Optimization of deep neural network : a functional perspective with applications in image classification

Simon Roburin

École des Ponts
ParisTech

THÈSE DE DOCTORAT

de l'École des Ponts ParisTech

Optimization of Deep Neural Networks: A Functional Perspective with Applications in Image Classification

École doctorale École Nationale des Ponts et Chaussées ParisTech: Mathématiques

Mathématiques Appliquées

Thèse préparée au sein du LIGM-IMAGINE
Financement: CIFRE, valeo.ai France

Thèse soutenue le 8 Novembre 2022, par
Simon Roburin

Composition du jury:

Patrick GALLINARI Sorbonne University , Criteo AI Lab	*Rapporteur*
Arnak S. DALALYAN ENSAE ParisTech - CREST	*Rapporteur*
Camille COUPRIE Meta	*Examinatrice*
Vicky KALOGEITON LIX, École Polytechnique/CNRS, Institut Polytechnique de Paris	*Examinatrice*
Mathieu AUBRY École des Ponts ParisTech	*Directeur de thèse*
Renaud MARLET École des Ponts ParisTech, valeo.ai	*Co-encadrant*
Patrick PÉREZ valeo.ai	*Co-encadrant*

École des Ponts ParisTech
LIGM-IMAGINE
6, Av Blaise Pascal - Cité Descartes
Champs-sur-Marne
77455 Marne-la-Vallée cedex 2
France

valeo.ai France
100 rue de Courcelles
75017 Paris France

Remerciements

En guise de préambule à ce manuscrit, je souhaiterais remercier toutes les personnes qui m'ont accompagné et soutenu au cours de ces trois années de dur labeur.

Je dédie ces travaux à mes parents qui par leurs sacrifices, leur amour et leur soutien indéfectible m'ont permis de conclure ces nombreuses années d'étude avec une thèse de doctorat.

Je tiens à remercier particulièrement mon trio de directeurs de thèse, Mathieu, Renaud et Patrick qui m'ont accompagné dans cette aventure. Renaud, pour sa bonne humeur, son humour ainsi que son application à relire et comprendre le moindre terme de mes innombrables équations. Patrick pour sa patience vis à vis de mes largesses concernant les tâches administratives, ses nombreux conseils et sa confiance en m'intégrant aux équipes de valeo.ai dès sa création. Et enfin Mathieu, pour son exemplarité tant éthique que professionnelle, son suivi pointilleux et sa bienveillance avec chacun de ses étudiants. Je remercie tous les membres du jury pour avoir pris le temps de s'échiner à relire mes travaux et de s'être libéré afin d'assister à ma soutenance, point d'orgue de ces trois dernières années.

Je remercie bien entendu les équipes des laboratoires entres lesquels j'ai oscillé, des rues pavées du 8ème arrondissement de Paris aux larges trottoirs de Champs sur Marne jusqu'aux grandes allées du campus de l'Ecole des Ponts. Chez valeo.ai, je pense évidemment aux collectif des Bezos, Charles, Arthur et Maxime, devenus des amis et sans qui l'expérience n'aurait sans aucun doute pas été aussi plaisante. Je remercie particulièrement Andrei pour ses conseils, ses suggestions toujours avisées et son soutien moral tout au long de cette thèse. Je remercie tous les membres de l'équipe avec qui j'ai pu partager de nombreux déjeuners et discussions au cours d'innombrables pauses café: Gabriel, Hédi, Matthieu, Eloi, Léon, Florent, Antoine, Alexandre, Laura, Bjorn, Spyros, Gilles, Oriane, Tuan-Hung, Himalaya, Huy. A l'École des Ponts, je remercie Tom pour ces heures passées à assurer conjointement dans la bonne humeur les TD de Deep Learning. Merci à Xi, François, Thibault, Michaël, Victor, Eliott, Abdou, Yuming, Yang, Benjamin, Théo pour leur disponibilité sans faille, leur sollicitude, et tous ces moments plaisants passés en leur compagnie des terrains de foot. Je remercie également mon ami et collègue Yann pour notre collaboration ayant mené à une première publication.

Enfin, je remercie évidemment mes fidèles amis Diego, Sabry, Corentin, Aziz et mon binôme André-Louis. Diego pour ces années d'amitié inaltérable, son humour et son accueil au cours de mes nombreuses escapades à Barcelone. Sabry pour nos ballades et discussions passionnantes à écumer les rues de Paris. Corentin pour nos explorations urbaines et nos longues nuits à sillonner les sentiers de la perdition. Aziz pour les moments de rigolade et de galères depuis le lycée, sa motivation communicative et sa générosité. André-Louis pour son enthousiasme dans cette compagnie de fortune dans le froid et la fatigue en Bretagne, dans la boue à Beynes et sur le sol brûlant à Toulon où nous faisions claquer les talons de ces chaussures qui nous broyaient les pieds. Je remercie également mon talentueux petit frère pour ses excellents cocktails, les nombreux repas partagés, son humour, son intelligence, et sa présence réconfortante. Un grand merci à Bobeik pour les micros aventures sans cesse renouvelées à Pigalle et au 82. Merci à Sophia pour ces nombreux cafés en terrasse. Je remercie les trois mousquetaires de Centrale, mes amis Manu, Sacha et Othmane pour notre épopée. Tous les étudiants qui ont côtoyé de près ou de loin cette chambre hors du temps qu'était la F221: Paul, Sofien, Larry, Doi. Je remercie bien évidemment mes frères d'armes de la section 4 et tout particulièrement les membres de la fine du 11. Un immense merci à tous mes guerriers, Amir, Marvin, Raph, Valentin, Saïku, Vincenzo, Marius, François, Romain, pour les coups échangés au cours de nos nombreux sparring et notre mentor Greg pour sa pédagogie, son intelligence et son humour. Enfin, je remercie Sophie pour m'avoir soutenu et accompagné avec tendresse une grande partie de cette thèse; Clara pour ces petites virées à Caen; Charlyn pour avoir bravi en ma compagnie les sommets Pyrénéens. Je remercie Caroline qui m'a grandement aidé au cours de cette dernière ligne droite.

Abstract

Despite numerous successes in a wide range of industrial and scientific applications, the learning process of deep neural networks is poorly understood. Loosely speaking, learning aims at finding the network parameters that not only minimize the network errors on a set of training examples but also yield correct predictions on unseen data. Under the prism of optimization, it boils down to minimizing a high dimensional non-convex function. Generalization can generally be expected when one has access to very large datasets and assumes that both training examples and unseen data are sampled from identically independently distributed random variables. The goal of this thesis is to develop analytical tools to better understand neural network optimization and to improve the design of training algorithms in the context of image classification.

To better understand deep networks training, we raise the issue of comparing neural networks. Instead of tackling this task by direct comparison of the networks' parameters, we study it from the functional perspective. In other words, all our work is guided by the idea that looking at the encoded function rather than parameters enables to take a fresh look at existing open problems. In this manuscript, we focus on two problems: the impact of normalization layers on deep neural network training and generalization properties of neural networks when the test data has a distribution different from the one of the training data.

Firstly, to study the optimization of deep neural networks with normalization layers, we exploit the radial invariance that stems from adding normalization layers in deep neural network architectures. To this end, we build an analytic framework referred to as the spherical framework. Concretely, we use the radial invariance of groups of parameters, such as filters for convolutional neural networks, to translate their optimization steps on the L_2 unit hypersphere. This formulation and the associated geometric interpretation shed new light on the training dynamics. It allows us to show that normalization layers transform simple gradient-based optimization schemes in a way that is equivalent to a scheduling of the learning rate (*effective learning rate*) and a change in the optimization direction (*effective direction*). In addition, we demonstrate that, in the presence of normalization layers, performing simple stochastic gradient descent alone is actually equivalent to train using a variant of a more complex and adaptive algorithm: Adam. Finally, this analysis outlines phenomena that previous variants of Adam act on and their importance in the optimization process is experimentally validated. Last, by using the

tools of optimization constrained to manifolds, we introduce new variants of Adam that significantly improve performances in the context of images classification with CNNs.

Secondly, we aim at improving generalization of deep networks when train and test set have different distributions. In the context of image classification, we focus on the case where data can be split into a majority group that includes images with a correlation between a visual element and a class and a minority group that does not. A neural network trained by minimizing the average errors on the training set will likely use the previous correlation as a prediction rule. As a result the model may perform poorly if the correlation does not hold on the test data. Such correlations are called *spurious correlations*. This challenge is referred to as Group Robustness. To avoid such an issue, we develop a method that identifies relevant splits of the train set with coherent data distributions on which the neural network function performs unevenly. Extensive attempts have been made to develop methods improving worst-group accuracy, *i.e.*, the accuracy computed on the group of data that does not display the aforementioned correlation. In general, all previous approaches require to know the presence of the spurious correlation for each training input. Such information may be expensive to get or unknown *a priori*. We therefore address the challenge of improving group robustness without such annotations during training. To this end, we first propose to partition the training dataset into groups based on Gram matrices of features extracted from an identification model and then apply a robust optimization objective based on these pseudo-groups. In the realistic context where no group labels are available, our experiments show that our approach not only improves group robustness over standard training with Empirical Risk Minimization (ERM) but also outperforms all recent baselines.

Résumé

Au cours des dernières décennies, l'apprentissage automatique a trouvé un nouveau souffle grâce au développement foudroyant de l'apprentissage profond. Discipline construite sur une classe de modèles paramétrés, les réseaux de neurones profonds, le processus d'apprentissage consiste à trouver des paramètres permettant au réseau de minimiser ses erreurs sur une base d'exemples dans le but de généraliser ses prédictions à de nouvelles données. Il s'agit de l'optimisation des réseaux de neurones profonds. Bien qu'ayant conduit à des améliorations notoires dans de nombreux domaines d'applications, comme la reconnaissance d'image, vocale ou de texte, la compréhension des mécanismes mis en jeu lors de l'apprentissage reste très superficielle. Les progrès techniques et pratiques du domaine sont fulgurants alors qu'un cadre théorique unifié peine à voir le jour. L'objectif de cette thèse est d'élaborer différents outils d'analyse pour améliorer notre compréhension de l'optimisation des réseaux de neurones d'une part, et l'améliorer d'autre part.

Les travaux présentés dans ce manuscrit ont été guidés par une problématique majeure en apprentissage profond: la comparaison de différents réseaux neuronaux. Au lieu d'adopter le point de vue classique qui consiste à comparer les paramètres associés aux différents réseaux, nous privilégions l'analyse des fonctions encodées. Cette interrogation, présente en toile de fond dans chacune de nos contributions, a permis d'apporter un nouveau regard sur certains problèmes ouverts de la discipline. Précisément, nous nous sommes focalisés sur:

- l'impact des couches de normalisation sur l'optimization des réseaux de neurones profonds;
- la robustesse des réseaux de neurones profonds.

En guise de préambule, dans le chapitre 2, nous nous appliquons à introduire les outils nécessaires aux notions abordées dans les chapitres suivants. Il s'agit tout d'abord de formaliser mathématiquement le problème d'apprentissage paramétrique supervisé. Le langage de la statistique couplé à celui de l'optimisation permet une formulation du problème. Théoriquement, on suppose d'abord l'existence de variables aléatoires indépendantes identiquement distribuées (*i.i.d.*) dont les réalisations sont les données d'entrées et de sorties. Ensuite, l'objectif est de minimiser l'espérance de la fonction de coût (quantifie les erreurs de prédiction du réseau par rapport à la sortie attendue) selon les lois des variables aléatoires d'entrées et de sorties. En pratique, les lois susmentionnées sont inconnues. Par conséquent, le problème de minimisation est approché

par l'estimateur empirique de l'espérance: la valeur moyenne de la fonction de coût sur les données d'entraînement (ERM). En apprentissage profond, la classe de modèles utilisés est celle des réseaux de neurones. Ils sont composés d'un enchevêtrement successif de fonctions linéaires paramétrées suivies de non linéarités quant à elles non paramétrées. On obtient ainsi une fonction non linéaire qui projette les entrées dans l'espace des sorties pour réaliser ainsi une prédiction. Dans ce contexte, ERM est un problème d'optimisation en haute dimension, non convexe, accompagné de son lot de challenges et de difficultés. Le paysage d'optimisation (l'hypersurface de la fonction de coût dans l'espace des paramètres) est parsemé de nombreuses régions et points atypiques comme les points selles, minima locaux, vallées, falaises ou plateaux. L'enjeu est alors de manoeuvrer à travers cette topographie complexe afin d'atteindre un minimum global. Rappelons ici que le but *in fine* est de généraliser: les erreurs du modèle, bien que minimisées sur le jeu d'entraînement, doivent également être faibles sur de nouvelles données (le jeu de test). Ainsi, l'algorithme d'optimisation doit non seulement se frayer un chemin jusqu'à un minimum mais aussi en trouver un capable de généraliser. A ce jour, les outils mathématiques développés ne garantissent ni la convergence, ni la généralisation. Pourtant, les méthodes de descente de gradient stochastique donnent des résultats très satisfaisants en pratique. Il s'agit de suivre la direction pointée par l'opposé du gradient de la fonction de coût, par rapport aux paramètres du réseau, estimé sur une fraction du représentative du jeu de données (appelé batch) jusqu'à convergence tout en contrôlant l'amplitude des pas à l'aide d'un hyperparamètre appellé learning rate. A celà s'ajoute une myriade de techniques (détaillées en section 2.3) comme, la pénalisation L_2, le momentum, les méthodes à learning rate adaptatif ou le early stopping. Ces améliorations incrémentales ont permis d'obtenir des réseaux toujours plus performants à l'issue de l'apprentissage. Nous clôturons ce chapitre de revue de l'état de l'art par la présentation des couches de normalisation. Ces techniques consistent schématiquement à normaliser les données selon les moments statistiques d'ordre 1 et 2 de certaines portions des tenseurs de sorties des couches intermédiaires du réseau. Sans les couches de normalisation, l'entraînement de réseaux de neurones profonds complexes munis d'un grand nombre de paramètres est impossible. De ce fait, elles se sont petit à petit imposées dans toutes les architectures modernes. Essentiellement introduites pour des raisons heuristiques, leur impact quantitatif sur l'entraînement des réseaux neuronaux reste un problème ouvert.

Dans le chapître 3, nous développons un nouvel outil d'analyse pour quantifier l'impact des couches de normalisations sur l'optimisation des réseaux de neurones. L'ajout de

couches de normalisation au sein d'une architecture induit une invariance fonctionnelle vis-à-vis de certains paramètres. Il s'agit de l'invariance radiale: toute multiplication par un scalaire du groupe de paramètres en question laisse la fonction encodée par le réseau de neurones inchangée. Les déplacements radiaux sont donc superflus en termes fonctionnels. Grâce au quotientage topologique de l'espace des paramètres du réseau par la relation d'équivalence associée à l'invariance radiale, on construit un espace qui reflète davantage les changements de fonction du réseau au cours de l'entraînement: l'hypersphère unité. Puisque l'entraînement des réseaux neuronaux est réalisé avec succès dans l'espace des paramètres, nous allons projeter les pas d'optimisation sur l'hypersphère unité. Cette formulation constitue un nouveau prisme pour analyser les couches de normalisations et comprendre, entre autres, le rôle des déplacements radiaux au cours de l'entraînement. Ainsi, nous montrons que les schémas d'optimisation classiques (descente de gradient stochastique avec ou sans pénalité, L_2, momentum ou Adam) sont transformés en schémas équivalents munis d'un learning rate adaptatif et d'une direction modifiée. Plus précisément, nous démontrons que, grâce aux couches de normalisation, utiliser une simple méthode de gradient stochastique est équivalent au fait d'optimiser le réseau avec une variante d'un schéma plus complexe, en l'occurrence: Adam. Ce résultat théorique est vérifié empiriquement avec une architecture moderne sur une tâche de classification d'images. Enfin, notre analyse permet également de mettre en lumière certains phénomènes géométriques quand l'algorithme Adam est utilisé pour optimiser un réseau de neurones muni de couches de normalisations. Après les avoir identifiés, nous utilisons les outils de l'optimisation contrainte à variété afin de proposer des variantes naturelles qui nulifient chacun des phénomènes précédents. Un volet expérimental avec différentes architectures et jeux de données de classification d'image, montre que, dans ce contexte, les variantes introduites améliorent les performances des réseaux obtenus et valide de ce fait, l'importance des phénomènes géométriques identifiés.

Le chapître 4 s'intéresse à l'optimisation des réseaux de neurones dans le cas ou les données dérogent à l'hypothèse *i.i.d.* Précisément, il s'agit de remarquer qu'il existe au sein de certains jeu de données des sous-groupes cohérents sur lesquels la fonction encodée par le modèle produit de faibles performances. Ce phénomène s'explique par l'existence de corrélations dites falacieuses dans un groupe majoritaire dans les données d'entrainement. Imaginons, par exemple, que l'on souhaite classifier des photos de vaches et de chameaux. En raison de biais de sélection évidents, la plupart des vaches sont représentées dans un environnement verdoyant alors que les chameaux auront tendance à se trouver davantage dans des contrées désertiques. Ainsi, une manière simple pour

discriminer les deux animaux est d'utiliser, non pas les caractéristiques propres de chacune des espèces, mais le milieu dans lequel ces dernières évoluent. On se pose le problème d'entrainer un modèle qui ne confondrait pas une vache dans un désert avec un chameau. De manière plus générale, il s'agit du challenge dit de robustesse par groupe. Certaines approches récentes ont développé des méthodes pour diminuer l'erreur de classification sur le pire groupe (un chameau sur de l'herbe dans l'exemple précédent), mais ces dernières requièrent toutes sans exception, l'information d'environnement *a priori* sur le jeu de données d'entraînement ou sur le jeu de validation. En général, ces informations supplémentaires sont soit coûteuses en annotation, ambigües (pensons à des données satellites par exemple avec de nombreuses métadonnées comme la localisation, l'heure ou l'année) voir même inconnues dans certains cas. Notre approche, propose un partitionnement du jeu de données en groupe grâce aux matrices de gram extraites d'un modèle exogène dit d'identification. Une fois clusterisé, on réalise un apprentissage robuste d'un modèle de classification grâce à un objectif d'optimisation régularisé par les pseudo groupes découverts lors de l'étape précédente. Un volet expérimental montre le succès de notre approche sur plusieurs jeux de données: non seulement nous améliorons la robustesse par groupe par rapport à un ERM standard mais nous surpassons en terme de performances toutes les méthodes récemment introduites.

Table of contents

List of figures

List of tables

Symbols

Roman Symbols

ℓ	loss function
$\mathcal{B}$	batches of input and target datas
$\mathcal{L}$	empirical training loss function
$\mathcal{M}$	manifold
$\mathcal{N}$	norm in finite vector space
$\mathcal{T}_{\mathbf{u}}(\mathcal{M})$	tangent space of the manifold $\mathcal{M}$ at point $\mathbf{u}$
$\mathbb{P}$	probability measure
$\mathbf{m}$	momentum
$\mathbf{s}$	input data
$\mathbf{t}$	target data
$\mathbf{u}$	parameter on the unit hypersphere
$\mathbf{v}$	order-2 moment
$\mathbf{x}$	parameters of the ML problem model
S	input random variables, each realisation is an input data
T	target random variables, each realisation is a target data

Greek Symbols

α	scaling parameter in BN

β momentum parameter

$\boldsymbol{\omega}$ parameters of the identification model

δ bias parameter in BN

ϵ small scalar quantity

η learning rate

γ geodesic

$\Gamma_{\mathbf{u}_{k-1}}^{\mathbf{u}_k}(\mathbf{a})$ parallel transport from $\mathbf{u}_{k-1}$ to $\mathbf{u}_k$ of $\mathbf{a}$

λ L_2 regularization parameter

μ mean

∇ gradient

ϕ model of the ML problem

π $\simeq 3.14\ldots$

ψ identification model

ρ scaling factor in the radial invariance

$\boldsymbol{\tau}$ intermediate batch output in the Deep Neural Network

σ standard deviation

Acronyms / Abbreviations

AI Artificial Intelligence

BN Batch Normalization

CNN Convolutional Neural Network

DL Deep Learning

DNN Deep Neural Network

EIIL Environement Inference for Invariant Learning

ERM Empirical Risk Minimization

GN Group Normalization

i.i.d. independently identically distributed

IN Instance Normalization

IRM Invariant Risk Minimization

JTT Just Train Minimization

LN Layer Twice

ML Machine Learning

MLP Multi Layer Perceptron

MMD Maximum Mean Discrepancy

NL Normalization Layer

SGD Stochastic Gradient Descent

SGD-M Stochastic Gradient Descent with Momentum

WD Weight Decay

WN Weight Normalization

Chapter 1

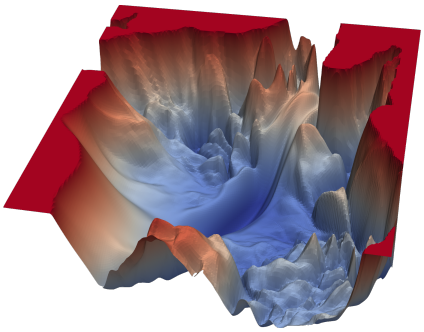
Introduction

1.1 Context

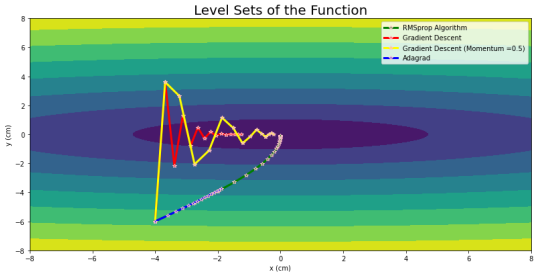
The stunning spread of Artificial Intelligence (AI) systems in a wide range of industrial and scientific applications[1] has been driven both by progresses in computer technology as well as significant improvements in Deep Learning (DL). DL is built upon Deep Neural Networks (DNNs). Loosely based on biological neural networks, DNNs are a class of high dimensional parametric models built from many basic computational blocks called neurons.

As an introductory example, let us consider the task of supervised image classification. The training of a DNN consists in finding the parameters that minimize its errors, i.e. misclassifcations, over the whole data distribution. In practice, we have: a DNN with parameters to select, a training set to train the DNN and a test set to assess its performance. Both train and test sets includes images with their associated classes. Then a tractable optimization objective is built to find the parameters that minimizes the DNN errors on the training samples. The overall purpose of the training process is to output a DNN that generalizes well i.e. results in low test error in practice. In other words, a desirable property for a DNN is to both classify correctly train and test images altought it has only been trained with train images. The particular procedure used to find these parameters is called a training algorithm, which relies on an optimization scheme. We differentiate:

[1] To grasp the impact of AI on today's world, the State of AI Report is published every year. It is a thorough synthesis of developments in research, industry and politics (see stateof.ai).



(a) Visualization of the loss surface.



(b) Comparison of trajectories from different optimization algorithms

Fig. 1.1 Loss surface and trajectories of training algorithms. In **(a)**, Li et al. (2018) provide a visualisation in a 3D space, for a 2D representation of the parameter space of the VGG-56 architecture (Simonyan and Zisserman, 2014) on the CIFAR-10 dataset (Krizhevsky et al., 2009). In **(b)**, a comparison of trajectories from different optimization algorithms on a simple 'bowl' shaped surface for 2 parameters represented by each of the axis is displayed. The values of the error function is visualized in color. The darker the color is the lower the error function values are. (*Image credit*: Tahavani).

- The optimization objective that defines what quantity is minimized. It is conveniently represented by a surface where each point is the value of the previous quantity for every single possible parameter (see illustration in Figure 1.1a).
- The training algorithm (or optimization scheme) that refers to how the quantity is minimized. It is conveniently represented by a discrete trajectory on the above surface (see illustration in Figure 1.1b on a different surface).

Despite the numerous successes of DL, there is a substantial divergence between theory and practice. While researchers have managed to improve drastically the performance of DNN over a variety of benchmarks, the theoretical analysis of the training process is lagging behind, often involving unrealistic assumptions and restricted settings. In this manuscript, we aim at reducing the gap between theory and practice by improving our understanding of specific mechanisms involved in DNNs training.

1.2 Motivations

The study of the training process of DNNs are motivated by the following points:

(a) Cow: 0.99, Pasture: 0.99, Grass: 0.99, No Person: 0.98, Mammal: 0.98

(b) No Person: 0.99, Water: 0.98, Beach: 0.97, Outdoors: 0.97, Seashore: 0.97

Fig. 1.2 Illustration of spurious correlation. Cows in canonical context (e.g. grass) are correctly classified while cows in rare contexts (e.g. water) are misclassified. Top five label and probability confidence produced by clarifai.com.

.

Understand the training mechanisms of DNNs. The optimization objective mentioned previously displays a tremendous amount of local minima with barely indistinguishable low error on the training data (see Figure 1.1a). This is commonly referred to as a non-convex optimization problem. In general, solving a non-convex optimization problem is NP-hard. Yet, practitioners have developed techniques that enable complex DNNs to be reliably trained with more or less simple training algorithms. Despite significant research efforts, our understanding of DNNs training mechanisms is still in its early stages.

To improve DNNs training. A myriad of optimization schemes have been developed to find the best network parameters. Among the parameters that minimize the training error, some of them generalize well, i.e. result in low test errors, while others lead to arbitrarily bad test performances. Unlike traditional optimization, what is of crucial interest is not so much whether the optimization scheme converges quickly to a minimum, but if it reaches a minimum with good generalization properties.

To avoid learning spurious correlations. With the advent of large-scale datasets, there is less and less control on the quality of data. Among others, there are growing concerns with data collection biases and confounding factors. Minimizing training errors leads DNNs into recklessly absorbing the correlations found in the training data. It is especially problematic if the trained DNN relies on spurious correlations, i.e. correlations that are not expected to hold in future test use cases. For example, imagine crowd-

sourcing images of cows. Due to selection biases, a high majority of cows stand in front of grass environments. A simple way to correctly classify images with cows would be to classify the background. This phenomena is observed empirically when training a DNN to perform object recognition (see Figure 1.2). It constitutes an undesirable shortcut that we want to avoid when training a DNN.

1.3 Goals

The concrete goals of this thesis are twofold: improve the understanding of the mechanisms behind DNNs training and develop new algorithms to enhance DNN generalization performance. More specifically, we focus on the following goals:

To understand better Normalization Layers (NLs) impact during training. Over the recent years, DNN architectures have significantly evolved, increasing at the same time both their complexity and performance. To ease the training of such complex architectures, a wide range of techniques have been introduced. Among them, Normalization Layers are the most critical: they have become necessary to train multiple modern architectures. Although essential, their impact on the training process is not fully understood and remains an active area of research.

To achieve Group Robustness. Spurious correlations are frequently observed in the following setting: data can be split into a majority group which includes a confounding factor and a minority group that does not. Under these circumstances, a DNN trained over the whole dataset will recklessly absorb these spurious correlation leading to bad performances on minority group. Our goal is to train a DNN with good performance both on the majority and minority group. This challenge is referred to as Group Robustness.

1.4 Challenges

This manuscript have been shaped by numerous challenges. Overall, they all relate to the following: comparing DNNs. Whether it is to assess DNN performance, or to understand DNNs dynamics during training, we are ineluctably brought to tackle the issue of comparing DNNs. There exists two main approaches to address this issue: comparison of the functions encoded by the DNNs or direct analysis of their associated parameters. Both approaches come with their fair share of challenges. Due to the high dimension of parameters, there exists no appropriate metric to efficiently compare the

DNN parameters. For the functional perspective, the choice of metrics and sets of data on which to compare DNNs constitutes a major issue. Yet distinct, both parameters and functional perspectives remain closely related. In chapter 3, we reconcile both views by analyzing training dynamics projected in the quotient space of the parameter space yielded by an invariance property of the functions encoded by DNN with NLs. In chapter 4, DNNs robust training and evaluation is performed by carefully choosing appropriate splits of the train and test sets.

1.5 Contributions

In response both to the goals and challenges introduced beforehand, our main contributions are as follows

The Spherical Framework. We introduce a generic framework that allows the analysis and comparison of any gradient based optimization scheme for DNNs with NLs. In short, we demonstrate the impact of NLs on optimization consists in a specific scheduling of the magnitude and direction of the steps by the training algorithm during the training phase. We also prove that, surprisingly, in presence of NLs, training with a simple stochastic gradient descent (SGD) is equivalent to using a variant of Adam a more complex and adaptive algorithm called Adam.

New optimization schemes for DNNs equipped with NLs. The Spherical Framework highlights geometrical phenomena occurring when training with Adam (Kingma and Ba, 2014). We leverage the tools of optimization constrained to a manifold to design variants of the training algorithm Adam. These variants consistently improves the generalization performance of the resulting DNNs over a variety of architectures and datasets.

Group Robustness with no labels. In the context of supervised image classification, we introduce a two-stage method that first partitions the training data into groups based on style features without any group supervision. Then a robust optimization objective is defined based on the previous discovered pseudo groups. The DNNs obtained with our method outperform with a significant margin all recent baselines addressing the problem of group robustness with no labels.

1.6 Outline

This manuscript revolves around the contributions set forth above and is organized as follows:

Chapter 2: Literature review Before delving into the core material of this manuscript, we present in chapter 2 an exhaustive overview of optimization in Deep Learning. All the necessary technical prerequisites to tackle the subsequent chapters are also introduced.

Chapter 3: Impact of Normalization Layers on Optimization This chapter introduces the first and second contributions of this thesis. The core idea is to leverage a shared property of DNNs with NLs: these architectures are invariant to a positive scaling of groups of parameters. We first provide a technical background on invariant models and optimization constrained to a manifold. With the help of the previous tools, we build the Spherical Framework and provide explicit expressions of the impact of NLs on the magnitude and direction of popular optimization schemes. We demonstrate the equivalence between SGD and a variant of Adam when training DNNs with NLs. Last, we empirically assess the influence of geometric phenomena in Adam highlighted by the Spherical Framework during the training process of DNNs with NLs. To this end, new variants of Adam are designed and we evaluate their performances over a variety of architectures and image classification datasets.

Chapter 4: Avoid learning spurious correlations This chapter introduces the third contribution of this thesis. First, we introduce existing attempts to define robust optimization objectives in order to perform Group Robustness. Although promising, these approaches all require group labels on the training set. We design an easy-to-scale method to split training images among distinct pseudo-groups, and train a robust optimization objective based on the pseudo-groups labels. After presenting, our method, we conduct extensive experiments on various image classification datasets with spurious correlations to show that our method outperforms every recent baselines.

Chapter 5: Conclusion We conclude this manuscript by summarizing the contributions of the thesis and discussing several potential perspectives for future work.

1.7 Publications

One journal article and one conference submission are presented in the manuscript:

- **Simon Roburin**[2], Yann de Mont-Marin[2], Andrei Bursuc, Renaud Marlet, Patrick Pérez and Mathieu Aubry. **Spherical Perspective on Learning with Normalization Layers**. Short version of the paper was accepted as a spotlight in Workshop on Optimization at *Advances in Neural Information Processing Systems (Neurips)* 2021 and long version was published as a journal article in *Neurocomputing* 2022. This is the subject of chapter 3 of this manuscript.

- **Simon Roburin**[2], Charles Corbières[2], Gilles Puy, Nicoles Thome, Mathieu Aubry, Renaud Marlet and Patrick Pérez. (2022) **Get One Gram of Neural Style Features, Get Enhanced Group Robustness**. Short version of the paper accepted in Workshop on Out of Distribution Detection at European Conference on Computer Vision (ECCV) 2022. This is the subject of chapter 4 of this manuscript.

During my PhD, I have also worked on an other project that is not presented in this thesis but that has led to a publication:

- Oriane Siméoni, Gilles Puy, Huy Van Vo, **Simon Roburin**, Spyros Gidaris, Andrei Bursuc, Patrick Pérez, Renaud Marlet and Jean Ponce. (2021) **Localizing Objects with Self-Supervised Transformers and no Labels**. In *British Machine Vision Conference (BMVC)*.

[2] Equal contribution

Chapter 2

Literature review

Abstract

In this chapter, a general overview of the training process of DNN is provided. First, the paradigm of statistical learning is formalized in section 2.1 under the scope of function approximation. The learning process, also referred to as the training of a DNN, is challenging. There exists no mathematical framework that explains the success of DNN and the error surface displays numerous complex regions that we discuss in section 2.2. However, a myriad of techniques has been developed in order to train properly DNN. Even if poorly understood, these techniques combined together allow DNNs to reach impressive performances on a wide variety of benchmarks. They all can be understood as defining general strategies to navigate on the loss surface until a model with good prediction performance is found. In particular, we enumerate the main training algorithms used in DL in section 2.3, explain the strategies to handle the learning rate and introduce the concept of early stopping to obtain minima with good generalization properties. Although these improvements are restricted to the training procedure, there exists also architecture modifications that improves significantly the training process. We conclude this chapter by providing an exhaustive review of one of the most prominent improvements in terms of architecture modification in DL: Normalization Layers (see section 2.4).

2.1 Preliminary

Deep Learning is a sub-field of a broader discipline: Machine Learning (ML). In order to solve a target task, usually too complex to be programmed from scratch, ML aims at extracting knowledge from data. Two main branches of the field are *supervised learning*

and *unsupervised learning*. In supervised learning, the ML algorithm dispose of both the inputs and expected outputs of the considered task whereas, in unsupervised learning only the inputs are available. In the manuscript, we will exclusively focus on supervised learning. A parametric ML problem is structured by the following mathematical objects:

- A class of parametric models;
- A set of samples;

The overall goal is to select among the parametric models, the model that better fits the data. The specificity of DL lies in the class of models used to solve the ML problem: DNNs. In this section, we first cover the main aspects of a supervised ML problem. Then we delve into the mathematical characteristics of a supervised parametric ML problem and introduce the basic concepts of optimization. Finally, we provide a succinct overview of DNN architectures and their associated functional expressivity. This section is inspired by the formalism developed both by Hastie et al. (2009) and Mallat (2020).

2.1.1 Statistical view of parametric machine learning problem

Whether we want to learn how to recognize objects, understand a spoken language or predict physical quantities, the underlying goal of any ML problem is to find the best candidate in a given class of parametric model that minimizes the model's loss function over the data distribution.

Statistical hypothesis. We denote respectively by $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ the training and test data. The latter are comprised of a finite number of input and output samples: $(\mathbf{s}, \mathbf{t}) \in \mathbb{R}^{\text{in}} \times \mathbb{R}^{\text{out}}$. ML problems are built upon the following fundamental hypothesis: the pairs $(\mathbf{s}, \mathbf{t}) \in \mathbb{R}^{\text{in}} \times \mathbb{R}^{\text{in}}$ are the realisation of independently and identically distributed (*i.i.d.*) random variables (S, T) according to some probability measure $\mathbb{P}_{(S,T)}$ over $\mathbb{R}^{\text{in}} \times \mathbb{R}^{\text{out}}$. We will throughout assume that the corresponding σ-algebra is a product of Borel σ-algebras w.r.t. the usual topologies. The statistical hypothesis ensures the existence of a common causal link between inputs and expected outputs both in $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$. Without such assumptions, it will be harder to find a model that generalizes from train samples ($\mathcal{D}_{\text{train}}$) to samples unseen during training ($\mathcal{D}_{\text{test}}$).

Definitions and context. We define the hypotheses space by a family of parametric model $\phi_x : \mathbb{R}^{\text{in}} \longrightarrow \mathbb{R}^{\text{out}}$ with parameters $\mathbf{x} \in \mathbb{R}^d$ where d is the number of parameters of the model. The loss function $\ell : \mathbb{R}^{\text{out}} \times \mathbb{R}^{\text{out}} \longrightarrow \mathbb{R}$ quantifies the discrepancy between a

model's prediction $\phi_{\mathbf{x}}(\mathbf{s})$ and the expected output $\mathbf{t}$. Overall, the purpose is to exploit the training data $\mathcal{D}_{\text{train}}$ in order to select the best candidate in $\mathcal{H}$ that that minimize the average loss over the data distribution $\mathbb{P}_{(S,T)}$. To assess the performance of a trained model, we disposes of a set of *unseen* input and output samples: $\mathcal{D}_{\text{test}}$.

Theoretical problem formulation. We want to find the parameters $\mathbf{x} \in \mathbb{R}^d$ that minimize the difference between $\phi_{\mathbf{x}}(\mathbf{s})$ and $\mathbf{t}$ over the data distribution $\mathbb{P}_{(S,T)}$. To this end we aim at finding the following quantity:

$$\arg\min_{\mathbf{x}\in\mathbb{R}^d} \mathbb{E}_{\mathbb{P}_{(S,T)}}\Big[\ell(\phi_{\mathbf{x}}(S),T)\Big] = \arg\min_{\mathbf{x}\in\mathbb{R}^d} \int_{\mathbb{R}^{\text{in}}\times\mathbb{R}^{\text{out}}} \ell(\phi_{\mathbf{x}}(\mathbf{s}),\mathbf{t})\mathbb{P}_{(S,T)}(d\mathbf{s},d\mathbf{t}). \tag{2.1}$$

Tractable optimization objective. Note that in practice, $\mathbb{P}_{(S,T)}$ is uknwon. To circumvent this issue, we construct the following estimator:

$$\widehat{\mathbb{E}}_{\mathbb{P}_{(S,T)}}\Big[\ell(\phi_{\mathbf{x}}(S),T)\Big] = \frac{1}{\#\mathcal{D}_{\text{train}}} \sum_{(\mathbf{s},t)\in\mathcal{D}_{\text{train}}} \ell(\phi_{\mathbf{x}}(\mathbf{s}),t), \tag{2.2}$$

where # denotes the cardinality of a set. We define the empirical training loss function by:

$$\mathcal{L} : \mathbf{x} \in \mathbb{R}^d \mapsto \sum_{(\mathbf{s},t)\in\mathcal{D}_{\text{train}}} \ell(\phi_{\mathbf{x}}(\mathbf{s}),t) \in \mathbb{R}. \tag{2.3}$$

Now, we can rewrite Eq 2.1 with a tractable optimization objective. Called Empirical Risk Minimization (ERM), it reads:

$$\text{argmin}_{\mathbf{x}\in\mathbb{R}^d}\mathcal{L}(\mathbf{x}) = \text{argmin}_{\mathbf{x}\in\mathbb{R}^d}\Big[\frac{1}{\#\mathcal{D}_{\text{train}}} \sum_{(\mathbf{s},t)\in\mathcal{D}} \ell(\phi_{\mathbf{x}}(\mathbf{s}),t)\Big]. \tag{2.4}$$

Generalization. The Holy Grail of ML is to find the parameters that minimize Eq 2.1. In practice, we are reduced to minimize Eq 2.4. A desirable property of a solution to Eq 2.1 is to yield small loss values not only on the training data $\mathcal{D}_{\text{train}}$ but also on the whole data distribution. Formally, for any parameters $\mathbf{x} \in \mathbb{R}^d$, it consists in narrowing the following difference:

$$\Big|\mathbb{E}_{\mathbb{P}_{(S,T)}}\Big[\ell(\phi_{\mathbf{x}}(S),T)\Big] - \mathcal{L}(\mathbf{x})\Big|. \tag{2.5}$$

The lower the quantity in Eq 2.5 is, the better the associated learned model $\phi_{\mathbf{x}}$ generalizes. Conversely, if the quantity in Eq 2.5 is high, the learned model overfits or underfits. In practice, we can approximate Eq 2.5 with the test set by the following:

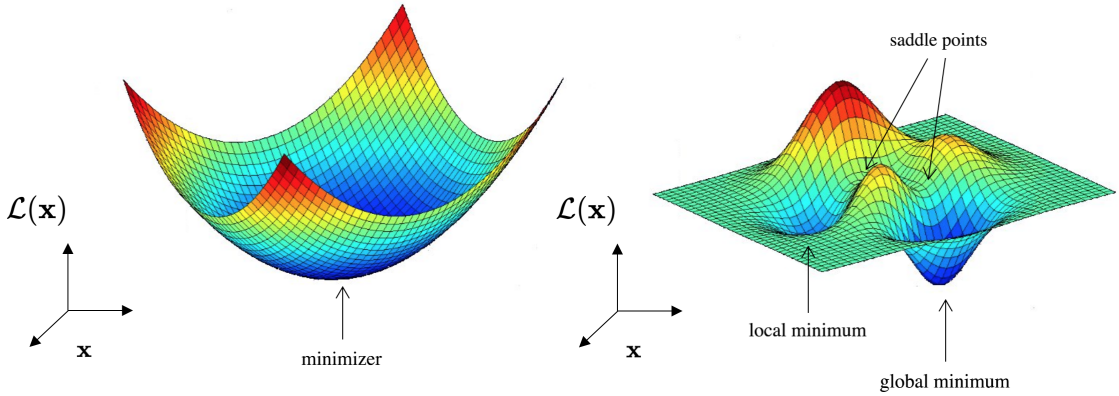


Fig. 2.1 Convex vs. non-convex function and critical points. We provide a 3D illustration of the 2D surface of a convex function (left) and a non-convex function (right) as well as the different critical points. *Image credit*: Kolev (2011).

.

$$\Big| \frac{1}{\#\mathcal{D}_{\text{test}}} \sum_{(\mathbf{s},t)\in\mathcal{D}_{\text{test}}} \ell(\phi_{\mathbf{x}}(\mathbf{s}),t) - \mathcal{L}(\mathbf{x}) \Big|. \tag{2.6}$$

2.1.2 Optimization: basic concepts

The learning process formalized in Eq 2.4 is an optimization problem. Here we detail the basic concepts of optimization.

Convexity vs non-convexity. An optimization problem is convex *iff* its objective function is convex. A function $\mathcal{L}$ is convex w.r.t the parameters $\mathbf{x}$ *iff* the line segment between two points $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{\text{in}}$ lies above the graph $\mathcal{G}$ of $\mathcal{L}$ (defined by $\mathcal{G} = \{(\mathbf{x}, \mathcal{L}(\mathbf{x})), \mathbf{x} \in \mathbb{R}^{\text{in}}\}$) between the two points. Otherwise, the function is non-convex (see illustration in Fig 2.3).

Critical points. Here, we assume that the objective function $\mathbf{x} \mapsto \mathcal{L}(\mathbf{x})$ is positive and almost everywhere twice differentiable. Note that this hypothesis is verified in most DL formulations. Let us consider a parameter $\mathbf{x} \in \mathbb{R}^d$ at which $\mathcal{L}$ is twice differentiable. If $\nabla\mathcal{L}(\mathbf{x}) \neq 0$, there exists a descent direction that leads to a smaller function value. By following the descent direction, we might reach a parameter $\mathbf{x}^*$ that verifies $\nabla\mathcal{L}(\mathbf{x}^*) = 0$. Depending on the eigenvalues of the Hessian of $\mathcal{L}$ as well as the values of $\mathcal{L}(\mathbf{x}^*)$, $\mathbf{x}^*$ is called either a local minimum, a global minimum or a saddle point (see illustration in Figure 2.1).

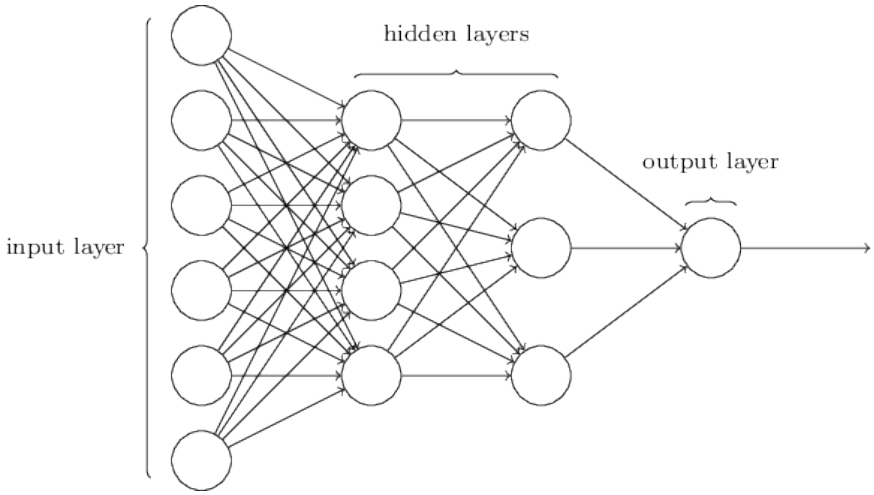


Fig. 2.2 Illustration of a MLP. We illustrate a 3-layer neural network with two hidden layers, the first one has four neurons and the second one has three neurons. There is no connection between the neurons of the same layer. *Image credit*: YashNita Github.

.

2.1.3 Deep Learning architectures

We provide a very succinct introduction to Deep Learning models. For an extensive coverage of the subject, we refer the reader to Goodfellow et al. (2016).

Inspired by a simplified model of a biological neuron, a DNN is composed of a succession of linear or non linear functions. These functions are called *layers*. Formally, for any $\mathbf{s} \in \mathbb{R}^{\text{in}}$, and a fixed parameter $\mathbf{x} \in \mathbb{R}^d$, it reads:

$$\phi_{\mathbf{x}}(\mathbf{s}) = \phi^{(L)}_{\mathbf{x}^{(L)}} \circ \phi^{(L-1)}_{\mathbf{x}^{(L-1)}} \circ \cdots \circ \phi^{(1)}_{\mathbf{x}^{(1)}}(\mathbf{s}) \in \mathbb{R}^{\text{out}}, \tag{2.7}$$

where, for all $l \in [\![1, L]\!]$ $\mathbf{x}^{(l)} \mapsto \phi^{(l)}_{\mathbf{x}^{(l)}}$ are the function layer with L the number of layers, $\mathbf{x}^{(l)} \in \mathbb{R}^{d_l}$ and $\mathbf{x} = [\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(L)}]$ the associated parameters. Note that for all $l \in [\![1, L]\!], \mathbf{x}^{(l)} \mapsto \phi^{(l)}_{\mathbf{x}^{(l)}}$ are differentiable almost everywhere. The latter property enables the computation of the intermediate gradients w.r.t. each parameter individually. As a consequence, for all $\mathbf{s} \in \mathbb{R}^{\text{in}}$, the function $\mathbf{x} \mapsto \phi_{\mathbf{x}}(\mathbf{s})$ is differentiable almost everywhere. It allows us to easily compute the gradients w.r.t. the network parameters via chain rule. The intermediate layers are called *hidden layers* and can be interpreted as a non linear projector of the input data. The number of layers L is called the *depth* of the DNN architecture.

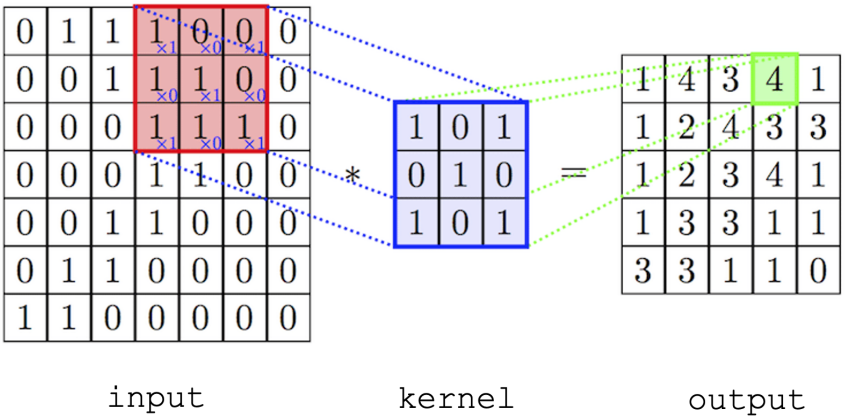


Fig. 2.3 Illustration of a convolution operation. A 2D input tensor is convolved by a kernel (blue) that produces a scalar value (green). The kernel results in a 2D feature maps once it has sliced the entire input. *Image credit:* Sun et al. (2020).

.

Multi Layer Perceptron (MLP). Initially introduced by Minsky and Papert (1969), an MLP is a type of DNN architecture in which the hidden layers are a parametric linear transformation of the input followed by a non-parametric linear transformation called an activation function. The last layer is a parametric linear transformation that is a mapping to the output space $\mathbb{R}^{\text{out}}$. Each node is called either *unit* or *neuron* (see illustration in Figure 2.2). The typical activations are the sigmoid functions, the hyperbolic tangent or the Rectified Linear Unit (ReLU), the latter being the most popular.

Convolutional Neural Networks (CNNs). CNNs (LeCun et al., 1989) substitutes the simple linear projection in MLPs by a convolutional layer. Still falling in the class of linear transformation, a convolutional layer convolves a kernel, called a *filter*, over the entire input. The convolution operation assumes the input has a specific structure. In this manuscript, we focus only on 2D-convolution on images (see illustration in Figure 2.3). Yet, note that there exists 3D-convolution (for LiDAR data for instance) or 1D-convolution (used in the case of text data). It a performs local scalar product between a group of local values and the kernel, producing a scalar value. The scalar values are then successively arranged to create an output tensor. This operation is particularly well suited to high dimensional data, structured along a vertical and an horizontal axis with translation invariant patterns, such as images.

Approximation Theorems. Here, we shortly summarize the main results in terms of approximation capabilities of DNNs: Hornik et al. (1989) show that the hypothesis class of MLPs, with a width of arbitrary size (maximum number of *neurons* in the architecture) and at least one hidden layer, is dense in the set of continuous functions on compact subset of $\mathbb{R}^{\text{in}}$; Yarotsky (2022) demonstrates an analogous result for CNNs and the set of functions invariant by translation. Although hinting us on expressive capacities of DNNs, the previous results assume an exponentially-large number of parameters which is infeasible in practice.

2.2 Challenges of training DNNs

Training DNN is the particular procedure that consists in minimizing Eq 2.1 w.r.t. the parameters of the network. It is accompanied by its fair share of challenges. Existing mathematical tools actually fail to explain the success of DNN suggesting that the current angle chosen to tackle learning problems must be reconsidered. In this section, we cover the main aspects of the encountered difficulties when training DNNs. First, we highlight what makes learning different from traditional optimization. Then, we detail the inherent problems of high-dimension function approximation. Last, an overview of the topography of the regions encountered in the training loss landscape is provided.

2.2.1 Learning differs from traditional optimization

Solving a ML/DL problem differs from traditional optimization. In most ML problems, the object of interest is usually a *performance measure* defined w.r.t. the *test set* (Hastie et al., 2009).

First, the test set is not accessible during training. The model is learned by exploiting data from the training set. Unlike traditional optimization, what is of crucial interest, is not so much whether we quickly reach a minimum of the empirical risk but if we reach a parameter value with good generalization properties (Goodfellow et al., 2016).

Second, in some cases, the performance measure cannot be optimized efficiently. To circumvent this issue, a surrogate loss function that acts as a proxy and does not suffer from the previous flaw, is defined. This is in contrast to traditional optimization where minimizing the loss function is a goal of itself. For instance, in the context of image classification, the performance measure is the accuracy (ratio of correctly classified samples with the total number of samples); the latter is also called the 0-1 loss. Marcotte

and Savard (1992) demonstrate that minimizing directly the 0-1 loss is intractable, even for linear classifiers. In this case, the negative log-likelihood of the correct class is used as a substitute for the 0-1 loss.

2.2.2 High dimension of the input data and black box analysis

Coarsely, training a DNN consists in finding the parameters in a *high-dimensional* space that yield to a low expected loss over an *unknown* data distribution (see Eq 2.1). In terms of dimension, the order of magnitude of the dimension of high-quality images is about 10^6. To obtain a good predictor, Signal Processing teaches us that samples have to be close to each other, and the speed of variations of the predictor between two samples must be bounded (Mallat, 2016). Yet, in such a high dimensional space, samples are typically far from each other. Imagine, that one wants to cover the unit hypercube in high dimension with a distance of ϵ between samples. To achieve that, we need $\epsilon^{-\text{in}}$ samples ! With $\epsilon = 10^{-1}$, it would require 10^{10^6} samples which is higher than the number of atoms in the observable universe. Needless to say, that it is infeasible in practice. In addition, the probability distribution $\mathbb{P}_{(S,T)}$ is unknown. It makes any attempt of theoretical analysis of the training process difficult without any additional hypothesis. There exists no clean mathematical characterisation of the loss landscape, this is referred to as *black-box analysis*.

2.2.3 High number of parameters and non-convexity of ERM

The training loss function of DNNs (cf. Eq 2.4) displays two serious flaws: it is at the same time highly-dimensional and highly non convex.

Blum and Rivest (1992) demonstrate that finding a global minimizer for a simple 2-layers, 3-neurons MLP is already an NP-hard problem. In modern DNNs, the typical number of parameters is $d = 10^8$. In such a context, it is infeasible to find global minima to Eq 2.1. Indeed, it would require to explore the whole parameter space $\mathbb{R}^d$. One can show that in $\mathbb{R}^d$, there exists $\exp d/\epsilon$ directions such that all directions have angle at most ϵ with one of these. In addition, traditional Learning Theory predicts that a model with such a high capacity leads to poor generalization performances (Bartlett and Mendelson, 2002; Bousquet and Elisseeff, 2002; Vapnik, 1998). Last, there exist transformations in the parameter space called symmetries $T : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ that leave the training loss function unchanged: $\forall \mathbf{x} \in \mathbb{R}^d, \mathcal{L}(T(\mathbf{x})) = \mathcal{L}(\mathbf{x})$ (Kornblith et al., 2019). These symmetries yield

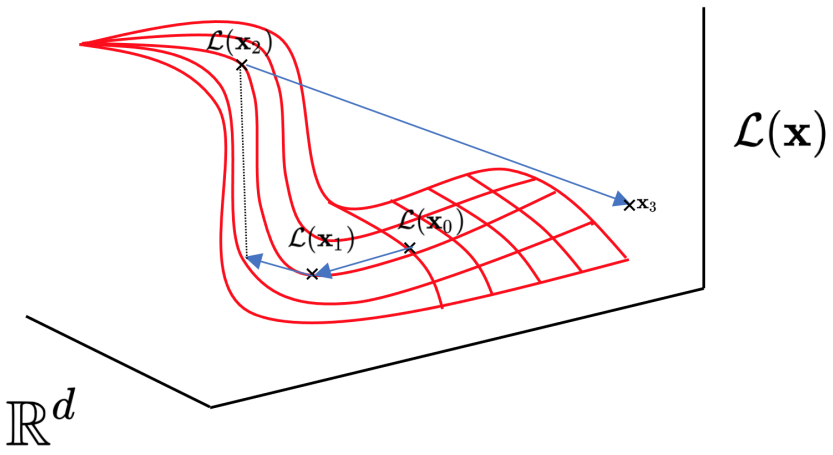


Fig. 2.4 Exploding steps in a cliff region. Starting from $\mathbf{x}_0$ with a fixed learning rate η, optimization steps bring the parameters close to a cliff region. At $\mathbf{x}_1$, by following the gradient direction, the parameters reach $\mathbf{x}_2$. From $\mathbf{x}_3$, the loss function gradient is high due to the steepness of the cliff region. Updates are therefore catapulted far away from the local minima in the bottom of the cliff.

.

to areas of constant loss values in the parameter space, thus to a large number of local minima.

The optimization objective also displays saddle points, cliffs or flat regions. In particular, motivated by arguments introduced in Statistical Physics, Dauphin et al. (2014) show empirically that a DNN optimization objective (ERM) displays a very large amount of saddle points compared to the number of local minima. The widespread use of non-smooth activation functions, gives rise to gradients with high magnitude in some regions. If, during the training process, the parameters come close to such a cliff, following the descent direction could catapult far away the parameters, making the whole process highly unstable (see Figure 2.4). Conversely, we also observe vanishing gradients that corresponds to region of the parameter space in which gradients are very small. As a result, parameters can be stuck in such a region while still displaying relatively high loss value.

2.3 Training techniques in Deep Learning

In response to the previous challenges, a myriad of techniques has been developed to properly train DNNs. In practice, DNNs reach surprisingly good performance on a high variety of benchmarks with relatively simple training algorithms. Yet, this behaviour is not universal: the trainability of DNNs is highly dependent on a variety of good practices. In this section, we cover the main strategies to properly train DNNs. We first delve into different parameter initialization strategies. Then, we introduce popular training algorithms, explain how to handle the amplitude of the steps throughout training. In addition, we highlight the stopping criterion referred to as early stopping in order to select parameters with low generalization gap (Eq. 2.5). Last, we detail the mechanisms of popular Normalization Layers that aim at improving the training of complex DNN architectures.

2.3.1 Parameter initialization

The stability of the training process of DNNs seems to heavily rely on the strategy chosen to initialize parameters of the network (Glorot and Bengio, 2010). Parameter initialization strategies in DL are mainly heuristic and little is known on how to properly initialize DNNs. The idea consists in avoiding vanishing and exploding gradients at the beginning of the training. To achieve that, Glorot and Bengio (2010) introduce an initialization strategy that aims at controlling both the variance of the activations and the variance of the gradient w.r.t. activations to avoid saturing parts of the non linearities. At a layer $l \in [\![1, L]\!]$, the parameters $\mathbf{x}^{(l)} \in \mathbb{R}^{d_l}$ are initialized according to the following uniform distribution $\mathcal{U}(-\frac{2}{d_{l-1}+d_l}, \frac{2}{d_{l-1}+d_l})$. When ReLU activations function are used, He et al. (2015) observe that the forward and backward pass act as if the parameters had half their variances which motivates multiplying the parameters by $\sqrt{2}$.

2.3.2 Gradient based optimization schemes

The optimization scheme also called training algorithm, is the cornerstone of DNNs training. It is the computational procedure used to select parameters $\mathbf{x} \in \mathbb{R}^d$ that minimize ERM (Eq 2.4). First-order method all rely on the same basic principle: follow the descent direction, i.e. the opposite of the gradient until a predefined stopping criterion is met. This technique is called *gradient descent*. The empirical training loss function $\mathbf{x} \mapsto \mathcal{L}(\mathbf{x})$ is assumed to be almost everywhere differentiable. Note that there also exists second-order methods based on approximate computation of the Hessian (Agarwal et al.,

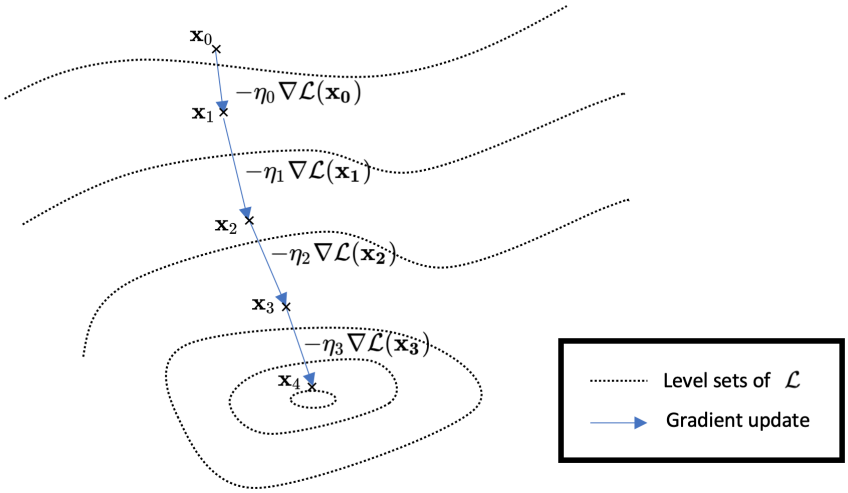


Fig. 2.5 Illustration of the gradient descent algorithm. Representation of the level sets of the training loss function in a 2D parameter space.

.

2017; Martens et al., 2010; Martens and Grosse, 2015) but they do not yield better minima compared to simpler first-order methods. Therefore, in this manuscript, we only focus on first-order optimization schemes. Formally, starting from a parameter initialization $\mathbf{x}_0 \in \mathbb{R}^d$ for all iterations $k \geq 0$, it reads:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \nabla \mathcal{L}(\mathbf{x}_k), \tag{2.8}$$

where $\eta_k \in \mathbb{R}$ is called the learning rate. We illustrate the mechanisms of gradient descent in Figure 2.5.

Stochastic Gradient Descent. In practice, the number of inputs in the training dataset is very large. To compute efficiently the empirical loss, the computational power of parallel computing in Graphical Processors Units (GPUs) is leveraged. Due to memory limitations, at each training step $k \in \mathbb{N}$, the gradient of the empirical training loss is evaluated on a different random portion of the training dataset $\mathcal{B}_k \subset \mathcal{D}_{\text{train}}$ called a *batch*. The resulting training algorithm is called Stochastic Gradient Descent (SGD). For all training iterations $k \geq 0$, it reads:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \nabla \mathcal{L}_{\mathcal{B}_k}(\mathbf{x}_k), \tag{2.9}$$

$$\mathcal{L}_{\mathcal{B}_k}(\mathbf{x}_k) = \frac{1}{\#\mathcal{B}_k} \sum_{(\mathbf{s},t)\in\mathcal{B}_k} \ell(\phi_{\mathbf{x}}(\mathbf{s}), t). \tag{2.10}$$

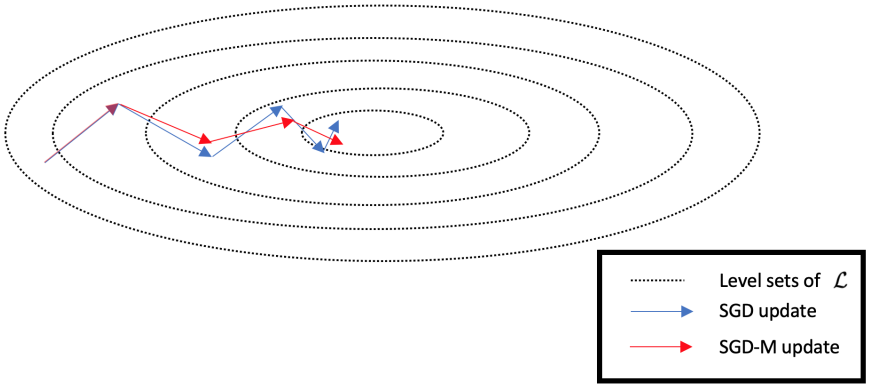


Fig. 2.6 Illustration of the effect of Momentum. Representation of the level sets of the training loss function in a 2D parameter space.

.

Depending on the *batch size* (i.e. $\#\mathcal{B}_k, \forall k$), the estimate of the empirical training loss is more or less noisy. In addition to its practical benefits in terms of memory usage, Ge et al. (2015) demonstrates that, under smoothness hypothesis of the empirial loss function $\mathbf{x} \mapsto \mathcal{L}(\mathbf{x})$, SGD escapes saddle points in a polynomial number of iterations. Therefore, SGD ensures the convergence to a local minimum. In the rest of the manuscript, to avoid heavy notations, we use $\mathcal{L}$ to actually denote the training loss function evaluated on a random batch of the dataset $\mathcal{L}_{\mathcal{B}_k}$ when referring to the gradient estimate used at an iteration k of the training procedure.

L_2 regularization. In traditional ML, L_2 regularization aims at mitigating overfitting. It consists in encouraging the optimization algorithm to reach solutions with small L_2 norm. Formally, it takes the form of as an additional term to the training loss function: $\lambda \|\mathbf{x}\|_{L_2}^2$ where λ (usually picked in $[0, 1]$) controls the amplitude of the L_2 regularization term. Combined with SGD, by differentiating the L_2 regularization term, we obtain the following update rule for all $k \geq 0$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k(\nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k) \tag{2.11}$$

Yet, when training DNNs the role of L_2 regularization appears to be less clear. Notably, Krizhevsky et al. (2012) observes not only improved test accuracy, but also training accuracy when using L_2 regularization. In chapter 4, we provide an extensive analysis of the impact of L_2 regularization when training DNNs with Normalization Layers (NLs).

Momentum. Despite SGD being a popular training algorithm in DL, it turns out to be quite slow. Introduced by Polyak (1964), momentum accumulates an exponentially decaying moving average of past gradients and follow the resulting direction. As a result, momentum accelerates drastically learning when confronted with high curvature, noisy gradients or small but consistent gradients in terms of direction. Combined with SGD, the resulting optimization scheme is dubbed SGD with Momentum (SGD-M). Its updates are:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \mathbf{m}_k, \tag{2.12}$$

$$\mathbf{m}_{k+1} = \beta \mathbf{m}_k + \nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k, \tag{2.13}$$

where $\mathbf{m}_k$ is the momentum, $\mathbf{m}_0$ (usually null in practice) the initial momentum, $\beta \in [0, 1]$ the momentum parameter and $\lambda \in [0, 1]$ is the L_2 regularization parameter. Eq 2.13 can be rewritten as: $\mathbf{m}_k = \beta^k \mathbf{m}_0 + \sum_{i=0}^{k} \beta^{k-i-1} \Big(\nabla \mathcal{L}(\mathbf{x}_i) + \lambda \mathbf{x}_i \Big)$. In SGD, the amplitude of the step is monitored by the learning rate η. In SGD-M, the size of the step depends on how large and how aligned the sequence of the past gradients are. The momentum parameter β controls the memory of the past gradients' direction and amplitude. In the extreme case where $\beta = 1$, there is no attenuation, each past gradient has the same contribution. If $\beta < 1$, the lower beta is, the lesser the past gradients influence the current step. In addition, there is a coupling of momentum and L_2 regularization parameters β and λ. To ease the search of best hyper parameters β and λ, Loshchilov and Hutter (2017) design a decoupled variant of SGD-M, dubbed SGD-W. Its updates are:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \mathbf{m}_k + \lambda \mathbf{x}_k, \tag{2.14}$$

$$\mathbf{m}_{k+1} = \beta \mathbf{m}_k + \nabla \mathcal{L}(\mathbf{x}_k). \tag{2.15}$$

2.3.3 Learning rate scheduling and adaptive optimization schemes.

Among all the hyperparameters, the learning rate η_k is undoubtedly one of the most difficult to set. A large learning rate can cause instability, whereas a too small learning rate can yield slow progress or cause being trapped in a bad local minimum. To address this issue, two main approaches were designed: *learning rate scheduling* and *adaptive optimization schemes*.

Learning rate scheduling. The particular procedure used to find a good learning rate value is described as follows: one tries a range of learning rate values and picks one that makes the training loss decrease significantly. Once the training loss plateaus, the

learning rate is decreased. Then, the previous steps are repeated until convergence is reached. This process is called a step-wise learning rate scheduling. The sequence $k \mapsto \eta_k$ is hence piece-wise constant. Note that there exists more exotic learning rate scheduling. For instance, Loshchilov and Hutter (2016) introduces periodical restarts of the learning rate followed by a cosine annealing decrease. As a results, cosine annealing scheduler improves slightly the performance of the obtained DNNs.

Since scheduling the learning rate during training is laborious and relies mainly on empirical heuristics, smarter optimization algorithms were introduced. These algorithms adapt the learning rate during learning for every parameter. They are called *adaptive optimization schemes*. We cover here the most popular adaptive training algorithms.

AdaGrad (Duchi et al., 2011). The training algorithm AdaGrad adapts the learning rates of every parameter in the network. Formally, the learning rate per parameter is scaled by the inverse of the square root of the sum of the past gradients' norm. The latter quantity is called the order-2 moment. The parameters with the largest gradients are associated to a rapid decrease in learning rate, while parameters with small gradients get a relatively small decrease in their learning rate. AdaGrad updates are:

$$\mathbf{v}_k = \mathbf{v}_{k-1} + (\nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k)^2, \tag{2.16}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \Big(\nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k \Big) \oslash \sqrt{\mathbf{v}_k + \epsilon}, \tag{2.17}$$

where $\mathbf{v}_k$ is the second-order moment (usually $\mathbf{v}_0$ is null), $\oslash$ is the element wise division and ϵ prevents division by zero. (Here and in the following, the square and the square root of a vector are to be understood as element-wise). Even if AdaGrad enjoys desirable convergence properties in the context of convex optimization, empirically, AdaGrad performs well for some but not all DL models. The accumulation of past gradient norm without decay can potentially cause premature decay of the amplitude of the steps.

Adam (Kingma and Ba, 2014). The optimization scheme Adam is a version of AdaGrad with momentum as well as additional correction terms to correct the estimator by exponential moving average of the momentum and the gradient's norm. Its updates

are:

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)(\nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k), \tag{2.18}$$

$$\mathbf{v}_k = \beta_2 \mathbf{v}_{k-1} + (1-\beta_2)(\nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k)^2, \tag{2.19}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \frac{\mathbf{m}_k}{1-\beta_1^{k+1}} \oslash \sqrt{\frac{\mathbf{v}_k}{1-\beta_2^{k+1}} + \epsilon}, , \tag{2.20}$$

where $\mathbf{m}_k$ is the momentum with parameter β_1 (usually $\mathbf{m}_0$ is null) , $\mathbf{v}_k$ is the second-order moment with parameter β_2 (usually $\mathbf{v}_0$ is null) , and ϵ prevents division by zero. Analogously to momentum, Adam estimates both order-1 (direction of the past gradients) and order-2 (norm of the past gradients) with an exponential moving average. To unbias the previous estimators using the exponential moving average, correction terms are introduced to account for the initialization of both $\mathbf{m}_0$ and $\mathbf{v}_0$, respectively $\frac{1}{1-\beta_1^{k+1}}$ and $\frac{1}{1-\beta_2^{k+1}}$. Again, β_1 and β_2 control the memory of their corresponding past moments. Empirically, Adam is fairly robust to the choice of the learning rate, making it one of the most popular training algorithm in DL.

Similarly to SGD with SGD-W, Loshchilov and Hutter (2017) introduces a decoupled version of Adam dubbed Adam-W. It reads for all $k \geq 0$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \left(\frac{\mathbf{m}_k}{1-\beta_1^{k+1}} \oslash \sqrt{\frac{\mathbf{v}_k}{1-\beta_2^{k+1}} + \epsilon} + \lambda \mathbf{x}_k \right), \tag{2.21}$$

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)\nabla \mathcal{L}(\mathbf{x}_k), \tag{2.22}$$

$$\mathbf{v}_k = \beta_2 \mathbf{v}_{k-1} + (1-\beta_2)(\nabla \mathcal{L}(\mathbf{x}_k))^2 \tag{2.23}$$

2.3.4 Early stopping

Optimization schemes aims at minimizing ERM (Eq 2.4) on the training set $\mathcal{D}_{\text{train}}$. *In fine*, the goal is to reach a minimum with good generalization properties. Quantitatively, we want a small generalization gap (Eq. 2.5). Since the test set $\mathcal{D}_{\text{test}}$ is not accessible, the training set can be split into two disjoint sets: $\mathcal{D}'_{\text{train}}$ and $\mathcal{D}_{\text{valid}}$. The performance measure (e.g., the accuracy in the context of classification) is computed on the validation set $\mathcal{D}_{\text{valid}}$. Since, the validation loss might increase while the train loss continues its decrease, the model associated to the parameters with highest validation accuracy during training is kept at the end of the training. This process is referred to as *early stopping*.

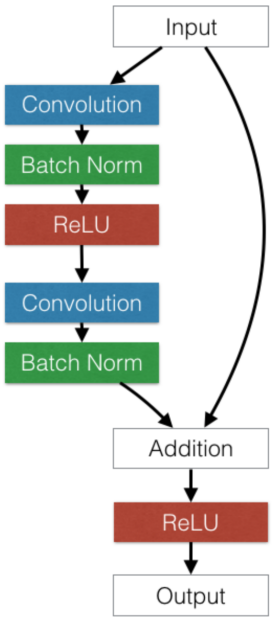


Fig. 2.7 BN in ResNet architecture. Each BN layer is arranged between the convolutional layer and the ReLu non linearity. *Image credit*: He et al. (2016a)

.

2.4 Normalization Layers

To ease the training of more and more complex architectures, significant efforts were made by researchers. Normalization Layers have drastically improved training of such highly parametric architectures. NLs are layers that are arranged between linear layers and the activation functions within DNN architectures (see illustration in Figure 2.7). Loosely speaking, they all consist in normalizing and centering the batch output of the linear layer before feeding it through the activation function. One of the most prominent is Batch Normalization (BN) (Ioffe and Szegedy, 2015). It improves significantly both the training speed and the prediction performance but has however a notable shortcoming: BN relies heavily on the batch size. To avoid such a dependency, normalization layers as LayerNorm (LN) (Ba et al., 2016), WeightNorm (WN) (Salimans and Kingma, 2016), InstanceNorm (IN) (Ulyanov et al., 2016) or GroupNorm (GN) (Wu and He, 2018) were introduced. In this section, we focus on NLs in the case of CNNs. Note that, without loss of generality, a linear layer can be expressed as a 1×1 convolution. Let us consider a layer with intermediate batch output $\boldsymbol{\tau} \in \mathbb{R}^{B \times C \times D}$ where B is the batch size, C the number of channels and D the spatial extension. We introduce the following notations for any slice of the tensor for all $(b, c, j) \in [\![1, B]\!] \times [\![1, C]\!] \times [\![1, D]\!]$ $\boldsymbol{\tau}_{b,c,\cdot} \in \mathbb{R}^D, \boldsymbol{\tau}_{b,\cdot,j} \in \mathbb{R}^C, \boldsymbol{\tau}_{\cdot,c,j} \in \mathbb{R}^B$

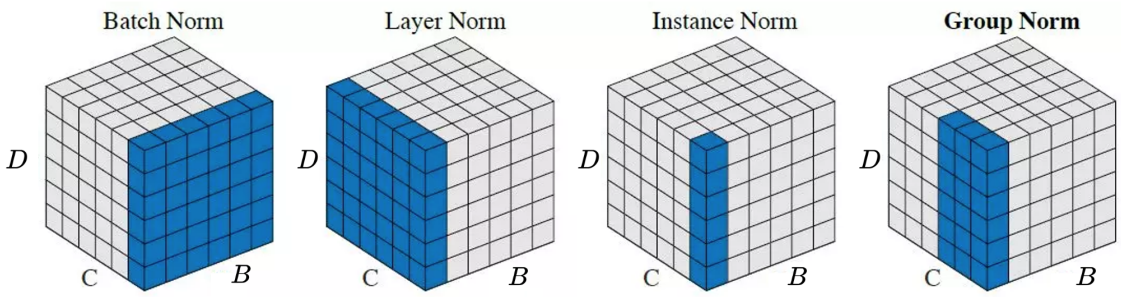


Fig. 2.8 Normalization Layers. Each subplot displays a input tensor with B as the batch axis, C the channel axis and D the spatial extension axis. The blue pixels are normalized by the same mean and standard deviations. *Image credit*: Wu and He (2018)

.

Batch Normalization (BN). BN normalizes and centers intermediate convolutional output along the batch and spatial dimension. It computes C means and standard deviations and applies the normalization to each channel independently. Formally, for all $c \in [\![1, C]\!]$, and all $b \in [\![1, B]\!]$ it corresponds to:

$$\mu_c = \frac{1}{BD} \sum_{b,j} \tau_{b,c,j} \tag{2.24}$$

$$\sigma_c^2 = \frac{1}{BD} \sum_{b,j} (\tau_{b,c,j} - \mu_c)^2 \tag{2.25}$$

$$\boldsymbol{\tau}^{(BN)}_{b,c,\cdot} \overset{\text{def}}{=} (\sigma_{\mathbf{c}}{}^2 + \epsilon)^{\frac{-1}{2}} (\boldsymbol{\tau}_{b,c,\cdot} - \mu_c \mathbf{1}_D) \in \mathbb{R}^D, \tag{2.26}$$

where $\mathbf{1}_D$ denotes the all-ones vector of dimension D and ϵ is a small constant used for numeric stability. Once normalized, the output is scaled as follows:

$$\forall (b, j) \in [\![1, B]\!] \times [\![1, D]\!], \alpha \odot \boldsymbol{\tau}^{(BN)}_{b,\cdot,j} + \delta, \tag{2.27}$$

where $\alpha \in \mathbb{R}^C$, $\delta \in \mathbb{R}^C$ are learnable parameters and $\odot$ is the element-wise mutliplication. At test time, BN uses running estimates of the mean and the standard deviation over all the training samples instead of μ_c and $\sigma_{\mathbf{c}}{}^2$.

Instance Normalization (IN). Based on a similar mechanism as BN, IN incorporates a slight difference. It normalizes and centers each channel and each batch independently.

The statistics are computed along the spatial dimension. It reads:

$$\mu_{b,c} = \frac{1}{D}\sum_j \tau_{b,c,j} \tag{2.28}$$

$$\sigma^2_{b,c} = \frac{1}{D}\sum_j (\tau_{b,c,j} - \mu_c)^2 \tag{2.29}$$

$$\boldsymbol{\tau}^{(IN)}_{b,c,\cdot} \overset{\text{def}}{=} (\sigma^2_{b,c} + \epsilon)^{\frac{-1}{2}}(\boldsymbol{\tau}_{b,c,\cdot} - \mu_{b,c}\mathbf{1}_D) \in \mathbb{R}^D, \tag{2.30}$$

Unlike BN, at run time, IN does not use the running estimates of the statistics accumulated during training. Instead, IN uses instance statistics from the considered test input at run time. The scaling and shift of the output after the IN layer is performed for each element of the batch and each channel independently whereas with BN it is applied only to each channel independently.

Layer Normalization (LN). LN normalizes and center each instance of a batch independently and computes its statistics along the spatial and channel dimension. It reads:

$$\mu_b = \frac{1}{DC}\sum_{c,j} \tau_{b,c,j} \tag{2.31}$$

$$\sigma^2_b = \frac{1}{DC}\sum_j (\tau_{b,c,j} - \mu_b)^2 \tag{2.32}$$

$$\boldsymbol{\tau}^{(LN)}_{b,c,\cdot} \overset{\text{def}}{=} (\sigma^2_b + \epsilon)^{\frac{-1}{2}}(\boldsymbol{\tau}_{b,c,\cdot} - \mu_b\mathbf{1}_D) \in \mathbb{R}^D, \tag{2.33}$$

Like IN, LN uses instance statistics at test time. As BN, it rescales and shift the previous output for each channel independently.

Group Normalization (GN). Similarly to LN, GN also computes its statistics along the spatial and channel dimensions but divides it into groups of fixed size and normalizes each group independently (see illustration in Fig 2.8). As BN, it rescales and shifts the previous output for each channel independently. To avoid heavy notations, we refer to the original paper (Wu and He, 2018) for a detailed review of the algorithm.

Weight Normalization (WN). WN (Salimans and Kingma, 2016) is simply a reparametrization of the parameters in a DNN that decouples their length from their direction. In the case of CNNs let us consider a layer $l \in [\![1, L]\!]$. The associated convolutional layer is parameterized by $\mathbf{x}^{(l)} \in \mathbb{R}^{F\times C\times K}$ where F is the number of filters, C the

number of channels of the intermediate input and K the kernel size of each filter. For each filter $f \in [\![1, F]\!]$, the parameters actually used in the convolution reads:

$$\frac{\mathbf{g}^{(l)}}{\|\mathbf{x}^{(l)}_{f,\cdot,\cdot}\|}\mathbf{x}^{(l)}_{f,\cdot,\cdot} \in \mathbb{R}^{C\times K}, \tag{2.34}$$

where $\mathbf{g}^{(l)} \in \mathbb{R}^F$ are learnable parameters that control the magnitude of the parameters. Note that unlike BN, there is no scaling and shift of the previous output, but only a decoupling of the length and direction of the parameters.

2.5 Conclusion

Although appearing, by many aspects, as a challenging task, training DNN has been made possible thanks to significant research efforts. Yet, many gray areas remain. The absence of any unified theoretical framework hardens any attempt of improvement of the optimization process. Significant breakthroughs to ease the training of more and more complex architectures have been achieved but are mostly backed up by researcher's intuition rather than formal theoretical analyses. Overall, existing mathematical tools fail to explain the success of DNN. It stresses out the urge to reconsider the mathematical framework to tackle learning problems. In chapter 3, we introduce an analytical tool built upon a mathematical invariance to analyse and compare the effects of NLs on the most popular optimization schemes. Despite its recent successes, training DNN still need to be improved in practice. To this end, we first introduce new optimization schemes suited to DNN equipped with NL in chapter 3. Last, in chapter 4, we introduce a new method to train DNN in the difficult context where data are not sampled from a single distribution.

Chapter 3

Impact of Normalization Layers on Optimization

The work described in this chapter is based on the following publication: **Simon Roburin**, Yann de Mont-Marin, Andrei Bursuc, Renaud Marlet, Patrick Pérez and Mathieu Aubry. **Spherical Perspective on Learning with Normalization Layers**.

- Short version of the paper was accepted as a spotlight in Workshop on Optimization at *Advances in Neural Information Processing Systems (Neurips)* 2021.
- Long version was published as a journal article in *Neurocomputing* 2022.

.

Abstract

Normalization Layers (NLs) are widely used in modern deep-learning architectures. Despite their apparent simplicity, NLs' effect on optimization is not yet fully understood. This chapter starts with a technical background on invariances in DNN and cover the fundamental concepts of quotient space as well as the basics of Riemannian Geometry. Then, we introduce the spherical framework to study the optimization of neural networks with NLs from a geometric perspective. Concretely, the radial invariance of groups of parameters, such as filters for convolutional neural networks, allows to translate the optimization steps on the L_2 unit hypersphere. This formulation and the associated geometric interpretation shed new light on the training dynamics. Firstly, the first effective learning rate expression of Adam is derived. Then the demonstration that, in the presence of NLs, performing SGD alone is actually equivalent to a variant of Adam

constrained to the unit hypersphere, stems from the framework. Finally, this analysis outlines phenomena that previous variants of Adam act on and their importance in the optimization process are experimentally validated.

3.1 Introduction

Inspite of significant research efforts, the optimization of deep neural networks is still poorly understood. Deep Neural Network (DNN) training involves minimizing a high-dimensional non-convex function, which has been proved to be a NP-hard problem (Blum and Rivest, 1989). Yet, elementary gradient-based methods show good results in practice. To improve the quality of reached minima, numerous NLs have stemmed in the last years and become common practices. Batch Normalization (BN) (Ioffe and Szegedy, 2015) is undoubtedly the most noteworthy. It improves drastically both the training speed, the prediction performance and allows deeper networks to be trained properly; BN has however a notable shortcoming: it relies heavily on the batch size. To alleviate the dependency w.r.t the batch size, normalization layers such as LayerNorm (LN) Ba et al. (2016), WeightNorm (WN) Salimans and Kingma (2016), InstanceNorm (IN) Ulyanov et al. (2016), or GroupNorm (GN) Wu and He (2018) were introduced. We refer the reader to section 2.4 for an extensive presentation of NLs. Despite their success, the interaction of NLs with optimization remains an open research topic.

Previous studies highlighted some of the mechanisms of the interaction between BN and Stochastic Gradient Descent (SGD), both empirically (Santurkar et al., 2018) and theoretically (Arora et al., 2019; Bjorck et al., 2018; Hoffer et al., 2018b). But none of them provides a generic framework, nor studies the interaction between NLs and the Adam optimizer (Kingma and Ba, 2014). In this chapter, we provide an extensive analysis of the relation between NLs and any order-1 optimization scheme. Moreover, we theoretically relate SGD with NLs to a variant of Adam (AdaGradG). It is of high interest since Adam is probably the most commonly-used adaptive scheme for Neural Networks (NNs). A shared effect of all mentioned NLs is to make NNs invariant to positive scalings of groups of parameters. These groups of parameters may differ from one NL method to another. The core idea of this chapter is precisely to focus on these groups of radially-invariant parameters and analyze their optimization projected on the L_2 unit hypersphere (see Figure 3.1), which is topologically equivalent to the quotient manifold of the parameter space by the scaling action. In fact, one could directly optimize parameters on the hypersphere as Cho and Lee (2017). Yet, most optimization methods

are still performed successfully in the original parameter space. Here we propose to study an optimization scheme for a given group of radially-invariant parameters through its image scheme on the unit hypersphere. This geometric perspective sheds light on the interaction between normalization layers and Adam.

The chapter is organized as follows. In **section** 3.2, we provide background on radial invariance in DNN, explicit its consequence on the parameter space via the concept of quotient space and introduce basic concepts of Riemanian Geometry. In **section 3.3**, we introduce our spherical framework to study the optimization of any radially-invariant model. We also define a generic optimization scheme that encompasses methods such as SGD with momentum (SGD-M) and Adam. We then derive its image step on the unit hypersphere, leading to definitions and expressions of *effective learning rate* and *effective learning direction*. These new definitions are explicit and have a clear interpretation, whereas the definition of van Laarhoven (2017) is asymptotic and the definitions of Arora et al. (2019) and of Hoffer et al. (2018b) are variational. In **section 3.4**, we leverage the tools of our spherical framework to demonstrate that, in presence of NLs, SGD is equivalent to AdaGradG, a combination of AdaGrad Duchi et al. (2011) (a special case of Adam without momentum) and AdamG (Cho and Lee, 2017) (a variant of Adam constrained to the unit hypersphere). In other words, AdaGradG is a variant of Adam without momentum and constrained to the unit hypersphere. In **section 3.5**, we analyze the effective learning direction for Adam. The spherical framework highlights phenomena that previous variants of Adam (Cho and Lee, 2017; Loshchilov and Hutter, 2019) act on. We perform an empirical study of these phenomena and show that they play a significant role in the training of convolutional neural networks (CNNs). In **section 3.6**, these results are put in perspective with related work.

Our main contributions are the following:

- A framework to analyze and compare order-1 optimization schemes of radially-invariant models;
- The first explicit expression of the effective learning rate for Adam;
- The demonstration that, in the presence of NLs, standard SGD is equivalent to AdaGradG, a variant of Adam without momentum and constrained to the unit hypersphere;
- The identification and the study of geometrical phenomena that occur with Adam and that impact significantly the training of CNNs with NLs.

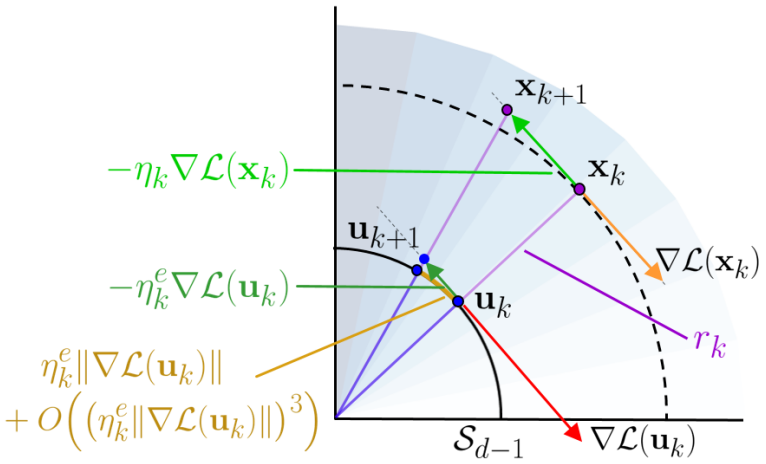


Fig. 3.1 Illustration of the spherical perspective for SGD. The loss function $\mathcal{L}$ of a NN w.r.t. the parameters $\mathbf{x}_k \in \mathbb{R}^d$ of a neuron followed by a BN is radially invariant. The neuron update $\mathbf{x}_k \to \mathbf{x}_{k+1}$ in the original space, with velocity $\eta_k \nabla \mathcal{L}(\mathbf{x}_k)$, corresponds to an update $\mathbf{u}_k \to \mathbf{u}_{k+1}$ of its projection through an exponential map on the unit hypersphere $\mathcal{S}_{d-1}$ with velocity $\eta_k^e \|\nabla \mathcal{L}(\mathbf{u}_k)\|$ at order 2 (see details in section 3.3). Note that in this figure, just as in section 2.3, to avoid heavy notations we use $\mathcal{L}$ to denote the training loss function evaluated on a random batch of the dataset $\mathcal{L}_{\mathcal{B}_k}$.

3.2 Technical background

In this section, we cover the main mathematical tools used to build our analytical framework to quantify the impact of NL on DNN optimization. We first give background on the radial invariance. Then, we detail how this radial invariance stems from the use of NL in DNN architectures. The concept of quotient space by the equivalence relation associated to radial invariance is leveraged to define a space that better reflects the model function than the full parameter space. Once built, we cover the basic intuitions behind Riemannian geometry to introduce the tools needed to perform projections.

3.2.1 Radial invariance

General case. We consider a family of parametric functions $\phi_{\mathbf{x}} : \mathbb{R}^{\text{in}} \longrightarrow \mathbb{R}^{\text{out}}$ parameterized by a group of radially-invariant parameters $\mathbf{x} \in \mathbb{R}^d \backslash \{\mathbf{0}\}$, i.e., $\forall \rho > 0, \phi_{\rho \mathbf{x}} = \phi_{\mathbf{x}}$, a dataset $\mathcal{D}_{\text{train}}$ comprised of a finite number of input and output samples:$(\mathbf{s}, \mathbf{t}) \in \mathbb{R}^{\text{in}} \times \mathbb{R}^{\text{out}}$, a loss function $\ell : \mathbb{R}^{\text{out}} \times \mathbb{R}^{\text{out}} \to \mathbb{R}$ and a training loss function $\mathcal{L} \colon \mathbb{R}^d \to \mathbb{R}$ defined as:

$$\mathcal{L}(\mathbf{x}) \overset{\text{def}}{=} \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\mathbf{s},\mathbf{t}) \in \mathcal{D}_{\text{train}}} \ell(\phi_{\mathbf{x}}(\mathbf{s}), \mathbf{t}). \tag{3.1}$$

It verifies: $\forall \rho > 0,\ \mathcal{L}(\rho \mathbf{x}) = \mathcal{L}(\mathbf{x})$. By using the previous, equality, the definition of the differential of $\mathcal{L}$ as well as the composition of differentials, we obtain the following lemma that states that the gradient of a radially-invariant loss function is tangential and -1 homogeneous (see proof in appendix 1):

Lemma 1 (Gradient of a function with radial invariance)**.** *If $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ is radially invariant and almost everywhere differentiable, then, for all $\rho > 0$ and all $\mathbf{x} \in \mathbb{R}^d$ where $\mathcal{L}$ is differentiable, we have:*

$$\langle \nabla \mathcal{L}(\mathbf{x}), \mathbf{x} \rangle = 0 \quad \text{and} \quad \nabla \mathcal{L}(\mathbf{x}) = \rho \, \nabla \mathcal{L}(\rho \mathbf{x}). \tag{3.2}$$

3.2.2 Radially invariant parameters in DNN with NL.

In the context of DNN equipped with NL, not all groups of parameters are radially invariant. The set of parameters $\mathbb{R}^d$ can be split into two disjoint sets $\mathbb{R}^d = \mathcal{F} \cup \mathcal{R}$ where $\mathcal{F}$ is the set of groups of radially-invariant parameters and $\mathcal{R}$ is the set of remaining parameters. Note that the set of remaining parameters in $\mathcal{R}$ differs from one architecture to another depending on the considered NLs. The following notations are restricted to this section only.

DNNs with BN. In this paragraph, we show the radial invariance of a set of filters equipped with BN. For the sake of simplicity, we only consider the case of a convolutional layer that preserves the spatial extension of the input. We also focus on a single filter. Since all filters act independently on input data, the following calculation holds for any filter. Let $\mathbf{x} \in \mathbb{R}^{C \times K}$ be the parameters of a single filter, where C is the number of input channels and K is the kernel size. During training, this layer is followed by BN and applied to a batch $\mathbf{s} \in \mathbb{R}^{B \times C \times D}$ of B inputs of spatial size D. The output of the convolution operator ϕ applied to a filter $\mathbf{x} \in \mathbb{R}^{C \times K}$ and to a given batch element $\mathbf{s}_b \in \mathbb{R}^{C \times D}$, with $b \in [\![1, B]\!]$, is thus:

$$\boldsymbol{\tau}_b \overset{\text{def}}{=} \phi(\mathbf{x}, \mathbf{s}_b) \in \mathbb{R}^D. \tag{3.3}$$

The application $(\mathbf{x}, \mathbf{s}_b) \mapsto \phi(\mathbf{x}, \mathbf{s}_b)$ is bilinear. BN then centers and normalizes the output

$\mathbf{t}$ using the mean and variance over the batch and the spatial dimension:

$$\mu = \frac{1}{BD} \sum_{b,j} \tau_{b,j}, \tag{3.4}$$

$$\sigma^2 = \frac{1}{BD} \sum_{b,j} (\tau_{b,j} - \mu)^2, \tag{3.5}$$

$$\boldsymbol{\tau}_b^{(\mathrm{BN})} \stackrel{\mathrm{def}}{=} (\sigma^2 + \epsilon)^{-1/2} (\boldsymbol{\tau}_b - \mu \mathbf{1}_D), \tag{3.6}$$

where $\mathbf{1}_D$ denotes the all-ones vector of dimension D and ϵ is a small constant.
Now if the coefficients of the filter are rescaled by $\rho > 0$, then, by bilinearity, the new output of the layer for this filter verifies:

$$\tilde{\boldsymbol{\tau}}_b = \phi(\rho \mathbf{x}, \mathbf{s}_b) = \rho \phi(\mathbf{x}, \mathbf{s}_b). \tag{3.7}$$

Since the variance of inputs is generally large in practice, for small ϵ, the mean and variance are:

$$\tilde{\mu} = \rho \mu, \tag{3.8}$$

$$\tilde{\sigma}^2 \approx \rho^2 \sigma^2. \tag{3.9}$$

It can then be considered that the subsequent BN layer is invariant to this rescaling, i.e., $\tilde{\boldsymbol{\tau}}_b^{(\mathrm{BN})} \approx \boldsymbol{\tau}_b^{(\mathrm{BN})}$. The model, at the convolutional layer is therefore radial invariant w.r.t. every single filter. In other words the set of radially parameters $\mathcal{F}$ is precisely the set of all convolutional filters in the network.

Other NLs. The above calculus is analogous for other normalization schemes. Only the considered group of radially invariant parameters differs from one architecture to another. The radial invariance for BN described above applies as well to IN and WN as the normalization is also done with respect to channels but without the batch dimension. Regarding LN, the normalization is performed over all channels and the entire weight layer can thus be rescaled too, without impacting the output. As for GN, it associates several channels for normalization; the radial invariance in this case concerns the corresponding group of filters. Thanks to this general property of radial invariance, the results in this chapter not only concern BN but also WN and IN. In fact, they apply as well to LN and GN when considering the suitable group of parameters. The optimization in this case concerns the proper slice of the parameter tensor of the layer, i.e., the whole tensor for LN, and the selected group of filters for GN.

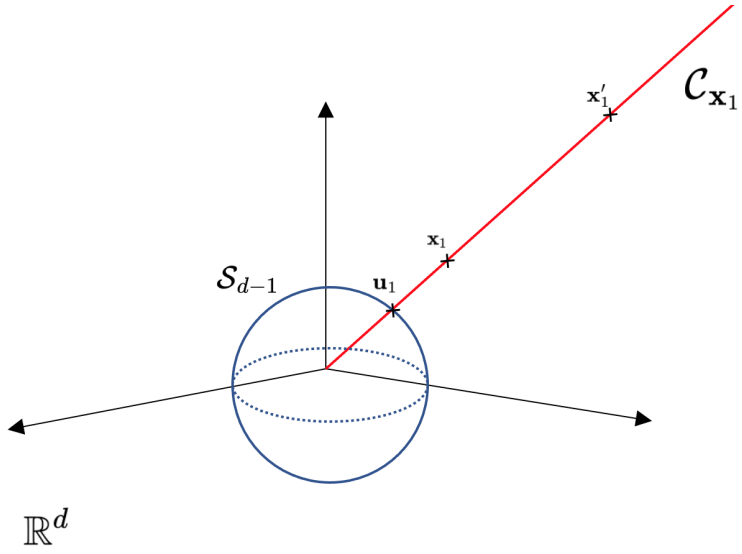


Fig. 3.2 3D illustration of equivalent classes and unit hypersphere. The ray $\mathcal{C}_{\mathbf{x}_1}$ is the equivalent class of the parameter $\mathbf{x}_1$. The parameter $\mathbf{x}_1'$ and $\mathbf{u}_1$ yield the same model and hence loss function value. To avoid such a redundancy, we map each parameter of $\mathbb{R}^d$ belonging to $\mathcal{C}_{\mathbf{x}_1}$ to $\mathbf{u}_1 \in \mathcal{S}_{d-1}$. By repeating the previous mapping for all $\mathbf{x} \in \mathbb{R}^d$, it leads to the unit hypersphere $\mathcal{S}_{d-1}$.

In the rest of this chapter, all results and calculus only concerns radially invariant parameters. In contrast to section 2.1.1, for the purpose of notations, we use $\mathbf{x} \in \mathbb{R}^d$ to denote a group of radially parameters of dimension d. The scope of our analysis is restricted to the set of identified radially invariant parameters in the corresponding DNN with NL architecture.

3.2.3 Quotient of the parameter space and hypersphere

The radial invariance implies redundancy among the parameters in $\mathbb{R}^d$. All parameters that belong to the same ray starting from the origin result in the same model function and hence the same loss function. Now, we construct a subset of the parameter space $\mathbb{R}^d$ in which each parameter corresponds to a different function value. Constructing such a set is of interest since it will better reflect the model function associated to the parameters. To achieve that, we need to quotient the parameter space by the equivalence relation associated to radial invariance. Intuitively, each element of the set defined by $\{\mathbf{x} \in \mathbb{R}^d | \mathcal{N}(\mathbf{x}) = 1\}$ is enough to represent the parameter space (where $\mathcal{N}$ is the Euclidean norm associated to the scalar product). Formally, the equivalence relation

associated to radial invariance $\mathcal{R}$ is defined by:

$$\forall \mathbf{x}, \tilde{\mathbf{x}} \in \mathbb{R}^d \setminus \{0_{\mathbb{R}^d}\}, \forall \rho > 0, \mathbf{x} \mathcal{R} \tilde{\mathbf{x}} \Leftrightarrow \mathbf{x} = \rho \tilde{\mathbf{x}}. \tag{3.10}$$

The quotient space by the equivalence relation associated to radial invariance is denoted $\mathbb{R}^d \setminus \{0_{\mathbb{R}^d}\}/\mathcal{R}$ and defined by the set of all equivalent classes:

$$\forall \mathbf{x} \in \mathbb{R}^d \setminus \{0_{\mathbb{R}^d}\}, \mathcal{C}_{\mathbf{x}} = \left\{\tilde{\mathbf{x}} \in \mathbb{R}^d | \mathbf{x} \mathcal{R} \tilde{\mathbf{x}}\right\}. \tag{3.11}$$

The quotient space by the equivalence relation associated to radial invariance $\mathbb{R}^d \setminus \{0_{\mathbb{R}^d}\}/\mathcal{R}$ is topologically equivalent to $\{\mathbf{x} \in \mathbb{R}^d | \mathcal{N}(\mathbf{x}) = 1\}$. Note that, since we aim at analyzing the optimization process, the choice of the norm $\mathcal{N}$ is of high importance. We consider here the L_2 sphere $\mathcal{S}_{d-1} = \{\mathbf{u} \in \mathbb{R}^d / \|\mathbf{u}\|_2 = 1\}$ whose canonical metric corresponds to angles: $d_{\mathcal{S}}(\mathbf{u}_1, \mathbf{u}_2) = \arccos(\langle \mathbf{u}_1, \mathbf{u}_2 \rangle)$. This choice of metric is relevant to study NNs since filters in CNNs or neurons in MLPs are applied through scalar product to input data. Besides, normalization in NLs is also performed using the L_2 norm directly in WeightNorm (Salimans and Kingma, 2016), and indirectly in other NLs because the standard deviation can be interpreted as a centered L_2 norm. To give a better intuition of the above-mentionned mathematical objects, we provide an illustration in Figure 3.2.

3.2.4 Riemannian geometry

We previously built a set that better reflects the model function associated to the parameters, which boils down to the hypersphere $\mathcal{S}_{d-1}$. From now on, we could directly optimize parameters on the unit hypersphere. Yet, order-1 optimization schemes already produce successful results in NL-equipped DNN architectures. In order to analyze their success and quantify the impact of NLs on the optimization of radially-invariant parameters, we are going to project the successive updates on the L_2 unit hypersphere during training. Precisely, when updating parameters via an optimization step in the parameter space we want to quantify the equivalent step taken on the unit hypersphere. To characterize properly such an equivalent step on $\mathcal{S}_{d-1}$, which is a nonlinear space, we need to introduce the basic concepts of Riemannian geometry. This theory articulates around the concept of Riemannian manifold as well as the notion of Riemannian distance. Overall, Riemannian geometry aims at extending methods from traditional geometry restricted to Euclidean space to smooth and possibly nonlinear space. The goal of this section is not to cover extensively this tpoic but only to give simple intuitions of the

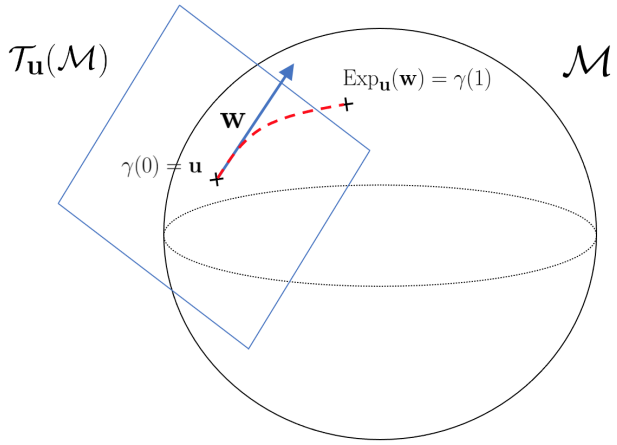


Fig. 3.3 Illustration of exponential map on the unit hypersphere in dimension 3. There is a unique geodesic $\gamma : [-1, 1] \to \mathcal{M}$ that is differentiable and such that $\gamma(0) = \mathbf{u}$ and $\gamma'(0) = \mathbf{w}$. Then, the exponential map along the tangential direction $\mathbf{w}$ that belongs to the tangent space $\mathbf{T}_{\mathbf{u}}(\mathcal{M})$ from the point on the manifold $\mathbf{u}$ is defined as $\mathrm{Exp}_{\mathbf{u}}(\mathbf{w}) = \gamma(1)$.

different concepts. We refer the reader to Lee (2006) for an in-depth review of Riemannian geometry.

Riemannian manifold. Intuitively, a smooth manifold $\mathcal{M}$ is a space that can be approximated locally by a linear space around every point. The linearization at such a point $\mathbf{u}$ is called the tangent space at $\mathbf{u}$: $\mathcal{T}_{\mathbf{u}}(\mathcal{M})$. To define a Riemannian structure on the manifold we equip each tangent space with its own inner product $\langle \cdot, \cdot \rangle_{\mathbf{u}}$ that varies smoothly with $\mathbf{u}$. Note that each inner product depends on the points $\mathbf{u}$ on the manifold. This choice of inner products is called a Riemannian metric. The resulting structure is called a Riemannian manifold. Note that a Riemannian manifold allows one to define properly the notion of gradients (as well as Hessians) on the manifold to perform optimization constrained to manifold.

Exponential mapping. To express the equivalent step on the manifold, the notion of distance on $\mathcal{M}$ needs to be introduced. The Riemannian metric induces a distance δ on the connected components of $\mathcal{M}$. Intuitively, $\delta(\mathbf{u}, \mathbf{u}')$ for all $\mathbf{u}, \mathbf{u}' \in \mathcal{M}$ is the length of the shortest smooth curve on $\mathcal{M}$ joining $\mathbf{u}$ and $\mathbf{u}'$. Formally the length of a smooth curve on $\mathcal{M}$ is the integral of the norm of the curve's speed computed with the Riemannian metric. Note that the latter quantity depends on the parametrization of the curve. The Riemannian distance δ is defined as the infinimum over all curves of $\mathcal{M}$ joining $\mathbf{u}$ and $\mathbf{u}'$.

By convention, a geodesic is the reached infinimum over all curves of $\mathcal{M}$ joining $\mathbf{u}$ and $\mathbf{u}'$ with constant tangent velocity 1. Geodesics are paths minimizing the local distance on $\mathcal{M}$ while conserving the tangent velocity. They are the generalization of straight line in linear spaces to manifolds. Using the tools from the study of ordinary differential equations, one can show that, given a Riemannian manifold $\mathcal{M}$, for a point $\mathbf{u} \in \mathcal{M}$ there exists an open set $\mathcal{O}$ of the tangent space $\mathcal{T}_{\mathbf{u}}\mathcal{M}$ containing $\mathbf{0}$, such that for any tangent vector $\mathbf{w} \in \mathcal{O}$, there is a unique geodesic $\gamma : [-1, 1] \to \mathcal{M}$ that is differentiable and such that $\gamma(0) = \mathbf{u}$ and $\gamma'(0) = \mathbf{w}$. Then, the exponential map of $\mathbf{w}$ from $\mathbf{u}$ is defined as $\mathrm{Exp}_{\mathbf{u}}(\mathbf{w}) = \gamma(1)$. The exponential map allows one to move away from $\mathbf{u}$ smoothly along a prescribed tangent direction $\mathbf{w}$ while remaining on the manifold (see illustration in Figure 3.3). This is precisely what we aim at, when quantifying the equivalent step on the hypersphere of an optimization step in the parameter space.

3.3 Spherical Framework

In this section, we first introduce a generic optimization scheme that encompasses SGD, SGD with momentum (SGD-M) and Adam, with and without L_2-regularization. Projecting the scheme update on the unit hypersphere leads to the formal definitions of effective learning rate and learning direction for any order-1 optimization scheme. This geometric perspective leads to the first explicit expression of the effective learning rate for Adam. These expressions cast new light on the interaction between optimization and radially-invariant parameters. The main notations are summarized in Figure 3.1.

3.3.1 Generic optimization scheme

There is a large body of literature on optimization schemes (Duchi et al., 2011; Kingma and Ba, 2014; Loshchilov and Hutter, 2019; Sutskever et al., 2013; Tieleman and Hinton, 2012). We focus here on two of the most popular ones, namely SGD and Adam (Kingma and Ba, 2014). Yet, to establish general results that may apply to a variety of other schemes, we introduce here a *generic optimization update*:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \mathbf{a}_k \oslash \mathbf{b}_k, \tag{3.12}$$

$$\mathbf{a}_k = \beta \mathbf{a}_{k-1} + \nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k, \tag{3.13}$$

where $\mathbf{x}_k \in \mathbb{R}^d$ is the group of radially-invariant parameters at iteration k, $\mathcal{L}$ is the group's loss estimated on a batch of input data (to ease the notation we use this notation instead of $\mathcal{L}_{\mathcal{B}_k}$), $\mathbf{a}_k \in \mathbb{R}^d$ is a momentum, $\mathbf{b}_k \in \mathbb{R}^d$ is a division vector that can depend on

the trajectory $(\mathbf{x}_i, \nabla\mathcal{L}(\mathbf{x}_i))_{i\in[\![0,k]\!]}$, $\eta_k \in \mathbb{R}$ is the scheduled trajectory-independent learning rate, $\oslash$ denotes the Hadamard element-wise division, β is the momentum parameter, and λ is the L_2-regularization parameter. We show how it encompasses several known optimization schemes.

Stochastic gradient descent (SGD) has proven to be an effective optimization method in deep learning. It can include L_2 regularization (also called weight decay) and momentum. Its updates are:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \mathbf{m}_k, \tag{3.14}$$

$$\mathbf{m}_k = \beta \mathbf{m}_{k-1} + \nabla\mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k, \tag{3.15}$$

where $\mathbf{m}_k$ is the momentum, β is the momentum parameter, and λ is the L_2-regularization parameter. It corresponds to our generic scheme (Eqs. 3.12-3.13) with $\mathbf{a}_k = \mathbf{m}_k$ and $\mathbf{b}_k = [1 \cdots 1]^\top$.

Adam is likely the most common adaptive scheme for NNs. Its updates are:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \frac{\mathbf{m}_k}{1-\beta_1^{k+1}} \oslash \sqrt{\frac{\mathbf{v}_k}{1-\beta_2^{k+1}} + \epsilon}, \tag{3.16}$$

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)(\nabla\mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k), \tag{3.17}$$

$$\mathbf{v}_k = \beta_2 \mathbf{v}_{k-1} + (1-\beta_2)(\nabla\mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k)^2, \tag{3.18}$$

omentum with parameter β_1, $\mathbf{v}_k$ is the second-order moment with parameter β_2, and ϵ prevents division by zero. (Here and in the following, the square and the square root of a vector are to be understood as element-wise.) It corresponds to our generic scheme (Eqs. 3.12-3.13) with $\beta{=}\beta_1$ and:

$$\mathbf{a}_k = \frac{\mathbf{m}_k}{1-\beta_1}, \tag{3.19}$$

$$\mathbf{b}_k = \frac{1-\beta_1^{k+1}}{1-\beta_1} \sqrt{\frac{\mathbf{v}_k}{1-\beta_2^{k+1}} + \epsilon}. \tag{3.20}$$

3.3.2 Image optimization on the hypersphere

The radial invariance implies that the radial part of the parameter update $\mathbf{x}$ does not change the function $\phi_{\mathbf{x}}$ encoded by the model, nor does it change the loss $\mathcal{L}(\mathbf{x})$. Due to radial invariance, the parameter space projected on the unit hypersphere is topologically

closer to the functional space of the network than the full parameter space. It hints that looking at optimization behaviour on the unit hypersphere might be interesting. To achieve this, we separate the quantities that can (tangential part) and cannot (radial part) change the model function. Theorem 2 formulates the spherical decomposition of our generic optimization scheme (Eqs. 3.12-3.13) in simple terms. It relates the update of radially-invariant parameters in the parameter space $\mathbb{R}^d$ and their update on $\mathcal{S}_{d-1}$ through an exponential map. Our framework relies on the decomposition of vectors into radial and tangential components. During optimization, we write the radially-invariant parameters at step $k \geq 0$ as $\mathbf{x}_k = r_k \mathbf{u}_k$ where $r_k = \|\mathbf{x}_k\|$ and $\mathbf{u}_k = \mathbf{x}_k / \|\mathbf{x}_k\|$. For any quantity $\mathbf{q}_k \in \mathbb{R}^d$ at step k, we write $\mathbf{q}_k^\perp = \mathbf{q}_k - \langle \mathbf{q}_k, \mathbf{u}_k \rangle \mathbf{u}_k$ its tangential component relatively to the current direction $\mathbf{u}_k$.

Theorem 2 (Image step on $\mathcal{S}_{d-1}$)**.** *Let's consider the update of a group of radially-invariant parameters $\mathbf{x}_k$ at step k following the generic optimization scheme (see Eqs. 3.12-3.13) and the corresponding update of its projection $\mathbf{u}_k$ on $\mathcal{S}_{d-1}$. If the following hypothesis are verified:*

- *(H1):* $1 - \frac{\eta_k \langle \mathbf{c}_k, \mathbf{u}_k \rangle}{r_k^2 d^{-1/2} \|\mathbf{b}_k\|} > 0$.
- *(H2):* $\eta_k^e \|\mathbf{c}_k^\perp\| < \pi$.

The update of a group of radially-invariant parameters $\mathbf{x}_k$ at step k following the generic optimization scheme (Eqs. 3.12-3.13) and the corresponding update of its projection $\mathbf{u}_k$ on $\mathcal{S}_{d-1}$ is given by an exponential map at $\mathbf{u}_k$ with velocity $\eta_k^e \mathbf{c}_k^\perp$:

$$\mathbf{u}_{k+1} = \mathrm{Exp}_{\mathbf{u}_k} \left(- \left[1 + O\left(\|\eta_k^e \mathbf{c}_k^\perp\|^2 \right) \right] \eta_k^e \mathbf{c}_k^\perp \right), \tag{3.21}$$

where $\mathrm{Exp}_{\mathbf{u}_k}$ is the exponential map on $\mathcal{S}_{d-1}$, and with:

$$\mathbf{c}_k \overset{\text{def}}{=} r_k \mathbf{a}_k \oslash \frac{\mathbf{b}_k}{d^{-1/2} \|\mathbf{b}_k\|}, \tag{3.22}$$

$$\eta_k^e \overset{\text{def}}{=} \frac{\eta_k}{r_k^2 d^{-1/2} \|\mathbf{b}_k\|} \left(1 - \frac{\eta_k \langle \mathbf{c}_k, \mathbf{u}_k \rangle}{r_k^2 d^{-1/2} \|\mathbf{b}_k\|} \right)^{-1}. \tag{3.23}$$

More precisely:

$$\mathbf{u}_{k+1} = \frac{\mathbf{u}_k - \eta_k^e \mathbf{c}_k^\perp}{\sqrt{1 + (\eta_k^e \|\mathbf{c}_k^\perp\|)^2}}. \tag{3.24}$$

Also, the learning rate is tuned by the dynamics of radiuses r_k which follow:

$$\frac{r_{k+1}}{r_k} = \left(1 - \frac{\eta_k \langle \mathbf{c}_k, \mathbf{u}_k \rangle}{r_k^2 d^{-1/2} \|\mathbf{b}_k\|}\right) \sqrt{1 + (\eta_k^e \|\mathbf{c}_k^\perp\|)^2}. \tag{3.25}$$

The theorem is illustrated in the case of SGD in Figure 3.1. The quantities defined in Theorem 2 are discussed in section 3.3.3. Note that for CNN training the hypothesis (H1) and (H2) with typical values of $1 - \frac{\eta_k \langle \mathbf{c}_k, \mathbf{u}_k \rangle}{r_k^2 d^{-1/2} \|\mathbf{b}_k\|} > 0$ (H1) are true. The other hypothesis $\eta_k^e \|\mathbf{c}_k^\perp\| < \pi$ (H2) where steps are supposed shorter than π is also true (see appendix B.1).

Proof. To simplify the calculation in the demonstration, we introduce the following notation:

$$A_k \stackrel{\text{def}}{=} \frac{\eta_k}{r_k^2 d^{-1/2} \|\mathbf{b}_k\|}. \tag{3.26}$$

We first demonstrate the expression for the radius dynamics in Eq. (3.25) and the precise step for $\mathbf{u}$ in Eq. (3.24). Then we use geometric arguments and a Taylor expansion to derive the update on the sphere stated in Eq.(3.21).

Radius dynamics. We first show Eq. (3.25), which we recall here using the A_k notation:

$$\frac{r_{k+1}}{r_k} = (1 - A_k \langle \mathbf{c}_k, \mathbf{u}_k \rangle) \sqrt{1 + (\eta_k^e \|\mathbf{c}_k^\perp\|)^2}. \tag{(3.25)}$$

First, we rewrite the step of a generic scheme in Eqs. (3.12-3.13) along the radial and tangential directions and separate the division vector $\mathbf{b}_k$ into its deformation $\frac{\mathbf{b}_k}{d^{-1/2}\|\mathbf{b}_k\|}$ and its scalar scheduling effect $d^{-1/2}\|\mathbf{b}_k\|$, as stated in the discussion:

$$\begin{aligned} r_{k+1}\mathbf{u}_{k+1} &= r_k \mathbf{u}_k - \frac{\eta_k}{d^{-1/2}\|\mathbf{b}_k\|} \mathbf{a}_k \oslash \frac{\mathbf{b}_k}{d^{-1/2}\|\mathbf{b}_k\|} \\ &= r_k \left[\mathbf{u}_k - \frac{\eta_k}{r_k^2 d^{-1/2}\|\mathbf{b}_k\|} r_k \mathbf{a}_k \oslash \frac{\mathbf{b}_k}{d^{-1/2}\|\mathbf{b}_k\|} \right] \\ &= r_k \left[\mathbf{u}_k - A_k r_k \mathbf{a}_k \oslash \frac{\mathbf{b}_k}{d^{-1/2}\|\mathbf{b}_k\|} \right]. \end{aligned} \tag{3.27}$$

We can note the appearance of a new term $r_k \mathbf{a}_k$. The vector $\mathbf{a}_k$ is a gradient momentum and therefore homogeneous to a gradient. Using Lemma 1, $r_k \mathbf{a}_k$ is homogeneous to a gradient on the hypersphere and can be interpreted as the momentum on the hypersphere. From Eq. (3.27), we introduce $\mathbf{c}_k$ (the deformed momentum on hypersphere) as in

Eq. (3.23) and decompose it into the radial and tangential components. We have:

$$\begin{aligned}\frac{r_{k+1}}{r_k}\mathbf{u}_{k+1} &= \mathbf{u}_k - A_k\mathbf{c}_k \\ &= (1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle)\,\mathbf{u}_k - A_k\mathbf{c}_k^{\perp}.\end{aligned} \tag{3.28}$$

By taking the squared norm of the equation, we obtain:

$$\frac{r_{k+1}^2}{r_k^2} = (1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle)^2 + \left(A_k\|\mathbf{c}_k^{\perp}\|\right)^2. \tag{3.29}$$

Making the assumption that $1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle > 0$, which is true in practice and discussed in the next subsection, we have:

$$\frac{r_{k+1}}{r_k} = (1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle)\sqrt{1 + \left(\frac{A_k}{1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle}\|\mathbf{c}_k^{\perp}\|\right)^2}. \tag{3.30}$$

After introducing $\eta_k^e = \frac{A_k}{(1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle)}$ as in Eq. (3.23), we obtain the result of (3.25).

Update of normalized parameters. We then show Eq. (3.24):

$$\mathbf{u}_{k+1} = \frac{\mathbf{u}_k - \eta_k^e\mathbf{c}_k^{\perp}}{\sqrt{1 + (\eta_k^e\|\mathbf{c}_k^{\perp}\|)^2}}. \tag{(3.24)}$$

Combining the radius dynamics previously calculated with Eq. (3.28), we have:

$$\mathbf{u}_{k+1} = \frac{(1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle)\mathbf{u}_k - A_k\mathbf{c}_k^{\perp}}{(1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle)\sqrt{1 + (\eta_k^e\|\mathbf{c}_k^{\perp}\|)^2}} \tag{3.31}$$

$$= \frac{\mathbf{u}_k - \frac{A_k}{1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle}\mathbf{c}_k^{\perp}}{\sqrt{1 + (\eta_k^e\|\mathbf{c}_k^{\perp}\|)^2}}. \tag{3.32}$$

Hence the result (3.24) using the definition of η_k^e. This result provides a unique decomposition of the generic step as a step in $\text{span}(\mathbf{u}_k, \mathbf{c}_k^{\perp})$ for the normalized filter (Eq. (3.24)) and as a radius update (Eq. (3.25)). We split the rest of the proof of the theorem in three parts.

Distance covered on the sphere. The distance covered on the hypersphere $\mathcal{S}_{d-1}$ by an optimization step is:

$$\text{dist}_{\mathcal{S}_{d-1}}(\mathbf{u}_{k+1}, \mathbf{u}_k) = \arccos(\langle \mathbf{u}_{k+1}, \mathbf{u}_k\rangle). \tag{3.33}$$

From Eq. (3.24) and with Lemma 1, we also have:

$$\langle \mathbf{u}_{k+1}, \mathbf{u}_k \rangle = \frac{1}{\sqrt{1 + (\eta_k^e \|\mathbf{c}_k^\perp\|)^2}}. \tag{3.34}$$

Therefore, $\text{dist}_{\mathcal{S}_{d-1}}(\mathbf{u}_{k+1}, \mathbf{u}_k) = \varphi(\eta_k^e \|\mathbf{c}_k^\perp\|)$ where $\varphi : z \mapsto \arccos\left(\frac{1}{\sqrt{1+z^2}}\right)$, which is equal to arctan on $\mathbb{R}_+$. Then a Taylor expansion at order 3 of arctan yields for $\eta_k^e \|\mathbf{c}_k^\perp\|$:

$$\text{dist}_{\mathcal{S}_{d-1}}(\mathbf{u}_{k+1}, \mathbf{u}_k) = \eta_k^e \|\mathbf{c}_k^\perp\| + O\left(\left(\eta_k^e \|\mathbf{c}_k^\perp\|\right)^3\right). \tag{3.35}$$

The Taylor expansion validity is discussed in the next subsection.

Exponential map on the sphere. Given a Riemannian manifold $\mathcal{M}$, for a point $\mathbf{u} \in \mathcal{M}$ there exists an open set $\mathcal{O}$ of the tangent space $\mathcal{T}_\mathbf{u}\mathcal{M}$ containing $\mathbf{0}$, such that for any tangent vector $\mathbf{w} \in \mathcal{O}$ there is a unique geodesic (a path minimizing the local distance on $\mathcal{M}$ when conserving the tangent velocity) $\gamma : [-1, 1] \to \mathcal{M}$ that is differentiable and such that $\gamma(0) = \mathbf{u}$ and $\gamma'(0) = \mathbf{w}$. Then, the exponential map of $\mathbf{w}$ from $\mathbf{u}$ is defined as $\text{Exp}_\mathbf{u}(\mathbf{w}) = \gamma(1)$. In the case of the manifold $\mathcal{S}_{d-1}$, the geodesics are complete (they are well defined for any point $\mathbf{u} \in \mathcal{S}_{d-1}$ and any velocity $\mathbf{w} \in \mathcal{T}_\mathbf{u}\mathcal{S}_{d-1}$) and are the great circles: for any $\mathbf{u} \in \mathcal{S}_{d-1}$ and any $\mathbf{w} \in \mathcal{T}_\mathbf{u}\mathcal{S}_{d-1}$, the map $\psi : t \in \mathbb{R} \mapsto \text{Exp}_\mathbf{u}(t\mathbf{w})$ verifies $\psi(\mathbb{R}) = \mathcal{S}_{d-1} \cap \text{span}(\{\mathbf{u}, \mathbf{w}\})$ which is a great circle passing through $\mathbf{u}$ with tangent $\mathbf{w}$ (see proofs in Lee (2006)). Furthermore, since the circumference of the great circle is 2π, we have that for any $\mathbf{p} \in \mathcal{S}_{d-1} \backslash \{-\mathbf{u}\}$ there is a unique $\mathbf{w}$ verifying $\|\mathbf{w}\| < \pi$ such that $\mathbf{p} = \text{Exp}_\mathbf{u}(\mathbf{w})$ and we have:

$$\text{dist}_{\mathcal{S}_{d-1}}(\mathbf{u}, \mathbf{p}) = \|\mathbf{w}\| \text{ and } \langle \mathbf{p}, \mathbf{w} \rangle \geq 0. \tag{3.36}$$

Optimization step as an exponential map. We will use the previously stated differential geometry properties to prove:

$$\mathbf{u}_{k+1} = \text{Exp}_{\mathbf{u}_k}\left(-\left[1 + O\left(\left(\eta_k^e \|\mathbf{c}_k^\perp\|\right)^2\right)\right] \eta_k^e \mathbf{c}_k^\perp\right). \tag{(3.21)}$$

For an optimization step we have:

- by construction, $\mathbf{c}_k^\perp \in \mathcal{T}_{\mathbf{u}_k}\mathcal{S}_{d-1}$;
- from Eq. (3.24), $\mathbf{u}_{k+1} \in \mathcal{S}_{d-1} \cap \text{span}(\{\mathbf{u}_k, \mathbf{c}_k^\perp\})$;
- from Eq. (3.24), $\langle \mathbf{u}_{k+1}, \mathbf{c}_k^\perp \rangle \leq 0$.

Then, there exists α that verifies $\|\alpha \mathbf{c}_k^\perp\| < \pi$ such that:

$$\mathbf{u}_{k+1} = \text{Exp}_{\mathbf{u}_k}\left(\alpha \mathbf{c}_k^\perp\right). \tag{3.37}$$

From Eq. (3.36), because of the inequality $\langle \mathbf{u}_{k+1}, \mathbf{c}_k^\perp \rangle \leq 0$, we have $\alpha < 0$. We also have that $\|\alpha \mathbf{c}_k^\perp\| = \text{dist}_{\mathcal{S}_{d-1}}(\mathbf{u}_{k+1}, \mathbf{u}_k)$. Then, using the distance previously calculated in Eq. (3.35), we have:

$$|\alpha| \|\mathbf{c}_k^\perp\| = \eta_k^e \|\mathbf{c}_k^\perp\| + O\left(\left(\eta_k^e \|\mathbf{c}_k^\perp\|\right)^3\right), \tag{3.38}$$

$$|\alpha| = \eta_k^e \left[1 + O\left(\left(\eta_k^e \|\mathbf{c}_k^\perp\|\right)^2\right)\right]. \tag{3.39}$$

Combining the sign and absolute value of α, we get the final exponential map expression:

$$\mathbf{u}_{k+1} = \text{Exp}_{\mathbf{u}_k}\left(-\left[1 + O\left(\left(\eta_k^e \|\mathbf{c}_k^\perp\|\right)^2\right)\right] \eta_k^e \mathbf{c}_k^\perp\right), \tag{(3.21)}$$

$$\approx \text{Exp}_{\mathbf{u}_k}\left(-\eta_k^e \mathbf{c}_k^\perp\right). \tag{3.40}$$

Note that we implicitly assume here that $|\alpha| \|\mathbf{c}_k^\perp\| \approx \eta_k^e \|\mathbf{c}_k^\perp\| < \pi$, which is discussed in appendix B.1. □

3.3.3 Effective quantities

In Theorem 2, the normalized parameters update in Eq. 3.21 can be read $\mathbf{u}_{k+1} \approx \text{Exp}_{\mathbf{u}_k}\left(-\eta_k^e \mathbf{c}_k^\perp\right)$, where η_k^e and $\mathbf{c}_k^\perp$ can then be respectively interpreted as the learning rate and the direction of an optimization step constrained to $\mathcal{S}_{d-1}$. Since $\mathbf{a}_k$ is the momentum and, with Lemma 1, the quantity $r_k \mathbf{a}_k$ in $\mathbf{c}_k$ can be seen as *a momentum on the hypersphere*. Due to the radial invariance, only the change of parameter on the unit hypersphere corresponds to a change of model function. Hence we can interpret η_k^e and $\mathbf{c}_k^\perp$ as *effective learning rate* and *effective learning direction*. In other words, these quantities correspond to the learning rate and direction on the hypersphere that reproduce the function update of the optimization step.

Using Theorem 2, we can derive actual effective learning rates for any optimization scheme that fits our generic framework. These expressions, summarized in Table 3.1 are explicit and have a clear interpretation, in contrast to learning rates in (van Laarhoven, 2017), which are approximate and asymptotic, and in (Arora et al., 2019; Hoffer et al., 2018a), which are variational and restricted to SGD without momentum only. In particular, we provide the first explicit expression of the effective learning rate for Adam:

$$\eta_k^e = \frac{\eta_k}{r_k \nu_k}\left(1 - \frac{\eta_k \langle \mathbf{c}_k, \mathbf{u}_k \rangle}{r_k \nu_k}\right)^{-1} \tag{3.41}$$

Table 3.1 Effective learning rate and direction for optimization schemes (we omit here the iteration index k).

Scheme	η^e	$\mathbf{c}^\perp$
SGD	$\frac{\eta}{r^2}$	$\nabla\mathcal{L}(\mathbf{u})$
SGD + L_2	$\frac{\eta}{r^2(1-\eta\lambda)}$	$\nabla\mathcal{L}(\mathbf{u})$
SGD-M	$\frac{\eta}{r^2}(1-\frac{\eta\langle\mathbf{c},\mathbf{u}\rangle}{r^2})^{-1}$	$\mathbf{c}^\perp$
Adam	$\frac{\eta}{r\nu}(1-\frac{\eta\langle\mathbf{c},\mathbf{u}\rangle}{r\nu})^{-1}$	$\mathbf{c}^\perp$

where $\nu_k = r_k d^{-1/2}\|\mathbf{b}_k\|$ is homogeneous to the norm of a gradient on the hypersphere and can be related to an *second-order moment on the hypersphere*. Indeed, with Eq. (3.65) and using Lemma 1, we can give the exact expression of the second-order moment on the sphere, defined as $\nu_k = r_k d^{-1/2}\|\mathbf{b}_k\|$:

$$\nu_k = d^{-1/2}\frac{1-\beta_1^{k+1}}{1-\beta_1}\left(\frac{1-\beta_2}{1-\beta_2^{k+1}}\right)^{1/2}\left(\sum_{i=0}^{k}\beta_2^{k-i}\frac{r_k^2}{r_i^2}\|\nabla\mathcal{L}(\mathbf{u}_i)+\lambda r_i^2\mathbf{u}_i\|^2\right)^{1/2}. \tag{3.42}$$

Using the variable ν also simplifies the in-depth analysis in section 3.5, allowing a better interpretation of formulas.

The expression of the effective learning rate of Adam, i.e., the amplitude of the step taken on the hypersphere, reveals a dependence on the dimension d (through ν) of the update of the considered group of radially-invariant parameters. In the case of an MLP or CNN that stacks layers with neurons or filters of different dimensions, the learning rate is thus tuned differently from one layer to another.

We can also see that for all schemes the learning rate is tuned by the dynamics of radiuses r_k, which follow Eq. 3.25. In contrast to previous studies (Arora et al., 2019; van Laarhoven, 2017), this result demonstrates that for momentum methods, $\langle\mathbf{c}_k, \mathbf{u}_k\rangle$, which involves accumulated gradients terms in the momentum as well as L_2 regularization, tunes the learning rate.

3.4 SGD is equivalent to AdaGradG

In this section, we leverage the tools introduced in the spherical framework of section 3.3 to find a scheme constrained to the hypersphere that is equivalent to SGD. We show that, for radially-invariant models, SGD is actually an adaptive optimization method. Formally,

SGD is equivalent to a special case of AdamG (Cho and Lee, 2017) without momentum, where AdamG is a variant of Adam adapted and constrained to the unit hypersphere. Alternatively, we can also say that SGD is equivalent to a variant of AdaGrad adapted and constrained to the hypersphere, where AdaGrad is a special case of Adam without momentum. Besides, we illustrate this theoretical equivalence empirically.

3.4.1 Equivalence between optimization schemes

Due to the radial invariance, the functional space of the model is encoded by $\mathcal{S}_{d-1}$. In other words, two schemes with the same sequence of groups of radially-invariant parameters on the hypersphere $(\mathbf{u}_k)_{k\geq 0}$ encode the same sequence of model functions. We say that two optimization schemes O and $\tilde{O}$ are equivalent if and only if $\forall k \geq 0, \mathbf{u}_k = \tilde{\mathbf{u}}_k$. Hence, starting from the same parameters, they reach the same optimum. By using Eq. 3.24, we obtain the following lemma, which is useful to prove the equivalence of two given optimization schemes:

Lemma 3 (Sufficient condition for the equivalence of optimization schemes)**.**

$$\begin{cases} \mathbf{u}_0 = \tilde{\mathbf{u}}_0 \\ \forall k \geq 0, \eta_k^e = \tilde{\eta}_k^e, \mathbf{c}_k^\perp = \tilde{\mathbf{c}}_k^\perp \end{cases} \Rightarrow \forall k \geq 0, \mathbf{u}_k = \tilde{\mathbf{u}}_k. \tag{3.43}$$

3.4.2 A hypersphere-constrained scheme equivalent to SGD

We now study, within our spherical framework, SGD with L_2 regularization, i.e., its associated update is: $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k(\nabla\mathcal{L}(\mathbf{x}_k) - \lambda_k\mathbf{x}_k)$. From the effective learning rate expression, we know that SGD yields an adaptive behaviour because it is scheduled by the radius dynamic, which depends on gradients. In fact, the tools in our framework allow us to find that SGD is equivalent to a variant of Adam constrained to the unit hypersphere, similar to AdamG (Cho and Lee, 2017), and without momentum, similar to AdaGrad. AdamG (Cho and Lee, 2017) uses the same updates as Adam (eq. 3.16-3.18) but project the weight on the hyper-sphere after each optimization step. More precisely, SGD is equivalent to AdamG with a null momentum factor $\beta_1 = 0$ (like AdaGrad), a non-null initial second-order moment v_0, an offset of the scalar second-order moment $k+1 \to k$ and without the bias correction term $1 - \beta_2^{k+1}$. Dubbed AdaGradG, this

scheme reads:

$$\text{(AdaGradG):} \quad \begin{cases} \hat{\mathbf{x}}_{k+1} = \mathbf{x}_k - \eta_k \frac{\nabla \mathcal{L}(\mathbf{x}_k)}{\sqrt{v_k}}, \\ \mathbf{x}_{k+1} = \frac{\hat{\mathbf{x}}_{k+1}}{\|\hat{\mathbf{x}}_{k+1}\|}, \\ v_{k+1} = \beta v_k + \|\nabla \mathcal{L}(\mathbf{x}_k)\|^2. \end{cases}$$

AdaGradG, like AdamG, is an adaptive method. Unlike Adam, which is adaptive with respect to the second-order moment for each parameter, AdaGradG and AdamG are adaptive for each group of radially-invariant parameters (e.g., filters for CNNs with BN or WN). In other words, each filter is adapted individually and independently by the optimization algorithm; it is not a global scheduling. Now if we call « equivalent at order 2 in the step » a scheme equivalence that holds when we use for r_k an expression that satisfies the radius dynamic with a Taylor expansion at order 2, then we have the following theorem:

Theorem 4 (SGD equivalent scheme on the unit hypersphere)**.** *For any $\lambda \geq 0, \eta > 0, r_0 > 0$, we have the following equivalence when using the radius dynamic at order 2 in $(\eta_k \|\nabla \mathcal{L}(\mathbf{u}_k)\|)^2 / r_k^2$:*

$$\begin{cases} \text{(SGD)} \\ \mathbf{x}_0 = r_0 \mathbf{u}_0 \\ \lambda_k = \lambda \\ \eta_k = \eta \end{cases} \text{ is scheme-equivalent at order 2 in step with } \begin{cases} \text{(AdaGradG)} \\ \mathbf{x}_0 = \mathbf{u}_0 \\ \beta = (1 - \eta\lambda)^4 \\ \eta_k = (2\beta)^{-1/2} \\ v_0 = r_0^4 (2\eta^2 \beta^{1/2})^{-1}. \end{cases}$$

This result is unexpected because SGD, which is not adaptive by itself, is equivalent to a second order moment adaptive method. The scheduling performed by the radius dynamics actually replicates the effect of dividing the learning rate by the second-order moment of the gradient norm: v_k. For standard values of hyperparameters $\lambda < 1$ (order of magnitude of 10^{-4}) and $\eta < 1$ (order of magnitude at most 10^{-1}), the higher-order terms of the radius in the Taylor expansion empirically become negligible in practice. Second, with standard values of the hyper-parameters, namely learning rate $\eta < 1$ and L_2 regularization $\lambda < 1$, we have $\beta \leq 1$ which corresponds to a standard value for a moment factor. Interestingly, the L_2 regularization parameter λ controls the memory of the past gradients' norm. If $\beta = 1$ (with $\lambda = 0$), there is no attenuation, each gradient norm has the same contribution in the order-2 moment. If $\lambda \neq 0$, there is a decay factor ($\beta < 1$) on past gradients' norm in the order-2 moment. This gives a new interpretation of the role of the L_2 regularization parameter λ in SGD with NLs.

Proof. Starting from SGD, we first use Lemma 3 to find a strict equivalence scheme with a simpler radius dynamic. We resolve this radius dynamic with a Taylor expansion at order 2 in $(\eta_k \|\nabla \mathcal{L}(\mathbf{u}_k)\|)^2 / r_k^2$. A second use of Lemma 3 finally leads to the scheme equivalence in Theorem 4. The fact that r_k is well approximated at order 2 in practice is illustrated and discussed in appendix C.1. As summarized in Table 3.1, the expressions of the effective learning rates and directions for SGD are $\mathbf{c}_k^\perp = r_k \nabla \mathcal{L}(\mathbf{x}_k) = \nabla \mathcal{L}(\mathbf{u}_k)$ and $\eta_k^e = \frac{\eta_k}{r_k^2(1-\eta_k\lambda_k)}$.

Equivalence with SGD and L_2 regularization. We look for conditions leading to an equivalence between SGD with L_2 regularization and SGD without L_2 regularization. Using Lemma 3, the equality of effective directions is trivial and the equality of effective learning rates for any step k yields the following equivalence:

$$\begin{cases} \text{(SGD)} \\ \tilde{\mathbf{x}}_0 = r_0 \mathbf{u}_0 \\ \tilde{\lambda}_k = \lambda \\ \tilde{\eta}_k = \eta \end{cases} \text{ is scheme-equivalent to } \begin{cases} \text{(SGD)} \\ \mathbf{x}_0 = r_0 \mathbf{u}_0 \\ \lambda_k = 0 \\ \eta_k = \eta(1-\eta\lambda)^{-2k-1} \end{cases} \tag{3.44}$$

L_2 regularization is equivalent to an exponential scheduling of the learning rate, as found in Li and Arora (2020). Here, we provide a proof in a constructive manner. We are going to use Lemma 3 and find a sufficient condition to have:

$$\begin{cases} \text{(i) } \mathbf{u}_0 = \tilde{\mathbf{u}}_0 \\ \text{(ii) } \forall k \geq 0, \eta_k^e = \tilde{\eta}_k^e, \mathbf{c}_k^\perp = \tilde{\mathbf{c}}_k^\perp. \end{cases}$$

Equation (i) is trivially satisfied by simply taking the same starting point: $\tilde{\mathbf{x}}_0 = \mathbf{x}_0$. Regarding (ii), because effective directions are the same and only depend on $\mathbf{u}_k$, we only need a sufficient condition on η_k^e. For effective learning rates, using Eq. ((3.25)) and expressions in Table 3.1, we have:

$$\eta_k^e = \tilde{\eta}_k^e \Leftrightarrow \frac{\eta_k}{r_k^2} = \frac{\tilde{\eta}_k}{\tilde{r}_k^2(1-\tilde{\eta}_k\lambda)}. \tag{3.45}$$

Since $\tilde{\eta}_k = \eta$, we obtain:

$$(3.45) \Leftrightarrow \eta_k = \left(\frac{r_k}{\tilde{r}_k}\right)^2 \frac{\eta}{(1-\eta\lambda)}.$$

Therefore:

$$\frac{\eta_{k+1}}{\eta_k} = \left(\frac{r_{k+1}\tilde{r}_k}{\tilde{r}_{k+1}r_k}\right)^2 = \left(\frac{r_{k+1}/r_k}{\tilde{r}_{k+1}/\tilde{r}_k}\right)^2.$$

By using the radius dynamics in Eq. (3.25) for the two schemes, SGD and SGD with L_2 regularization, and by the equality of effective learning rates and directions, we have:

$$\begin{aligned}\frac{\eta_{k+1}}{\eta_k} &= \left(\frac{\sqrt{1+(\eta_k^e\|\mathbf{c}_k^\perp\|)^2}}{(1-\eta\lambda)\sqrt{1+(\tilde{\eta}_k^e\|\tilde{\mathbf{c}}_k^\perp\|)^2}}\right)^2 \\ &= (1-\eta\lambda)^{-2}.\end{aligned}$$

By taking Eq. (3.45) for $k=0$, because $r_0=\tilde{r}_0$ we have: $\eta_0=\eta(1-\eta\lambda)^{-1}$. Combining the previous relation and the initialization case, we derive by induction that $\eta_k = \eta(1-\eta\lambda)^{-2k-1}$ is a sufficient condition. We can conclude, using Lemma 3, the equivalence stated in Eq. (3.44).

Resolution of the radius dynamics. Without L_2 regularization, the absence of radial component in $\mathbf{c}_k$ makes the radius dynamics simple:

$$r_{k+1}^2 = r_k^2 + \frac{(\eta_k\|\nabla\mathcal{L}(\mathbf{u}_k)\|)^2}{r_k^2}. \tag{3.46}$$

With a Taylor expansion at order 2, we can show that for $k\geq 1$ the solution

$$r_k^2 = \sqrt{2\sum_{i=0}^{k-1}(\eta_i\|\nabla\mathcal{L}(\mathbf{u}_i)|)^2 + r_0^4}$$

satisfies the previous equation. Indeed using the expression at step $k+1$ gives:

$$\begin{aligned}r_{k+1}^2 &= \sqrt{2\sum_{i=0}^{k-1}(\eta_i\|\nabla\mathcal{L}(\mathbf{u}_i)|)^2 + r_0^4 + 2(\eta_k\|\nabla\mathcal{L}(\mathbf{u}_k)\|)^2} \\ &= r_k^2\sqrt{1+2\frac{(\eta_k\|\nabla\mathcal{L}(\mathbf{u}_k)\|)^2}{r_k^4}} \\ &= r_k^2\left(1+(1/2)2\frac{(\eta_k\|\nabla\mathcal{L}(\mathbf{u}_k)\|)^2}{r_k^4} + o\left(\frac{(\eta_k\|\nabla\mathcal{L}(\mathbf{u}_k)\|)^2}{r_k^4}\right)\right) \\ &= r_k^2 + \frac{(\eta_k\|\nabla\mathcal{L}(\mathbf{u}_k)\|)^2}{r_k^2} + o\left(\frac{(\eta_k\|\nabla\mathcal{L}(\mathbf{u}_k)\|)^2}{r_k^2}\right).\end{aligned}$$

Using $\eta_k = \eta(1-\eta\lambda)^{-2k-1}$, introducing $\beta = (1-\eta\lambda)^4$, omitting the $o\left(\frac{(\eta_k\|\nabla\mathcal{L}(\mathbf{u}_k)\|)^2}{r_k^2}\right)$ and injecting the previous solution in the effective learning rate, we obtain the closed form:

$$\begin{aligned}\eta_k^e &= \frac{\eta(1-\eta\lambda)^{-2k-1}}{\sqrt{2\sum_{i=0}^{k-1}\eta^2(1-\eta\lambda)^{-4i-2}\|\nabla\mathcal{L}(\mathbf{u}_i)\|^2 + r_0^4}} \\ &= \frac{(2\beta)^{-\frac{1}{2}}}{\sqrt{\sum_{i=0}^{k-1}\beta^{(k-1)-i}\|\nabla\mathcal{L}(\mathbf{u}_i)\|^2 + \beta^k\frac{r_0^4}{2\eta^2\beta^{\frac{1}{2}}}}}.\end{aligned} \tag{3.47}$$

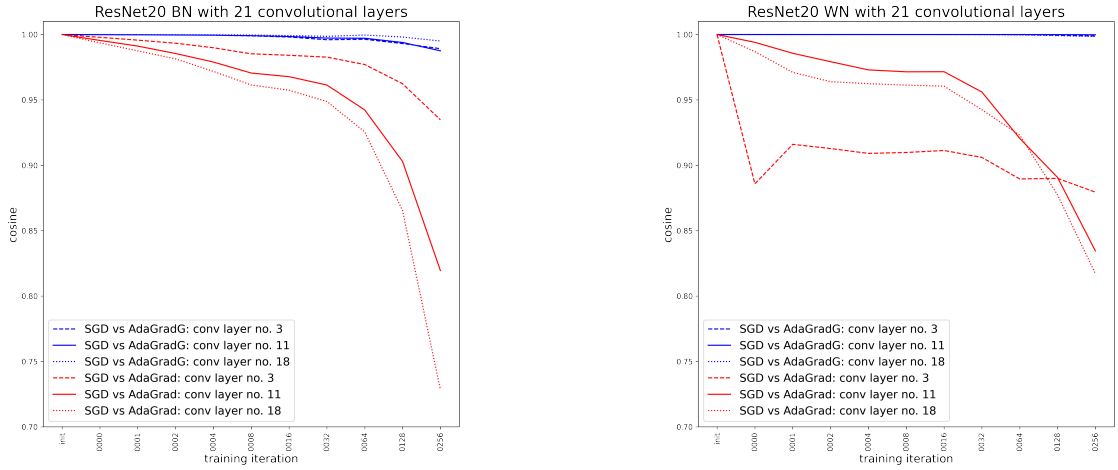
AdaGradG. The AdaGradG scheme is constrained on the hypersphere thanks to the normalization; the radius is therefore constant and equal to 1. The absence of radial component in the update gives: $\mathbf{c}_k^{\perp} = \nabla\mathcal{L}(\mathbf{u}_k)$ and $\eta_k^e = \frac{\eta_k}{\sqrt{v_k}}$. Thus, the resolution of the induction on v_k leads to the the closed form:

$$\eta_k^e = \frac{\eta_k}{\sqrt{\sum_{i=0}^{k-1}\beta^{(k-1)-i}\|\nabla\mathcal{L}(\mathbf{u}_i)\|^2 + \beta^k v_0}}. \tag{3.48}$$

Hence the final theorem, when identifying the closed-form expressions of effective learning rates and using Lemma 3. □

3.4.3 Empirical validation

In order to illustrate the equivalence in Theorem 4, we experiment with learning an image classifier on CIFAR10 using different optimization schemes. We consider two architectures: ResNet20 with BN and ResNet20 with WN (He et al., 2016a). We recall the set of parameters $\boldsymbol{\theta}$ of the above architectures can be split into two disjoint subsets: $\boldsymbol{\theta} = \mathcal{F} \cup \mathcal{R}$, where $\mathcal{F}$ is the set of groups of radially-invariant parameters and $\mathcal{R}$ is the set of remaining parameters. Note that the set of remaining parameters in $\mathcal{R}$ differs from one architecture to another: for ResNet20 BN, it includes the last linear layer as well as the scaling and bias of BN layers; for ResNet20 WN, it includes the magnitude parameters in each convolutional layer as well as the last linear layer. For each architecture, we experiment with tracking the trajectory of parameters under different optimization schemes: SGD, AdaGradG and AdaGrad. As our analysis is restricted to radially-invariant parameters, we only track the trajectory of parameters belonging to $\mathcal{F}$, while the remaining parameters, belonging to $\mathcal{R}$, are always optimized in the same way, i.e., with SGD. For stability purposes, we finetune a previously trained architecture with SGD. The order of batches as well as the random seed for data augmentation are fixed to obtain comparable trajectories. The hyperparameters are chosen for SGD and



(a) ResNet20 BN. **(b)** ResNet20 WN.

Fig. 3.4 Comparison of the trajectories of radially-invariant parameters using different optimization schemes. For three randomly selected filters in each block of a ResNet20 architecture, with BN (left) or WN (right), we compute the cosine similarity between the parameter values obtained with SGD and the parameters values obtained respectively by AdaGradG and AdaGrad, at different iteration stages of a classification training on CIFAR10.

AdaGrad so that gradient steps have the same order of magnitude (see appendix C.1 for details); the hyperparameters for AdaGradG are provided by the equivalence in Theorem 4. In Figure 3.4, we show the angle between the training trajectories using SGD and AdaGradG (resp. SGD and AdaGrad), for three different filters in each block of the ResNet20 architectures. We observe that the trajectories on the L_2 unit hypersphere remain aligned for SGD and AdaGradG whereas, for SGD and AdaGrad, they quickly diverge. It empirically validates the equivalence mentioned in Theorem 4.

3.5 Geometric phenomena in Adam

Our framework with its geometrical interpretation reveals intriguing behaviors occurring in Adam. Indeed, since the unit hypersphere is enough to represent the functional space encoded by the network, from the perspective of manifold optimization, the optimization direction should only depend on the trajectory on that manifold. In the case of Adam, the effective direction not only depends on the trajectory on the hypersphere but also on the deformed gradients and additional radial terms. These terms are thus likely to play a role in Adam optimization. In order to understand their role, we describe these geometrical

phenomena in section 3.5.1. Interestingly, previous variants of Adam, AdamW (Loshchilov and Hutter, 2019) and AdamG (Cho and Lee, 2017) are related to these phenomena. To study empirically their importance, we consider in section 3.5.2 variants of Adam that first provide a direction intrinsic to the unit hypersphere, without deformation of the gradients, and then where radial terms are decoupled from the direction. The empirical study of these variants over a variety of datasets and architectures suggests that these behaviors do play a significant role in CNNs training with BN.

3.5.1 Identification of geometrical phenomena in Adam

Here, we perform an in-depth analysis of the effective learning direction of Adam.

(a) Deformed gradients. Considering the quantities defined for a generic scheme in Eq. 3.23, $\mathbf{b}_k$ has a deformation effect on $\mathbf{a}_k$, due to the Hadamard division by $\frac{\mathbf{b}_k}{d^{-1/2}\|\mathbf{b}_k\|}$, and a scheduling effect $d^{-1/2}\|\mathbf{b}_k\|$ on the effective learning rate. In the case where the momentum factor is null $\beta_1 = 0$, the direction of the update at step k is $\nabla\mathcal{L}(\mathbf{u}_k) \oslash \frac{\mathbf{b}_k}{d^{-1/2}\|\mathbf{b}_k\|}$ (Eq. 3.23) and the deformation $\frac{\mathbf{b}_k}{d^{-1/2}\|\mathbf{b}_k\|}$ may push the direction of the update outside the tangent space of $\mathcal{S}_{d-1}$ at $\mathbf{u}_k$, whereas the gradient itself lies in the tangent space. This deformation is in fact not isotropic: the displacement of the gradient from the tangent space depends on the position of $\mathbf{u}_k$ on the sphere. We illustrate this anisotropy in Figure 3.5(b).

(b) Additional radial terms. In the momentum on the sphere $\mathbf{c}_k$, quantities that are radial (resp. orthogonal) at a point on the sphere may not be radial (resp. orthogonal) at another point. To clarify the contribution of $\mathbf{c}_k$ in the effective learning direction $\mathbf{c}_k^{\perp}$, we perform the following decomposition. We recall that:

$$\mathbf{c}_k = r_k \mathbf{a}_k \oslash \frac{\mathbf{b}_k}{d^{-1/2}\|\mathbf{b}_k\|},$$

We start by the recurrence in Eq (3.13):

$$\mathbf{a}_k = \sum_{i=0}^{k} \beta^{k-i} \left(\nabla\mathcal{L}(\mathbf{x}_i) + \lambda \mathbf{x}_i\right). \tag{3.49}$$

Using Lemma 1 and decomposing on $\nabla\mathcal{L}(\mathbf{u}_i)$ and $\mathbf{u}_i$, we have:

$$\mathbf{a}_k = \sum_{i=0}^{k} \beta^{k-i} \left(\frac{1}{r_i} \nabla\mathcal{L}(\mathbf{u}_i) + \lambda r_i \mathbf{u}_i \right) \tag{3.50}$$

$$= \frac{1}{r_k} \left(\sum_{i=0}^{k} \beta^{k-i} \left(\frac{r_k}{r_i} \nabla\mathcal{L}(\mathbf{u}_i) + \lambda r_k r_i \mathbf{u}_i \right) \right). \tag{3.51}$$

Thus:

$$r_k \mathbf{a}_k = \sum_{i=0}^{k} \beta^{k-i} \frac{r_k}{r_i} \nabla\mathcal{L}(\mathbf{u}_i) + \lambda r_k^2 \sum_{i=0}^{k} \beta^{k-i} \frac{r_i}{r_k} \mathbf{u}_i, \tag{3.52}$$

Overall, we obtain the decomposition:

$$\mathbf{c}_k = (\mathbf{c}_k^{\text{grad}} + \lambda r_k^2 \mathbf{c}_k^{L_2}) \oslash \frac{\mathbf{b}_k}{d^{-1/2}\|\mathbf{b}_k\|} \quad \text{with:} \tag{3.53}$$

$$\mathbf{c}_k^{\text{grad}} \overset{\text{def}}{=} \nabla\mathcal{L}(\mathbf{u}_k) + \sum_{i=0}^{k-1} \beta^{k-i} \frac{r_k}{r_i} \nabla\mathcal{L}(\mathbf{u}_i) \tag{3.54}$$

$$\mathbf{c}_k^{L_2} \overset{\text{def}}{=} \mathbf{u}_k + \sum_{i=0}^{k-1} \beta^{k-i} \frac{r_i}{r_k} \mathbf{u}_i. \tag{3.55}$$

b1. Contribution of $\mathbf{c}_k^{\text{grad}}$. At step k, the contribution of each past gradient corresponds to the orthogonal part $\nabla\mathcal{L}(\mathbf{u}_i) - \langle \nabla\mathcal{L}(\mathbf{u}_i), \mathbf{u}_k \rangle \mathbf{u}_k$. It impacts the effective learning direction depending on its orientation relatively to $\mathbf{u}_k$. Two past points, although equally distant from $\mathbf{u}_k$ on the sphere and with equal gradient amplitude may thus contribute differently in $\mathbf{c}_k^{\perp}$ due to their orientation (*cf.* Figure 3.5(c)).

b2. Contribution of $\mathbf{c}_k^{L_2}$. Naturally, the current point $\mathbf{u}_k$ does not contribute to the effective learning direction $\mathbf{c}_k^{\perp}$, unlike the history of points in $\sum_{i=0}^{k-1} \beta^{k-i} \frac{r_i}{r_k} \mathbf{u}_i$, which does. This dependency can be avoided if we decouple the L_2 regularization, in which case we do not accumulate L_2 terms in the momentum. This shows that the decoupling proposed in AdamW (Loshchilov and Hutter, 2019) actually removes the contribution of L_2 regularization in the effective learning direction.

(c) The radius ratio $\frac{r_k}{r_i}$ present in both $\mathbf{c}_k^{\text{grad}}$ and $\mathbf{c}_k^{L_2}$ (in inverse proportion) impacts the effective learning direction $\mathbf{c}_k^{\perp}$: it can differ for identical sequences $(\mathbf{u}_i)_{i \leq k}$ on the sphere but with distinct radius histories $(r_i)_{i \leq k}$. Since the radius is closely related to the effective learning rate, it means that the effective learning direction $\mathbf{c}_k^{\perp}$ is adjusted according to the learning rates history.

Note that AdamG (Cho and Lee, 2017), by constraining the optimization to the unit hypersphere and thus removing L_2 regularization, neutralizes all the above phenomena. However, this method has no scheduling effect allowed by the radius dynamics (*cf.* Eq.3.25) since it is kept constant during training.

3.5.2 New variants of Adam.

We recall that AdamW neutralizes **(b2)** and that AdamG neutralizes all of above phenomena but loses the scheduling effect identified in Eq. 3.25. To complete our analysis, we use geometrical tools to design variations of Adam which neutralizes sequentially each phenomenon while preserving the natural scheduling effect in Theorem 2. We neutralize **(a)** by replacing the element-wise second-order moment, **(b1)** and **(b2)** by transporting the momentum from a current point to the new one, **(c)** by re-scaling the momentum at step k.

Adam without deformation of gradients (a). Following Theorem 2, the division vector $\mathbf{b}_k$ has two contributions in the decomposition:

- a deformation in $\mathbf{c}_k$ applied to $\mathbf{a}_k$: $\mathbf{c}_k = r_k \mathbf{a}_k \oslash \frac{\mathbf{b}_k}{d^{-1/2}\|\mathbf{b}_k\|}$;
- a scheduling effect in the effective learning rate $d^{-1/2}\|\mathbf{b}_k\|$ (Eq. (3.23)).

The goal is to find a new division vector $\mathsf{S}(\mathbf{b}_k)$ that does not create a deformation while preserving the scheduling effect of $\mathbf{b}_k$ in the effective learning rate. This means:

$$\frac{\mathsf{S}(\mathbf{b}_k)}{d^{-1/2}\|\mathsf{S}(\mathbf{b}_k)\|} = [1 \cdots 1]^\top, \tag{3.56}$$

$$d^{-1/2}\|\mathsf{S}(\mathbf{b}_k)\| = d^{-1/2}\|\mathbf{b}_k\|. \tag{3.57}$$

This leads to $\mathsf{S}(\mathbf{b}_k) = d^{-1/2}\|\mathbf{b}_k\|[1 \cdots 1]^\top$.

In the case of $\beta_1 = 0$, $\mathbf{a}_k = \nabla\mathcal{L}(\mathbf{x}_k)$, for any $\mathbf{b}_k$. When we apply the standardization, we obtain:

$$\mathbf{c}_k = r_k \nabla\mathcal{L}(\mathbf{x}_k) \oslash \frac{\mathsf{S}(\mathbf{b}_k)}{d^{-1/2}\|\mathsf{S}(\mathbf{b}_k)\|} = \nabla\mathcal{L}(\mathbf{u}_k) \oslash [1 \cdots 1]^\top = \nabla\mathcal{L}(\mathbf{u}_k). \tag{3.58}$$

The direction lies in the tangent space because, by Lemma 1, the gradient belongs to it. In the generic scheme, using the standardization gives:

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k - \eta_k \mathbf{a}_k \oslash \mathsf{S}(\mathbf{b}_k) && (3.59)\\ &= \mathbf{x}_k - \eta_k \mathbf{a}_k \oslash (d^{-1/2}\|\mathbf{b}_k\|[1\cdots 1]^\top) && (3.60)\\ &= \mathbf{x}_k - \eta_k \mathbf{a}_k / (d^{-1/2}\|\mathbf{b}_k\|). && (3.61)\end{aligned}$$

This means that the standardization consists in replacing the Hadamard division by $\mathbf{b}_k$ with a scalar division by $d^{-1/2}\|\mathbf{b}_k\|$. In the case of Adam, we recall that:

$$\mathbf{b}_k = \frac{1-\beta_1^{k+1}}{1-\beta_1}\sqrt{\frac{\mathbf{v}_k}{1-\beta_2^{k+1}} + \epsilon}\ . \qquad ((3.20))$$

Omitting ϵ for simplicity we have:

$$d^{-1/2}\|\mathbf{b}_k\| = \frac{1-\beta_1^{k+1}}{1-\beta_1}\left(\frac{1}{1-\beta_2^{k+1}}\right)^{\frac{1}{2}} d^{-1/2}\|\sqrt{\mathbf{v}_k}\|. \qquad (3.62)$$

Let us calculate $\|\sqrt{\mathbf{v}_k}\|$. Developing the recursion of $\mathbf{v}_k$, as defined in Eq. (3.18), leads to:

$$\mathbf{v}_k = (1-\beta_2)\sum_{i=0}^{k}\beta_2^{k-i}\left(\nabla\mathcal{L}(\mathbf{x}_i) + \lambda\mathbf{x}_i\right)^2, \qquad (3.63)$$

$$\sqrt{\mathbf{v}_k} = \sqrt{1-\beta_2}\sqrt{\sum_{i=0}^{k}\beta_2^{k-i}\left(\nabla\mathcal{L}(\mathbf{x}_i) + \lambda\mathbf{x}_i\right)^2}, \qquad (3.64)$$

where the square and the square-root are element-wise operations. Hence, if we take the square norm:

$$\begin{aligned}\|\sqrt{\mathbf{v}_k}\|^2 &= (1-\beta_2)\sum_{j=1}^{d}\left(\sqrt{\sum_{i=0}^{k}\beta_2^{k-i}\left(\nabla\mathcal{L}(\mathbf{x}_i) + \lambda\mathbf{x}_i\right)^2}\right)_j^2\\ &= (1-\beta_2)\sum_{j=1}^{d}\sum_{i=0}^{k}\beta_2^{k-i}\left(\nabla\mathcal{L}(\mathbf{x}_i) + \lambda\mathbf{x}_i\right)_j^2\\ &= (1-\beta_2)\sum_{i=0}^{k}\beta_2^{k-i}\sum_{j=1}^{d}\left(\nabla\mathcal{L}(\mathbf{x}_i) + \lambda\mathbf{x}_i\right)_j^2\\ &= (1-\beta_2)\sum_{i=0}^{k}\beta_2^{k-i}\|\nabla\mathcal{L}(\mathbf{x}_i) + \lambda\mathbf{x}_i\|^2, && (3.65)\end{aligned}$$

where the j subscript denotes the j-th element of the vector. It is exactly the order-2 moment of the gradient norm. Therefore, we define the scalar v_k:

$$v_k = \beta_2 v_{k-1} + (1-\beta_2) d^{-1} \|\nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k\|^2, \tag{3.66}$$

which is the order-2 moment of the gradient norm with a factor d^{-1}. It verifies $\sqrt{v_k} = d^{-1/2}\|\sqrt{\mathbf{v}_k}\|$, needed for the scalar division stated in Eq. (3.62). By applying the bias correction, it gives the formula given in the chapter of Adam w/o (a):

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \frac{\mathbf{m}_k}{1-\beta_1^{k+1}} \Big/ \sqrt{\frac{v_k}{1-\beta_2^{k+1}} + \epsilon}, \tag{3.67}$$

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)(\nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k), \tag{3.68}$$

$$v_k = \beta_2 v_{k-1} + (1-\beta_2) d^{-1} \|\nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k\|^2. \tag{3.69}$$

Note that the previous demonstration makes the factor d^{-1} appear in v_k to have exactly the scheduling effect of Adam without the deformation.

Adam without deformed gradients and no additional radial terms (ab). We introduce the rescaling and transport transformation of the momentum to neutralize the identified effects on the effective direction. The resulting, new $\mathbf{c}_k$ is orthogonal to $\mathbf{u}_k$ and does not contribute in the effective learning rate tuning with its radial part. To avoid gradient history leaving the tangent space and thus neutralize **(b)**, we perform a parallel transport of the momentum $\mathbf{a}_{k-1}$ from the corresponding point on the sphere $\mathbf{u}_{k-1}$ to the new point $\mathbf{u}_k$ denoted as $\Gamma_{\mathbf{u}_{k-1}}^{\mathbf{u}_k}(\mathbf{a}_{k-1})$ at each iteration $k \geq 1$. Figure 3.5(c) illustrates the transport of a gradient. The parallel transport between two points associates each vector of the tangent space of the first point to a vector of the second tangent space by preserving the scalar product with the derivatives along the geodesic. Consequently, the gradients accumulated in the resulting momentum now lie in the tangent space of $\mathbf{u}_k$ at each step. This neutralizes the additional radial terms phenomena from $\mathbf{c}_k^{\text{grad}}$. Since $\mathbf{u}_{k-1}$, $\mathbf{u}_k$ and $\mathbf{a}_k$ are coplanar, the transport of the momentum on the hypersphere can be expressed as a rotation:

$$\mathsf{T}(\mathbf{a}_{k-1}) \overset{\text{def}}{=} \Gamma_{\mathbf{u}_{k-1}}^{\mathbf{u}_k}(\mathbf{a}_{k-1}) = \langle \mathbf{u}_{k-1}, \mathbf{u}_k \rangle \mathbf{a}_{k-1} - \langle \mathbf{a}_{k-1}, \mathbf{u}_k \rangle \mathbf{u}_{k-1}, \tag{3.70}$$

$$\mathbf{a}_k = \beta \mathsf{T}(\mathbf{a}_{k-1}) + \nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k. \tag{3.71}$$

Although the transport operation is strictly defined on the tangent space only, the scalar product formulation enables its extension to the whole space. The transformation is

linear and $\mathsf{T}(\mathbf{u}_{k-1}) = 0$. We thus have:

$$\mathsf{T}(\mathbf{a}_{k-1} - \lambda \mathbf{u}_{k-1}) = \mathsf{T}(\mathbf{a}_{k-1}). \tag{3.72}$$

In the previous formulation, we see that the L_2 component is not transported and does not contribute in the new momentum. Finally, the momentum only contains the contribution of the current L_2 regularization. This means that the RT transformation decouples the L_2 regularization and thus neutralizes the additional radial terms from $\mathbf{c}_k^{L_2}$.

Adam without deformed gradients, no additional radial terms and no radius ratio (abc). To avoid the ratio $\frac{r_k}{r_i}$ in the effective learning direction and thus to cancel $(\mathbf{c})$, we rescale the momentum in the update by the factor $\frac{r_{k-1}}{r_k}$ at each iteration $k \geq 1$. From Lemma. 1, we obtain:

$$\mathsf{R}(\mathbf{a}_{k-1}) \overset{\text{def}}{=} \frac{r_{k-1}}{r_k}\mathbf{a}_{k-1} \tag{3.73}$$

$$\mathbf{a}_k = \beta \mathsf{R}(\mathbf{a}_{k-1}) + \nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k \tag{3.74}$$

$$= \frac{1}{r_k}\Big(\sum_{i=0}^{k} \beta^{k-i} (\nabla \mathcal{L}(\mathbf{u}_k) + \lambda r_k r_i \mathbf{u}_i) \Big). \tag{3.75}$$

Note that now, the factor $\frac{r_k}{r_i}$ is not contained anymore in the gradient contribution of $\mathbf{c}_k = r_k \mathbf{a}_k$, which neutralizes the radius ratio phenomenon. We can note that R and T are commutative and that we can combine them in a simple concise scalar expression:

$$\mathsf{RT}(\mathbf{a}_{k-1}) \overset{\text{def}}{=} \frac{\langle \mathbf{x}_k, \mathbf{x}_{k-1} \rangle \mathbf{a}_{k-1} - \langle \mathbf{x}_k, \mathbf{a}_{k-1} \rangle \mathbf{x}_{k-1}}{\langle \mathbf{x}_k, \mathbf{x}_k \rangle}, \tag{3.76}$$

$$\mathbf{a}_k = \beta \mathsf{RT}(\mathbf{a}_{k-1}) + \nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k. \tag{3.77}$$

This new momentum leads to $\mathbf{c}_k = \mathbf{c}_k^{\mathsf{RT}} + r_k^2 \lambda \mathbf{u}_k$ with $\langle \mathbf{c}_k, \mathbf{u}_k \rangle = \lambda r_k^2$ and $\mathbf{c}_k^{\perp} = \mathbf{c}_k^{\mathsf{RT}}$. The latter relies only on the trajectory on the hypersphere and always lies in the tangent space:

$$\mathbf{c}_k^{\mathsf{RT}} = \beta \Gamma_{\mathbf{u}_{k-1}}^{\mathbf{u}_k}(\mathbf{c}_{k-1}^{\mathsf{RT}}) + \nabla \mathcal{L}(\mathbf{u}_k). \tag{3.78}$$

Algorithm 1 Adam w/o (abc) and its algorithm illustrated for a filter $\mathbf{x} \in \mathbb{R}^d$ followed by BN. Steps that are different from Adam are shown in highlight. For non-convolutional layers we use standard Adam.

procedure ADAM W/O (ABC) ▷ Require: $\beta_1, \beta_2 \in [0,1]$; $\lambda, \eta \in \mathbb{R}$; $\mathcal{L}(\mathbf{x})$
  while *stopping criterion not met* **do**
    $k \leftarrow k+1$
    $\mathbf{g} \leftarrow \nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k$
    $\mathbf{m}_k \leftarrow \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)\mathbf{g}$
    $v_k \leftarrow \beta_2 v_{k-1} + (1-\beta_2) d^{-1} \mathbf{g}^\top \mathbf{g}$
    $\hat{\mathbf{m}} \leftarrow \mathbf{m}_k / (1-\beta_1^{k+1})$
    $\hat{v} \leftarrow v_k / (1-\beta_2^{k+1})$
    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - \eta \hat{\mathbf{m}} / (\sqrt{\hat{v}} + \epsilon)$
    $\mathbf{m}_k \leftarrow \mathbf{m}_k (\mathbf{x}_{k+1}^\top \mathbf{x}_k \mathbf{m}_k - \mathbf{m}_k^\top \mathbf{x}_{k+1} \mathbf{x}_k) / (\mathbf{x}_{k+1}^\top \mathbf{x}_{k+1})$
    $v_k \leftarrow v_k (\mathbf{x}_k^\top \mathbf{x}_k / \mathbf{x}_{k+1}^\top \mathbf{x}_{k+1})$
  end while
  return parameters $\mathbf{x}_k$
end procedure

The final Adam w/o (abc) scheme reads:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \frac{\mathbf{m}_k}{1-\beta_1^{k+1}} \Big/ \sqrt{\frac{v_k}{1-\beta_2^{k+1}} + \epsilon}, \tag{3.79}$$

$$\mathbf{m}_k = \beta_1 \mathsf{RT}(\mathbf{m}_{k-1}) + (1-\beta_1)(\nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k), \tag{3.80}$$

$$v_k = \beta_2 \frac{r_{k-1}^2}{r_k^2} v_{k-1} + (1-\beta_2) d^{-1} \|\nabla \mathcal{L}(\mathbf{x}_k) + \lambda \mathbf{x}_k\|^2. \tag{3.81}$$

We also rescale the introduced scalar v_k at each step with the factor $\frac{r_{k-1}^2}{r_k^2}$. This removes the radius from the gradient contribution of the scheduling $\nu^{\mathsf{R}} = r_k v_k$, in contrast with ν_k from Eq. (3.42). The new scheduling effect reads:

$$\nu_k^{\mathsf{R}} = d^{-1/2} \frac{1-\beta_1^{k+1}}{1-\beta_1} \sqrt{\frac{1-\beta_2}{1-\beta_2^{k+1}}} \Big(\sum_{i=0}^{k} \beta_2^{k-i} \|\nabla \mathcal{L}(\mathbf{u}_i) + \lambda r_i r_k \mathbf{u}_i\|^2 \Big)^{1/2}.$$

3.5.3 Empirical study.

To assess empirically the significance of the above phenomena in the context of CNNs with BN and BN w/o affine, we evaluate the different variants of AdamW, AdamG, Adam w/o (a), w/o (ab), w/o (abc) over a variety of datasets and architectures.

Protocol. For evaluation, we conduct experiments on two architectures: VGG16 (Simonyan and Zisserman, 2014) and ResNet (He et al., 2016a) – more precisely ResNet20, a simple variant designed for small images (He et al., 2016a), and ResNet18, a popular variant for image classification. We consider three datasets: SVHN (Netzer et al., 2011), CIFAR10 and CIFAR100 (Krizhevsky et al., 2009). Since our goal is to evaluate the significance of phenomena on radially-invariant parameters, i.e., the convolution filters followed by BN, we only apply variants of Adam including AdamG and AdamW on convolution layers. The algorithm of Adam w/o (abc) is illustrated in Algorithm 1. Note, the optimization schemes introduced in this chapter do not change the complexity in time of the algorithm. During the update of parameters in a layer, we only do a temporary copy of the parameter tensor just before the update to perform the RT transformation. This temporary copy is flushed after the RT transformation. Nothing permanent is stored in the optimizer. For comparison consistency, we keep standard Adam on the remaining parameters, and we use a fixed grid hyperparameter search budget and frequency for each method and each architecture. For each optimization scheme, each dataset and each architecture, the same grid search range and budget was performed while mini-batch size was fixed. We used a mini-batch size of 128 for SVHN, CIFAR10 and CIFAR100. The learning rates η varied in $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$, the weight decay in $10^{-3} \cdot \{0, \frac{1}{128}, \frac{1}{64}, \frac{1}{32}, \frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}\}$ (similar to Loshchilov and Hutter (2019)), the momentum was fixed to 0.9 (β_1 for variants of Adam) and the order-two moment β_2 in $\{0.99, 0.999, 0.9999\}$ (as in Kingma and Ba (2014)). We used the same step-wise learning rate scheduler for each method. For SVHN, CIFAR10 and CIFAR100, models were trained for 405 epochs, and the learning rate multiplied by 0.1 at epochs 135, 225 and 315. For each architecture and each dataset, the same learning rate was systematically found for each method while the momentum factor was fixed at 0.9 (cf. appendix table B.1). Best other hyperparameters, i.e., L_2 regularization and order-2 moment, are shown in appendix table B.2. Besides, please also remember that the impact of the scaling and bias parameters (which belongs to the remaining non radially-invariant parameters) is out of the scope of this study. Nevertheless, we additionally evaluate the variants of Adam on CNNs with BN without scaling and bias parameters (BN w/o affine) in table 3.2 and observe marginal performance difference in comparison to CNNs with standard BN. It hints that the normalization layer in BN plays the most important role regarding the model performance.

Results. In Table 3.3, we report quantitative results of Adam variants across architectures and datasets. To indicate that AdamW and AdamG are actually only used on convolutional filters, while Adam is used for the other parameters, we denote the

Table 3.2 Accuracy of Adam and its variants when training with BN w/o affine layers. The figures in this table are the mean top1 accuracy ± the standard deviation over 5 seeds on the test set for CIFAR10, CIFAR100 and on the validation set for SVHN. † indicates that the original method is only used on convolutional filters while Adam is used for other parameters.

	CIFAR10			CIFAR100		SVHN	
Method	ResNet20	ResNet18	VGG16	ResNet18	VGG16	ResNet18	VGG16
Adam	90.41 ± 0.06	93.67 ± 0.15	92.62 ± 0.15	71.60 ± 0.22	68.28 ± 0.19	95.29 ± 0.11	95.56 ± 0.18
AdamW†	90.36 ± 0.11	93.7 ± 0.16	93.03 ± 0.12	70.11 ± 0.31	69.68 ± 0.12	89.83 ± 0.28	95.63 ± 0.11
AdamG†	91.12 ± 0.09	93.62 ± 0.14	93.20 ± 0.20	69.96 ± 0.34	70.07 ± 0.23	95.12 ± 0.09	95.62 ± 0.21
Adam w/o (a)	91.15 ± 0.11	93.98 ± 0.18	93.12 ± 0.14	75.43 ± 0.13	70.01 ± 0.24	95.75 ± 0.09	95.64 ± 0.10
Adam w/o (ab)	**91.38** ± 0.08	**94.63** ± 0.08	93.45 ± 0.06	**75.68** ± 0.22	71.74 ± 0.15	**95.77 ± 0.08**	**95.78** ± 0.07
Adam w/o (abc)	91.11 ± 0.11	94.02 ± 0.10	**93.56** ± 0.09	75.38 ± 0.21	**72.08** ± 0.22	95.51 ± 0.08	95.69 ± 0.09

Table 3.3 Accuracy of Adam and its variants when training with BN layers.

The figures in this table are the mean top1 accuracy ± the standard deviation over 5 seeds on the test set for CIFAR10, CIFAR100 and on the validation set for SVHN. † indicates that the original method is only used on convolutional filters while Adam is used for other parameters.

	CIFAR10			CIFAR100		SVHN	
Method	ResNet20	ResNet18	VGG16	ResNet18	VGG16	ResNet18	VGG16
Adam	90.98 ± 0.06	93.77 ± 0.20	92.83 ± 0.17	71.30 ± 0.36	68.43 ± 0.16	95.32 ± 0.23	95.57 ± 0.20
AdamW†	90.19 ± 0.24	93.61 ± 0.12	92.53 ± 0.25	67.39 ± 0.27	71.37 ± 0.22	95.13 ± 0.15	94.97 ± 0.08
AdamG†	91.64 ± 0.17	94.67 ± 0.12	93.41 ± 0.17	73.76 ± 0.34	70.17 ± 0.20	95.73 ± 0.05	95.70 ± 0.25
Adam w/o (a)	91.15 ± 0.11	93.95 ± 0.23	92.92 ± 0.11	74.44 ± 0.22	68.73 ± 0.27	95.75 ± 0.09	95.66 ± 0.09
Adam w/o (ab)	**91.92** ± 0.18	**95.11** ± 0.10	**93.89** ± 0.09	**76.15** ± 0.25	**71.53** ± 0.19	**96.05** ± 0.12	**96.22** ± 0.09
Adam w/o (abc)	91.81 ± 0.20	94.92 ± 0.05	93.75 ± 0.06	75.28 ± 0.35	71.45 ± 0.13	95.84 ± 0.07	95.82 ± 0.05

experimented methods as AdamW† and AdamG†. In addition, we compare the evolution of the training loss in Figure 3.6 and the top1 accuracy on the validation set in Figure 3.7. We observe that each phenomenon displays a specific trade-off between generalization (accuracy on the test set) and training speed, as following. Neutralizing **(a)** has little effect on the speed over Adam, yet achieves better accuracy on the train set. Although it slows down training, neutralizing **(ab)** leads to minima with the overall best accuracy on test set in the case of BN equipped CNNS. Note that AdamW† neutralizes **(b2)** with its decoupling and is the fastest method, but finds minima with overall worst generalization properties. By constraining the optimization to the hypersphere, AdamG† speeds up training over the other variants. Finally, neutralizing **(c)** with Adam w/o **(abc)** brings a slight acceleration, though reaches lower accuracy than Adam w/o **(ab)**. In terms of generalization, before the first decrease of the learning rate, neutralizing **(a)** displays the same speed in terms of reached accuracy on the valid set compared to Adam. When neutralizing **(ab)** and **(abc)**, we observe a slight increase, comparable with AdamG†.

After the first decrease of the learning rate, we observe the same hierarchy as in Table 3.3. These results show that the geometrical phenomena revealed by our analysis in the spherical framework have a significant impact on the training of BN-equipped CNNs.

3.6 Related work

Understanding Batch Normalization. Albeit conceptually simple, BN has been shown to have complex implications over optimization. The argument of Internal Covariate Shift reduction (Ioffe and Szegedy, 2015) has been challenged and shown to be secondary to smoothing of optimization landscape (Ghorbani et al., 2019; Santurkar et al., 2018) or its modification by creating a different objective function (Lian and Liu, 2019), enabling of high learning rates through improved conditioning (Bjorck et al., 2018), or alleviating the sharpness of the Fisher information matrix Karakida et al. (2019). In an analysis of a variant of GD, Kohler et al. (2019) show that BN accelerates optimization by decoupling the optimization of the direction and length of parameters. Daneshmand et al. (2020) argue that BN is an effective strategy to ensure that activations generated by a randomly-initialized network have high rank, unlike vanilla networks where the rank in the final layers collapses with depth Saxe et al. (2013). Arora et al. (2019) demonstrate that (S)GD with BN is robust to the choice of the learning rate, with guaranteed asymptotic convergence, while a similar finding for GD with BN is made in Cai et al. (2019).

Invariances in neural networks. Cho and Lee (2017) propose optimizing over the Grassmann manifold using Riemannian GD. Liu et al. (2017) project weights and activations on the unit hypersphere and compute a function of the angle between them instead of inner products, and subsequently generalize these operators by scaling the angle (Liu et al., 2018). In Li and Arora (2020) the radial invariance is leveraged to prove that weight decay (WD) can be replaced by an exponential learning-rate scheduling for SGD with or without momentum. Arora et al. (2019) investigate the radial invariance and show that radius dynamics depends on the past gradients, offering an adaptive behavior to the learning rate. Here we go further and show that SGD projected on the unit hypersphere corresponds to Adam constrained to the hypersphere, and we give an accurate definition of this adaptive behavior.

Effective learning rate. Due to its scale invariance, BN can adaptively adjust the learning rate (Arora et al., 2019; Cho and Lee, 2017; Li and Arora, 2020; van Laarhoven,

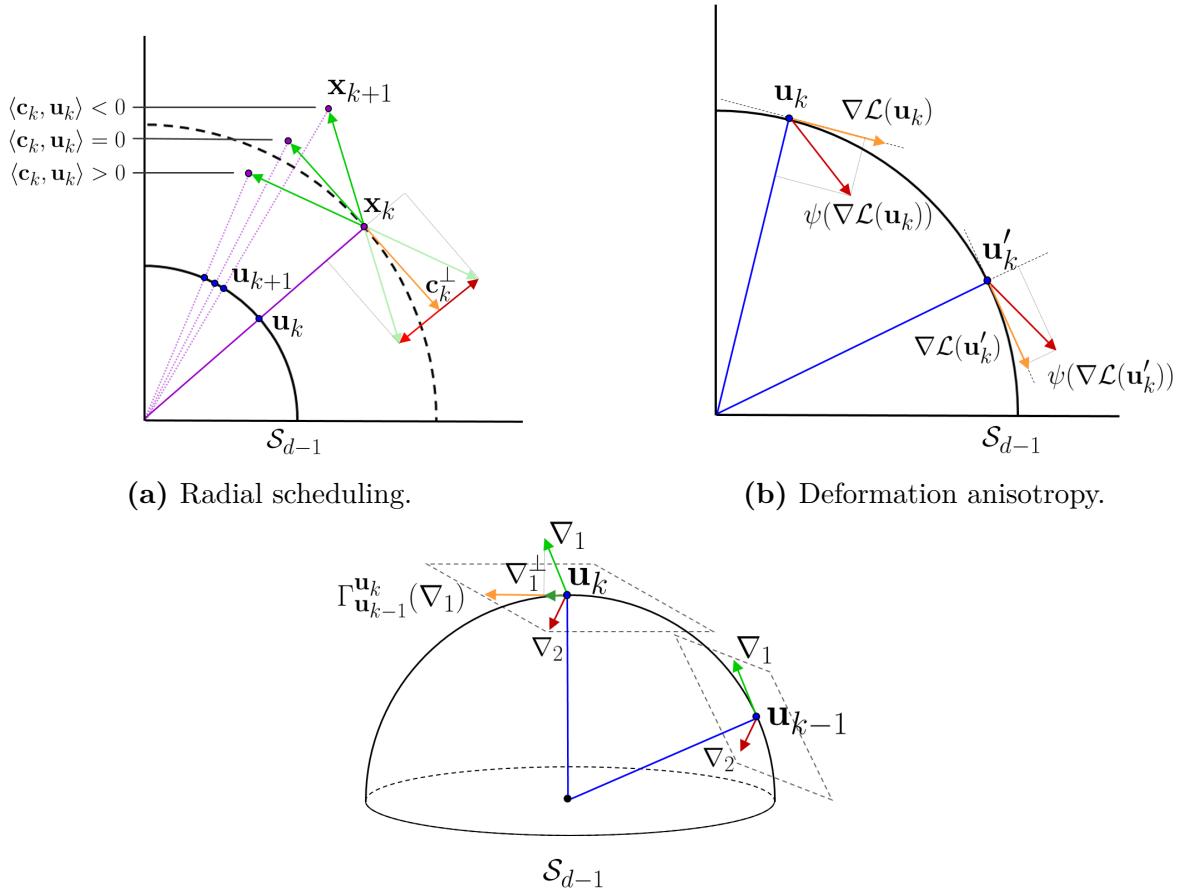
2017). van Laarhoven (2017) shows that in BN-equipped networks, WD increases the effective learning rate by reducing the norm of the weights. Conversely, without WD, the norm grows unbounded (Soudry et al., 2018), decreasing the effective learning rate. Zhang et al. (2019) bring additional evidence supporting hypothesis in van Laarhoven (2017), while Hoffer et al. (2018a) find an exact formulation of the effective learning rate for SGD in normalized networks. In contrast with prior work, we find generic definitions of the effective learning rate with exact expressions for SGD and Adam.

3.7 Limitations

Our study only concerns the optimization of radially-invariant parameters. It does not include the impact on other parameters, e.g., for CNNs with BN, scaling and bias in BN layers, and last linear layer.

3.8 Conclusion

The spherical framework introduced in this study provides a powerful tool to analyse Adam optimization scheme through its projection on the L_2 unit hypersphere. It allows us to give a precise definition and expression of the effective learning rate for Adam, to relate SGD to a variant of Adam, and to identify geometric phenomena which empirically impact training. The framework also brings light to existing variations of Adam, such as L_2-regularization decoupling. The geometry of invariance properties appears as a promising research direction toward a better understanding of their impact on optimization.



(a) Radial scheduling.

(b) Deformation anisotropy.

(c) Gradient history contribution.

Fig. 3.5 Geometrical phenomena in Adam. (a) Effect of the radial part of $\mathbf{c}_k$ on the displacement on $\mathcal{S}_{d-1}$; (b) Example of anisotropy and sign instability for the deformation $\psi(\nabla\mathcal{L}(\mathbf{u}_k)) = \nabla\mathcal{L}(\mathbf{u}_k) \oslash \frac{|\nabla\mathcal{L}(\mathbf{u}_k)|}{d^{-1/2}\|\nabla\mathcal{L}(\mathbf{u}_k)\|}$ (where $|\cdot|$ is the element-wise absolute value) occurring in Adam's first optimization step; (c) Different contribution in $\mathbf{c}_k^\perp$ of two past gradients ∇_1 and ∇_2 of equal norm, depending on their orientation. Illustration of the transport of ∇_1 from $\mathbf{u}_{k-1}$ to $\mathbf{u}_k$: $\Gamma_{\mathbf{u}_{k-1}}^{\mathbf{u}_k}(\nabla_1)$ (*cf.* 3.5.2 for details).

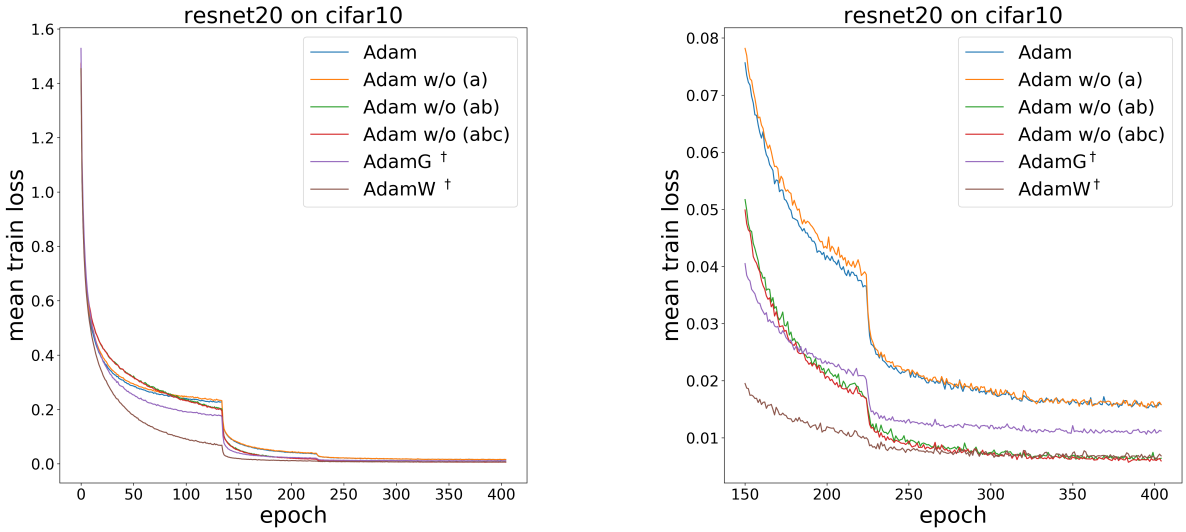


Fig. 3.6 Training speed comparison with ResNet20 BN on CIFAR10. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

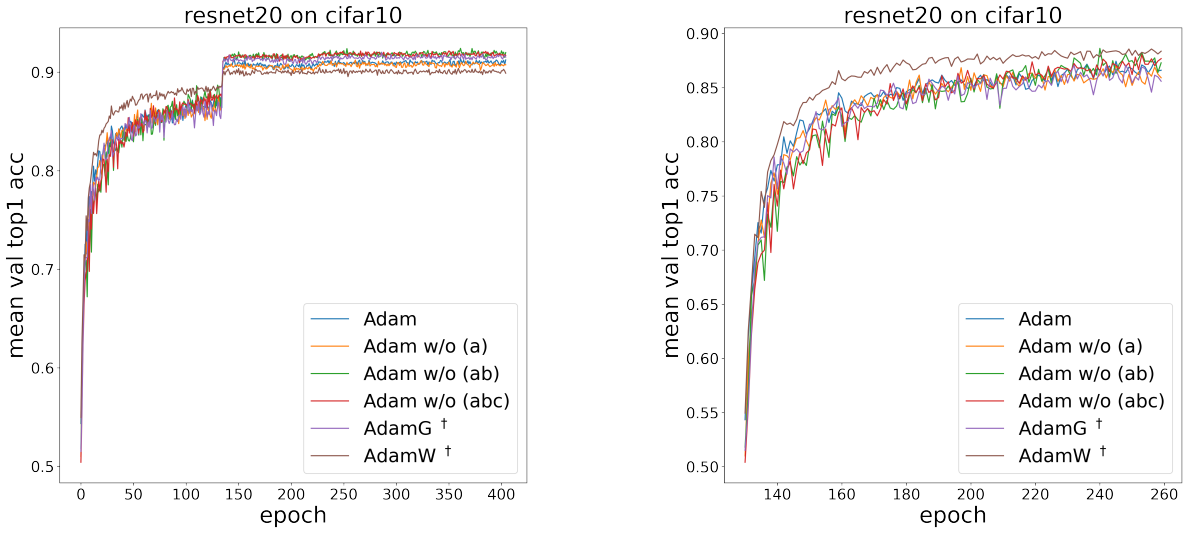


Fig. 3.7 Valid accuracy comparison with ResNet20 BN on CIFAR10. *Left:* Mean valid top1 acc over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the first epochs. Please refer to Table 3.3 for the corresponding accuracies.

Chapter 4

Avoid learning spurious correlations

The work described in this chapter is based on the following publication: **Simon Roburin**, Charles Corbières, Gilles Puy, Nicoles Thome, Mathieu Aubry, Renaud Marlet and Patrick Pérez. **Get One Gram of Neural Style Features, Get Enhanced Group Robustness**.

- Short version of the paper accepted in the Workshop on Out Of Distribution Detection at European Conference on Computer Vision (ECCV) 2022.
- Long version of the paper sumbited at Internation Conference on Learning Representation (ICLR) 2023.

Abstract

Predictive performance of machine learning models trained with empirical risk minimization (ERM) can degrade considerably under distribution shifts. In particular, the presence of spurious correlations in training datasets leads ERM-trained models to display high loss when evaluated on minority groups not presenting such correlations in test sets. Extensive attempts have been made to develop methods improving worst-group robustness. However, they require group information for each training input or at least, a validation set with group labels to tune their hyperparameters, which may be expensive to get or unknown a priori. In this chapter, we address the challenge of improving group robustness *without group annotations during training*. To this end, we propose to partition automatically the training dataset into groups based on Gram matrices of features extracted from an identification model and to apply robust optimization based on these pseudo-groups. In the realistic context where no group labels are available, our

experiments show that our approach not only improves group robustness over ERM but also outperforms all recent baselines.

4.1 Introduction

Empirical Risk Minimization (ERM) (Vapnik, 1991) is the most standard machine learning (ML) formulation, which assumes that training and testing samples are independent and identically distributed (*i.i.d.*). While academic datasets are mainly built to respect the *i.i.d.* assumption, practical settings display more challenging configurations with distribution shifts. Training data might be affected by selection biases and confounding factors, also called *spurious correlations* (Duchi et al., 2019; Woodward, 2005). For example, imagine crowd-sourcing an image dataset of camels and cows (Beery et al., 2018). Due to selection biases, a high majority of cows stand in front of grass environments and camels in the desert. Therefore, a simple way to differentiate cows from camels would be to classify the background, an undesirable shortcut that ERM will naturally exploit. ERM may perform poorly on minority groups that do not display such spurious correlation (Duchi et al., 2019; Hashimoto et al., 2018; Tatman, 2017), *e.g.*, a cow standing in the desert.

Extensive attempts (Ahmed et al., 2021; Arjovsky et al., 2020; Sagawa* et al., 2020; Zhang et al., 2021) have been made to develop new training objectives that are robust to spurious correlations, *e.g.*, by ensuring high worst-group accuracy. However, these approaches require a prior knowledge about the confounding factors during training. This is a major limitation since these factors might be a priori unknown and, if known, ambiguous to define and expensive to annotate. Recent works Ahmed et al. (2021); Creager et al. (2021); Liu et al. (2021); Matsuura and Harada (2020); Sohoni et al. (2020) rely on two-stage schemes, first automatic environment discovery (*e.g.*, based on deep feature clustering), then robust optimization based on environment pseudo-labels. However, all these approaches still require the availability of ground-truth environment labels on a validation set in order to properly tune their hyperparameters.

This chapter addresses the problem of learning a robust classifier, which, for instance, would not confuse a cow standing in the desert with a camel although not given any annotation about grass or desert. In computer vision, many identified spurious correlations are closely related to visual aspects, such as background (Beery et al., 2018), texture (Geirhos et al., 2019), image style (Hendrycks et al., 2021), physic attributes (Liu et al., 2015) or camera characteristics (Koh et al., 2021). In this work, we assume that relevant environment labels can be inferred from visual feature statistics, and demonstrate they lead to meaningful environments and robust classifiers for standard datasets used to evaluate robust classification. We propose a two-stage approach, GramClust, which

first assigns a group label, *i.e.*, a class-environment pair label, by partitioning a training dataset into clusters of images with similar visual statistics and then trains a robust classifier based on these pseudo-group labels. Our approach is summarized in Figure 4.1. We use Gram matrices as visual descriptive statistics, which are second-order moments of neural activation. Gram matrices are well known for displaying impressive results in style transfer techniques (Gatys et al., 2016), but more importantly for the interpretation of our approach, Li et al. (2017) demonstrate that matching Gram matrices between two groups of images is equivalent to aligning the respective distribution of each group, minimizing the Maximum Mean Discrepancy. Therefore, our method can be interpreted as grouping images into clusters of similar feature distributions that are sensible candidates for environments

The chapter is organized as follows. In **section 4.2**, we provide background on group robustness methods with or without group annotation as well as neural style transfer. In **section 4.3**, our two-stages approach is detailed and formalized: firstly we discover without any group supervision pseudo-environment labels, secondly we perform robust training of a model from scratch by minimizing the worst pseud-group accuracy. In **section 4.4**, our method is evaluated on a standard benchmark in group-robustness composed of three datasets. In addition, we provide various empirical analysis of our approach whether it is the importance of using Gram matrices as styles features, the impact of the choice of layers to extract features from or the impact of the number of cluster.

Our main contributions are:

- We introduce an easy-to-scale method to split training images among distinct pseudo-environments, based on feature Gram matrices extracted by a specifically-trained identification model;
- GramClust alleviates the need of ground-truth group labels altogether, even in the validation set, as hyperparameters are set based on validation performance computed from our pseudo-groups;
- Extensive experiments on various image classification datasets with spurious correlations show that GramClust outperforms all recent baselines addressing robustness without group annotation. In particular, on the realistic large-scale CelebA dataset (Liu et al., 2015), we improve worst-group test accuracy by +24.3 points.

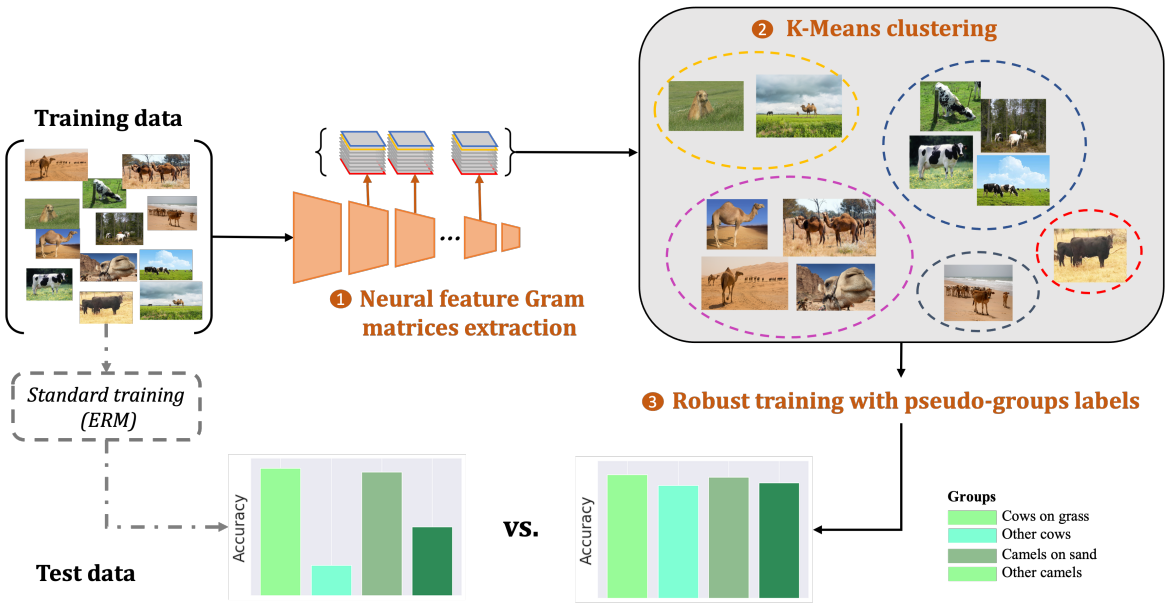


Fig. 4.1 Overview of the proposed approach for robust classification with unsupervised group discovery. (1) We first extract deep image features using an identification model and (2) we cluster the training dataset based on their feature Gram matrices (their "style"'); (3) then, we train the targeted classifier with a robust optimization that exploits the assigned pseudo-group labels.

4.2 Related work

The reliability of ML learning process assumes that the associations between inputs and targets remain similar between training and test distributions (see section 2.1.1). However useful correlations between visual elements of images and their associated classes at training time that results into correct prediction over the training distribution can become useless or even harmful at test time. Such correlations are called *spurious correlations*. Group robustness focus on the specific case where data can be split into a majority group which includes a confounding factor and a minority group that does not. Under these circumstances, a DNN trained over the whole dataset will recklessly absorb these spurious correlation leading to bad performances on minority group. The goal is to train a DNN with good performance both on the majority and minority group. In this section, we cover the main methods developed to perform group-robutsness with or without group annotation. In section 4.4, we evaluate our method against these methods. Then, a quick overview of neural style transfer is provided.

4.2.1 Group Robustness with group annotation.

To improve group robustness, many recent approaches propose to leverage group annotations during training. IRM Arjovsky et al. (2020) augments the standard ERM term with invariance penalties across data from different groups. Similarly, Ahmed et al. (2021) promotes, through a simple penalty, identical prediction behaviour across groups. Other works such as Sagawa* et al. (2020); Zhang et al. (2021) minimize explicitly the worst-group loss during training; Sagawa et al. (2020) re-balances majority and minority groups via re-weighting and sub-sampling.

4.2.2 Group Robustness without group annotation

Here, we focus on a more realistic setting in which group annotations are not available on the training data. Existing approaches undergo two distinct phases: groups discovery and robust training by minimizing the worst pseudo-group accuracy with GroupDRO. Environment Inference for Invariant Learning (EIIL) Creager et al. (2021) derives a group inference objective from a trained identification model that maximizes variability across environments, and is differentiable w.r.t a distribution over group assignments. Just Train Twice (JTT) Liu et al. (2021) is a simple method in which environments are defined by images on which a trained identification model performs poorly. GEORGE Sohoni et al. (2020) is based on an unsupervised clustering algorithm in the feature space of a trained identification model. These methods require implicitly or explicitly a small validation set with group annotation. JTT explicitly tunes its hyperparameters on a small validation set with group annotation while EIIL and GEORGE use best hyperparameters found in the GroupDRO paper Sagawa* et al. (2020) to minimize the worst-group accuracy on their inferred groups. These best hyperparameters were actually found using a validation set with true-group labels in the original study.

4.2.3 Gram matrices

The original work of Gatys et al. (2016) demonstrated impressive results to generate images with the style of an existing image. The style of a first image is transferred to a second one by matching Gram matrices of features extracted by a convolutional neural network. Sastry and Oore (2020) also used Gram matrices in out-of-distribution detection to identify an anomaly by comparing their values to the respective range observed over the training data. Interestingly, Li et al. (2017) demonstrate a formal equivalence between matching Gram matrices of neural activations with an L_2 norm and the MMD with the second-order polynomial kernel. This shows that Gram matrices are also implicitly used

in the process of distribution alignment between images. This finding motivates our approach, which consists in discovering pseudo-groups using Gram matrices.

4.3 GramClust: A Clustering Approach for Robust Optimization

Our method, GramClust, consists of two main steps. First, we discover pseudo-environments among the images of a given dataset (see section 4.3.2). Second, we train a robust classifier that leverages the inferred pseudo-environment labels to reduce classification errors due to spurious environment correlations (see section 4.3.3). To discover environments, we train during a few iterations an exogenous "identification model". Then, using this model, we compute for each image its Gram matrix representation from different layers and apply random projections to reduce dimension. The resulting concatenated features are then fed to an unsupervised clustering algorithm (k-means) to produce pseudo-environment labels. This allows us to define pseudo-groups as the intersection of pseudo-environments and classes. Last, we train the target classifier by minimizing the standard cross-entropy classification loss on the worst pseudo-group with GroupDRO (Sagawa* et al., 2020).

4.3.1 Problem formulation

Let us consider a dataset $\mathcal{D} = \{(\mathbf{s}_i, t_i)\}_{i=1}^N \in (\mathbb{R}^{\text{in}} \times \mathbb{R}^{\text{out}})$ of N samples where $\mathbb{R}^{\text{in}}$ represents the input space and $\mathbb{R}^{\text{out}} = [\![1, K]\!]$ is a set of labels. We assume the data is sampled from random variables (S_e, T_e) in $\mathbb{R}^{\text{in}} \times \mathbb{R}^{\text{out}}$ with probability law $\mathbb{P}_{(S_e,T_e)}$ for all $e \in [\![1, E]\!]$, where E is the number of *environments*. The full dataset can then be seen as the union of subsets associated to each random variable, *i.e.*, $\mathcal{D} = \bigcup_{e=1}^E \mathcal{D}_e$ where each $\mathcal{D}_e$ is composed of *i.i.d.* realisations of a random variable with joint probability law $\mathbb{P}_{(S_e,T_e)}$. For notation purposes, we choose the following equivalent formulation for the dataset $\mathcal{D} = \{(\mathbf{s}_i, t_i, e_i)\}_{i=1}^N \in (\mathbb{R}^{\text{in}} \times \mathbb{R}^{\text{out}} \times [\![1, E]\!])^N$ where e_i refers to the environment from which $\mathbf{s}_i$ and t_i were sampled.

Our goal is to find a model m in a given hypothesis space $\mathcal{H}$ which minimizes the error on the worst group. A *group* is defined as a set of samples from the same class and

in the same environment. Formally, we introduce group distributions:

$$\mathbb{P}_{G_{1,1}} = \mathbb{P}(S_1|T_1 = 1), \tag{4.1}$$

$$\vdots$$

$$\mathbb{P}_{G_{E,K}} = \mathbb{P}(S_E|T_E = K). \tag{4.2}$$

The purpose is then to minimize the following objective:

$$\hat{m} \in \arg\min_{m \in \mathcal{H}} \left\{ \max_{g \in [\![1,E]\!] \times [\![1,K]\!]} \mathbb{E}_{(\mathbf{s},t)\sim\mathbb{P}_{G_g}} \Big[\ell(m(\mathbf{s}), t) \Big] \right\}, \tag{4.3}$$

where $\ell : \mathbb{R}^{\text{out}} \times \mathbb{R}^{\text{out}} \to \mathbb{R}^+$ is the cross-entropy loss between the model's prediction and the true label. Note that we do not have access to any of the environment labels. To circumvent this issue, we first discover pseudo-environment labels and then estimate the pseudo-group distributions to be used in Eq. 4.3.

4.3.2 Dataset partition

In this section, we describe the first stage of GramClust, which aims at environment discovery. The method is illustrated in Figure 4.2.

Identification model. Our approach starts by initializing a convolutional neural network Φ for the classification task at hand; it is composed of L layers with parameters $\boldsymbol{\omega}$ and is pre-trained on ImageNet (Deng et al., 2009).

Previous work Liu et al. (2021) observed that ERM tends to fit models on data presenting easy-to-learn spurious correlations at the beginning of the learning process. It is only after a significant number of epochs that the model starts to learn more difficult patterns. Hence, we only train Φ during a few iterations, minimizing the following empirical loss function:

$$\min_{\boldsymbol{\omega}} \frac{1}{N} \sum_{i=1}^{N} \ell(\Phi_{\boldsymbol{\omega}}(\mathbf{s}_i), t_i), \tag{4.4}$$

where $\ell : \mathbb{R}^{\text{out}} \times \mathbb{R}^{\text{out}} \to \mathbb{R}^+$ is the cross-entropy loss between the model's predicted label $\Phi_{\boldsymbol{\omega}}(\mathbf{s}_i)$ and the true label t associated with sample $\mathbf{s}$. In the following, we call Φ the *identification model* as our clustering is based on features extracted from this model. The idea is to leverage the biases learned by Φ to identify relevant environments and efficiently partition the training dataset into groups of images presenting spurious correlations on one side and groups of images free from these correlations on the other side. Hence, after

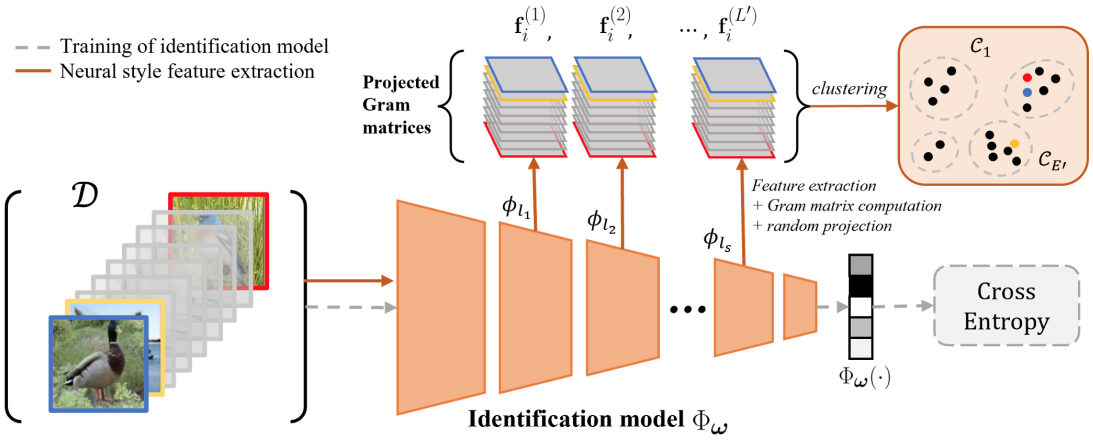


Fig. 4.2 Illustration of the style-based dataset partition. An identification model Φ with parameters $\boldsymbol{\omega}$ is trained for a limited number of epochs T with ERM to fit groups with easy-to-learn spurious correlations. Then, for each image $\mathbf{s}_i \in \mathbb{R}^{\text{in}}$, we extract intermediate features $\phi^{(l)}$ and compute their Gram matrix $\mathbf{G}_l$ with a random projection. These projected Gram matrix representations are used as features to cluster the training dataset $\mathcal{D}_{\text{train}}$ in E' environments.

this initial training and in the rest of the chapter, the parameters $\boldsymbol{\omega}$ of the identification model Φ are frozen.

Features Gram matrix. We denote the feature map of an image $\mathbf{s}$ at layer l of Φ by $\phi^{(l)}(\mathbf{s}) \in \mathbb{R}^{M_l \times C_l}$, where C_l is the number of channels and M_l is the spatial dimension of the feature map. For each image $\mathbf{s}_i \in \mathbb{R}^{\text{in}}$, we extract its feature maps at $L' \leq L$ different and fixed layers $\{l_1, \cdots l_{L'}\}$, and compute the Gram matrices defined as:

$$\mathbf{G}_l(\mathbf{s}_i) = \frac{1}{M_l}\, \phi^{(l)}(\mathbf{s}_i)^\intercal \phi^{(l)}(\mathbf{s}_i) \in \mathbb{R}^{C_l \times C_l}, \quad l \in \{l_1, \cdots l_{L'}\}. \tag{4.5}$$

Given input $\mathbf{s}_i$ and identification model Φ, the Gram matrix of its feature $\phi^{(l)}(\mathbf{s}_i)$ ecodes visual correlations via an inner product between each pair of vectorized feature maps. In visual style transfer (Gatys et al., 2016), these Gram matrices have been shown to encode the "style" of an image, that is, loosely speaking, its textures and color palette, by contrast with its "structure".

Clustering with k-means. For each image $\mathbf{s}_i$, we vectorize and normalize its L' associated Gram matrices: $\mathbf{f}_i^{(l)} = \text{vec}(\mathbf{G}_l(\mathbf{s}_i))/\|\text{vec}(\mathbf{G}_l(\mathbf{s}_i))\|_2 \in \mathbb{R}^{C_l^2}$. The normalization

permits us to have all the Gram matrices contributing equally in the clustering loss. Each image $\mathbf{s}_i$ is thus encoded by the vector $\mathbf{f}_i = \left[\mathbf{f}_i^{(l)}, \ldots, \mathbf{f}_i^{(L')}\right] \in \mathbb{R}^C$, where $C = \sum_l C_l^2$. Relying on our assumption that the environments can be identified from visual feature statistics, we propose to discover E' environments by clustering the N training images into E' clusters $\{\mathcal{C}_1, \ldots \mathcal{C}_{E'}\}$ via k-means clustering, i.e., by computing a solution to:

$$\min_{\{\mathcal{C}_1, \ldots \mathcal{C}_{E'}\}} \sum_{e=1}^{E'} \frac{1}{2|\mathcal{C}_e|} \sum_{i,j \in \mathcal{C}_e} \|\mathbf{f}_i - \mathbf{f}_j\|_2^2, \tag{4.6}$$

where $\|\mathbf{f}_i - \mathbf{f}_j\|_2^2 = \sum_{l=1}^{S} \|\mathbf{f}_i^{(l)} - \mathbf{f}_j^{(l)}\|_2^2$. Li et al. (2017) demonstrate that matching Gram matrices is actually equivalent to distribution alignment using the Maximum Mean Discrepancy distance with the second-order polynomial kernel. Built on the Euclidean distance of normalized Gram matrices (Eq.4.6), our method can hence be interpreted as grouping images into clusters of similar feature distributions that are likely candidates for environments.

Scaling with random projections. Storing all these vectors and computing distances between them in a high-dimensional space is computationally and memory expensive on large datasets. We overcome this difficulty by projecting the vectors $\mathbf{f}_i^{(l)}$ in a lower-dimensional space as proposed in Achlioptas (2003). We build a matrix $\mathbf{P} \in \mathbb{R}^{\ell_0 \times C}$ whose entries $\mathbf{P}_{mn}$ are the realisation of independent random variables: $\mathbf{P}_{mn} = 1$ or $\mathbf{P}_{mn} = -1$ with probability $1/2$. Then we compute:

$$\tilde{\mathbf{f}}_i^{(l)} = \frac{1}{\sqrt{\ell_0}} \mathbf{P} \mathbf{f}_i^{(l)} \tag{4.7}$$

and substitute $\tilde{\mathbf{f}}_i^{(l)}$ for $\mathbf{f}_i^{(l)}$ in Eq. 4.6. We justify this choice by the fact that this projection preserves the distances $\|\mathbf{f}_i^{(l)} - \mathbf{f}_j^{(l)}\|_2^2$ involved in the k-means objective of Eq. 4.6. Indeed, let $\epsilon \in]0, 1[$ and $\ell_0 \propto \log(N)$, then with high probability,[1]

$$(1-\epsilon)\|\mathbf{f}_i^{(l)} - \mathbf{f}_{j,l}\|_2 \;\leq\; \|\tilde{\mathbf{f}}_i^{(l)} - \tilde{\mathbf{f}}_j^{(l)}\|_2 \;\leq\; (1+\epsilon)\|\mathbf{f}_i^{(l)} - \mathbf{f}_j^{(l)}\|_2, \tag{4.8}$$

for all $(i, j) \in [\![1, N]\!]^2$. In practice, we choose $\ell_0 = \lfloor 100 \log(N) \rfloor$ which yields dimensions for $\tilde{\mathbf{f}}_i^{(l)}$ much lower than typical values of C. We remark that this choice of projection is independent of all $\mathbf{f}_i^{(l)}$ and thus can be defined and fixed before any feature extraction.

[1] We let the reader refer to Theorem 1.1 in Achlioptas (2003) for the exact expression of this probability as a function of ϵ, N, and ℓ_0.

4.3.3 Robust optimization with pseudo-group labels

For all $i \in [\![1, N]\!]$, based on our estimated environments labels $\hat{e}_i$, we define for each image, pseudo-groups $\hat{g}_i$ as the intersection of a pseudo-environment and a class. Formally, $\hat{e}_i \in [\![1, E']\!]$ and therefore $\hat{g}_i = (\hat{e}_i, t_i) \in [\![1, E']\!] \times [\![1, K]\!]$. Going back to Eq. 4.3, the distribution over the groups $\mathbb{P}_{G_{\hat{g}}}$ are estimated by

$$\hat{\mathbb{P}}_{G_{\hat{g}}} = \delta(\mathcal{G}(\hat{g})) \qquad \text{for all } \hat{g} \in [\![1, E']\!] \times [\![1, K]\!], \tag{4.9}$$

where δ is the Dirac distribution, and:

$$\mathcal{G}(1,1) = \{(\mathbf{s}_i, t_i), i \in [\![1, N]\!] \mid t_i = 1, \mathbf{s}_i \in \mathcal{C}_1, \} \tag{4.10}$$

$$\vdots$$

$$\mathcal{G}(E', K) = \{(\mathbf{s}_i, t_i), i \in [\![1, N]\!] \mid t_i = K, \mathbf{s}_i \in \mathcal{C}_{E'}\} \tag{4.11}$$

are the sets of images and labels associated with the pseudo-group labels. Each training point $\mathbf{s}_i \in \mathbb{R}^{\text{in}}$ is now associated with a class label t_i and a pseudo-group annotation $\hat{g}_i$. We train a robust classifier h with parameters $\mathbf{x}$ by minimizing the worst-group risk on the training dataset (*GroupDRO* Sagawa* et al. (2020)):

$$\hat{\mathbf{x}} \in \arg\min_{\mathbf{x}} \left\{ \max_{\hat{g} \in [\![1,E']\!] \times [\![1,K]\!]} \frac{1}{|\mathcal{G}(\hat{g})|} \sum_{(\mathbf{s},t) \in \mathcal{G}(\hat{g})} \left[\ell(h_{\mathbf{x}}(\mathbf{s}), t) \right] \right\}, \tag{4.12}$$

where the loss $\ell : \mathbb{R}^{\text{out}} \times \mathbb{R}^{\text{out}} \to \mathbb{R}^+$ remains the cross-entropy between robust classifier's predicted label $h_{\mathbf{x}}(\mathbf{s})$ and the true label t associated with sample $\mathbf{s}$.

4.3.4 Model selection via cross-validation on validation data

Setting relevant hyperparameters is important in optimization algorithms to ensure a proper convergence. Hyperparameters tuning is performed with cross-validation using a held-out subset of training data. With robust optimization, worst-group accuracy of the final classifier is the go-to metric for model selection. Previous approaches rely on *true* group labels of the validation set to define and assess performance on the worst group. In contrast, we do not rely on such a prior information. We partition the validation set using the clusters found on the training set and we conduct cross-validation based on the resulting pseudo-groups. In our experiments, we observe that this type of model selection is effective to achieve proper group robustness.

Fig. 4.3 Illustration of the three datasets.

4.4 Experiments

In this section, we evaluate the capacity of GramClust to improve group robustness on image classification datasets with spurious correlations. In section 4.4.2, we empirically show that it outperforms, on three datasets, other baselines addressing robustness without group annotation. We then present in section 4.4.3 an empirical analysis of our approach, including: the importance of using Gram matrices as clustering features, the impact of the choice of layers to extract features from, and the impact of the number of clusters.

4.4.1 Setup

Datsets. We experiment with three image classification datasets on which previous works evaluate worst-group performance. These datasets are illustrated in Figure 4.3.

- **Waterbirds** Sagawa* et al. (2020) is a dataset composed of images combining bird photographs from the CUB dataset (Welinder et al., 2010) with background scenes taken from the Places365 dataset (Zhou et al., 2018). The target labels are "landbirds" and "waterbirds" which are spuriously correlated with the background images of either "land" or "water". The train set is composed of 4,795 images and the validation and test set are respectively composed of 1,199 and 2,897 images.

- **CelebA** (Liu et al., 2015) is a celebrity face dataset with 202,599 images. Sagawa* et al. (2020) considered the task of classifying the hair color of the individual as "blond" or "not blond". The authors observed that there exists a spurious correlation between the hair color and the gender ("male" or "female") of a person. For instance, in the dataset, only 2% of blond people are male. We use the official train-val-test split from Liu et al. (2015).

- **COCO-on-Places-224** is a dataset of 10 segmented COCO (Lin et al., 2014) objects superimposed on scenes from the Places365 dataset. Training set has 7,200 training images – 800 images per category – and validation and test sets are composed of 900 images – 100 images per category. As in the Waterbirds dataset, the background has spurious correlation with the object classification. At test time, the evaluation is performed either on images that possess the same biases as the training set (*in-distribution*) or images with objects correlated to backgrounds that were not seen at training time (*systematically-shifted*). We rebuilt this dataset based on the code provided by Ahmed et al. (2021)[2] but with images resized to 224×224 (instead of 64×64 in the original paper, which considerably degrades visual features of object and background).

Baselines. We compare our approach against the standard **ERM** baseline and recent methods that aim at robust predictions across groups without the use of train group annotations (**EIIL** Creager et al. (2021), **GEORGE** Sohoni et al. (2020) and **JTT** Liu et al. (2021)). We also include robust methods that use train group annotations (**IRM** Arjovsky et al. (2020), importance weighting and **GroupDRO** Sagawa* et al. (2020)). The latter methods and ERM were already implemented and we took care to reproduce results for all methods. Our results with baselines are in line with those reported respectively in each original paper. Note that our approach and GroupDRO share the same robust optimization objective (Eq. 4.3). Hence, GramClust would boil down to GroupDRO if discovered pseudo-groups were to match exactly the ones annotated in the dataset.

Training details. All methods use a ResNet-50 architecture pre-trained on ImageNet Deng et al. (2009) as the robust classifier. Models are optimized using SGD with momentum. For GroupDRO and ERM, we use the hyperparameters reported by the authors on Waterbirds and CelebA datasets. Further training details are available in appendix C.2. Note that hyperparameters have been selected with the use of a validation set with group labels. Regarding our approach, we select a VGG-19 Simonyan and Zisserman (2015) architecture for the identification model Φ and train it for 1 epoch using SGD with momentum. Among usual layers used to compute style representations in neural style transfer, we observed improved performance by selecting deeper layers in the network (see section 4.4.4). Consequently, for each dataset, we consistently extract features from the *conv5_1* layer, i.e., the first convolutional layer of block 5. We include

[2] https://github.com/Faruk-Ahmed/predictive_group_invariance

Table 4.1 Comparative results on Waterbirds, CelebA and COCO-on-Places-224 (COCO-on-P). Worst-group (*w-g*) and average (*avg*) test accuracies (% mean and std.) for Waterbirds and CelebA datasets; systematically-shifted (*shift*) and in-distribution (*ind*) test-set accuracies (% mean and std.) for COCO-on-Places dataset. Experiments with ResNet-50 models. Underlined and **bold** type indicate respectively best and per-block best performance (with significance $p < 0.05$ according to paired t-test on five runs).

Method	*Group labels* train	val	**Waterbirds** (w-g)	(avg)	**CelebA** (w-g)	(avg)	**COCO-on-P** (sys)	(ind)
ERM		✓	65.0$_{\pm 2.7}$	97.3$_{\pm 0.1}$	42.4$_{\pm 1.5}$	94.8$_{\pm 0.1}$	71.9$_{\pm 0.3}$	95.5$_{\pm 0.1}$
IRM Arjovsky et al. (2020)	✓	✓	77.4$_{\pm 0.3}$	**97.3**$_{\pm 0.1}$	75.1$_{\pm 0.6}$	**94.5**$_{\pm 0.1}$	78.8$_{\pm 0.3}$	**95.1**$_{\pm 0.2}$
Imp. Weighting	✓	✓	74.4$_{\pm 0.6}$	**97.4**$_{\pm 0.1}$	72.4$_{\pm 1.4}$	**94.4**$_{\pm 0.2}$	71.7$_{\pm 0.5}$	93.7$_{\pm 0.2}$
GroupDRO Sagawa* et al. (2020)	✓	✓	**83.9**$_{\pm 0.3}$	96.8$_{\pm 0.1}$	**85.7**$_{\pm 2.0}$	93.7$_{\pm 0.2}$	**79.0**$_{\pm 0.4}$	95.2$_{\pm 0.2}$
EIIL Creager et al. (2021)		✓	78.7$_{\pm 0.3}$	**96.9**$_{\pm 0.1}$	-	-	68.5$_{\pm 0.4}$	94.8$_{\pm 0.3}$
GEORGE Sohoni et al. (2020)		✓	76.2$_{\pm 2.0}$	95.7$_{\pm 0.5}$	53.7$_{\pm 1.3}$	**94.6**$_{\pm 0.2}$	71.6$_{\pm 0.3}$	**95.1**$_{\pm 0.1}$
JTT[3] Liu et al. (2021)		✓	82.9$_{\pm 0.3}$	96.4$_{\pm 0.2}$	56.0$_{\pm 0.7}$	93.6$_{\pm 0.0}$	69.2$_{\pm 0.4}$	94.7$_{\pm 0.3}$
GramClust-*orig* (Ours)		✓	**85.3**$_{\pm 1.1}$	96.6$_{\pm 0.1}$	**77.9**$_{\pm 2.2}$	94.2$_{\pm 0.2}$	**72.4**$_{\pm 0.4}$	**95.0**$_{\pm 0.2}$
GramClust-*cv* (Ours)			**85.3**$_{\pm 1.1}$	96.6$_{\pm 0.1}$	**80.3**$_{\pm 1.9}$	93.4$_{\pm 0.1}$	**73.2**$_{\pm 0.3}$	95.3$_{\pm 0.3}$

results with two types of model selection via cross-validation: (i) based on validation set with true-group annotations (GramClust-*orig*), and; (ii) based on pseudo-group labels (GramClust-*cv*) predicted by our clustering (see section 4.3.4).

Metrics. Following previous works, we report worst-group and average test accuracy (%) for Waterbirds and CelebA datasets. On COCO-on-Places-224, we follow the evaluation protocol proposed by Ahmed et al. (2021) and report predictive performances on the in-distribution test set, which follows the same distribution as the training set, and on the systematically-shifted test set, where the spurious correlations have been removed and COCO objects are composed with uniformly-sampled random backgrounds.

4.4.2 Comparative results

We observe that GramClust improves worst-group test accuracy over ERM baseline on Waterbirds and CelebA and systematic generalisation on COCO-on-Places224. More importantly, GramClust-*cv* achieves state-of-the-art performance on group robustness compared to all methods that do not use group labels on the training set. This results show empirically that our proposed approach, using Gram matrices of feature to discover pseudo-groups, which are then used for robust optimization and hyperparameter cross-validation, is very effective for group robustness. It also supports that Gram matrices are well suited to capture various types of dataset biases (background for Waterbirds, physical

Table 4.2 Study of the clustering features. Results in worst-group (Waterbirds, CelebA) and systematically-shifted (COCO-on-P) test-set accuracies (%). Gram matrix show to be the most effective type of clustering features to obtain improved group robustness.

Clustering features	Architecture	Layer	Waterbirds	CelebA	COCO-on-P
Standard	ResNet-50	*AvgPool*	$76.2_{\pm 2.0}$	$53.7_{\pm 1.3}$	$71.6_{\pm 0.3}$
MeanVar	VGG-19	*Conv5_1*	$\mathbf{85.3}_{\pm 1.2}$	$69.8_{\pm 1.0}$	$71.4_{\pm 0.5}$
Gram matrix	VGG-19	*Conv5_1*	$\mathbf{85.3}_{\pm 1.1}$	$\mathbf{77.9}_{\pm 2.2}$	$\mathbf{72.4}_{\pm 0.4}$

attribute in CelebA, multiple backgrounds in COCO-on-Places-224). For instance, on Waterbirds, GramClust-*cv* achieves 85.3% worst-group accuracy compared to the second-best method, JTT, which reaches 82.9%. The gap is even more pronounced on CelebA where our approach outperforms JTT by 24.3 pts. CelebA constitutes an interesting dataset to evaluate the scalability of methods as the training dataset is composed of 200k images. For instance, we were not able to scale EIIL on this dataset. Note that GramClust-*orig* uses the same hyperparameters as EIIL, GEORGE and JTT for robust training of the target classifier from predicted group labels, and still displays significant improvements on the three datasets in terms of worst-group accuracy. Liu et al. (2021) reported results that were obtained with early stopping thanks to a small validation set annotated with group labels. The authors selected models before convergence (around epoch 3) with low average accuracy on the test set but high worst-group accuracy. We argue that it is not a suitable property for a model and prefer models with high accuracy both in average and on the worst group of the test set. Surprisingly, GramClust-*cv* and GramClust-*orig* outperform GroupDRO on Waterbirds with 85.3% vs. 83.9%, while the latter method uses true-group labels during training. Our intuition is that it may be due to the ambiguity of the background in some Waterbirds images. We further discuss this result in section 4.4.5. Overall, these results show that our pseudo-groups on the validation set are relevant to select good hyperparameters and more importantly, that GramClust does not require any group labels during training to achieve group robustness.

4.4.3 Study of the clustering features

In this section, we compare the performances obtained when clustering images with different features. As noted in section 4.4.1, Huang and Belongie (2017) uses the channel-

[3]Results with JTT differ from the original paper as the scores that we report correspond to models trained without early-stopping.

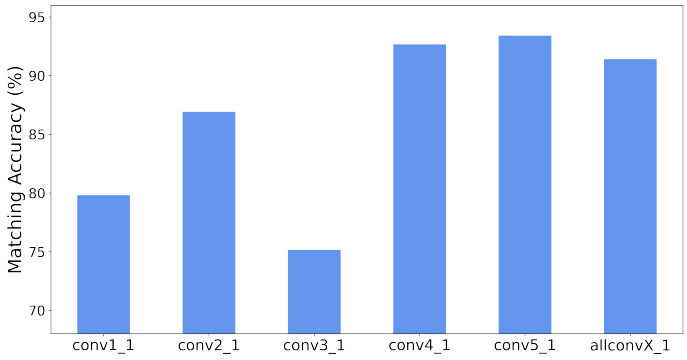


Fig. 4.4 Impact of the layer choice to extract style features. Results in matching accuracy on the validation set for GramClust on Waterbirds.

wise mean and variance of image features, instead of Gram matrices as in Gatys et al. (2016), to perform style transfer. We thus compare the use of such features (*'MeanVar'*) against our use of Gram matrices.

We also compared our use of VGG-19 *conv5_1* features with the direct use of the penultimate (*'AvgPool'*) representation of a ResNet-50 identification model. Indeed, our clustering features are extracted using a VGG-19, but our robust classifier is a ResNet-50. One may thus wonder if using directly the deepest features (*'Standard'*) before the classification head in a ResNet-50 could be better than using VGG-19 features. For fair comparison, we trained the robust classifier with the same hyperparameters for each method, which are consistent with those found by Sagawa* et al. (2020) with GroupDRO.

The results are available in Table 4.2 for Waterbirds, CelebA and COCO-on-Places-224. First, we observe that using the penultimate layer of a ResNet-50 as features for the clustering produces poorer performance than Gram matrices of VGG-19 features in every configuration. *MeanVar* reaches test worst-group accuracy on-par with *Gram matrix* on Waterbirds but degrades significantly performances on CelebA: 69.8% in average compared to 77.9% with *Gram matrix*. Gram matrices provide more information than *MeanVar* as their diagonals already contain the information about the channel-wise mean and variance of the deep features (see Eq. 4.5). Hence, these results show that, when scaling on large datasets such as CelebA, keeping all the correlations between different channels is important for group robustness.

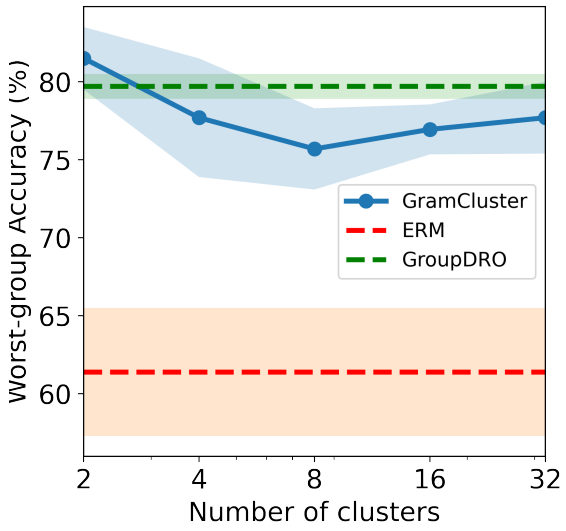


Fig. 4.5 Impact of the number of clusters. Results in worst-group val accuracies of GramClust on Waterbirds.

4.4.4 Clustering analysis

In this section, we study the behavior of our clustering algorithm with respect to the layers selected to extract features and to the number of clusters. This analysis is conducted on the Waterbirds dataset.

Effect of the selected layers for clustering features. We evaluate the impact of the selection of VGG-19 layers to extract the features in the clustering stage. To this end, we study the matching of the predicted environments to the true environment labels on the validation set. The assignment problem is solved via Hungarian matching Kuhn (1955) and we measure the global matching accuracy across all validation samples. In Figure 4.4, we compare results between using all five layers commonly used in neural style transfer (*conv1_1*,*conv2_1*,*conv3_1*,*conv4_1*,*conv5_1*) and each layer taken independently on Waterbirds dataset. Results show that: (i) Features from deeper layers correlate with better matching accuracy; (ii) Our approach is robust to the choice of deep layers either taken together (*allconvX_1*) or individually such as *conv4_1* and *conv5_1*; (iii) Using *conv5_1* outperforms selecting all traditional style layers and results in the highest score of 93.41%. We found consistent conclusions on the CelebA dataset. Results are reported in appendix B.5.

(a) True land background predicted as water background by GramClust

(b) True water background predicted as land background by GramClust

Fig. 4.6 Visualisation of confusing samples in Waterbirds dataset and wrongly predicted by GramClust. (a) Samples of confusing land-background images predicted as water background; (b) Samples of confusing land-background images predicted as water background. In each case, the actual image background is confusing due to the joint presence of elements reflecting land background (forest, heavy vegetation, sand) and water background (water surface, rainfalls, mist).

Impact of the number of clusters. . We now study the impact of the number of clusters as hyperparameter in the clustering algorithm. Worst-group accuracy on the validation set for $E' \in \{2, 4, 8, 16, 32\}$ clusters are reported in Figure 4.5 for Waterbirds datasets. Overall, our method is robust to a variation in the number of clusters: GramClust with higher numbers of clusters produces a slight drop in performance but still improves performance over ERM. It also has on-par performances with GroupDRO Sagawa* et al. (2020).

4.4.5 Discussion about improved results of GramClust over GroupDRO on Waterbirds

Comparative results in Table 4.1 actually show that GramClust-*orig* outperforms GroupDRO on the Waterbirds dataset. The difference between the approaches lies in the

usage of true-group labels on the training dataset for GroupDRO while GramClust-*orig* leverages its predicted pseudo-groups. Hence, this might be surprising given that the evaluation is performed on true test group labels and that both GramClust-*orig* and GroupDRO methods share the same robust optimization algorithm and hyperparameters. We intuit that this behavior, which occurs only on Waterbirds dataset, is related to the dataset group labels. In Figure 4.6, we show some examples of confusing images that were not correctly assigned with our predicted group labels with GramClust. These images were taken from the set of mismatches between true-group labels and our pseudo-group labels after the Hungarian matching. We can see that some of these samples present dominant characteristic elements from land background, such as heavy vegetation and sand, while being labeled as water background. Conversely, some samples labeled as land background display a high percentage of water surfaces in the image. As mentioned in section 4.4.1, the Waterbird dataset was created by combining bird photographs with background scenes taken from the Places365 dataset. But the latter dataset is actually composed of very diverse images which might not reflect the expected background for a category. This unwanted behavior raises the issue of creating benchmarks including datasets with spurious correlations from real-world data, such as hair color/gender detected in CelebA dataset. We leave this for future work. More importantly, it outlines the difficulty of annotating groups on web-crawled images. It motivates the need to push for future works towards automatically discovering groups in data, as proposed in our method.

4.5 Conclusion

In this chapter, we introduce GramClust, a two-stage method that first partitions a training dataset into clusters via k-means clustering based on Gram matrices computed from image features, which are extracted from a identification model trained to catch spurious correlations in a biased dataset. This first stage is then followed by learning a robust classifier which minimizes the error on the worst pseudo-group labels previously discovered. GramClust demonstrates to be an effective approach to tackle group robustness and outperforms every baseline on standard datasets with spurious correlations. The usage of Gram matrices of features is crucial to capture pertinent visual statistics of the image and enables a relevant partition for robust training. Our approach also alleviates the need to label a validation set of images with group information and is able to tune its hyperparameters in an unsupervised fashion by applying its clustering algorithm on the validation set.

Chapter 5

Conclusion

In this chapter, we summarize the contributions presented in the thesis. We then discuss a few research directions opened by this work.

5.1 Summary of contributions

This manuscript revolves around a central issue in DL: comparing neural networks. Instead of tackling this task by direct comparison of the networks' parameters, we study it from the functional perspective. All our work is guided by the idea that looking at the encoded function rather than its parameters enables to take a fresh look at existing open problems. Namely, we focus on the impact of **(i)** Normalization Layers (NLs) on Deep Neural Network (DNN) training and **(ii)** DNN robustness. Firstly, regarding **(i)**, we exploit the radial invariance that stems from adding NLs in DNN architectures to shed a new light on the NL's impact on training dynamics (chapter 3). Secondly, regarding **(ii)**, we develop a method that identifies relevant splits of the training set with similar data distribution on which the neural network's function performs unevenly. The pseudo labels yielded by the splits previously defined allow us to perform robust training in order to obtain more robust models (chapter 4).

(i) Impact of Normalization Layers on DNN training. To quantify the impact of NLs, we introduce an analytical tool dubbed the spherical framework. Built on the radial invariance that stems from adding NLs to DNN architectures, it enables the analysis and comparison of order-1 optimization schemes. This perspective shows that NLs transform gradient-based optimization schemes into algorithms with an adapted learning rate scheduling (*effective learning rate*) and a regularized direction (*effective learning rate*). We provide not only the first expression of the effective learning rate and

direction for Adam, but also the demonstration that, in the presence of NLs, standard (Stochastict Gradient Descent) SGD is equivalent to AdaGradG, a variant of Adam without momentum and constrained to the unit hypersphere. This result is unexpected because SGD, which is not adaptive by itself, is equivalent to a second-order moment adaptive method. The scheduling performed by the radius dynamics actually replicates the effect of dividing the learning rate by the second-order moment of the gradient norm. Overall, NLs allow us to relate a standard simple SGD with a variant of a more complex algorithm, and one of the most popular optimization schemes in DL: Adam.

New optimization schemes suited to the training of DNN with NLs. The spherical framework reveals intriguing geometrical behaviours occurring in Adam. In the context of manifold optimization, the optimization direction should only depend on the trajectory on that manifold. With Adam, the effective direction not only depends on the trajectory on the hypersphere but also on the deformed gradients and additional radial terms. By using the tools of optimization on manifold, we introduce natural variants of Adam that better respect the geometry of the hypersphere. Thereby, we are able to **(a)** identify geometrical phenomena that play empirically a significant role in CNNs training with BN, and **(b)** define new optimization schemes that are better suited to the training of CNNs with BN layers.

(ii) Group-robustness without environment labels. We introduce an easy-to-scale method to split training images among distinct groups of images with similar data distribution. It is based on Gram-matrix features extracted by an exogenous identification model. These splits, called pseudo groups, are subsequently leveraged to perform robust training of a neural network. Our method, GramStyle, is evaluated on the standard benchmarks of group robustness. It is composed of datasets that include a majority group of data displaying correlations between a visual element in images and the target classes and a minority group that does not exhibit such correlations. In addition to the class label for each image, there is a group label: the information whether an image belongs to the majority or minority group. Unlike recent approaches, our method alleviates the need for these ground-truth group labels altogether, even in the validation set as hyperparameters are set based on the validation performance computed from our pseudo-groups. Overall, GramStyle outperforms by a significant margin all recent baselines addressing group robustness without group annotation.

5.2 Future work

Let us now discuss interesting directions that could be addressed in future work.

Stochasticity in Optimization. The study of both stochasticity and functional variability in the training process is of interest for various reasons. Stochasticity seems to play a role in the generalization capability of DNNs. As mentioned in section 2.2.1, not all minima of the training loss have the same generalization capabilities. Various recent works (Barrett and Dherin, 2020; Smith et al., 2021) suggest that SGD preferably selects minima with good generalization due to its intrinsic stochasticity. Note also that Picard (2021) shows that the stochasticity's influence on DNN performance (which is an indirect measure of functional variability) is underestimated. Improving our understanding of its impact on the training process is therefore of importance to explain generalization in DL and design better benchmarks in order to assess properly the performance of the different models. Concretely, the sources of stochasticity in the training of DNNs are plural. They are: the random initialization of DNNs parameters; the noise in the empirical loss estimation using random batches; the use of stochastic data augmentation to improve NNs performances; the noise in practical implementation of certain operations on Graphical Computer Units (GPUs). The different contributions of these stochastic components result in a very noisy training process. The leading interrogation is to precisely analyze how these sources of stochasticity impact the obtained models. Due to the existence of symmetries in the parameter space of the network, changes in the parameter space do not necessary imply a change of function encoded by the corresponding parameters. Instead of comparing the obtained parameters, one could adopt the functional perspective and focus on analyzing to what extent the obtained functions are similar or different. In other words, the purpose would be to quantify how the training process of DNNs is functionally stable. Such an approach would address the following interrogations: **(1)** To what extent is the training process stable w.r.t. functional variability? **(2)** What is the influence of the different sources of stochasticity on training stability? **(3)** Is it possible to find processes that mitigate the functional instability? Functional variability during training informs us indirectly over trajectories followed during the optimization process. We think here of regions of stability or instability, of the impact of regularization techniques over the trajectories or of differences between training algorithms. This approach could also shed new lights on intriguing techniques such as knowledge distillation (Hinton et al., 2015) or the Lottery Ticket Hypothesis (Frankle and Carbin, 2018; Frankle et al., 2019), that manages to train highly-sparse and iteratively-pruned architectures by rewinding the parameters to their initial values for simple architectures, or at an iteration close

to the beginning of the original training for more complex DNNs. Finally, note that function variability is also presumably playing an important role in ensembles of DNNs. Ensembling consists in averaging the predictions of several DNNs obtained from different training processes. It results in significant improvements in terms of performances and provides state-of-the-art performance in out-of-distribution detection (Lakshminarayanan et al., 2017).

ERM and path of least efforts. Our empirical study in chapter 4 shows that architectures trained with ERM using the cross entropy loss lead to poor worst group accuracy. Arjovsky et al. (2020) argue that this phenomenon is explained by the fact that simple correlations found in data are used as predictive rules by the model. Overall, this phenomenon is often backed up by the blurry intuition that ERM naturally selects the path of least efforts by converging to the 'simplest' solutions. Currently, there exists no mathematical characterization of a so called 'simple' solution. Let us reflect on the Colored MNIST dataset (Arjovsky et al., 2020) as a starting point. In this dataset, a specific color in the background is correlated to a given digit. One can easily be convinced that using the background is simpler than learning invariant features for each digit. Somehow, the 'simple' criterion seems to be related to the function with the fewest invariances w.r.t. the input data. Formally, given parameters in $\mathbb{R}^d$, finding a mathematical method to approximate the size of the set of invariances w.r.t. the input data is very challenging. However, trying to find properties that characterize models with few invariances could be a way to circumvent the previous challenge. In the domain of network similarity, previous works (Charpiat et al., 2019; Kornblith et al., 2019) have developed metrics to analyze a neural network's internal representation. In addition, neural networks trained in Table 4.1 provide a variety of functions that perform unevenly in terms of worst group accuracy. Analyzing and comparing the internal representation of neural networks with different generalization properties could be used as a starting point for such a study. If a satisfying quantitative criterion that discriminates networks w.r.t. their robustness is found, one could also imagine to regularize the training process of neural networks by penalizing the optimization objective to get *in fine* more robust models.

Bibliography

Achlioptas, D. (2003). Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, pages 671—-687.

Agarwal, N., Bullins, B., and Hazan, E. (2017). Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research*, 18(1):4148–4187.

Ahmed, F., Bengio, Y., van Seijen, H., and Courville, A. (2021). Systematic generalisation with group invariant predictions. In *International Conference on Learning Representations (ICLR)*.

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. (2020). Invariant risk minimization.

Arora, S., Li, Z., and Lyu, K. (2019). Theoretical analysis of auto rate-tuning by batch normalization. In *International Conference on Learning Representations (ICLR)*.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Barrett, D. G. and Dherin, B. (2020). Implicit gradient regularization. *arXiv preprint arXiv:2009.11162*.

Bartlett, P. L. and Mendelson, S. (2002). Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482.

Beery, S., Horn, G. V., and Perona, P. (2018). Recognition in terra incognita. In *European Conference on Computer Vision (ECCV)*.

Bjorck, N., Gomes, C. P., Selman, B., and Weinberger, K. Q. (2018). Understanding batch normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Blum, A. and Rivest, R. L. (1989). Training a 3-node neural network is np-complete. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Blum, A. L. and Rivest, R. L. (1992). Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117–127.

Bousquet, O. and Elisseeff, A. (2002). Stability and generalization. *The Journal of Machine Learning Research*, 2:499–526.

Cai, Y., Li, Q., and Shen, Z. (2019). A quantitative analysis of the effect of batch normalization on gradient descent. In *36th International Conference on Machine Learning (ICML)*.

Charpiat, G., Girard, N., Felardos, L., and Tarabalka, Y. (2019). Input similarity from the neural network perspective. *Advances in Neural Information Processing Systems*, 32.

Cho, M. and Lee, J. (2017). Riemannian approach to batch normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Creager, E., Jacobsen, J.-H., and Zemel, R. (2021). Environment inference for invariant learning. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*.

Daneshmand, H., Kohler, J., Bach, F., Hofmann, T., and Lucchi, A. (2020). Batch normalization provably avoids rank collapse for randomly initialised deep networks. In *NeurIPS*.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*, 27.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

Duchi, J. C., Hashimoto, T., and Namkoong, H. (2019). Distributionally robust losses against mixture covariate shifts. *Under review*, 2.

Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. (2019). Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ge, R., Huang, F., Jin, C., and Yuan, Y. (2015). Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on learning theory*, pages 797–842. PMLR.

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. (2019). Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations (ICLR)*.

Ghorbani, B., Krishnan, S., and Xiao, Y. (2019). An investigation into neural net optimization via hessian eigenvalue density. In *36th International Conference on Machine Learning (ICML)*.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Hashimoto, T., Srivastava, M., Namkoong, H., and Liang, P. (2018). Fairness without demographics in repeated loss minimization. In *International Conference on Machine Learning*, pages 1929–1938. PMLR.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Deep residual learning for image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hendrycks, D., Basart, S., Mu, N., Kadavath, S., Wang, F., Dorundo, E., Desai, R., Zhu, T., Parajuli, S., Guo, M., Song, D., Steinhardt, J., and Gilmer, J. (2021). The many faces of robustness: A critical analysis of out-of-distribution generalization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Hinton, G., Vinyals, O., Dean, J., et al. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7).

Hoffer, E., Banner, R., Golan, I., and Soudry, D. (2018a). Norm matters: efficient and accurate normalization schemes in deep networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Hoffer, E., Hubara, I., and Soudry, D. (2018b). Fix your classifier: the marginal value of training the last weight layer. In *International Conference on Learning Representations (ICLR)*.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Huang, X. and Belongie, S. (2017). Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.

Karakida, R., Akaho, S., and Amari, S.-i. (2019). The normalization method for alleviating pathological sharpness in wide neural networks. In *NeurIPS*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koh, P. W., Sagawa, S., Marklund, H., Xie, S. M., Zhang, M., Balsubramani, A., Hu, W., Yasunaga, M., Phillips, R. L., Gao, I., et al. (2021). Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, pages 5637–5664. PMLR.

Kohler, J., Daneshmand, H., Lucchi, A., Hofmann, T., Zhou, M., and Neymeyr, K. (2019). Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. In *AISTATS*.

Kolev, K. (2011). Convexity in image-based 3d surface reconstruction.

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. (2019). Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

Kuhn, H. W. (1955). The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2:83–97.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

Lee, J. M. (2006). *Riemannian manifolds: an introduction to curvature*, volume 176. Springer Science & Business Media.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31.

Li, Y., Wang, N., Liu, J., and Hou, X. (2017). Demystifying neural style transfer. In *International Joint Conference on Artificial Intelligence*, page 2230–2236.

Li, Z. and Arora, S. (2020). An exponential learning rate schedule for deep learning. In *International Conference on Learning Representations (ICLR)*.

Lian, X. and Liu, J. (2019). Revisit batch normalization: New understanding and refinement via composition optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2014). Microsoft coco: Common objects in context.

Liu, E. Z., Haghgoo, B., Chen, A. S., Raghunathan, A., Koh, P. W., Sagawa, S., Liang, P., and Finn, C. (2021). Just train twice: Improving group robustness without training group information. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*.

Liu, W., Liu, Z., Yu, Z., Dai, B., Lin, R., Wang, Y., Rehg, J. M., and Song, L. (2018). Decoupled networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, W., Zhang, Y.-M., Li, X., Yu, Z., Dai, B., Zhao, T., and Song, L. (2017). Deep hyperspherical learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Loshchilov, I. and Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

Mallat, S. (2016). Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150203.

Mallat, S. (2020). Sciences des données. *L'annuaire du Collège de France. Cours et travaux*, (118):31–42.

Marcotte, P. and Savard, G. (1992). Novel approaches to the discrimination problem. *Zeitschrift für Operations Research*, 36(6):517–545.

Martens, J. et al. (2010). Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742.

Martens, J. and Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR.

Matsuura, T. and Harada, T. (2020). Domain generalization using a mixture of multiple latent domains. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*.

Minsky, M. and Papert, S. (1969). An introduction to computational geometry. *Cambridge tiass., HIT*, 479:480.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Picard, D. (2021). Torch. manual_seed (3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision. *arXiv preprint arXiv:2109.08203*.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17.

Sagawa*, S., Koh*, P. W., Hashimoto, T. B., and Liang, P. (2020). Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. In *International Conference on Learning Representations (ICLR)*.

Sagawa, S., Raghunathan, A., Koh, P. W., and Liang, P. (2020). An investigation of why overparameterization exacerbates spurious correlations. In *International Conference on Machine Learning*, pages 8346–8356. PMLR.

Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). How does batch normalization help optimization? In *Advances in Neural Information Processing Systems (NeurIPS)*.

Sastry, C. S. and Oore, S. (2020). Detecting out-of-distribution examples with Gram matrices. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*.

Smith, S. L., Dherin, B., Barrett, D. G., and De, S. (2021). On the origin of implicit regularization in stochastic gradient descent. *arXiv preprint arXiv:2101.12176*.

Sohoni, N. S., Dunnmon, J. A., Angus, G., Gu, A., and Ré, C. (2020). No subclass left behind: Fine-grained robustness in coarse-grained classification problems. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*.

Soudry, D., Hoffer, E., Nacson, M. S., Gunasekar, S., and Srebro, N. (2018). The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research (JMLR)*.

Sun, X., Peng, J., Shen, Y., and Kang, H. (2020). Tobacco plant detection in rgb aerial images. *Agriculture*, 10(3):57.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *30th International Conference on Machine Learning (ICML)*, Atlanta, Georgia, USA.

Tatman, R. (2017). Gender and dialect bias in youtube's automatic captions. In *Proceedings of the first ACL workshop on ethics in natural language processing*, pages 53–59.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*.

van Laarhoven, T. (2017). L2 regularization versus batch and weight normalization. arXiv preprint arXiv:1706.05350.

Vapnik, V. (1991). Principles of risk minimization for learning theory. In Moody, J., Hanson, S., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann.

Vapnik, V. N. (1998). Adaptive and learning systems for signal processing communications, and control. *Statistical learning theory*.

Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., and Perona, P. (2010). Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology.

Woodward, J. (2005). *Making things happen: A theory of causal explanation*. Oxford university press.

Wu, Y. and He, K. (2018). Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19.

Yarotsky, D. (2022). Universal approximations of invariant maps by neural networks. *Constructive Approximation*, 55(1):407–474.

Zhang, G., Wang, C., Xu, B., and Grosse, R. (2019). Three mechanisms of weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

Zhang, J., Menon, A. K., Veit, A., Bhojanapalli, S., Kumar, S., and Sra, S. (2021). Coping with label shift via distributionally robust optimisation. In *International Conference on Learning Representations*.

Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., and Torralba, A. (2018). Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(6):1452–1464.

Appendix A

Omitted proofs

A.1 Proof of Lemma 1

Lemma 1 (Gradient of a function with radial invariance) If $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ is radially invariant and almost everywhere differentiable, then, for all $\rho > 0$ and all $\mathbf{x} \in \mathbb{R}^d$ where $\mathcal{L}$ is differentiable, we have:

$$\langle \nabla \mathcal{L}(\mathbf{x}), \mathbf{x} \rangle = 0 \quad \text{and} \quad \nabla \mathcal{L}(\mathbf{x}) = \rho \, \nabla \mathcal{L}(\rho \mathbf{x}).$$

Proof. Let us consider $\mathbf{x} \in \mathbb{R}^d \setminus \{0_{\mathbb{R}^d}\}$ a parameter on which $\mathcal{L}$ is differentiable. By using the definition of the radial invariance and the composition of differentials we obtain:

$$\nabla \mathcal{L}(\mathbf{x}) = \nabla \mathcal{L}(\rho \mathbf{x}), \tag{A.1}$$
$$\nabla \mathcal{L}(\mathbf{x}) = \rho \nabla \mathcal{L}(\rho \mathbf{x}). \tag{A.2}$$

Then:

$$\langle \nabla \mathcal{L}(\mathbf{x}), \mathbf{x} \rangle = \langle \nabla \mathcal{L}(\rho \mathbf{x}), \mathbf{x} \rangle, \tag{A.3}$$
$$\Leftrightarrow \langle \nabla \mathcal{L}(\rho \mathbf{x}), \mathbf{x} \rangle = \rho \langle \nabla \mathcal{L}(\rho \mathbf{x}), \mathbf{x} \rangle, \tag{A.4}$$
$$\Leftrightarrow \langle \nabla \mathcal{L}(\rho \mathbf{x}), \mathbf{x} \rangle - \rho \langle \nabla \mathcal{L}(\rho \mathbf{x}), \mathbf{x} \rangle, \tag{A.5}$$
$$\Leftrightarrow (1 - \rho) \langle \nabla \mathcal{L}(\mathbf{x}), \mathbf{x} \rangle = 0. \tag{A.6}$$

Since $1 - \rho \neq 0$, we have: $\langle \nabla \mathcal{L}(\mathbf{x}), \mathbf{x} \rangle = 0$ □

Appendix B

Additional experiments

B.1 Theorem 2 assumptions validity

Sign of $1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle$. We tracked the maximum of the quantity $A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle$ for all the filters of a ResNet20 CIFAR trained on CIFAR10 and optimized with SGD-M or Adam with learning rate parameter of 0.1 (SGD) and 0.01 (Adam) as well as L_2 regularization of 10^{-4}, momentum parameter of 0.9 for SGD and 0.9 for both the order-1 and order-2 moment with Adam. As can be seen on Figure B.1, this quantity is always small compared to 1, making $1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle$ always positive in practice. The order of magnitude of this quantity is roughly the same for different architectures and datasets.

Taylor expansion. We tracked the maximum of the quantity $\eta_k^e \|\mathbf{c}_k^\perp\|$ for all the filters of a ResNet20 CIFAR trained on CIFAR10 and optimized with SGD-M or Adam. The observed values justify the Taylor expansion and validate the assumption $|\alpha|\|\mathbf{c}_k^\perp\| \approx \eta_k^e \|\mathbf{c}_k^\perp\| < \pi$. (cf. Figure B.2). The order of magnitude of this quantity is roughly the same for other different architectures and datasets.

B.2 Theorem 4 assumptions validity

Validity of the Taylor expansion. For a CNN trained with SGD optimization, we tracked the quantity $(\eta_k \|\nabla \mathcal{L}(\mathbf{u}_k)\|)^2 / r_k^2$, which is the variable of the Taylor expansion. As can be seen in Figure B.3, the typical order of magnitude is 10^{-2}, justifying the Taylor expansion.

A quick formal analysis also suggests the validity of this hypothesis. Thanks to the expression of $\eta_k = (1 - \eta\lambda)^{-2i-k}\eta$ shown in the previous section, if we replace $\|\nabla \mathcal{L}(\mathbf{u}_k)\|$

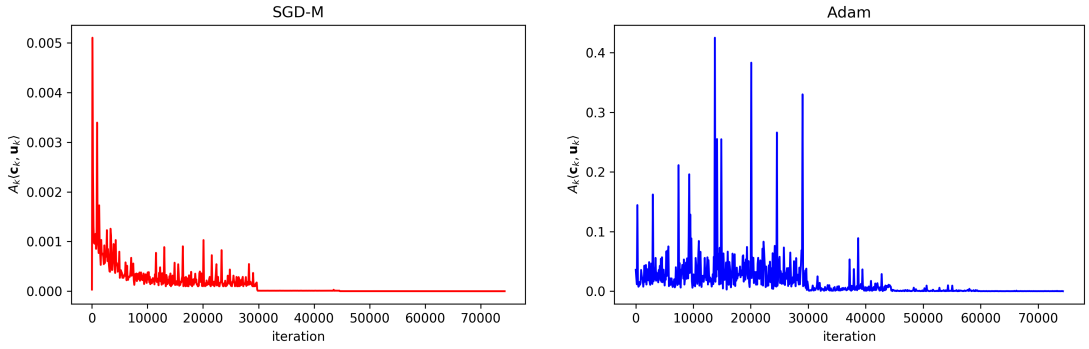


Fig. B.1 Tracking of $A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle$ for SGD-M and Adam. The above graphs show the maximum of the absolute value of $A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle$ for all filters in all layers of a ResNet20 CIFAR trained on CIFAR10 and optimized with SGD-M (left) or Adam (right). The quantity is always small compared to 1. Therefore we may assume that $1 - A_k\langle \mathbf{c}_k, \mathbf{u}_k\rangle \geq 0$.

by a constant for asymptotic analysis, the comparison becomes:

$$(1-\eta\lambda)^{-4k-2} \ll (1-\eta\lambda)^{-2}\frac{1-(1-\eta\lambda)^{-4k}}{1-(1-\eta\lambda)^{-4}} \tag{B.1}$$

$$1 \ll \frac{1-(1-\eta\lambda)^{4k}}{(1-\eta\lambda)^{-4}-1}. \tag{B.2}$$

It is asymptotically true.

B.3 Best hyperparameters for optimizers in table 3.3 and table 3.2.

This section provides the best hyperparameters found for each optimizer of the benchmark. Learning rates and momentum factors can be found in table B.1 while L_2 regularization and order-2 moment parameters are in table B.2.

B.4 Extended results from section 3.5.3 to other datasets and architectures

In this section, we observe the mean loss training curves associated to Adam, AdamW, AdamG, Adam w/o (a), Adam w/o (ab), Adam w/o (abc) on datasets CIFAR10, CIFAR100 and SVHN with architecture ResNet20, ResNet18 or VGG16 with BN,

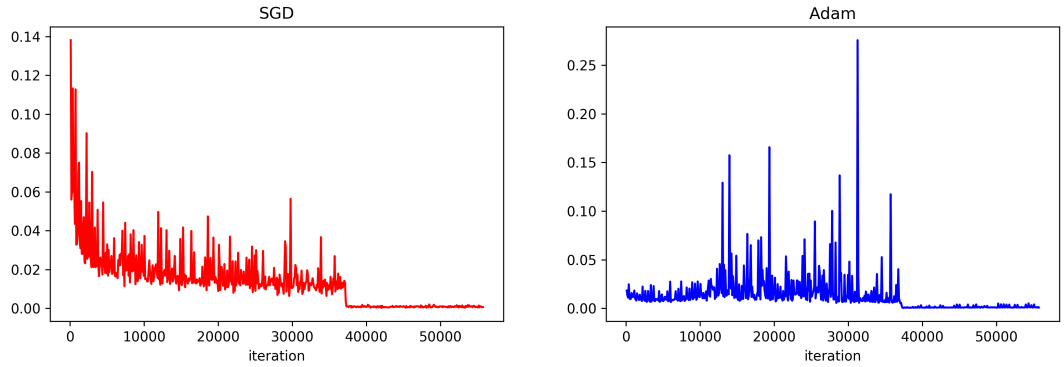


Fig. B.2 Tracking of $\eta_k^e \|\mathbf{c}_k^\perp\|$ for SGD-M and Adam. The above graphs show the maximum of the absolute value of $\eta_k^e \|\mathbf{c}_k^\perp\|$ for all filters in all layers of a ResNet20 CIFAR trained on CIFAR10 and optimized with SGD-M (left) or Adam (right).

Table B.1 Best learning rate and momentum factor. We systematically found the same learning rate for each dataset and architecture while the momentum factor was fixed to 0.9.

Method	η_0	β, β_1
Adam	0.001	0.9
AdamW	0.001	0.9
AdamG	0.01	0.9
Adam w/o (a)	0.001	0.9
Adam w/o (ab)	0.001	0.9
Adam w/o (abc)	0.001	0.9

corresponding to the accuracies given in Table 3.3.These curves are illustrated in Figures B.4 B.5-B.6 B.6 B.7 B.8 B.9 B.10 B.12 B.13 B.14 B.15 B.16. The case of ResNet20 is illustrated in Figure 3.6 of the manuscript.

B.5 Clustering analysis on CelebA

We present, in Figure B.17, the matching accuracy between the ground-truth environments and the environments discovered with our method on the validation set of CelebA for different layers of the VGG-19. As on Waterbirds, we notice that the best result is obtained when using the layer *conv5_1*.

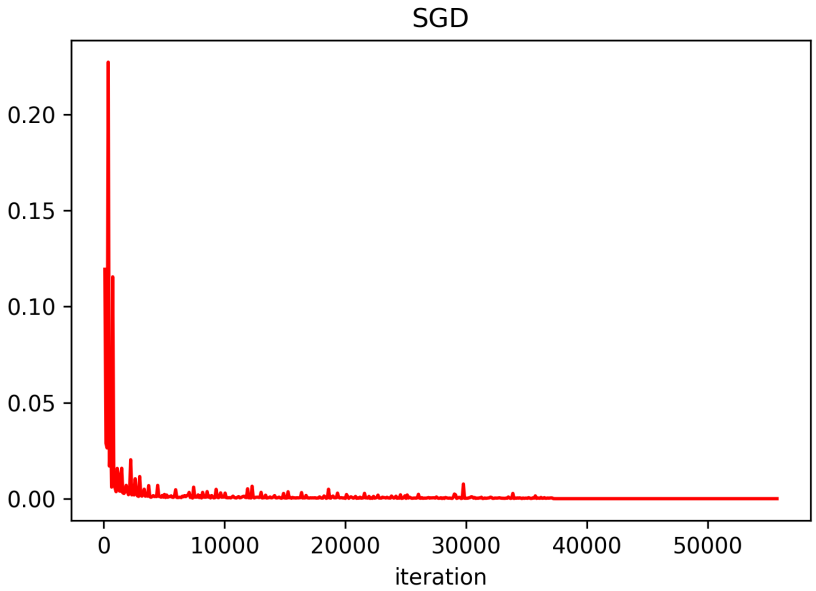


Fig. B.3 Validity of Taylor expansion. We tracked the maximum value of $(\eta_k \|\nabla \mathcal{L}(\mathbf{u}_k)\|)^2 / r_k^2$ for all filters in all layers of a ResNet20 CIFAR trained on CIFAR10 with SGD. The order of magnitude of the gradient is roughly the same for other architectures or datasets. It empirically validates the approximation by the Taylor expansion.

Table B.2 Best L_2 regularization (λ) and order-2 moment factors (β_2).

Setup			Adam	AdamW	AdamG	Adam w/o (a)	Adam w/o (ab)	Adam w/o (abc)
CIFAR10	ResNet20	$\lambda \times 10^4$	2.5	5	1.25	0.31	1.25	5
		β_2	0.99	0.99	0.99	0.99	0.99	0.99
	ResNet18	$\lambda \times 10^4$	2.5	0.08	0.63	2.5	1.25	0.16
		β_2	0.999	0.99	0.99	0.99	0.99	0.99
	VGG16	$\lambda \times 10^4$	2.5	0.31	2.5	0.63	0.00	0.31
		β_2	0.999	0.999	0.999	0.999	0.99	0.999
CIFAR100	ResNet18	$\lambda \times 10^4$	1.25	1.25	1.25	1.25	1.25	0.00
		β_2	0.999	0.99	0.99	0.99	0.99	0.999
	VGG16	$\lambda \times 10^4$	0.63	0.16	0.63	0.63	1.25	0.08
		β_2	0.99	0.99	0.99	0.99	0.99	0.99
SVHN	ResNet18	$\lambda \times 10^4$	0.00	0.08	5	0.31	5	0.08
		β_2	0.999	0.999	0.99	0.99	0.999	0.999
	VGG16	$\lambda \times 10^4$	0.00	0.31	5	0.08	2.5	2.5
		β_2	0.99	0.99	0.99	0.99	0.99	0.999

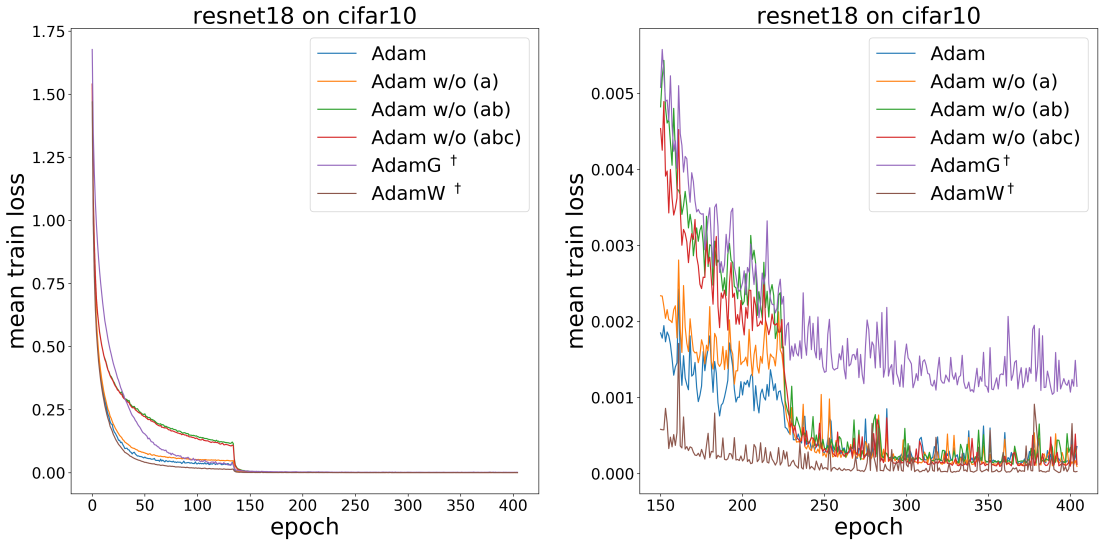


Fig. B.4 Training speed comparison with ResNet18 BN on CIFAR10. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

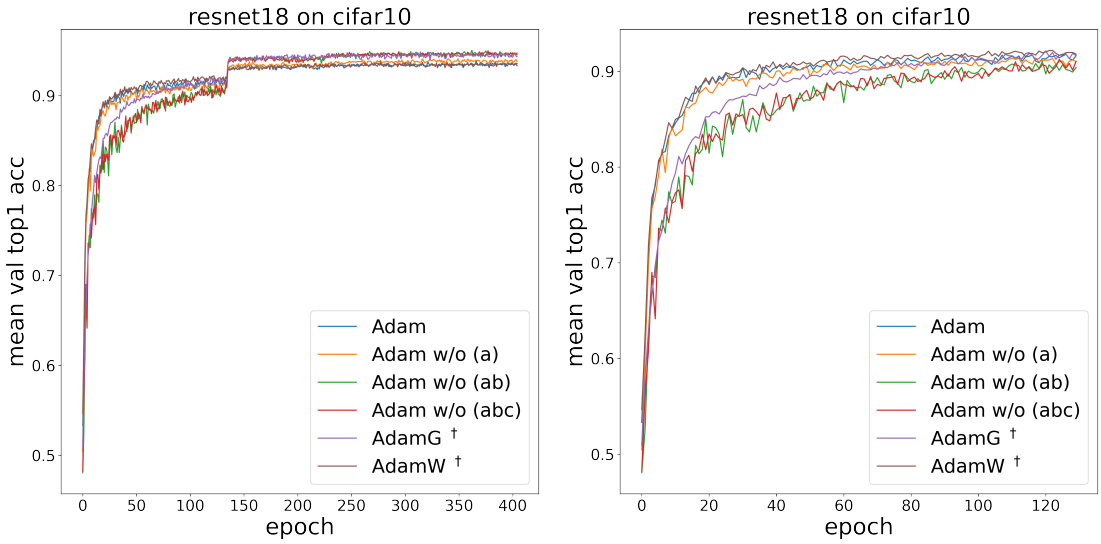


Fig. B.5 Accuracy comparison on the validation set with ResNet18 BN on CIFAR10. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the first epochs. Please refer to Table 3.3 for the corresponding accuracies.

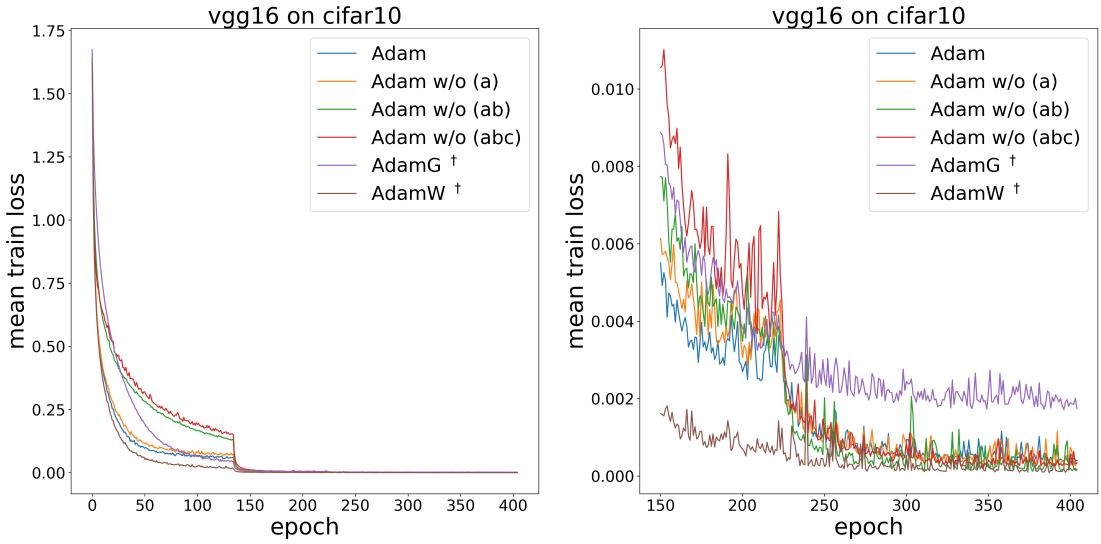


Fig. B.6 Training speed comparison with VGG16 on CIFAR10. *Left:* Mean accuracy on the validation set over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

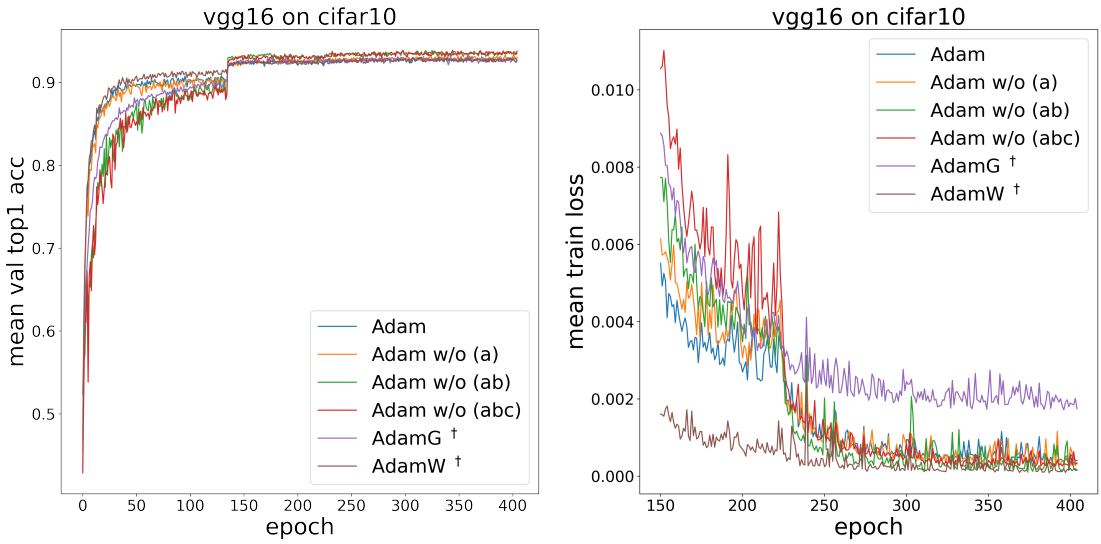


Fig. B.7 Accuracy comparison on the validation set with VGG16 BN on CIFAR10. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

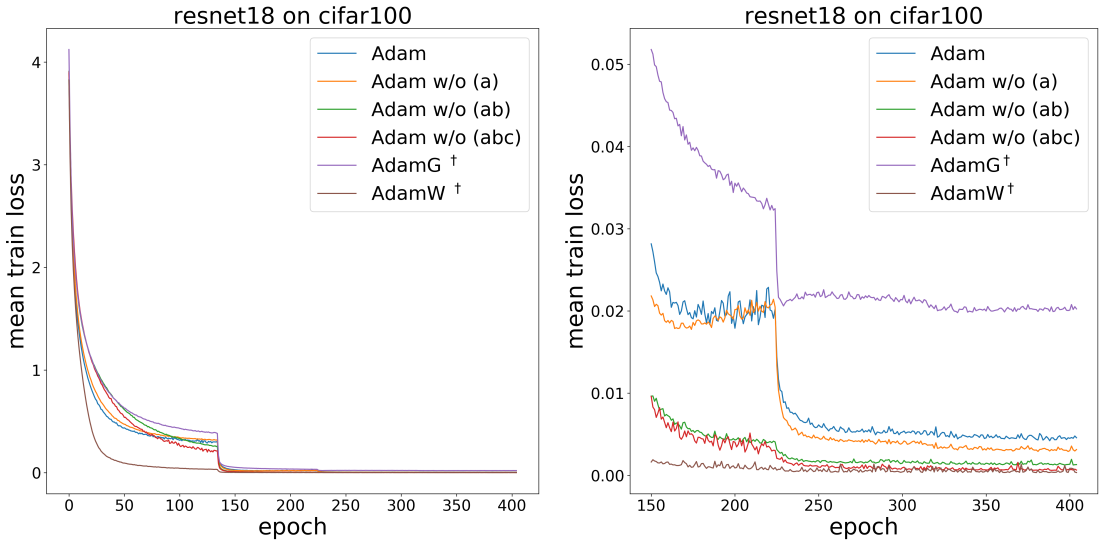


Fig. B.8 Training speed comparison with ResNet18 on CIFAR100. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

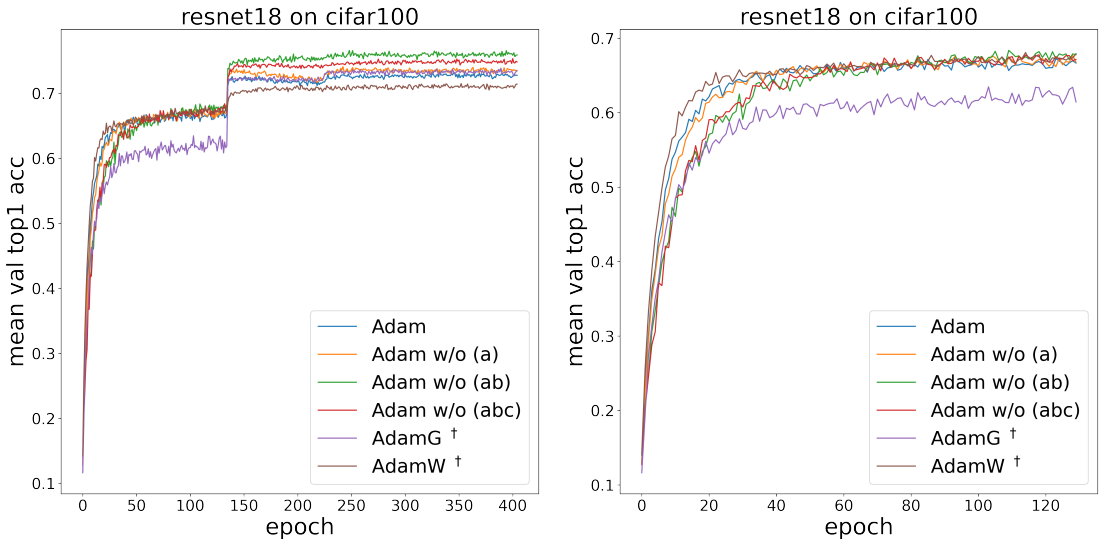


Fig. B.9 Accuracy comparison on the validation set with ResNet18 BN on CIFAR100. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

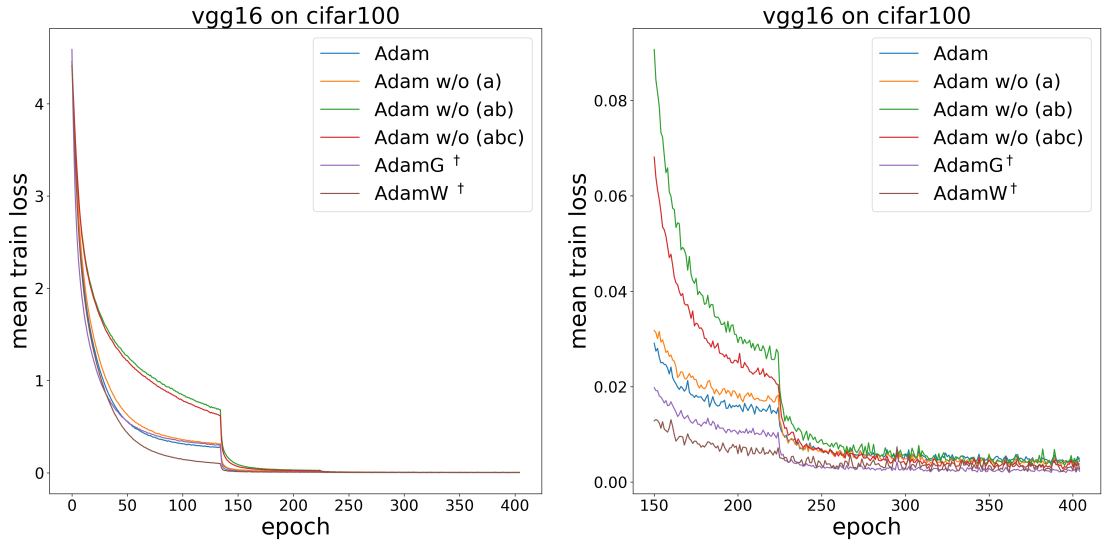


Fig. B.10 Training speed comparison with VGG16 on CIFAR100. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

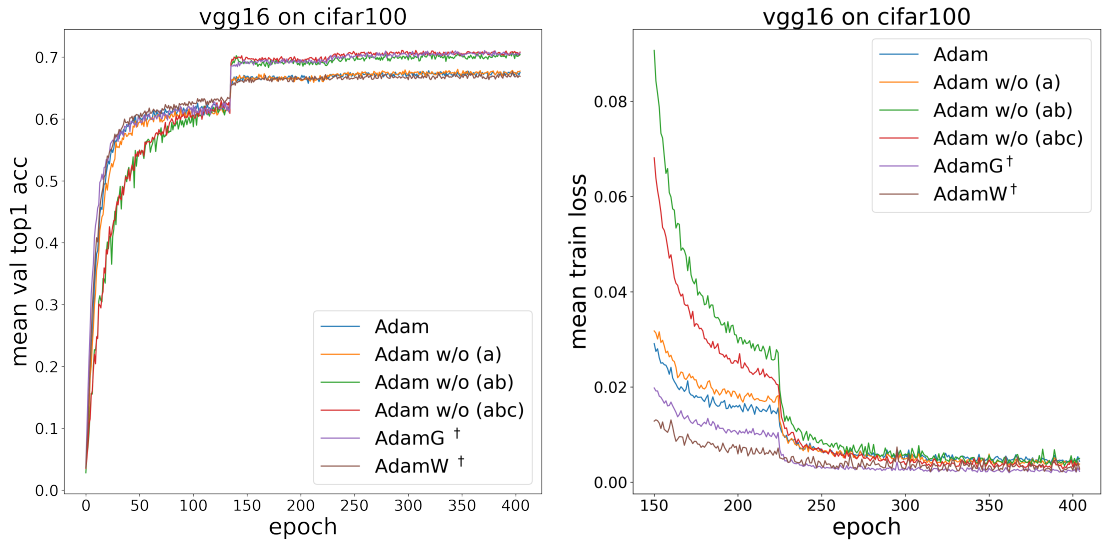


Fig. B.11 Accuracy comparison on the validation set with VGG16 BN on CIFAR100. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

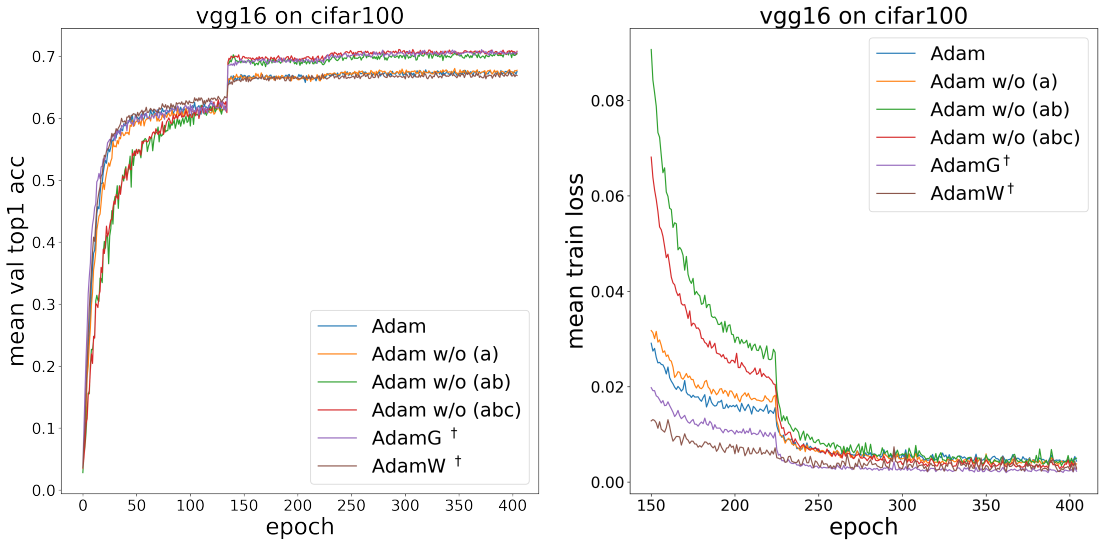


Fig. B.12 Accuracy comparison on the validation set with VGG16 BN on CIFAR100. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

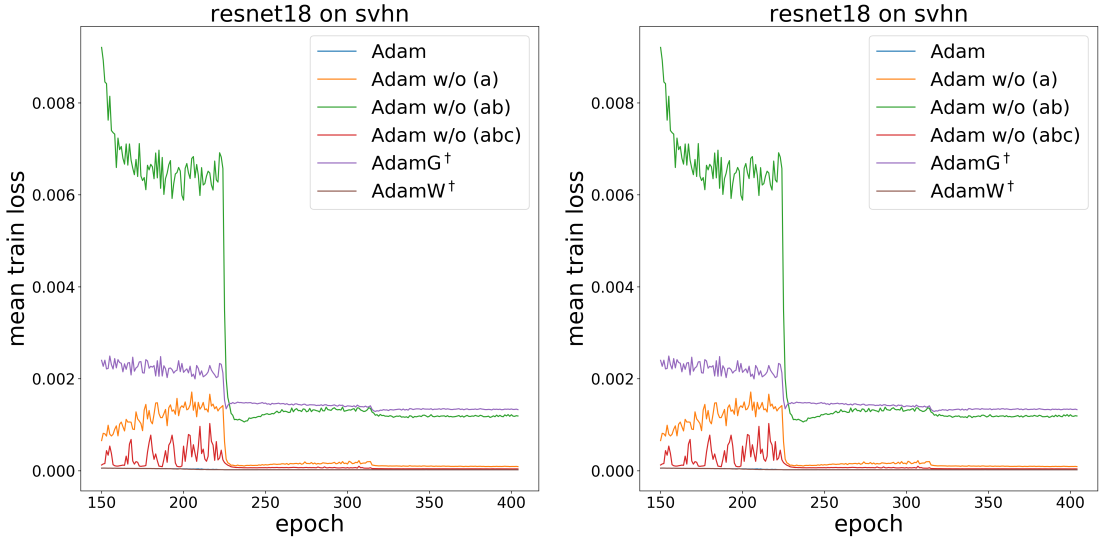


Fig. B.13 Training speed comparison with ResNet18 on SVHN. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

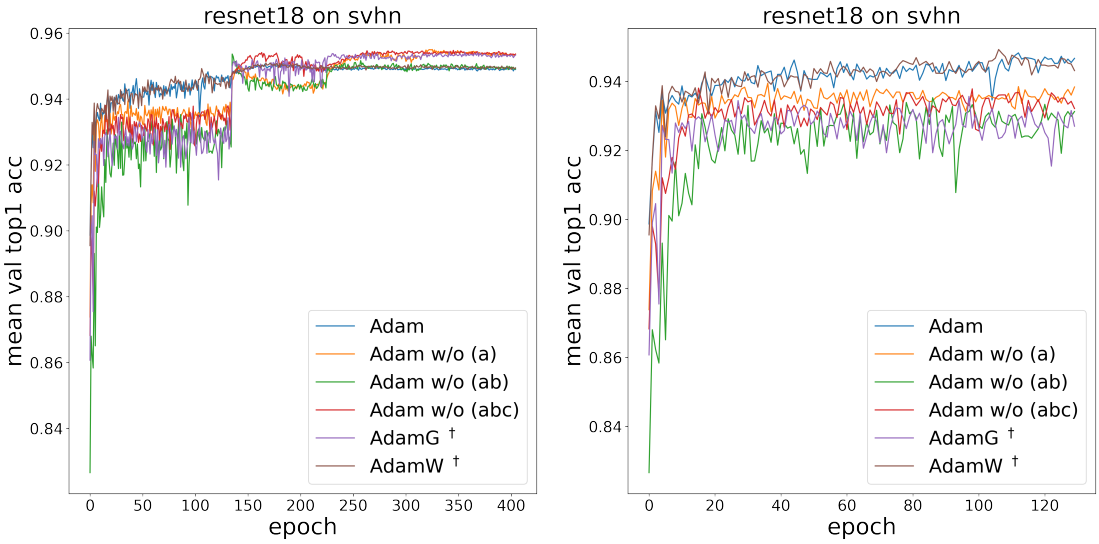


Fig. B.14 Accuracy comparison on the validation set with ResNet18 BN on SVHN. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

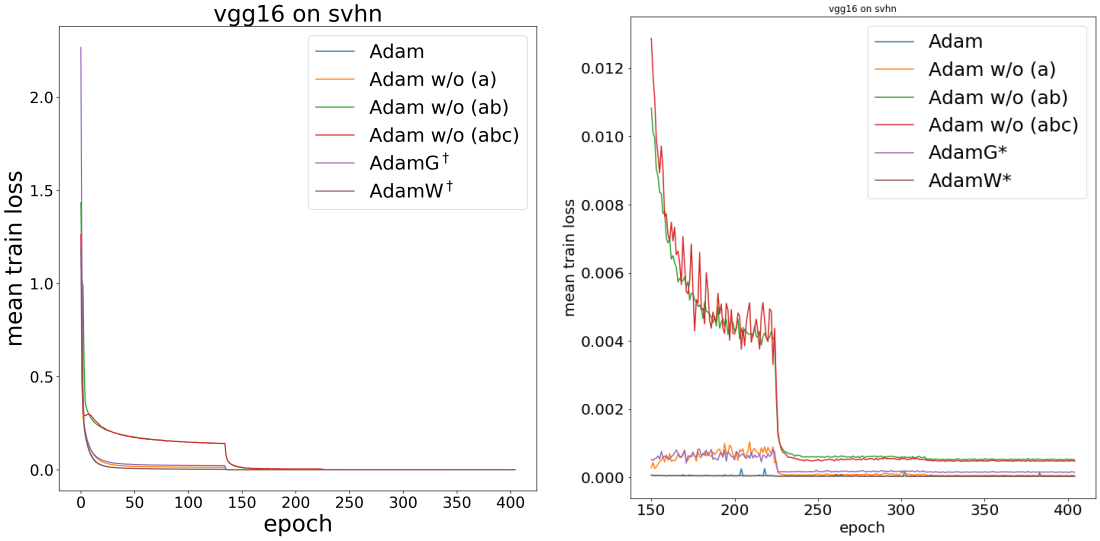


Fig. B.15 Training speed comparison with VGG16 on SVHN. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

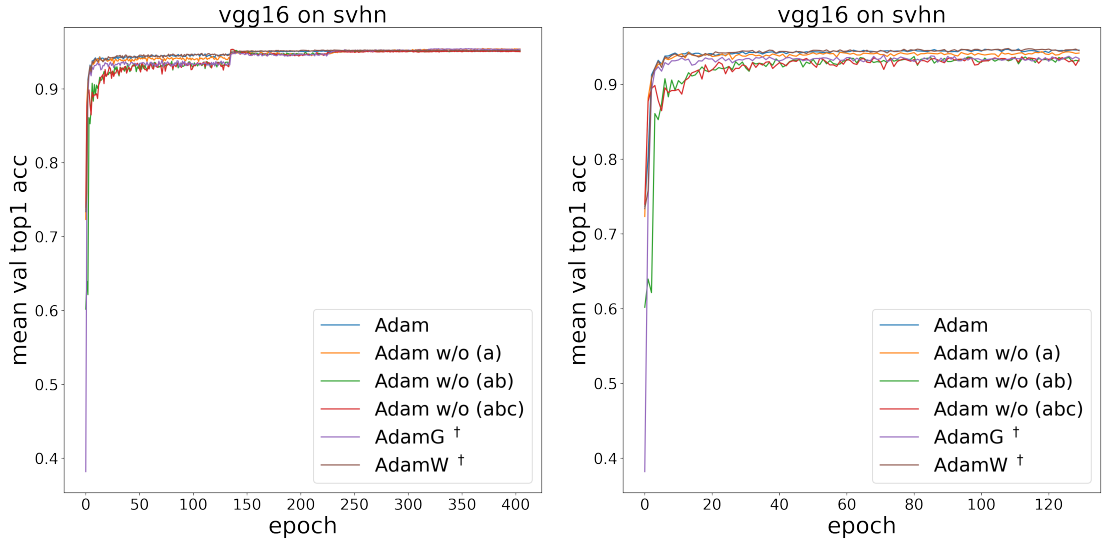


Fig. B.16 Accuracy comparison on the validation set with VGG16 BN on SVHN. *Left:* Mean training loss over all training epochs (averaged across 5 seeds) for different Adam variants. *Right:* Zoom-in on the last epochs. Please refer to Table 3.3 for the corresponding accuracies.

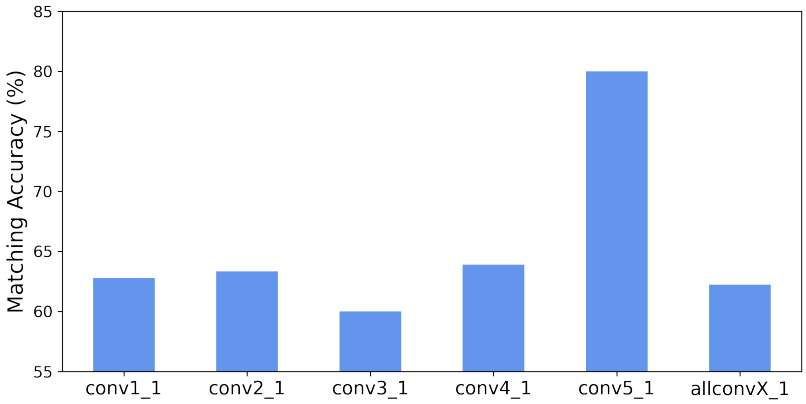


Fig. B.17 Impact of the layer choice to extract style features on CelebA. We show the matching accuracy between the ground-truth environments on the validation set CelebA and the discovered ones with GramStyle when using different VGG-19 layers. The result denoted *allconvX_1* is obtained when using all the layers *conv1_1*, *conv2_1*, *conv3_1*, *conv4_1*, *conv5_1* in our method.

Appendix C

Implementation details

C.1 Weight trajectory tracking from section 3.4.3

Due to the high non-convexity of the optimization landscape, we choose to start from a relatively stable point in the parameter space. The finetuning of each architecture (ResNet20 BN, ResNet20 BN w/o affine and ResNet WN) starts from previously trained architectures on CIFAR10 via a simple SGD with an initial learning rate of 10^{-1}, a L_2-regularization parameter of 10^{-4} and a momentum parameter of 0.9. The training is performed during 200 epochs, and the learning rate is multiplied by 0.1 at epochs 80, 120 and 160.

Then we track the trajectory obtained with SGD, AdaGrad and AdaGradG. The effective learning rate for SGD is fixed to 10^{-2} and the L_2-regularization parameter is set to 10^{-3} during finetuning. It gives us the following equivalent parameters for AdaGradG: order-2 moment parameter $\beta \approx 0.99996$ and learning rate $\eta \approx 0.71$. Since the effective direction is the same for both SGD and AdaGrad (Adam without momentum), in order to have the same order of magnitude for the gradient steps we need to have effective learning rates of same order of magnitude. From Table 3.1, in the case of SGD we have $\eta^e_{k\text{SGD}} = \frac{\eta_k}{r_k^2}$, and in the case of AdaGrad we have $\eta^e_{k\text{AdaGrad}} = \frac{\eta_k}{r_k \nu_k} = \frac{\eta_k}{r_k^2 d^{-1/2} \|\mathbf{b}_k\|} = \eta^e_{k\text{SGD}} \frac{1}{d^{-1/2}\|\mathbf{b}_k\|}$.We track the quantity $\frac{1}{d^{-1/2}\|\mathbf{b}_k\|}$ during training, which is roughly in the order of magnitude of 10^{-1}. Therefore, to have gradient steps of equivalent order of magnitude between SGD and AdaGrad, we have to choose a learning rate of 10^{-3} for AdaGrad.

C.2 Implementation details of experiments in section 4.4.1

This section focuses on implementation details used to produce the results in the main text of our paper. The code that we used is provided along with this appendix. Our implementation builds upon the WILDS framework[1] released with the paper of Koh *et al.* Koh et al. (2021).

C.2.1 Construction of COCO-on-Places-224

We generated the dataset using the code[2] of Ahmed *et al.* Ahmed et al. (2021) but, as explained in the main paper, we modified it to produce images of size 224×224 instead of 64×64. The reader can refer to the appendix of Ahmed et al. (2021) for more details regarding the generation of the COCO-on-Places dataset.

C.2.2 Details about robust optimization

We trained all models on one NVIDIA® V100 Tensor Core with 16GB of memory, using PyTorch 1.10 and CUDA 10.2.

We used the implementations of IRM Arjovsky et al. (2020), Importance Weighting and GroupDRO Sagawa* et al. (2020) available in WILDS Koh et al. (2021), our own implementations of JTT Liu et al. (2021) and of GEORGE Sohoni et al. (2020) (while making sure that we could reproduce the original performances on Waterbirds and CelebA), and the official implementation[3] of EIIL Creager et al. (2021). Concerning EIIL, we recall that we were not able to make this method scale to large datasets such as CelebA.

For all methods, we used a ResNet-50 He et al. (2016b) architecture trained using stochastic gradient descent with momentum (SGD-M) and L_2 regularization, but without any learning rate scheduler. We used a momentum of 0.9 and a batch size of 128 for all datasets and all methods. The learning rate η and L_2 regularization parameters λ are set as detailed below.

JTT, GEORGE, EIIL, GramStyle all use GroupDRO Sagawa* et al. (2020) as robust optimization step. On Waterbirds and CelebA, we did not redo any grid search and used the hyperparameters found in Sagawa* et al. (2020). These hyperparameters were

[1] https://github.com/p-lambda/wilds
[2] https://github.com/Faruk-Ahmed/predictive_group_invariance
[3] https://github.com/ecreager/eiil

Table C.1 SGD-M hyperparameters for GroupDRO training.

SGD-M hyperparameters	Waterbirds	CelebA	COCO-on-Places-224
Learning rate η	10^{-5}	10^{-5}	$5 \cdot 10^{-5}$
L_2 regularization λ	1.0	0.1	10^{-2}

Table C.2 SGD-M hyperparameters for ERM training.

SGD-M hyperparameters	Waterbirds	CelebA	COCO-on-Places-224
Learning rate η	10^{-4}	10^{-4}	10^{-4}
L_2 regularization λ	10^{-3}	10^{-4}	10^{-4}

optimized using a small validation set annotated with true group labels. To produce the results on COCO-on-Places-224, we performed our own grid search using the annotated validation set. We considered values of η and λ close to those used in Sagawa* et al. (2020): $\lambda \in \{10^{-4}, 10^{-2}, 10^{-1}, 1\}$ and $\eta \in \{10^{-5}, 5{\cdot}10^{-5}, 10^{-4}\}$. The best hyperparameters for GroupDRO are summarized in Table C.1.

To ensure fair comparisons, we also performed the same grid search over η and λ for ERM, IRM and Importance Weighting. The best hyperparameters for ERM and IRM are summarized for each dataset in Table C.2 and Table C.3, respectively. Note that they correspond to those reported in Sagawa* et al. (2020) for Waterbirds and CelebA.

C.2.3 Group discovery details

For GramStyle, we follow standard practice of neural style transfer Gatys et al. (2016) and use the VGG-19 Simonyan and Zisserman (2015) architecture for the identification model. This model is trained during 1 epoch on the training dataset with ERM using a batch size of 128 and SGD-M. In the experiments of Section 4.2 in the main paper, we set the number of clusters to 2, and use the layer *conv5_1* to extract Gram Matrices. For EIIL and GEORGE, the identification model is a ResNet-50 He et al. (2016b) as used in the original methods. We train the model for 1 epoch with ERM using SGD-M, as for GramStyle. Note that the activation at the output of the last layer is a sigmoid in EIIL Creager et al. (2021) while it is a softmax in GEORGE Sohoni et al. (2020). As for GramStyle, the best results were obtained when using 2 clusters for EIIL and GEORGE. We refer the reader to Creager et al. (2021) and Sohoni et al. (2020) for other implementation details specific to EIIL and GEORGE, respectively.

Table C.3 SGD-M hyperparameters for IRM training.

SGD-M hyperparameters	Waterbirds	CelebA	COCO-on-Places-224
Learning rate η	10^{-4}	10^{-5}	$5 \cdot 10^{-5}$
L_2 regularization λ	10^{-3}	0.1	0.1

Table C.4 Grid search results on the validation sets of Waterbirds, CelebA and COCO-on-Places-224 with pseudo-group labels. We report the worst-group ('w-g') and average ('avg') accuracies for Waterbirds and CelebA datasets, and the systematically-shifted ('shift') and in-distribution ('ind') accuracies for COCO-on-Places ('COCO-on-P') dataset.

Method	Hyperparam. λ	Hyperparam. η	Waterbirds w-g	Waterbirds avg	CelebA w-g	CelebA avg	COCO-on-P sys	COCO-on-P ind
GramStyle-*cv*	0.01	$1 \cdot 10^{-5}$	74.6	82.4	**86.0**	93.2	62.8	92.3
	0.01	$5 \cdot 10^{-5}$	69.2	79.9	53.5	**94.6**	70.7	76.5
	0.01	$1 \cdot 10^{-4}$	70.0	80.6	-	-	78.5	82.7
	0.1	$1 \cdot 10^{-5}$	75.4	82.6	85.6	93.7	**78.7**	83.3
	0.1	$5 \cdot 10^{-5}$	73.8	82.4	85.0	89.1	70.4	76.4
	0.1	$1 \cdot 10^{-4}$	76.9	85.8	-	-	76.2	81.2
	1	$1 \cdot 10^{-5}$	**80.8**	**86.4**	-	-	65.5	72.6
	1	$5 \cdot 10^{-5}$	0.0	23.1	-	-	0.1	11.1
	1	$1 \cdot 10^{-4}$	0.0	23.1	-	-	0.2	11.1

C.2.4 Cross validation on pseudo-group annotations

We report in Table C.4 the results of our grid search on the validation set of each dataset using the *pseudo-annotations* discovered with our method, i.e., using our discovered environments instead of the ground-truth ones. Hence, the average and worst group accuracies in Table C.4 are computed using the discovered pseudo-groups. The hyperparameters used in GramStyle-*cv* correspond to those which yield the best worst-group accuracy in this table.