THE UNIVERSITY *of* EDINBURGH

Nesting optimization with adversarial games, meta-learning, and deep equilibrium models

Paul Micaelli

Doctor of Philosophy

The University of Edinburgh

2022

Abstract

Nested optimization, whereby an optimization problem is constrained by the solutions of other optimization problems, has recently seen a surge in its application to Deep Learning. While the study of such problems started nearly a century ago in the context of market theory, many of the algorithms developed since do not scale to modern Deep Learning applications. In this thesis, I push the understanding and applicability of nested optimization to three machine learning domains: 1) adversarial games, 2) meta-learning and 3) deep equilibrium models. For each domain, I tackle a particular goal. In 1) I adversarially learn model compression, in the case where training data isn't available, in 2) I meta-learn hyperparameters for long optimization processes without introducing greediness, and in 3) I use deep equilibrium models to improve temporal coherence in video landmark detection.

The first part of my thesis deals with casting model compression as an adversarial game. Performing knowledge transfer from a large teacher network to a smaller student is a popular task in deep learning. However, due to growing dataset sizes and stricter privacy regulations, it is increasingly common not to have access to the data that was used to train the teacher. I propose a novel method which trains a student to match the predictions of its teacher without using any data or metadata. This is achieved by nesting the training optimization of the student with that of an adversarial generator, which searches for images on which the student poorly matches the teacher. These images are used to train the student in an online fashion. The student closely approximates its teacher for simple datasets like SVHN, and on CIFAR10 I improve on the state-of-the-art for few-shot distillation (with 100 images per class), despite using no

data. Finally, I also propose a metric to quantify the degree of belief matching between teacher and student in the vicinity of decision boundaries, and observe a significantly higher match between the zero-shot student and the teacher, than between a student distilled with real data and the teacher.

The second part of my thesis deals with meta-learning hyperparameters in the case when the nested optimization to be differentiated is itself solved by many gradient steps. Gradient-based hyperparameter optimization has earned a widespread popularity in the context of few-shot meta-learning, but remains broadly impractical for tasks with long horizons (many gradient steps), due to memory scaling and gradient degradation issues. A common workaround is to learn hyperparameters online, but this introduces greediness which comes with a significant performance drop. I propose forward-mode differentiation with sharing (FDS), a simple and efficient algorithm which tackles memory scaling issues with forward-mode differentiation, and gradient degradation issues by sharing hyperparameters that are contiguous in time. I provide theoretical guarantees about the noise reduction properties of my algorithm, and demonstrate its efficiency empirically by differentiating through $\sim 10^4$ gradient steps of unrolled optimization. I consider large hyperparameter search ranges on CIFAR-10 where I significantly outperform greedy gradient-based alternatives, while achieving $\times 20$ speedups compared to the state-of-the-art black-box methods.

The third part of my thesis deals with converting deep equilibrium models to a form of nested optimization in order to perform robust video landmark detection. Cascaded computation, whereby predictions are recurrently refined over several stages, has been a persistent theme throughout the development of landmark detection models. I show that the recently proposed deep equilibrium model (DEQ) can be naturally adapted to this form of computation, given appropriate regularization. My landmark model achieves state-of-the-art performance on the challenging WFLW facial landmark dataset, reaching 3.92 normalized mean error with fewer parameters and a training memory cost of $\mathcal{O}(1)$ in the number of recurrent modules. Furthermore, I show that DEQs are particularly suited for landmark detection in videos. In this setting, it is

typical to train on still images due to the lack of labeled videos. This can lead to a "flickering" effect at inference time on video, whereby a model can rapidly oscillate between different plausible solutions across consecutive frames. I show that the DEQ root solving problem can be turned into a constrained optimization problem in a way that emulates recurrence at inference time, despite not having access to temporal data at training time. I call this "Recurrence without Recurrence", and demonstrate that it helps reduce landmark flicker by introducing a new metric, and contributing a new facial landmark video dataset targeting landmark uncertainty. On the hard subset of this new dataset, made up of 500 videos, my model improves the accuracy and temporal coherence by 10 and 13% respectively, compared to the strongest previously published model using a hand-tuned conventional filter.

Lay Summary

In an optimization problem, you are trying to find the best *strategy* to solve a problem. As humans we use our brain to do this all the time. For example, you solve an optimization problem when you decide the best time to show up to a party. Too early and you'll risk standing around awkwardly. Too late and you'll risk missing the fun. You may consider various factors in making the call, like how many of your friends are coming and how big the venue is. You may also rely on previous experience. Last time you showed up so early the host asked you to help in the kitchen. No thanks. This time it has to be just right. It has to be *optimal*.

Finding the optimal strategy by learning from previous experience is the cornerstone of deep learning. Basically, we use computers to learn strategies from vast amounts of data automatically, and use these strategies to make predictions in previously unseen scenarios. In this thesis I look at a particular flavor of optimization called *nested* optimization, which, as the name suggests, is an optimization problem that contains another optimization problem. More specifically, I consider three types of nested optimization problems: adversarial games, meta-learning, and equilibrium models.

First, let's look at nested optimization for adversarial games. Here, the host of the party knows that you're planning to arrive later than the official start time. The thing is, her objective is different than yours; she wants everyone to arrive exactly on time. She is your *adversary*. The strategy she chooses is to tell you a fake start time, which is early by the same amount that she reckons you will be late. Smart, but hard. For her strategy to be optimal, she has to figure out your optimal strategy first, which relies on all your past experience relating to parties. Therefore, though you are completely unaware of her treachery, your optimization problem is nested within her optimization problem.

Consider now a meta-learning approach to nested optimization. As before, you are trying to learn from previous party experiences what the optimal turn up time is for this party. The difference now is that you would like to leverage knowledge from other *related* problems you have successfully learned to solve in the past. For instance you once figured out the optimal morning routine, and the optimal strategy to get your dad Christmas presents that he likes. Say that in both of these problems, you found that communication with other people involved was key to your success. This insight can be helpful in the context of timing your arrival to this party: maybe telling your plans to your friends will help. In general, figuring out this problem-agnostic insight, aka *meta-knowledge*, is a form of nested optimization because it involves solving lots of individual nested problems, before you can learn something about the big picture.

The third nested optimization model we consider is deep equilibrium models, and our analogy starts to require more imagination. In this scenario, you decide to take a closer look at what your brain does when it's trying to reflect upon a single party experience you had years ago. Your brain doesn't immediately jump to a conclusion about what went right and what went wrong that day. Instead, it starts with an intuition, and then refines it several times by considering various perspectives about the problem. Eventually, your brain converges to a conclusion: it has reached an *equilibrium*. Here, the very process of producing an opinion about a single memory required solving an optimization problem, to find this equilibrium. This problem is nested within the broader problem you seek to solve, which is to make this opinion accurate.

While the above examples are illustrative, they hopefully demonstrate the breadth and variety of nested optimization. In this thesis, I focus on applications to nested problems within deep learning, where more traditional methods scale poorly.

Acknowledgements

PhDs are hard, and I am grateful for the help I received along the way. On an academic level, let me start by thanking my office mates, who engaged in endless coffee-sponsored brainstorming sessions with me, and became great friends along the way: Miguel Jaques, Luke Darlow, Ben Rhodes, Etienne Toussaint, Michael Camilleri and Emmanuel Bernieri. Spending a year of my PhD without these people during Covid revelead the value of these relationships in a humbling way. I also thank the Bayeswatch research group, led by my supervisor Amos Storkey, who was particularly helpful at the start of this PhD when I was sharpening my research tools.

On a personal level, I first want to thank Claire Micaelli. Claire was a complete stranger when I wrote the opening words to this thesis, and she now shares my last name for life. I thank my parents for raising a rebellious child with steadfast love, and being an anchor through the PhD storms. I thank my brother and my sister for pretending to care about my research. Too often, my PhD failures have gone to my heart, and my successes have gone to my head. And so finally, for shepherding me through the futility of self-worth from achievements, I thank my church, and for offering me an identity beyond self-determination, I thank my God.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Paul Micaelli

Contents

Chapter 1

Introduction

1.1 Prelude

The work presented in this thesis falls under the field of Machine Learning (ML), or more specifically Deep Learning (DL), the subset of ML that relies on large artificial neural networks. A popular definition of machine learning is given by (Mitchell, 1997): "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". Note that performance P (usually measured on test data) isn't necessarily what is being optimized for when learning experience E (e.g. one can learn from E on training data). This definition allows making a key distinction between machine learning and previous related fields (like optimization and statistics), namely the concept of generalization, which is paramount throughout this thesis. Indeed, experience E is typically finite (in time and memory) while the tasks in T are typically infinite and partially observed (e.g. if they contain samples from probabilistic distributions), and so experience E needs to be relevant for unseen tasks drawn from the same task *class*. In this thesis, I assume that the reader has a general knowledge of Deep Learning. A comprehensive overview can otherwise be found in (Goodfellow et al., 2016).

Despite its poorly streamlined early development starting in the 1960s, the success of Machine Learning (ML) in the past ten years has been second to none. In 2012, the era of learning machines was kick-started by the convergence of rising computational power and efficient convolutional neural networks (Krizhevsky et al., 2012), which dominated alternative pattern recognition approaches on the challenging ImageNet dataset (Deng

et al., 2009). Since then, machine learning models of ever increasing size have become the industry standard in fields like computer vision (He et al., 2016), speech recognition (Oord et al., 2016), generative models (Brock et al., 2019; Ho et al., 2020), and natural language processing (Brown et al., 2020).

For all this progress, the latest machine learning models still pale in comparison to the initial tacit ambition of the field, namely building machines with a general purpose Artificial Intelligence (AI). Putting aside the potentially unrealistic nature of this endeavor, it remains useful to make statements about the limitations of the machine learning state-of-the-art by comparing it to human intelligence. While this discussion was once confined to the academic setting, the recent surge of interest in large language models, sparked by Chat-GPT (OpenAI, 2023), has helped in making this discussion more global.

On a superficial level, human intelligence differs from machine intelligence by being embodied, and naturally augmented with ethics. On a more technical level, it is generally argued that other major sources of difference fall under 1) adaptability or 2) scalability. The issue of adaptability refers to the over-specialization of ML models. Indeed, the most common ML pipeline consists in training a model from scratch on a single specific task, such as the classification of dog breeds. This "tabula rasa" approach means that such a model is completely inadequate for other tasks, even if somewhat related. The lack of adaptation to unseen tasks is in stark contrast with the intelligence of humans, who are able to reason quickly on a continuum of unseen problems, by leveraging previous related experiences. Secondly, the issue of scalability refers to the fact that increasing the size of machine learning models and training datasets exhibits sharply diminishing return. Indeed, the performance on a task v.s. model size graph is typically highly sublinear. Considering language translation as an example, scaling up a small 100M-parameter GPT3 model by 1B parameters increases the BLUE score by 25 points, while scaling it further by 100B parameters increases the BLUE score by less than 10 points (Brown et al., 2020).

The nested optimization methods I develop in this thesis, though they are showcased within specific applications, can be understood as a part of the solutions that the machine learning community is building to address the adaptability and scalability issues. In Chapter 3 I tackle the problem of diminishing reward by compressing large neural networks into smaller ones. More specifically, I show that no data is needed for effective distillation. Chapter 5 also falls under scalability solutions, as deep equilibrium models can be seen as a way to represent infinitely deep neural networks with a finite computation cost and parameter count.

In Chapter 4 I extend meta-learning tools to real-world problems with many gradient steps. Meta-learning, or learning to learn, is arguably the most promising solution to the adaptability problem of machine learning, as it explicitly learns task-agnostic knowledge which can be leveraged for quickly adapting to unseen tasks. In the specific case of hyperparameter optimization, I make the differentiation of the unrolled learning procedure more efficient when it consists of thousands of gradient steps.

Nested optimization is a mature field, which far predates the applications tackled in this thesis. That being said, much of the literature in nested optimization considers toy problems and make hard assumptions which don't hold in most machine learning problems. As such, one of the contribution of this thesis is to extend nested optimization tools to the deep learning setting. This should be considered an independent contribution to the broader application-specific contributions we make in each chapter. These include demonstrating that networks can be trained without any training data in Chapter 3, showing the superiority of non-greedy algorithms in Chapter 4, and the importance of temporal coherence for landmark detection in Chapter 5.

Symbol	Meaning
$\boldsymbol{x}$	single datapoint input
$\boldsymbol{y}$	single datapoint label
$\boldsymbol{X}$	all dataset inputs
$\boldsymbol{Y}$	all dataset labels
$\mathcal{D} = (\boldsymbol{X}, \boldsymbol{Y})$	dataset
$\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\omega}$	learnable parameters
$\boldsymbol{\theta}^*, \boldsymbol{\phi}^*, \boldsymbol{\omega}^*$	optimum or learned parameter
$p(\boldsymbol{u})$, $q(\boldsymbol{u})$	probability distributions of $\boldsymbol{u}$
$\mathbb{E}_{\boldsymbol{u}\sim p(\boldsymbol{u})}\, f(\boldsymbol{u})$	expectation of $f(\boldsymbol{u})$ under $p(\boldsymbol{u})$
$\mathcal{L}(\cdot)$	loss function
$\mathcal{L}_{\text{CE}}(p(\boldsymbol{u}), q(\boldsymbol{u}))$	cross-entropy of $p(\boldsymbol{u})$ and $q(\boldsymbol{u})$
$D_{KL}((p(\boldsymbol{u}) \mid\mid q(\boldsymbol{u})))$	Kullback–Leibler divergence of $p(\boldsymbol{u})$ and $q(\boldsymbol{u})$
$\|\boldsymbol{v}\|_{\text{p}}$	L-p norm of $\boldsymbol{v}$

Table 1.1: Common meaning of symbols used

1.2 Notation and conventions

The notation and conventions used are consistent throughout this thesis. Vectors are denoted by bold lowercase (e.g. $\boldsymbol{u}$) and matrices by bold uppercase (e.g. $\boldsymbol{A}$). By convention, $\boldsymbol{u}$ denotes a column vector while $\boldsymbol{u}^\top$ denotes a row vector. Scalars are written as normal uppercase (e.g. N) when they are fixed properties of the problem that the practitioner cannot change, or as normal lowercase (e.g. α) when they are properties of the algorithm which the practitioner can change. A function $f(\cdot)$ or $F(\cdot)$ is written as $f(\boldsymbol{u}, \boldsymbol{v}; \boldsymbol{\theta})$ or $F(\boldsymbol{u}, \boldsymbol{v}; \boldsymbol{\theta})$ to indicate that it inputs $\boldsymbol{u}$ and $\boldsymbol{v}$, and is parameterized by $\boldsymbol{\theta}$. The most common meaning of certain symbols is assumed, as defined in Tab. 1.1.

1.3 Supervised learning basics

Supervised machine learning refers to the setting where a label $\boldsymbol{y}$ is available for each input $\boldsymbol{x}$, and our task is to predict $\boldsymbol{y}$ from $\boldsymbol{x}$. More specifically, the tasks considered in this thesis are either 1) image classification, whereby $\boldsymbol{x}$ is an image and $\boldsymbol{y}$ is a one-hot probability vector indicating the class of $\boldsymbol{x}$, or 2) landmark regression, whereby $\boldsymbol{y}$ represents 2D coordinates for several keypoints of $\boldsymbol{x}$.

This section covers some basic machine learning tools relevant to the models introduced in this thesis. In particular, maximum likelihood estimation is important for Chapters 3-5, maximum a posteriori estimation is important for Chapter 5, while gradient descent update rules are particularly important for Chapter 4. While these tools aren't limited to the supervised learning setting, they are used in this context throughout the thesis.

1.3.1 Maximum likelihood estimation

Our objective is to learn a model $p_{\mathrm{m}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$ which approximates the true (conditional) data distribution $p_{\mathrm{d}}(\boldsymbol{y}|\boldsymbol{x})$:

$$\boldsymbol{\theta}^*_{\mathrm{MLE}} = \arg\min_{\boldsymbol{\theta}} D_{KL}(p_{\mathrm{d}}(\boldsymbol{y}|\boldsymbol{x}) \,||\, p_{\mathrm{m}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})) \tag{1.1}$$

where the KL divergence between two densities $p(\boldsymbol{y})$ and $q(\boldsymbol{y})$ is given by:

$$D_{KL}(p(\boldsymbol{y}) \,||\, q(\boldsymbol{y})) = \int p(\boldsymbol{y}) \log \frac{p(\boldsymbol{y})}{q(\boldsymbol{y})}\, dx \tag{1.2}$$

Now since $p_{\mathrm{d}}(\boldsymbol{y}|\boldsymbol{x})$ doesn't depend on $\boldsymbol{\theta}$ the above reduces to

$$\boldsymbol{\theta}^*_{\mathrm{MLE}} = \arg\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim p_{\mathrm{d}}(\boldsymbol{x},\boldsymbol{y})} - \log p_{\mathrm{m}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) \tag{1.3}$$

which is the maximum likelihood estimate (MLE), or equivalently the parameters that minimize the negative (conditional) log likelihood under the model. In practice we don't have access to the joint data generating distribution $p_{\mathrm{d}}(\boldsymbol{x},\boldsymbol{y})$ and so we minimize a Monte Carlo estimate of the above expectation over some independent and identically distributed (iid) samples, which we call the *training* datapoints: $\mathcal{D}_{\mathrm{train}} = (\boldsymbol{X}_{\mathrm{train}}, \boldsymbol{Y}_{\mathrm{train}})$. In the case of classification, a neural network outputs a vector of probabilities over classes, and the above is called the cross entropy loss $\mathcal{L}_{\mathrm{CE}}$ between $p_{\mathrm{d}}(\boldsymbol{y}|\boldsymbol{x})$ and $p_{\mathrm{m}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$:

$$\boldsymbol{\theta}^*_{\mathrm{MLE}} = \arg\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{D}_{\mathrm{train}}} \mathcal{L}_{\mathrm{CE}}(p_{\mathrm{d}}(\boldsymbol{y}|\boldsymbol{x}), p_{\mathrm{m}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})) \tag{1.4}$$

In the case of regression, making the assumption that $p_{\mathrm{m}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) \propto \mathcal{N}(\boldsymbol{y};\hat{\boldsymbol{y}},\sigma^2\mathbb{1})$, where $\hat{\boldsymbol{y}} = \hat{\boldsymbol{y}}(\boldsymbol{x};\boldsymbol{\theta})$ is the output of our neural network and σ some scalar, Eq. (1.3) leads to minimizing the mean squared error loss $\mathcal{L}_{\mathrm{MSE}}$:

$$\begin{aligned}
\boldsymbol{\theta}^*_{\mathrm{MLE}} &= \arg\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{D}_{\mathrm{train}}} -\log \mathcal{N}(\boldsymbol{y};\hat{\boldsymbol{y}},\sigma^2\mathbb{1}) && (1.5)\\
&= \arg\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{D}_{\mathrm{train}}} C + \frac{1}{2\sigma^2}(\boldsymbol{y}-\hat{\boldsymbol{y}})^T(\boldsymbol{y}-\hat{\boldsymbol{y}}) && (1.6)\\
&= \arg\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{D}_{\mathrm{train}}} \|\boldsymbol{y}-\hat{\boldsymbol{y}}\|_2^2 && (1.7)\\
&= \arg\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{D}_{\mathrm{train}}} \mathcal{L}_{\mathrm{MSE}}(\boldsymbol{y},\hat{\boldsymbol{y}}) && (1.8)
\end{aligned}$$

To evaluate the generalization performance of the learned model, we measure the loss $\mathcal{L}_{\mathrm{CE}}$ or $\mathcal{L}_{\mathrm{MSE}}$ on samples from the data generating distribution which weren't used during training, referred to as *test* datapoints: $\boldsymbol{X}_{\mathrm{test}}, \boldsymbol{Y}_{\mathrm{test}}$.

1.3.2 Maximum a posteriori estimation

In some cases, we would like to encourage the learned parameters $\boldsymbol{\theta}^*$ to be close to some meaningful value $\boldsymbol{\theta}_0$. This is best achieved in a Bayesian framework, where we use Bayes' rule to write the posterior over parameters as being proportional to the likelihood and a prior:

$$p(\boldsymbol{\theta}|\boldsymbol{y},\boldsymbol{x}) \propto p_{\mathrm{m}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{1.9}$$

Now in the special case where we assume a Gaussian prior, namely $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta};\boldsymbol{\theta}_0,\epsilon^2\mathbb{1})$, maximizing the posterior in the case of regression leads to the following the maximum a posteriori estimate (MAP):

$$\begin{aligned}
\boldsymbol{\theta}^*_{\mathrm{MAP}} &= \arg\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{D}_{\mathrm{train}}} -\log p(\boldsymbol{\theta}|\boldsymbol{y},\boldsymbol{x}) && (1.10)\\
&= \arg\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{D}_{\mathrm{train}}} -\log p_{\mathrm{m}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) - \log p(\boldsymbol{\theta}) && (1.11)\\
&= \arg\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y}\sim\mathcal{D}_{\mathrm{train}}} \mathcal{L}_{\mathrm{MSE}}(\boldsymbol{y},\hat{\boldsymbol{y}}) + \frac{\lambda}{2}\|\boldsymbol{\theta}-\boldsymbol{\theta}_0\|_2^2 && (1.12)
\end{aligned}$$

where $\lambda \propto \sigma^2\epsilon^{-2}$

1.3.3 Gradient descent

Both maximum likelihood and maximum a posteriori estimations boil down to the minimization of some training loss function $\mathcal{L}_{\text{train}}$, which includes the outputs of some neural network parameterized by $\boldsymbol{\theta}$. Due to the complexity and size of typical neural networks, the minimizer $\boldsymbol{\theta}^*$ of $\mathcal{L}_{\text{train}}$ cannot be found analytically, and we must rely on iterative algorithms like gradient descent. In its simplest form, gradient descent updates parameters $\boldsymbol{\theta}_t$ at step t by letting:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \left(\frac{\partial \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}} \right)^\top \tag{1.13}$$

where α is the learning rate. Note that I use the numerator convention of matrix calculus, which means that given some scalar s and two vectors $\boldsymbol{u} \in \mathbb{R}^U$ and $\boldsymbol{v} \in \mathbb{R}^V$ then $\frac{\partial s}{\partial \boldsymbol{v}} \in \mathbb{R}^{1 \times V}$ and $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{v}} \in \mathbb{R}^{U \times V}$. In practice, modern optimizers rely on variants of stochastic gradient descent, whereby the gradient term is calculated on minibatches of data rather than the whole training dataset.

1.4 Thesis structure and list of contributions

The work I produced for this PhD thesis includes two first author NeurIPS publications, and one first author paper currently under review at CVPR 2023. I also significantly contributed towards a PAMI survey paper as a coauthor. In Chapter 2 I give a more technical overview of the various flavours of nested optimization relevant to my work. This chapter leverages concepts and perspectives that were contributed towards in the making of the following survey:

> *T. Hospedales, A. Antoniou, P. Micaelli and A. Storkey, "Meta-Learning in Neural Networks: A Survey" in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. , no. 01, pp. 1-1, 5555, 2020.*

This survey has become the most popular survey on meta-learning in the past two years, sparking much discussion within the community since publication. It provides an up to date literature review for a fast moving field, as well as a perspective on the future of meta-learning and a new taxonomy which enables faster prototyping of meta-learning solutions.

The remaining chapters correspond to one paper each, with extra background relating to the specific application domains of each paper. In Chapter 3 I cover my work on knowledge distillation in the absence of training dataset. The idea is to train a generator adversarially against a student network: the generator looks for images for which the student maximally mismatches the teacher network, and the student learns on those images. This work formed the basis of a spotlight paper published at NeurIPS 2019:

> *P. Micaelli and A. Storkey, "Zero-shot Knowledge Transfer via Adversarial Belief Matching", Advances in Neural Information Processing Systems, 2019.*

In this chapter I also give background on model compression, knowledge distillation, and adversarial attacks. While the field that has been building on this publication the most deals with the privacy of modern neural networks, much of this literature is tangential to this thesis and will therefore be given a cursory glance.

In Chapter 4 I present my work on gradient based hyperparameter optimization for problems with many optimization steps. The forward-mode algorithm proposed therein tackles a memory limitation of many meta-learning algorithms, and was published at NeurIPS 2021:

> *P. Micaelli and A. Storkey, "Gradient-based Hyperparameter Optimization Over Long Horizons", Advances in Neural Information Processing Systems, 2021.*

In this chapter, I also give an overview of traditional hyperparameter optimization techniques (which serve as our baselines) and a discussion of forward-mode vs. reverse-mode differentiation.

Finally, in Chapter 5 I present work done on video landmark detection using deep equilibrium models. This work was carried out in partnership with NVIDIA, and was published at CVPR 2023:

> *P. Micaelli, A. Vahdat, H. Yin, J. Kautz, P. Molchanov "Recurrence without Recurrence: Stable Video Landmark Detection with Deep Equilibrium Models", under review, 2022.*

I conclude in Chapter 6 with the main insights uncovered during this PhD, as well as potential future work.

Chapter 2

Nested Optimization Models

2.1 Overview

An optimization problem is the task of finding the best solution to a problem, according to some measurable criterion, from a set of feasible solutions. In machine learning, optimization often reduces to the task of finding the minimum (or maximum) of some differentiable function $F(\boldsymbol{\omega}) : \mathcal{R}^D \rightarrow \mathcal{R}$. Since $\min F(\boldsymbol{\omega}) = -\max(-F(\boldsymbol{\omega}))\ \forall F$ we write the general form of an optimization problem without loss of generality as follows:

$$\boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}} F(\boldsymbol{\omega}) \tag{2.1}$$

$$s.t.\ \ G^{(i)}(\boldsymbol{\omega}) \leq 0 \ \ \text{for} \ \ i = 1, 2, \cdots \tag{2.2}$$

where $G^{(i)}$s are an arbitrary number of constraints which define the feasible set (or search space) for $\boldsymbol{\omega}^*$. Note that we omit writing equality constraints explicitly in the above equation for the sake of notation clarity, and because any equality constraint can be expressed by two inequality constraints without loss of generality.

Many relevant problems in machine learning can be expressed as a nested optimization. This is a type of optimization whose constraints contain another optimization. The degree of nesting is usually one, in which case nested optimization is synonymous with *bilevel optimization*, a term coined in (Candler and Norton, 1977). The general form

for the bilevel optimization problem is as follows:

$$\boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}} F(\boldsymbol{\omega}, \boldsymbol{\theta}^*) \tag{2.3}$$

$$s.t. \quad G^{(i)}(\boldsymbol{\omega}, \boldsymbol{\theta}) \leq 0 \quad \text{for} \quad i = 1, 2, \cdots \tag{2.4}$$

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \boldsymbol{\omega}) \tag{2.5}$$

$$s.t. \quad g^{(j)}(\boldsymbol{\theta}, \boldsymbol{\omega}) \leq 0 \quad \text{for} \quad j = 1, 2, \cdots \tag{2.6}$$

where Eq. (2.3) is called the *upper* (or *outer*) optimization and Eq. (2.5) is called the *lower* (or *inner*) optimization. The lower objective $f(\boldsymbol{\theta}, \boldsymbol{\omega})$ under lower constraints $g^{(j)}$ has an optimum $\boldsymbol{\theta}^*$ which determines the upper objective function $F(\boldsymbol{\omega}, \boldsymbol{\theta}^*)$. While it was omitted in the equations above for clarity, note that the lower level optimum is a function of the upper variable, i.e. $\boldsymbol{\theta}^* = \boldsymbol{\theta}^*(\boldsymbol{\omega})$. This is often called the *best-response* function, and it is particularly relevant in the context of meta-learning where it is challenging to differentiate.

This type of optimization was first investigated by Heinrich Freiherr von Stackelberg (von Stackelberg et al., 1934; Stackelberg, 1952) in the context of economic game theory and market equilibrium. There, the problem is posed as a two player game with a single move per player, with one player (the *leader*) making their move before the other (the *follower*). Each player has their own objective: the upper objective in Eq. (2.3) corresponds to the leader's problem, while the lower objective in Eq. (2.5) is the follower's problem. The leader $\boldsymbol{\omega}$ has the advantage, because it moves first with full knowledge of the follower's strategy. The follower $\boldsymbol{\theta}$ always optimizes for their objective, but that objective is determined by the leader's action, and the leader takes that into account when choosing their strategy. It is never in the follower's advantage not to optimize for their objective, given that there is only a single turn to the game, which is why the leader has full knowledge of what the follower will do. The above setting is often referred to as a *Stackelberg game*.

This paradigm is best illustrated with an example, inspired from (Colson et al., 2007). Consider the problem of a highway toll company, whose task is to set the price for each section of highway under their management, so as to maximize profit. The company knows that the road users will only use their highways if they correspond to a competitive offer in terms of speed and cost compared to alternatives like trains and country roads. To find the best price, the toll company thus has to solve another nested optimization problem, namely the follower's problem, who simply finds which mode of transportation minimizes some function of time and cost. In order to be a bilevel optimization problem, it must be the case that the toll company has full knowledge of the follower's objective, so that the follower's strategy is entirely determined by the price chosen by the toll company.

The game theory perspective of bilevel optimization can often give us relevant insight for machine learning applications. For instance, one thing it points to is that the basic formulation in Eq. 2.3 – 2.6 lacks clarity when there are several solutions possible to the lower level problem, i.e. $\boldsymbol{\theta}^* \in \{\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, \cdots\}$. In this case, it is common to make one of two assumptions, referred to as optimistic and pessimistic. In the optimistic (pessimistic) assumption, the follower chooses the solution which leads to the best (worse) upper level objective for the leader. This can be relevant in meta-learning applications (Sec. 2.3) where a large gap between the optimistic and pessimistic upper level objective function can lead to a higher generalization error.

Soon after its inception by H. Stackelberg, which focused on applications to markets, research in bilevel optimization continued as part of the field of mathematical programming (Bracken and McGill, 1973; Candler and Norton, 1977; Fortuny-Amat and McCarl, 1981). There, interest was driven by the surprising complexity of bilevel optimization problems, whose hierarchical nature typically produces non-convex problems with disconnected feasible sets, even when the lower level problem is convex. In fact, even basic bilevel optimization problems were proven to be NP-hard (Hansen et al., 1992), and merely checking a proposed solution for bilevel optimality is itself NP-hard (Vicente et al., 1994).

Theoretical research in bilevel optimization is still very much active today, and has branched out into a myriad of sub problems and method classes. This includes evolutionary bilevel optimization (Mathieu et al., 1994; Yin, 2000; Sinha et al., 2014), discrete bilevel optimization (Vicente et al., 1996; Moore and Bard, 1990; Bard and Moore, 1992), and multi-objective bilevel optimization (Gadhi and Dempe, 2012; Eichfelder, 2007; Ye, 2011). Applications to bilevel optimization have grown exponentially since being used for market simulations. The most common applications where a leader-follower Stackleberg game arises include environmental research (Amouzegar and Moshirvaziri, 1999; Sinha et al., 2013; Whittaker et al., 2017), homeland security (An et al., 2013; Wein, 2009; Brown et al., 2005; Brown et al., 2009), chemistry (Seider and White III, 1985; Clark and Westerberg, 1990; Halter and Mostaghim, 2006), engineering structure design (Bendsøe, 1995; Herskovits et al., 2000; Christiansen et al., 2001) and control problems / robotics (Mombaur et al., 2010; Albrecht et al., 2011; Suryan et al., 2016). Note that virtually all of the tools used in these research and application domains do not scale to the deep learning setting, which is the context of this thesis. As such, I do not cover these individually in detail, and instead refer the interested reader to the following surveys (Sinha et al., 2018; Dempe, 2018; Dempe and Zemkoho, 2021).

2.2 Adversarial learning

Adversarial machine learning usually refers to one of two things. First, it can refer to the idea of perturbing the training or inference input data in a way that degrades the performance of a model. For instance, it was observed that high-dimensional input images to neural networks can be perturbed in small amounts imperceptible to humans, such that the corresponding network output changes drastically (Szegedy et al., 2014; Goodfellow et al., 2015). It was also shown that perturbations can be made to the training dataset in a way that disrupts the learning process and leads to poor model performance, a method referred to as data poisoning (Zhang et al., 2021a). Adversarial

attacks cannot always be cast as a nested optimization problem, and are less relevant to the methods devised in this thesis. In Chapter 3 I use adversarial attacks as a way to probe decision boundaries of distilled neural networks, and more information on this attack is given in Sec. 3.1.3.

Another facet of adversarial learning more generally relevant to this thesis is the joint training of two models, such that each model's performance is negatively correlated to the other. For instance, this is at the heart of Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) where the generator network has a higher loss when the discriminator's loss is low and vice versa. This paradigm has shown to be useful in various applications where images must be generated like in super-resolution (Ledig et al., 2016), but also in related settings like density estimation (Mescheder et al., 2017). The network presented in Chapter 3 is adversarially trained with a generator in a minimax game, which is a specific example of a nested optimization problem. The following overview is given in the context of GANs without loss of generality.

2.2.1 Minimax training

GANs consist of a generator G and a discriminator D trained adversarially. The discriminator inputs an image $\boldsymbol{x}$ and outputs a probability of this image being real. It is trained to classify *real* images, which are sampled $p_{\text{data}}(\boldsymbol{x})$, from *fake* images, which are sampled from $p_G(\boldsymbol{x})$. The generator inputs a noise vector $\boldsymbol{z}$ and outputs an image, which is equivalent to a sample from $p_G(\boldsymbol{x})$. It is trained to produce images that look real enough to fool the discriminator. In the simplest GAN formulation, the above dynamics amount to solving the following minimax objective:

$$\min_G \max_D V(G, D) \tag{2.7}$$

where

$$V(G, D) = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log D(\boldsymbol{x}) + \mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})} \log(1 - D(G(\boldsymbol{z})) \tag{2.8}$$

and where I use the letters D and G to refer to both the functions and their parameters to make notation clearer, as per the original GAN formulation in (Goodfellow et al., 2014).

The minimax game in Eq. (2.7) has a leader follower structure, just like the Stackelberg games introduced in Sec. 2.1. This can be made more evident by rewriting Eq. (2.7) as:

$$\min_{G} V(G, D^*) \tag{2.9}$$

$$s.t. \quad D^* = \arg\max_{D} V(G, D) \tag{2.10}$$

For any fixed generator G, the optimal discriminator is given by

$$D^* = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})} \tag{2.11}$$

Importantly, note that because $V(G, D)$ in a DL setting is highly non-convex non-concave, the minimax problem isn't equivalent to a maximin problem (corresponding to the upper and lower levels being swapped in Eq. 2.9 – 2.10. In the minimax case as shown above, the generator seeks to fool an optimal discriminator, which corresponds to learning $p_G(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$ as wanted. If we were to consider the maximin objective, the generator would instead seek to map every $\boldsymbol{z}$ value to the single image that maximizes the discriminator's output probability. This is called *mode collapse*, and corresponds to all of the inputs $\boldsymbol{z}$ being mapped to a single type of image.

2.2.2 Online adversarial learning

In practice, exactly solving the objective of Eq. (2.9) with gradient methods is intractable, because that would require differentiating through the algorithm used to find D^*. Instead, the original GAN model was trained using the gradient descent-ascent (GDA) online algorithm, whereby both discriminator and generator are trained iteratively by taking simultaneous gradient update steps. This approach makes the distinction between minimax and maximin ambiguous, and thus encourages mode

collapse (Goodfellow et al., 2014; Goodfellow, 2017). In fact, several authors have shown that GDA for Stackelberg games like GANs fails to capture the hierarchy of the nested optimization problem (Daskalakis and Panageas, 2018; Jin et al., 2020), and fail to converge to a local optimum unless strong assumptions are made.

Some attempts have been made to improve on GDA. In unrolled GANs (Metz et al., 2017), the authors take several gradient steps on D before taking a single step on G. By tracking these gradient update steps themselves in autograd and differentiating through them, this allows for a better approximation of D^*, and for G to take into account how D will adapt to its update. While this approach can help prevent mode collapse, it is highly limited in memory due to backpropagation having to store all gradient updates in memory. As a result, only ~ 10 update steps are unrolled for the discriminator, which falls short of the thousands of steps needed in real-world GAN training.

The memory limitation of backpropagation when differentiating through many unrolled gradient steps is a recurring theme in this thesis. For the adversarial nested optimization in Chapter 3, I rely on a modified version of GDA that does not unroll the updates of the discriminator. However, for the meta-learning nested optimization of Chapter 4 I do unroll the lower level optimization, and circumvent the memory issues of reverse-mode differentiation by using forward-mode differentiation. More justification for this choice is given in Sec. 2.3.5.

2.3 Meta-learning

Broadly speaking, meta-learning is best summarized as *learning to learn*. In most cases relevant to this thesis, this can be paraphrased as *learning how to train a neural network*, and in Chapter 4 I focus on learning optimization hyperparameters as an example. As the success of deep learning was driven by replacing manual feature extraction algorithms (Lowe, 2004) with learned features, meta-learning aims to provide a further revolution by jointly learning the model, features and training algorithm.

As defined in the recent meta-learning survey (Hospedales et al., 2021), which was coauthored during this thesis, I take the view that meta-learning can be defined in the the multi-task or single task setting. In the multi-task setting, the aim is to meta-learn some task agnostic knowledge that can then be used to quickly solve unseen tasks from the same task distribution. In the single task setting, the aim is to improve how to solve a given task by repeating several training episodes. The method presented in Chapter 4 is concerned with the single task setting, but its insights are relevant for the multi-task setting as well.

Meta-learning allows us to directly tackle the main limitations of deep learning, namely the need for large quantities of data and computational resources. Indeed, since the learning algorithm itself can be learned to minimize some meta-objective, this objective can explicitly target data efficiency or quick convergence. Additionally, the multi-task view of meta-learning also enables for models that are better at reusing knowledge learned from past experience, which is another main limitation of the current tabula rasa paradigm of deep learning.

It is worth noting that the boundaries of what constitutes a meta-learning algorithm can be somewhat blurry. In the most general sense, many conventional algorithms such as cross-validation could be defined as a type of "learning to learn" algorithm. Again, I take the view defined in our survey (Hospedales et al., 2021) that the salient characteristic of contemporary meta-learning is an explicitly defined meta-level objective, and end-to-end optimization of the inner algorithm with respect to this objective.

Over the past five years, the applications of meta-learning have grown exponentially, spanning domains like few-shot learning (Snell et al., 2017), convergence speed (Duan et al., 2017), unsupervised learning (Hsu et al., 2019), neural architecture search (Liu et al., 2019), optimizer learning (Ravi and Larochelle, 2017) and many more. Many of these meta-learning applications are gradient-based, which means that the meta-learning step requires differentiating over optimization itself. Since reverse-mode differentiation requires activations to be stored during the forward pass, these meta-learning algorithms typically scale poorly in memory. In particular, they only allow

learning over a handful of gradient steps, such as the few-shot learning setting (Finn et al., 2017), or they make coarse approximations by e.g. truncating the graph to be differentiated (Shaban et al., 2019). The algorithm presented in Chapter 4 proposes an efficient forward-mode differentiation algorithm that directly addresses this wide-spread meta-learning challenge.

2.3.1 The many-task view

While the model in Chapter 4 is showcased on the single-task meta-learning setting, it could be applied to the many-task setting, for which I now give a brief overview. I follow the notation used in our survey (Hospedales et al., 2021).

Let's contrast the standard machine learning pipeline with the meta-learning pipeline. Recall that in the conventional supervised setting, we have a training dataset $\mathcal{D}^{\text{train}} = \{(x_i^{\text{train}}, y_i^{\text{train}})\}_{i=1}^N$ and we train a predictive model $\hat{y} = f(x; \boldsymbol{\theta})$ by solving:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}^{\text{train}}; \boldsymbol{\theta}, \boldsymbol{\omega}) \tag{2.12}$$

where $\mathcal{L}$ is a loss function. The conditioning on $\boldsymbol{\omega}$ denotes the dependence of this solution on assumptions about 'how to learn', which are specified manually by the practitioner. This includes the choice of optimizer to use, its hyperparameters, the function class and architecture of f etc.. After training, the generalization performance of $f(x; \boldsymbol{\theta}^*)$ is measured by evaluating this loss on test data:

$$\text{test score} = \mathcal{L}(\mathcal{D}^{\text{test}}; \boldsymbol{\theta}^*) \tag{2.13}$$

In the many-task meta-learning setting, we seek to learn $\boldsymbol{\omega}$ across a multitude of tasks. Formally, a task $\mathcal{T}$ is defined as a dataset and loss function tuple: $\mathcal{T} = \{\mathcal{D}, \mathcal{L}\}$. In the analysis below, I assume the loss is the same for all tasks to simplify notation.

During the *meta-training* stage we have access to a set of M *source* tasks sampled from a task distribution $p(\mathcal{T})$: $\mathscr{D}_{source} = \{(\mathcal{D}^{train}_{source}, \mathcal{D}^{val}_{source})^{(i)}\}_{i=1}^{M}$. Note that each task has both training and validation data. The source train and validation datasets are often respectively called *support* and *query* sets, and $\boldsymbol{\omega}$ is often called the *meta-knowledge* or *across-task* knowledge. The meta-training step of 'learning how to learn' can be written as the following nested optimization:

$$\boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}} \sum_{i=1}^{M} \mathcal{L}(\boldsymbol{\theta}^{*(i)}(\boldsymbol{\omega}), \boldsymbol{\omega}, \mathcal{D}^{val\ (i)}_{source}) \tag{2.14}$$

$$s.t. \quad \boldsymbol{\theta}^{*(i)}(\boldsymbol{\omega}) = \arg\min_{\theta} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\omega}, \mathcal{D}^{train\ (i)}_{source}) \tag{2.15}$$

One would best summarize the meta-training step above as follows: the outer level optimization learns $\boldsymbol{\omega}$ such that it produces models $\boldsymbol{\theta}^{*\ (i)}(\boldsymbol{\omega})$ that perform well on their validation sets after training, where the training pipeline is determined by $\boldsymbol{\omega}$.

In the meta-testing stage, we test the performance of $\boldsymbol{\omega}$ on Q *target tasks* $\mathscr{D}_{target} = \{(\mathcal{D}^{train}_{target}, \mathcal{D}^{test}_{target})^{(i)}\}_{i=1}^{Q}$ where each task also has both training and test data. More specifically, we use the learned meta-knowledge $\boldsymbol{\omega}^*$ to train the base model on each previously unseen target task i:

$$\text{test score} = \sum_{i=1}^{Q} \mathcal{L}(\boldsymbol{\theta}^{*(i)}, \boldsymbol{\omega}^*, \mathcal{D}^{val\ (i)}_{target}) \tag{2.16}$$

$$\text{where} \quad \boldsymbol{\theta}^{*(i)} = \arg\min_{\theta} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\omega}^*, \mathcal{D}^{train\ (i)}_{target}) \tag{2.17}$$

Contrary to the conventional pipeline, learning on the training set of a target task i now benefits from meta-knowledge $\boldsymbol{\omega}^*$ about the algorithm to use. For instance $\boldsymbol{\omega}$ could be an estimate of the initial parameters such that the some gradient based optimizer converges to a good solution in a few steps (Finn et al., 2017), a hyper-parameter such as regularization strength (Franceschi et al., 2018), a parameterization of the loss function to meta-train on (Li et al., 2019), or an optimizer (Ravi and Larochelle, 2017).

2.3.2 The single-task view

The above formalization of meta-training uses the notion of a distribution over tasks, and seeks to learn a single meta-knowledge $\boldsymbol{\omega}$ that works well for any task sampled from this distribution. While having many tasks is most common in the meta-learning literature, it is not a necessary condition for meta-learning. More formally, if we are given a single train and test dataset, we can artificially create M "source tasks" by creating M train-val random splits from the single train dataset. At meta-test time, once $\boldsymbol{\omega}^*$ is learned, we can simply train on the whole original training set and test on the test set.

To be more precise, since all train splits above are from the same dataset, we are learning $\boldsymbol{\omega}$ over several episodes rather than several tasks per se. But allowing this extension makes meta-learning tools more evidently relevant to more deep learning applications, where the training dataset is unique and $\boldsymbol{\omega}$ is usually set by hand.

2.3.3 Online meta learning

A naive implementation of the bilevel optimization in Eq. 2.14 – 2.15 is expensive in both time (because each outer step requires several inner steps) and memory (because reverse-mode differentiation requires storing the intermediate inner states). For this reason, much of meta-learning has focused on the few-shot regime (Finn et al., 2017). However, there is an increasing focus on methods which seek to extend optimization-based meta-learning to the many-shot regime.

The simplest solution is to alternate updates on $\boldsymbol{\omega}$ and $\boldsymbol{\theta}$. This is particularly common in gradient-based meta-learning models, which I focus on in this thesis. This solution shortens the *horizon* of the lower level problem, which refers to the number of gradient updates taken to find $\boldsymbol{\theta}^*$ before one update of $\boldsymbol{\omega}$ is done. For instance, this type of online meta-learning is used to perform differentiable neural architecture search (Liu et al., 2019; Zela et al., 2020), to learn optimizer hyperparameters (Baydin et al., 2018)

or learn optimizers directly (Li et al., 2019). In Chapter 4, I build on the observations of (Wu et al., 2018b) to reject this approach altogether. Indeed, I argue that online meta-learning methods introduce greediness in the meta objective function which results in a poor approximation to $\boldsymbol{\omega}^*$.

2.3.4 Long horizon meta-learning

There exists alternative methods to online meta-learning which do not shorten the horizon of the inner loop, but instead make approximations to the gradient calculation in the upper level. These methods include implicit differentiation of $\boldsymbol{\omega}$ (Pedregosa, 2016; Rajeswaran et al., 2019; Lorraine et al., 2019), gradient preconditioning (Flennerhag et al., 2020), truncation (Shaban et al., 2019), shortcuts (Fu et al., 2016) and inner loop inversion (Maclaurin et al., 2015). Another family of approaches accelerate meta-training via closed-form solvers in the inner loop (Bertinetto et al., 2019; Lee et al., 2019). In contrast to the above approximate methods, in Chapter 4 I use forward-mode differentiation (Williams and Zipser, 1989; Franceschi et al., 2017) which calculates upper level gradients exactly.

Each method has different limitations. Implicit gradients scale to large dimensions of $\boldsymbol{\omega}$ but only provide approximate gradients for it, and require the inner task loss to be a function of $\boldsymbol{\omega}$. Forward-mode differentiation is exact and doesn't have such constraints, but scales poorly with the dimension of $\boldsymbol{\omega}$. Gradient degradation is also a challenge in the many-shot regime, and solutions include warp layers (Flennerhag et al., 2020), or gradient averaging, as proposed in Chapter 4.

2.3.5 Greediness: adversarial vs. meta-learning setting

In the adversarial model of Chapter 3, I decide against unrolling the lower level of the nested optimization problem, which means that I use greedy online updates, as per the original GAN paper (Goodfellow et al., 2014). However, in the meta-learning approach of Chapter 4 I propose to use forward-mode differentiation to allow for unrolling. The reason for this dichotomy isn't just empirically motivated. In adversarial

learning, we typically care for the last value of the upper level variable. In the case of GANs for instance, we only care for the final state of the generator to produce realistic images. In contrast, in gradient-based HPO we often care for the intermediate hyperparameter values visited during training. For instance, the entire learning rate schedule matters, not just the final learning rate value. As such, online approximations to nested optimization is generally more forgiving in the context of adversarial models (Chapter 3) than meta-learning models (Chapter 4).

2.4 Deep equilibrium models

Deep Equilibrium Models (DEQs) (Bai et al., 2019) are a type of implicit model that naturally lend themselves to a constrained optimization during the training phase. In Chapter 5 I modify the DEQ formulation to cast it as a nested optimization for video landmark detection. In what follows I give a background information on DEQs relevant to understanding Chapter 5.

2.4.1 Implicit models

DEQs are a type of implicit model, which is a model that is trained to represent an implicit functions. A function is called *implicit* in the sense that its output cannot be explicitly written as a function of its input. Implicit models include three main families: neural ODEs (Chen et al., 2018; Dupont et al., 2019), optimization networks (Amos and Kolter, 2017; Djolonga and Krause, 2017; Wang et al., 2019a) and DEQs (Bai et al., 2019; Bai et al., 2020). Implicit functions are best explained with a toy example. Consider the 1D map $F : x \to y$. In traditional ML models like MLPs or ConvNets, we can write this map as a function f of the input and some parameters θ, *e.g.* $y = f(x;\theta) = x^2+3x\theta-1$ if F represents a polynomial. In contrast, $f(x;\theta)$ does not exist for an implicit function, and we instead have an equality condition $g(x, y, \theta) = 0$ relating inputs, parameters and outputs, *e.g.* $y^5 + 16y - 32x^3 + 8\theta^2 = 0$.

During the forward pass, we must solve the $g(x, y, \theta) = 0$ equation for y. This is usually done using a black-box root solver, i.e. a solver whose operations do not need to be tracked by autograd. This can be an ODE solver for Neural ODEs, an optimizer for optimization networks, or a fixed point solver for DEQs. In the backward pass we use gradient computation methods that only input the function $g(x, y, \theta)$ and the final result of the forward pass (no intermediate steps needed as in reverse-mode differentiation). Typically we use the adjoint sensitivity method (Pontryagin et al., 1962) for Neural ODEs, and the implicit function theorem (IFT) (Krantz and Parks, 2003) for optimization networks and DEQs.

Note that every explicit function can be written as an implicit function, but not the other way around. In that sense, the set of implicit functions supersets that of explicit functions, which is one of the reasons they are attractive in the context of designing ML models with high representational power.

Let's consider the type of implicit functions that are commonly implemented. Each of them can be plugged in as a layer F into a conventional neural network pipeline, in the sense that they express a differentiable forward pass:

Neural ODEs. Here the implicit function is an ordinary differential equation relating time, inputs, parameters, and rate of change of input over time. Specifically, a neural ODE takes the form

$$F : \boldsymbol{x}_0 \to \boldsymbol{y} \tag{2.18}$$

$$s.t. \quad \dot{\boldsymbol{x}} = f(\boldsymbol{x}(t), \boldsymbol{\theta}, t) \tag{2.19}$$

$$\text{where } \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \text{ and } \boldsymbol{x}(t_1) = \boldsymbol{y} \tag{2.20}$$

where f is some differentiable function modelled with a neural network, most famously a ResNet. Note that the equality conditions above can be written as $g_i(\boldsymbol{x}_0, \boldsymbol{y}, \boldsymbol{\theta}) = 0$ constraints trivially. In the forward pass, an ODE solver is used to solve for $\boldsymbol{y}$. In the backward pass, the adjoint sensitivity method is used to compute gradients w.r.t. to $\boldsymbol{\theta}$, t_0 and t_1. The main advantage of Neural ODEs is their parameter efficiency and their

support of adaptive compute and adaptive depth. Their main disadvantage is their limitation to learning smooth homeomorphisms as well as their slow speed compared to conventional neural networks. Applications include learning a model on irregularly sampled time series (Rubanova et al., 2019), inferring dynamics of physical systems for control (Zhong et al., 2020), or motion prediction under uncertainty (Yildiz et al., 2019).

Optimization networks. These use the optimum of some function $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$ as the output to F:

$$F : \boldsymbol{x} \rightarrow \boldsymbol{y}^* \tag{2.21}$$

$$s.t. \quad \boldsymbol{y}^* = \arg\min_{\boldsymbol{y}} f(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) \tag{2.22}$$

In the forward pass, we solve for the minimum of f using some closed form solution or some gradient-based optimizer. In the backward pass, we use the fact that $(\partial f / \partial \boldsymbol{y})|_{\boldsymbol{y}^*} = \mathbf{0}$ and the IFT to derive the gradients w.r.t. $\boldsymbol{\theta}$. It is common to add simple constraints on the minimization problem to simplify computation; typically $f(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$ is chosen to be a convex function so that its minimum is unique and cheaper to find. Applications of optimization networks include signal denoising (Amos and Kolter, 2017) and the design of exotic non-linear functions for neural networks (Martins and Astudillo, 2016). Their main limitation comes from their cubic complexity in the number of variables and constraints, and the practical restriction to applications relying on convex optimization problems.

Deep Equilibrium Models. In the case of DEQs, the output is the fixed point $\boldsymbol{y}^*$ of a function $f(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$, which is a value of $\boldsymbol{y}$ for which the input and output of f are equal:

$$F : \boldsymbol{x} \rightarrow \boldsymbol{y}^* \tag{2.23}$$

$$s.t. \quad \boldsymbol{y}^* = f(\boldsymbol{x}, \boldsymbol{y}^*; \boldsymbol{\theta}) \tag{2.24}$$

In the forward pass, we find the root of $g(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = f(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) - \boldsymbol{y}$ using an off-the-shelf root solver. In the backward pass we use the implicit function theorem and the fact that $g(\boldsymbol{x}, \boldsymbol{y}^*; \boldsymbol{\theta}) = \mathbf{0}$ to derive gradients w.r.t. $\boldsymbol{\theta}$. Much like Neural ODEs, DEQs are parameter efficient and naturally support adaptive compute at test time (trading off compute for accuracy by varying the precision with which the root is found). In practice, their main limitations are also comparable to that of Neural ODEs, namely inference speed (due to the cost of root solving), and restrictions to homeomorphisms (the dimension of the input and output to a DEQ layer must be the same). Since DEQ layers can be plugged into any deep learning model, their applications are broad. In practice, they have been particularly successful on optical flow estimation (Bai et al., 2022) and image classification (Bai et al., 2021; Bai et al., 2020), but usually fall short of state-of-the-art conventional architectures.

Implicit functions also have a history in statistical modeling (Diggle and Gratton, 1984), specifically for simulator-based statistical models with intractable likelihood functions (Gutmann and Corander, 2016). These approaches typically don't rely on gradient computation and are less popular compared to the above deep learning models.

2.4.2 How to train your DEQ

Deep equilibrium models are central to Chapter 5 and so a more technical background about them is given in this section.

Forward pass. Consider the process of root solving for a fixed point $\boldsymbol{y}^* = f(\boldsymbol{x}, \boldsymbol{y}^*; \boldsymbol{\theta})$. In the simplest case, f is a contraction mapping and so $f \circ f \circ f \circ \cdots \circ f(\boldsymbol{y}_0)$ converges to a unique fixed point $\boldsymbol{y}^*$ for any initial guess $\boldsymbol{y}_0$. In practice, it is neither tractable or helpful to directly take an infinite number of fixed point iteration steps. Instead, it is common to achieve the same result by leveraging quasi-Newtonian root solvers like Broyden's method (Broyden, 1965) or Anderson acceleration (Anderson, 1965), which find $\boldsymbol{y}^*$ in fewer iterations by solving for the root of $g(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = f(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) - \boldsymbol{y}$. These solvers are "quasi-Newtonian" in the sense that they approximate the inverse Jacobian $J_n^{-1} = \left(\frac{\partial \boldsymbol{f}(\boldsymbol{y}_n)}{\partial \boldsymbol{y}_n}\right)^{-1}$ used in Newtonian solvers by $\hat{J}_n^{-1}$ for each solver iteration n, such

that their update takes the form

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n - \hat{J}_n^{-1} f(\boldsymbol{y}_n) \tag{2.25}$$

Using solvers as opposed to conventional feed-forward operations means that we can model a potentially infinite number of feed-forward operations (albeit all using the same weights) with a finite number of solver steps.

The original deep equilibrium model (Bai et al., 2019) makes no attempt at proving the existence or the uniqueness of $f(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$. Later DEQ variants seek to design f functions that are guaranteed a unique fixed point (Winston and Kolter, 2020; Revay et al., 2020), but this is cumbersome and better performance can be obtained by relying on regularization heuristics that are conducive to convergence, such as weight normalization and variational dropout (Bai et al., 2020).

Backward pass. An important property of DEQs is that they are agnostic to the root solver used. This is because the operations of the root solver aren't tracked by autograd during the forward pass. Indeed, gradients are computed from the implicit function theorem (IFT) and the final $\boldsymbol{y}^*$ state only. To see this, first consider the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^*} \frac{\partial \boldsymbol{y}^*}{\partial \boldsymbol{\theta}} \tag{2.26}$$

where the first term on the RHS is determined by operations outside of the DEQ layer, and is given by conventional reverse-mode differentiation. The difficulty is thus in finding $\partial \boldsymbol{y}^* / \partial \boldsymbol{\theta}$. Explicitly writing out the dependence of $\boldsymbol{y}^*$ on $\boldsymbol{\theta}$ we can write the following:

$$g(\boldsymbol{x}, \boldsymbol{y}^*(\boldsymbol{\theta}); \boldsymbol{\theta}) = \mathbf{0} \tag{2.27}$$

$$\frac{dg}{d\boldsymbol{\theta}} = \mathbf{0} \tag{2.28}$$

$$\frac{\partial g}{\partial \boldsymbol{\theta}} + \frac{\partial g}{\partial \boldsymbol{y}^*} \frac{\partial \boldsymbol{y}^*}{\partial \boldsymbol{\theta}} = \mathbf{0} \tag{2.29}$$

and rearrange terms to get:

$$\frac{\partial \boldsymbol{y}^*}{\partial \boldsymbol{\theta}} = -\left(\frac{\partial g}{\partial \boldsymbol{y}^*}\right)^{-1} \frac{\partial g}{\partial \boldsymbol{\theta}} \tag{2.30}$$

$$\frac{\partial \boldsymbol{y}^*}{\partial \boldsymbol{\theta}} = -\left(\frac{\partial g}{\partial \boldsymbol{y}^*}\right)^{-1} \frac{\partial f}{\partial \boldsymbol{\theta}} \tag{2.31}$$

which can be recognized as an application of the IFT to $g(\boldsymbol{x}, \boldsymbol{y}^*(\boldsymbol{\theta}); \boldsymbol{\theta}) = \mathbf{0}$. Putting Eq. (2.31) and Eq. (2.26) together we obtain:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = -\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^*}\left(\frac{\partial g}{\partial \boldsymbol{y}^*}\right)^{-1} \frac{\partial f}{\partial \boldsymbol{\theta}} \tag{2.32}$$

At first sight, the above gradient computation is intractable due to the inverse Jacobian which cannot be computed or stored in memory for deep learning sized problems. Thankfully, we can compute the above by casting it as the solution to another fixed point problem (to be solved for each backward pass). We first write it as:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \boldsymbol{u}^T \frac{\partial f}{\partial \boldsymbol{\theta}} \tag{2.33}$$

where

$$\boldsymbol{u}^T = -\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^*}\left(\frac{\partial g}{\partial \boldsymbol{y}^*}\right)^{-1} \tag{2.34}$$

$$\boldsymbol{u}^T \left(\frac{\partial g}{\partial \boldsymbol{y}^*}\right) + \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^*} = 0 \tag{2.35}$$

which can be further simplified to:

$$\boldsymbol{u}^T \left(\frac{\partial f}{\partial \boldsymbol{y}^*} - \mathbb{1}\right) + \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^*} = 0 \tag{2.36}$$

$$\boldsymbol{u}^T = \boldsymbol{u}^T \frac{\partial f}{\partial \boldsymbol{y}^*} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^*} \tag{2.37}$$

This is a fixed point problem to be solved for $\boldsymbol{u}^T$. The first term on the RHS is a vector-Jacobian product which doesn't require computing the Jacobian directly, and the second term is provided by reverse-mode differentiation as usual.

2.4.3 DEQ: from constrained to nested optimization

Consider that a DEQ is used to predict a ground truth label $\boldsymbol{y}_{GT}$. By restricting the output of a DEQ layer to be a fixed point, training a DEQ is equivalent to constrained optimization:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{MSE}(\boldsymbol{y}^*(\boldsymbol{\theta}), \boldsymbol{y}_{GT}) \tag{2.38}$$

$$s.t. \;\; g(\boldsymbol{x}, \boldsymbol{y}^*(\boldsymbol{\theta}); \boldsymbol{\theta}) = \mathbf{0} \tag{2.39}$$

In Chapter 5 I extend this optimization problem by adding constraints on which fixed points are used, in the case when many fixed points exist. The first step is to recast the above as a nested optimization:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{MSE}(\boldsymbol{y}^*(\boldsymbol{\theta}), \boldsymbol{y}_{GT}) \tag{2.40}$$

$$s.t. \;\; \boldsymbol{y}^* = \arg\min_{\boldsymbol{y}} \|g(\boldsymbol{x}, \boldsymbol{y}^*(\boldsymbol{\theta}); \boldsymbol{\theta})\|_2^2 \tag{2.41}$$

This isn't exactly equivalent to the problem in Eq. 2.38 – 2.39 because we now support cases when exact fixed points don't exist, in which case an approximate solution $\boldsymbol{y}^* \simeq f(\boldsymbol{x}, \boldsymbol{y}^*; \boldsymbol{\theta})$ can be helpful nonetheless. In the case where several fixed points exist, we can disambiguate the above by adding a regularizer loss term $h(\boldsymbol{x}, \boldsymbol{y}^*)$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{MSE}(\boldsymbol{y}^*(\boldsymbol{\theta}), \boldsymbol{y}_{GT}) \tag{2.42}$$

$$s.t. \;\; \boldsymbol{y}^* = \arg\min_{\boldsymbol{y}} \|g(\boldsymbol{x}, \boldsymbol{y}^*(\boldsymbol{\theta}); \boldsymbol{\theta})\|_2^2 + \lambda h(\boldsymbol{x}, \boldsymbol{y}^*) \tag{2.43}$$

where λ is a hyperparameter scalar. This formulation of DEQs approaches that of optimization networks in Eq. 2.21 – 2.22. The key difference is that when solutions to Eq. (2.43) are approximately keypoints we can still use fast fixed-point solvers like Anderson acceleration (Anderson, 1965) to find them. This is much faster than optimizers and means we're not limited to convex lower level objectives. I build on this strategy in Chapter 5 and demonstrate how fixed point solvers can be used in the context of landmark detection in video.

Chapter 3

Zero-shot Knowledge Transfer via Adversarial Belief Matching

This chapter is about my paper "Zero-shot Knowledge Transfer via Adversarial Belief Matching", which was published in NeurIPS 2019 with spotlight distinction. I start by providing a background section on the application domains relevant to this paper, which is more exhaustive that the background material included in the original publication. Following this, the paper itself is included with minor changes.

3.1 Background

3.1.1 Model compression basics

In its simplest form, model compression inputs a pretrained neural network A, and outputs a network B that is smaller in size, while retaining most of the performance of model A. A smaller size refers to a smaller latency and/or memory cost at inference, which is key to enable many applications like real-time ML on smartphones. It is common to call model A the teacher, and model B the student. Here we give a summary of model compression techniques that are relevant to this thesis, but we refer the reader to the survey from (Menghani, 2021) for an in-depth overview. There are several ways to achieve model compression:

Network pruning. This consists in identifying and excluding the parameters of the teacher network that are the least relevant towards its performance. Unstructured pruning removes individual weights by setting them to zero (LeCun et al., 1990; Han et al., 2015), which leads to sparse architectures that are cheaper to store in memory

and are likely to overfit less. Unfortunately current ML hardware like GPUs struggle to convert sparse weights into performance improvement, and since the main memory cost of modern networks lies in the forward pass activations rather that the weights, the benefits of weight pruning remain very limited. On the other hand, structured pruning (Li et al., 2017a; Molchanov et al., 2017) removes parameter blocks (like whole convolutional filters or layers) and typically provides an improvement in latency and memory that is linear with the amount pruned. In both structured and unstructured pruning, it is common to change the teacher network progressively, by alternating pruning and retraining steps. The parameters to prune are identified by designing a saliency score, which is usually based on second order derivatives of the weights, their magnitude or their momentum.

Network quantization The standard deep learning libraries like Pytorch (Paszke et al., 2019) and Tensorflow (Abadi et al., 2015) rely on 32-bit floating-point values for all operations by default. Quantization refers to lowering the precision of the weights and activations, usually down to 16 or 8 bits. Quantization can be performed on a 32-bit teacher after training it (Vanhoucke et al., 2011), but quantization aware training (Jacob et al., 2017) is also possible, where the quantization range is adapted while training the student to make sure it matches the teacher best. It is increasingly common to discard the teacher altogether, and simply train a network in lower precision from scratch. For instance, Pytorch now supports seamless 16-bit training in a way that matches 32-bit precision in most deep learning applications. This option has recently surged in popularity due to the increase in the number of half precision cores in modern Nvidia GPUs, making 16-bit quantization half cheaper in both memory and latency.

3.1.2 Knowledge distillation

Knowledge distillation can be seen as a third way to do model compression. Here the idea is to use data labels along with the additional outputs of the teacher to train a student. The intuition is that the outputs of the teacher contain useful information about the relationship between different classes. For instance, for the CIFAR-10 dataset

(Krizhevsky, 2009) the car and truck classes are expected to be semantically closer with each other than with the frog class. Knowledge distillation was first proposed by (Buciluǎ et al., 2006) as a way to compress a large ensemble into a single network, and was later made popular by (Ba and Caruana, 2014) and then (Hinton et al., 2015), who proposed smoothing the teacher's probability outputs. Since then, the focus has mostly been on improving distillation efficiency by designing better students (Romero et al., 2015; Crowley et al., 2018), or getting small performance gains with extra loss terms, e.g. an attention transfer loss (AT) (Zagoruyko and Komodakis, 2016a). Knowledge distillation isn't image specific, and has shown to be useful in various ML domains. Most famously, DistillBERT (Sanh et al., 2019) uses a custom distillation loss to compress the natural language model BERT (Devlin et al., 2018) by nearly half while retaining most of its performance.

Let $S(\boldsymbol{x}; \boldsymbol{\theta})$ be the outputs of a student network parameterized by $\boldsymbol{\theta}$. We can use the output predictions of a pretrained teacher network $T(\boldsymbol{x}; \boldsymbol{\omega}^*)$ parameterized by $\boldsymbol{\omega}^*$ to train the student network by using the knowledge distillation loss function:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{KD}} \tag{3.1}$$

where

$$\mathcal{L}_{\text{KD}} = \alpha \mathcal{L}_{\text{CE}}(S(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y})) \; + \; (1 - \alpha) D_{KL}(S(\boldsymbol{x}; \boldsymbol{\theta}) \; || \; T(\boldsymbol{x}; \boldsymbol{\omega}^*)) \tag{3.2}$$

and where α is a hyperparameter. The first term is the standard cross entropy loss term, while the second term encourages the student to match the outputs of the teacher network, which contain more entropy than the ground truth labels. Since the predictions of modern neural networks are often very confident, it is common to soften the predictions of the teacher by increasing the temperature of the softmax that is applied to its logits (Hinton et al., 2015).

Using the output probabilities of the teacher to guide the student into learning a richer representation of the data may be too weak of a signal. Fortunately, it is common for the teacher and student networks to have architectures somewhat similar in structure. For instance, ResNets are made of 4 stages, where each stage has a varying number of layers but always is half the resolution and twice the channel count than the previous stage. This means that a small ResNet-18 student and a large ResNet-152 teacher both share extra structure in their activations, which can be used to distill the teacher better, a process called attention transfer (Zagoruyko and Komodakis, 2016a). Specifically, consider aligning teacher and student activation blocks, written $A_b^{(t)}$ and $A_b^{(s)}$ respectively, where b is the block index $b \in \{1, 2, ..., B\}$. These blocks typically correspond to the last last activations of each stage in the ResNet. The attention transfer procedure is given by:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{AT}} \tag{3.3}$$

where

$$\mathcal{L}_{\text{AT}} = \alpha \mathcal{L}_{\text{CE}}(S(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y})) \; + \; (1-\alpha)\frac{1}{B}\sum_{b}^{B} \left\| \frac{\nu(A_b^{(t)})}{\left\|\nu(A_b^{(t)})\right\|_2} - \frac{\nu(A_b^{(s)})}{\left\|\nu(A_b^{(s)})\right\|_2} \right\|_2 \tag{3.4}$$

where $\nu(\cdot)$ is an aggregation function applied to a given activation block. The original authors suggested using $\nu(A_b) = (1/C_b)\sum_c \boldsymbol{a}_{bc}^2$ where $\boldsymbol{a}_{bc}$ is the cth channel of activation block A_b and C_b the number of channels of block b. The loss function used by the model in this chapter uses both $\mathcal{L}_{\text{KD}}$ and $\mathcal{L}_{\text{AT}}$ as defined above.

3.1.3 Adversarial attacks

Adversarial attacks refers to a type of algorithms aiming to perturb the inputs to a neural network, so as to maximize the model's error. Most often, this perturbation is crafted on test images and is imperceptible to humans, as was first demonstrated in (Szegedy et al., 2014; Goodfellow et al., 2015). Designing new types of adversarial

attacks (Kurakin et al., 2017; Papernot et al., 2016) and defenses (Lyu et al., 2015; Tramèr et al., 2018; Papernot and McDaniel, 2017) has been a major focus of the Deep Learning community in the last decade. A full review of adversarial attacks is beyond the scope of this thesis, and can be found in (Chakraborty et al., 2021).

In the following paper, we are interested in targeted white-box attacks. In this case we have full access to the model under attack, and the aim is to perturb some input image by a small amount such that it changes model predictions towards another class. Our aim in the following paper is to monitor the characteristics of a trained neural network as some input crosses its decision boundaries. As such, we consider the most basic form of adversarial attack, using a few steps of gradient descent. Consider inputs $\boldsymbol{x}$, class predictions from a model $\hat{\boldsymbol{y}}$, and target class $\boldsymbol{y} \neq \hat{\boldsymbol{y}}$. We can simply update $\boldsymbol{x} \leftarrow \boldsymbol{x} - \xi \partial \mathcal{L}_{CE}(\hat{\boldsymbol{y}}, \boldsymbol{y})/\partial \boldsymbol{x}$ for K times in a row, for a small learning rate ξ. Here the size of the total change in $\boldsymbol{x}$ is kept small by using a low value of K and ξ. In practice this is enough to move model predictions from $\hat{\boldsymbol{y}}$ to $\boldsymbol{y}$.

3.2 Introduction

Large neural networks are ubiquitous in modern deep learning applications, including computer vision (He et al., 2016), speech recognition (Oord et al., 2016) and natural language understanding (Devlin et al., 2018). While their size allows learning from big datasets, it is a limitation for users without the appropriate hardware, or for internet-of-things applications. As such, the deep learning community has seen a focus on model compression techniques, including knowledge distillation (Ba and Caruana, 2014; Hinton et al., 2015), network pruning (Li et al., 2017a; Han et al., 2016) and quantization (Gupta et al., 2015; Hubara et al., 2016).

These methods typically rely on labeled data drawn from the training distribution of the model that needs compressed. Distillation does so by construction, and pruning or quantization need to fine-tune networks on training data to get good performance. We argue that this is a strong limitation because pretrained models are often released

without training data, an increasingly common trend that has been grounds for controversy in the deep learning community (Radford et al., 2019). We identify four main reasons why datasets aren't released: privacy, property, size, and transience. Respective examples include Facebook's DeepFace network trained on four million confidential user images (Taigman et al., 2014), Google's Neural Machine Translation System trained on internal datasets (Wu et al., 2016) and regarded as intellectual property, the JFT-300 dataset which contains 300 million images across more than 18k classes (Sun et al., 2017), and finally the field of policy distillation in reinforcement learning (Rusu et al., 2016), where one requires observations from the original training environment which may not exist anymore. One could argue that missing datasets can be emulated with proxy data for distillation, but in practice that is problematic for two reasons. First, there is a correlation between data that is not publicly released and data that is hard to emulate, such as medical datasets of various diseases (Burton et al., 2015), or datasets containing several thousand classes like JFT. Secondly, it has been shown in the semi supervised setting that out-of-distribution samples can cause significant performance drop when used for training (Oliver et al., 2018).

As such, we believe that a focus on zero-shot knowledge transfer is justified, and our paper makes the following contributions: 1) we propose a novel adversarial algorithm that distills a large teacher into a smaller student without any data or metadata, 2) we show its effectiveness on two common datasets, and 3) we define a measure of belief match between two networks in the vicinity of one's decision boundaries, and demonstrate that our zero-shot student closely matches its teacher.

3.3 Related work

Inducing point methods and dataset distillation. Inducing point methods (Snelson and Ghahramani, 2005) were introduced to make Gaussian Processes (GP) more tractable. The idea is to choose a set of inducing points that is smaller than the input dataset, in order to reduce inference cost. While early techniques used a subset of the training

data as inducing points (Candela and Rasmussen, 2005), creating pseudo data using variational techniques was later shown more efficient (Titsias, 2009). Dataset distillation is related to this idea, and uses a bi-level optimization scheme to learn a small subset of pseudo images, such that training on those images yields representations that generalize well to real data (Wang et al., 2018). The main difference with these methods and ours is that we do not need the training data to generate pseudo data, but we rely on a pretrained teacher instead.

Privacy attacks. There are a few approaches to making privacy attacks that are related to our method. In model extraction (Tramèr et al., 2016) we have access to the probability predictions of a black-box model and the aim is to extract an equivalent model. The limitation of black-box access makes this task harder and often limited to simple datasets. In model inversion (Fredrikson et al., 2014; Fredrikson et al., 2015) we have white-box access to a pretrained model, and we wish to recreate training images from the weights alone. This is a different aim from that of our task, because we wish to produce images that are relevant for training regardless of whether or not they resemble the training data.

Zero-shot learning. In zero-shot learning (Larochelle et al., 2008; Socher et al., 2013), we are typically given training images with labels and some additional intermediate semantic representation T, such as textual descriptions. The task is then to classify images at test time that are represented in T but whose classes were never observed during training. In our model, the additional intermediate information can be considered to be the teacher, but none of the classes are formally observed during training because no samples from the training set are used.

Zero and few-shot distillation. More recently, the relationship between data quantity and distillation performance has started being addressed. In the few-shot setting, (Li et al., 2018) obtain a student by pruning a teacher, and align both networks with 1x1 convolutions using a few samples. (Kimura et al., 2018) distill a GP to a neural network by adversarially learning pseudo data. In their setting however, the teacher itself has access to little data and is added to guide the student. Concurrent to our work, (Ahn

et al., 2019) formulated knowledge transfer as variational information distillation (VID) and to the best of our knowledge obtained state-of-the-art in few-shot performance. We show that our method gets better performance than they do even when they use an extra 100 images per class on CIFAR-10. The zero-shot setting has been a lot more challenging: most methods in the literature use some type of metadata and are limited to simple datasets like MNIST. (Lopes et al., 2017) showed that training images could be reconstructed from a record of the teacher's training activations, but in practice releasing training activations instead of data is unlikely. Concurrent to our work, (Nayak et al., 2019) synthesize pseudo data from the weights of the teacher alone and use it to train a student in zero-shot. Their model is not trained end-to-end, and on CIFAR-10 we obtain a performance 17% higher than they do for a comparable sized teacher and student.

3.4 Zero-shot knowledge transfer

3.4.1 Algorithm

Let $T(\boldsymbol{x};\boldsymbol{\omega}^*)$ be a pretrained teacher network with weights $\boldsymbol{\omega}^*$, which maps some input image $\boldsymbol{x}$ to a probability vector $\boldsymbol{t}$. Similarly, $S(\boldsymbol{x};\theta)$ is a student network parameterized by weights $\boldsymbol{\theta}$, which outputs probability vector $\boldsymbol{s}$. Let $G(\boldsymbol{z};\boldsymbol{\phi})$ be a generator parameterized by weights $\boldsymbol{\phi}$, which produces pseudo data $\boldsymbol{x}_p$ from a noise vector $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The main loss function we use is the forward Kullback–Leibler (KL) divergence between the outputs of the teacher and student networks on pseudo data, namely $D_{KL}(T(\boldsymbol{x}_p) \;||\; S(\boldsymbol{x}_p)) = \sum_i \boldsymbol{t}_p^{(i)} \log(\boldsymbol{t}_p^{(i)}/\boldsymbol{s}_p^{(i)})$ where i corresponds to image classes.

Our zero-shot training algorithm is described in Algorithm 1. For N iterations we sample one batch of $\boldsymbol{z}$, and take n_G gradient updates on the generator with learning rate η, such that it produces pseudo samples $\boldsymbol{x}_p$ that maximize $D_{KL}(T(\boldsymbol{x}_p) \;||\; S(\boldsymbol{x}_p))$. We then take n_S gradient steps on the student with $\boldsymbol{x}_p$ fixed, such that it matches the

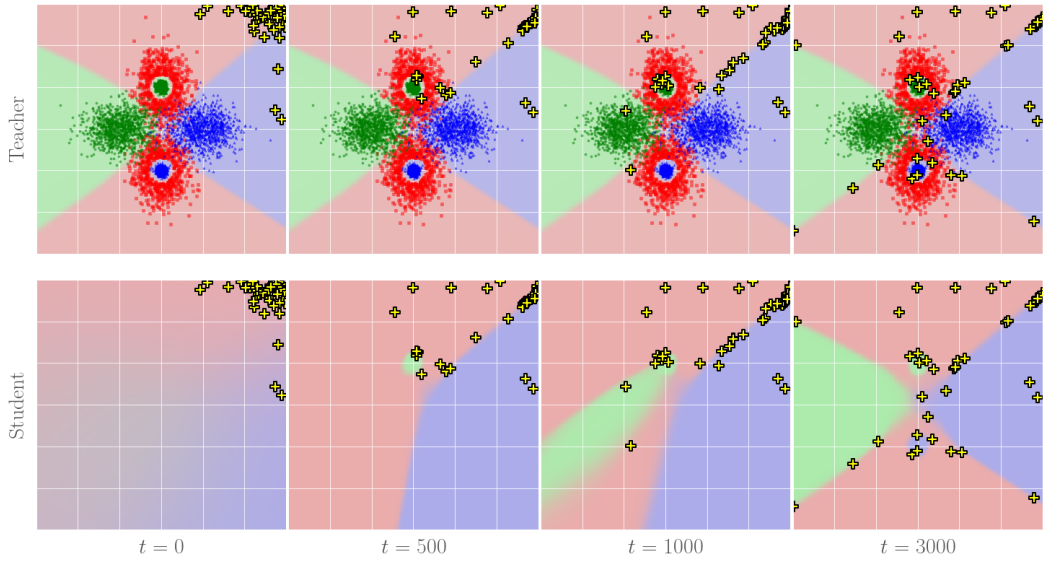

Figure 3.1: A simplified version of our method on a three-class toy problem. The teacher and student decision boundaries are shown in the top and bottom rows respectively. The knowledge transfer unfolds left to right and never uses the real data points, which are shown for visualization purposes. We initialize random pseudo points (yellow/black crosses) away from the data manifold, and train them to maximize the KL divergence between the student and teacher. At the same time we train the student to achieve the opposite. Note how pseudo points use the decision boundaries as channels to explore the input space, but can also explore regions away from them such as the two isolated green and blue pockets. After a few steps the student and teacher decision boundaries are indistinguishable.

teacher's predictions on $\boldsymbol{x}_p$. The idea of taking several steps on the two adversaries has proven effective in balancing their relative strengths. In practice we use $n_S > n_G$, which gives more time to the student to match the teacher on $\boldsymbol{x}_p$, and encourages the generator to explore other regions of the input space at the next iteration.

3.4.2 Extra loss functions

Using the forward KL divergence as the main loss function encourages the student to spread its density over the input space and gives non-zero class probabilities for all images. This high student entropy is a vital component to our method since it makes it hard for the generator to fool the student too easily; we observe significant drops in student test accuracy when using the reverse KL divergence or Jensen–Shannon divergence. In practice, many student-teacher pairs have similar block structures, and

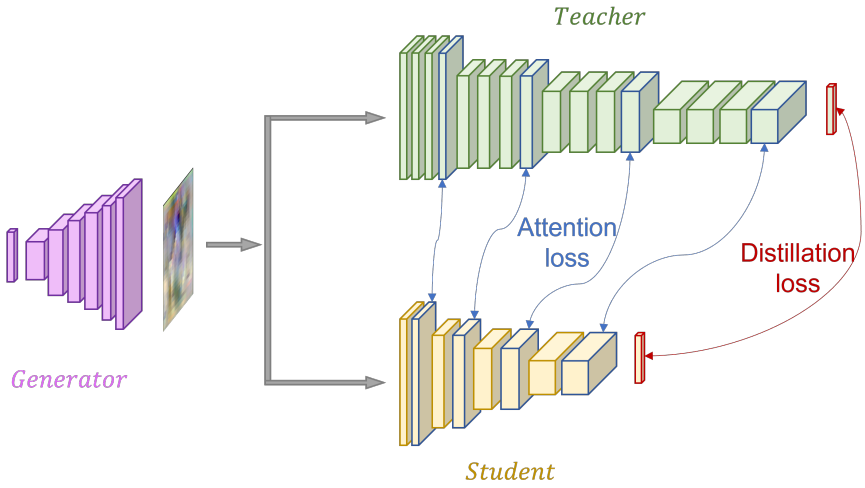

Figure 3.2: Illustration of the proposed model, whereby a large teacher is distilled to a student without using any training data. The generator is trained to produce pseudo images that maximize the distillation loss while the student is trained to minimize both the distillation loss and the attention loss.

so we can add an attention term to the student loss:

$$\mathcal{L}_S = D_{KL}(T(\boldsymbol{x}_p) \,||\, S(\boldsymbol{x}_p)) + \beta \sum_{l}^{N_L} \left\| \frac{f(A_l^{(t)})}{\left\| f(A_l^{(t)}) \right\|_2} - \frac{f(A_l^{(s)})}{\left\| f(A_l^{(s)}) \right\|_2} \right\|_2 \tag{3.5}$$

where β is a hyperparameter. We take the sum over some subset of N_L layers. Here, $A_l^{(t)}$ and $A_l^{(s)}$ are the teacher and student activation blocks for layer l , both made up of N_{A_l} channels. If we denote by $\boldsymbol{a}_{lc}$ the cth channel of activation block A_l, then we use the spatial attention map $f(A_l) = (1/N_{A_l}) \sum_c \boldsymbol{a}_{lc}^2$ as suggested by the authors of AT (Zagoruyko and Komodakis, 2016a). We don't use attention for the generator loss $\mathcal{L}_G$ because it makes it too easy to fool the student. We illustrate our training pipeline in Fig. 3.2.

Algorithm 1: Zero-shot KT (Sec. 3.4.1)

pretrain: $T(\cdot)$
initialize: $G(\cdot;\boldsymbol{\phi})$
initialize: $S(\cdot;\theta)$

for $1, 2, ..., N$ **do**

$\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

for $1, 2, ..., n_G$ **do**

$\boldsymbol{x}_p \leftarrow G(\boldsymbol{z};\boldsymbol{\phi})$

$\mathcal{L}_G \leftarrow -D_{KL}(T(\boldsymbol{x}_p) \mid\mid S(\boldsymbol{x}_p))$

$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \eta\frac{\partial L_G}{\partial \boldsymbol{\phi}}$

end

for $1, 2, ..., n_S$ **do**

$\mathcal{L}_S \leftarrow D_{KL}(T(\boldsymbol{x}_p) \mid\mid S(\boldsymbol{x}_p)) + \beta \sum_l^{N_L} \left\| \frac{f(A_l^{(t)})}{\left\|f(A_l^{(t)})\right\|_2} - \frac{f(A_l^{(s)})}{\left\|f(A_l^{(s)})\right\|_2} \right\|_2$

$\theta \leftarrow \theta - \eta\frac{\partial L_S}{\partial \theta}$

end

decay η

end

Many other loss terms were investigated but did not help performance, including sample diversity, sample consistency, and teacher or student entropy (see Appendix 1). These losses seek to promote properties of the generator that already occur in the plain model described above. This is an important difference with competing models such as that of (Kimura et al., 2018) where authors must include hand designed losses like carbon copy memory replay (which freezes some pseudo samples in time), or fidelity (Dehghani et al., 2017).

3.4.3 Toy experiment

The dynamics of our algorithm is illustrated in Fig. 3.1, where we use two layer MLPs for both the teacher and student, and learn the pseudo points directly (no generator). These are initialized away from the real data manifold, because we assume that no information about the dataset is known in practice. During training, pseudo points can be seen to explore the input space, typically running along decision boundaries where the student is most likely to match the teacher poorly. At the same time, the student is trained to match the teacher on the pseudo points, and so they must keep changing locations. When the decision boundaries between student and teacher are well aligned, some pseudo points will naturally depart from them and search for new high teacher mismatch regions, which allows disconnected decision boundaries to be explored as well.

3.4.4 Potential conceptual concerns

Here we address a number of potential conceptual concerns when dealing with the application of this approach to higher dimensional input spaces.

Focus. The first potential concern is that $G(\boldsymbol{z}; \boldsymbol{\phi})$ is not constrained to produce real or bounded images, and so it may prefer to explore the larger region of space where the teacher has not been trained. Assuming that the teacher's outputs outside of real images is irrelevant for the classification task, the student would never receive useful signal. In practice we observe that this assumption does not hold. On MNIST for instance, preliminary experiments showed that a simple student can achieve 90% test accuracy when trained to match a teacher on random noise. This is purely due to the simplicity of the MNIST dataset and corresponding teacher decision boundaries. Indeed, the distribution for the classes of random noise according to the teacher is close to uniform. On more diverse datasets like CIFAR-10 however, we observe that uniform noise is mostly classified as the same class across networks (see Appendix 2). This suggests that the density of decision boundaries is smaller outside of the real image manifold, and so $G(\boldsymbol{z}; \boldsymbol{\phi})$ may struggle to fool the student in that space due to the teacher being too predictable.

Algorithm 2: Compute transition curves of networks A and B, when stepping across decision boundaries of network A (Sec. 3.5.4)

pretrain: net_A
pretrain: net_B

for $\boldsymbol{x} \in \boldsymbol{X}_{test}$ **do**
  $i_A \equiv$ class of $\boldsymbol{x}$ according to net_A
  $i_B \equiv$ class of $\boldsymbol{x}$ according to net_B
  if $i_A = i_B = i$ **then**
    $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$
    for $j \neq i$ **do**
      $\boldsymbol{x}_{adv} \leftarrow \boldsymbol{x}_0$
      for $1, 2, ..., K$ **do**
        $\boldsymbol{y}_A, \boldsymbol{y}_B \leftarrow net_A(\boldsymbol{x}_{adv}), net_B(\boldsymbol{x}_{adv})$
        $\boldsymbol{x}_{adv} \leftarrow \boldsymbol{x}_{adv} - \xi \frac{\partial \mathcal{L}_{CE}(\boldsymbol{y}_A, j)}{\partial \boldsymbol{x}_{adv}}$
        save: $\boldsymbol{y}_A, \boldsymbol{y}_B$
      end
    end
  end
end

Adversaries. Another potential concern is that $G(\boldsymbol{z}; \boldsymbol{\phi})$ could simply iterate over adversarial examples, which in this context corresponds to images that are all very similar in pixel space and yet are classified differently by the teacher. Here we refer the reader to recent work by (Ilyas et al., 2019), who isolate adversarial features and show that they are enough to learn classifiers that generalize well to real data. The bottom line is that non-robust features, in a human sense, still contain most of the task-relevant knowledge. Therefore, this suggests that our generator can produce adversarial examples (in practice, we observe that it does) while still feeding useful signal to the student.

3.5 Experiments

In the few-shot setting, researchers have typically relied on validation data to tune hyperparameters. Since our method is zero-shot, assuming that validation data exists is problematic. In order to demonstrate zero-shot KT we thus find coarse hyperparameters in one setting (CIFAR-10, WRN-40-2 teacher, WRN-16-1 student) and use the same parameters for all other experiments of this paper. In practice, we find that this makes the other experiments only slightly sub optimal, because our method is very robust to hyperparameters and dataset change: in fact, we find that halving or doubling most of our hyperparameters has no effect on student performance. For each experiment we run three seeds and report the mean with one standard deviation. When applicable, seeds cover networks initialization, order of training images, and the subset of M images per class in few-shot.

3.5.1 CIFAR-10 and SVHN

We focus our experiments on two common datasets, SVHN (Netzer et al., 2011) and CIFAR-10 (Krizhevsky, 2009). SVHN contains over 73k training images of 10 digits taken from house numbers in Google Street images. It is interesting for our task because most images contain several digits, the ground truth being the most central one, and so ambiguous images are easily generated. CIFAR-10 contains 50k training images across 10 classes, and is substantially more diverse than SVHN, which makes the predictions of the teacher harder to match. For both datasets we use WideResNet (WRN) architectures (Zagoruyko and Komodakis, 2016b) since they are ubiquitous in the distillation literature and easily allow changing the depth and parameter count.

Our distillation results are shown in Fig. 3.3 for a WRN-40-2 teacher and WRN-16-1 student, when using $\mathcal{L}_S$ as defined in Eq. (3.5). We include the few-shot performance of our method as a comparison, by naively finetuning our zero-shot model with M samples per class. As baselines we show the student performance when trained from scratch (no teacher supervision), and the student performance when trained with both knowledge distillation and attention transfer, since that was observed to be better than

either techniques alone. We also plot the equivalent result of VID (Ahn et al., 2019); to the best of our knowledge, they are the state-of-the-art in the few-shot setting at the time of writing. On CIFAR-10, we obtain a test accuracy of $83.69 \pm 0.58\%$ if we use the student loss described in Eq. (3.5), which is 2% better than VID's performance when it uses an extra $M = 100$ images per class. By finetuning our model with $M = 100$ our accuracy increases to $85.91 \pm 0.24\%$, pushing the previous few-shot state-of-the-art by more than 4%. If we do not use attention ($\beta = 0$) we observe an average drop of 2% across all the architectures in Tab. 3.1.

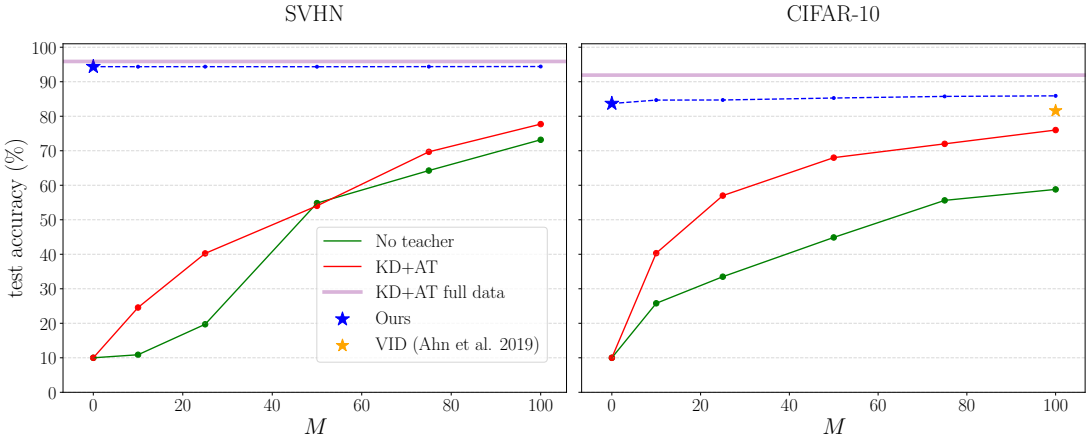

Figure 3.3: Performance of our model for a WRN-40-2 teacher and WRN-16-1 student, on the SVHN and CIFAR-10 datasets, for M images per class. We compare this to the student when trained from scratch (no teacher), the student when trained with knowledge distillation and attention transfer from the teacher (KD+AT), and the performance of (Ahn et al., 2019) (81.59% for $M = 100$). Our method reaches $83.69 \pm 0.58\%$ without using any real data, and increases to $85.91 \pm 0.24\%$ when finetuned with $M = 100$ images per class.

Using all the same settings on SVHN yields a test accuracy of $94.06 \pm 0.27\%$. This is quite close to $95.88 \pm 0.15\%$, the accuracy obtained when using the full 73k images during KD+AT distillation, even though the hyperparameters and generator architecture were tuned on CIFAR-10. This shows that our model can be used on new datasets without needing a hyperparameter search every time, which is desirable for zero-shot tasks where validation data may not be available. If hyperparameters are tuned on SVHN specifically, our zero-shot performance is on par with full data distillation.

Implementation details. For the zero-shot experiments, we choose the number of iterations N to match the one used when training the teachers on SVHN and CIFAR-10 from scratch, namely 50k and 80k respectively. For each iteration we set $n_G = 1$ and $n_S = 10$. We use a generic generator with only three convolutional layers, and our input noise $\boldsymbol{z}$ has 100 dimensions. We use Adam (Kingma and Ba, 2015) with cosine annealing, with an initial learning rate of 2×10^{-3}. We set $\beta = 250$ unless otherwise stated. For our baselines, we choose the same settings used to train the teacher and student in the literature, namely SGD with momentum 0.9, and weight decay 5×10^{-4}. We scale the number of epochs such that the number of iterations is the same for all M. The initial learning rate is set to 0.1 and is divided by 5 at $30\%, 60\%$, and 80% of the run.

Table 3.1: Zero shot performance on various WRN teacher and student pairs for CIFAR-10. Our zero-shot technique ($M = 0$) is on par with KD+AT distillation when it's performed with $M = 200$ images per class. Teacher scratch and student scratch are trained with $M = 5000$. We report mean and standard deviation over 3 seeds.

Teacher (# params)	Student (# params)	Teacher scratch	Student scratch	KD+AT $M = 200$	Ours $M = 0$
WRN-16-2 (0.7M)	WRN-16-1 (0.2M)	93.97 ± 0.11	91.04 ± 0.04	**84.54 ± 0.21**	82.44 ± 0.21
WRN-40-1 (0.6M)	WRN-16-1 (0.2M)	93.18 ± 0.08	91.04 ± 0.04	**81.71 ± 0.25**	79.93 ± 1.11
WRN-40-2 (2.2M)	WRN-16-1 (0.2M)	94.73 ± 0.02	91.04 ± 0.04	81.25 ± 0.67	**83.69 ± 0.58**
WRN-40-1 (0.6M)	WRN-16-2 (0.7M)	93.18 ± 0.08	93.97 ± 0.11	85.74 ± 0.47	**86.60 ± 0.56**
WRN-40-2 (2.2M)	WRN-16-2 (0.7M)	94.73 ± 0.02	93.97 ± 0.11	86.39 ± 0.33	**89.71 ± 0.10**
WRN-40-2 (2.2M)	WRN-40-1 (0.6M)	94.73 ± 0.02	93.18 ± 0.08	**87.35 ± 0.12**	86.60 ± 1.79

3.5.2 Architecture dependence

While our model is robust to the choice of hyperparameters and generator, we observe that some teacher-student pairs tend to work better than others, as is the case for few-shot distillation. We compare our zero-shot performance with $M = 200$ KD+AT distillation across a range of network depths and widths. The results are shown in Tab. 3.1. The specific factors that make for a good match between teacher and student are to be explored in future work. In zero-shot, deep students with more parameters don't necessarily help: the WRN-40-2 teacher distills 3.1% better to WRN-16-2 than to WRN-40-1, even though WRN-16-2 has less than half the number of layers, and

a similar parameter count than WRN-40-1. Furthermore, the pairs that are strongest for few-shot knowledge distillation are not the same as for zero-shot. More modern distillation experiments usually use a teacher whose structure is very different to the student, in the hope to teach patterns in the data that are specifically hard for the student to learn. For example, early vision transformer models did not use convolutions and therefore relied on large convolutional teachers (Touvron et al., 2021).

Finally, note that concurrent work by (Nayak et al., 2019) obtains 69.56% for $M = 0$ on CIFAR-10, despite using a hand-designed teacher/student pair that has more parameters than the WRN-40-1/WRN-16-2 pair we use. Our method thus yields a 17% improvement, but this is in part attributed to the difference of efficiency in the architectures chosen.

3.5.3 Nature of the pseudo data

Samples from $G(\boldsymbol{z}; \boldsymbol{\phi})$ during training are shown in Fig. 3.4. We notice that early in training the samples look like coarse textures, and are reasonably diverse. Textures have long been understood to have a particular value in training neural networks, and have recently been shown to be more informative than shapes (Geirhos et al., 2018). After about 10% of the training run, most images produced by $G(\boldsymbol{z}; \boldsymbol{\phi})$ look like high frequency patterns that have little meaning to humans.

During training, the average probability of the class predicted by the teacher is about 0.8. On the other hand, the most likely class according to student has an average probability of around 0.3. These confidence levels suggest that the generator focuses on pseudo data that is close to the boundary decisions of the student, or uses them as channels (local optima) to more efficiently search for regions where the student decision boundaries differ from those of the teacher, which is what we observed in the toy experiment of Fig. 3.1. It also suggests that the teacher wouldn't be robust to adversarial attacks outside of the data manifold, namely noisy images can easily be classified confidently as an actual class. Finally, we also observe that the classes of

pseudo samples are close to uniformly distributed during training, for both the teacher and student. Again this is not surprising: the generator seeks to make the teacher less predictable on the pseudo data in order to fool the student, and spreading its mass across all the classes available is the optimal solution.

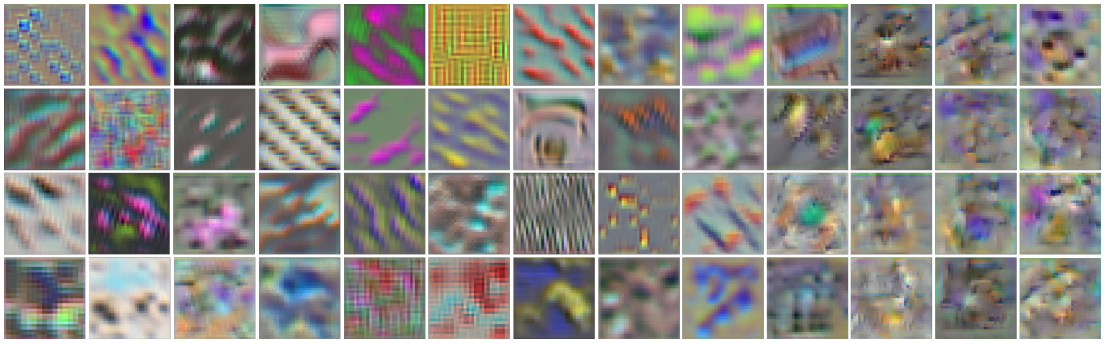

Figure 3.4: Pseudo images sampled from the generator across several seeds and hyperparameters. As the training progresses (left to right), pseudo data goes from coarse and diverse textures to complex high frequency patterns. Note that time is not to scale and most images look like the last four columns.

3.5.4 Measuring belief match near decision boundaries

Our understanding of the adversarial dynamics at play suggests that the student is implicitly trained to match the teacher's predictions close to decision boundaries. To gain deeper insight into our method, we would like to verify that this is indeed the case, in particular for decision boundaries near real images. Let $\mathcal{L}_{CE}$ be the cross entropy loss. In Algorithm 2, we propose a way to probe the difference between the beliefs of network A and B near the decision boundaries of A. First, we sample a real image $\boldsymbol{x}$ from the test set $\boldsymbol{X}_{test}$ such that network A and B both give the same class prediction i. Then, for each class $j \neq i$ we update $\boldsymbol{x}$ by taking K adversarial steps on network A, with learning rate ξ, to go from class i to class j. The probability p_i^A of $\boldsymbol{x}$ belonging to class i according to network A quickly reduces, with a concurrent increase in p_j^A. During this process, we also record p_j^B, the probability that $\boldsymbol{x}$ belongs to class j according to network B, and can compare p_j^A and p_j^B. In essence, we are asking the following question: *as we perturb $\boldsymbol{x}$ to move from class i to j according to network A, to what degree do we also move from class i to j according to network B?*

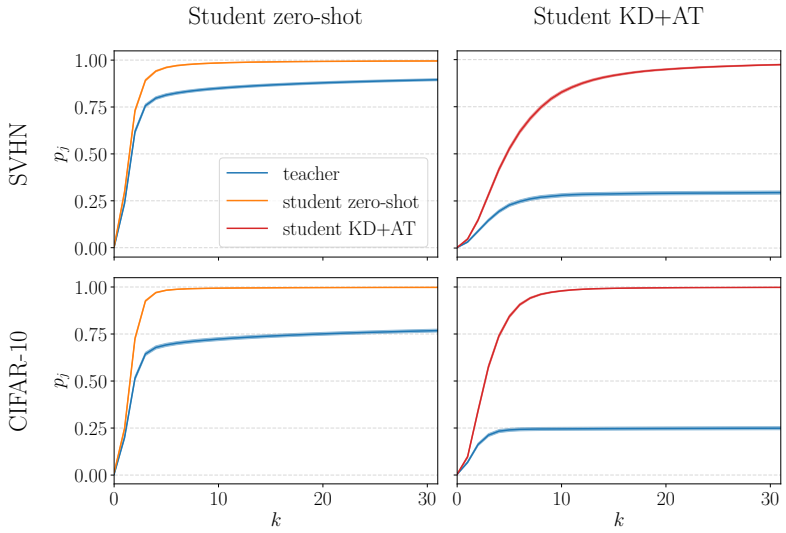

Figure 3.5: Average transition curves over 9 classes and 1000 images for both SVHN and CIFAR-10. LEFT, taking adversarial steps w.r.t. the zero shot student model, and RIGHT, w.r.t. the normal distilled student model. We note that the average of p_j is much more similar between teacher and student when targeting the zero-shot boundary than when targeting the boundary of the student learnt via normal KD+AT distillation. Line thickness shows ± 2 times the standard error of the mean.

We refer to p_j curves as transition curves. For a dataset of C classes, we obtain $C-1$ transition curves for each image $\boldsymbol{x} \in \boldsymbol{X}_{test}$, and for each network A and B. We show the average transition curves in Fig. 3.5, in the case where network B is the teacher, and network A is either our zero-shot student or a standard student distilled with KD+AT. We observe that, on average, updating images to move from class i to class j on our zero-shot student also corresponds to moving from class i to class j according to the teacher. This is true to a much lesser extent for a student distilled from the teacher with KD+AT, which we observed to have flat $p_j = 0$ curves for several images. This is particularly surprising because the KD+AT student was trained on real data, and the transition curves are also calculated for real data.

We can more explicitly quantify the belief match between networks A and B as we take steps to cross the decision boundaries of network A. We define the Mean Transition Error (MTE) as the absolute probability difference between p_j^A and p_j^B, averaged over K steps, N_{test} test images and $C-1$ classes:

$$\text{MTE}(net_A, net_B) = \frac{1}{N_{test}} \sum_{n}^{N_{test}} \frac{1}{C-1} \sum_{c}^{C-1} \frac{1}{K} \sum_{k}^{K} \left| p_j^A - p_j^B \right| \tag{3.6}$$

The mean transition errors are reported in Tab. 3.2. Our zero-shot student has much lower transition errors, with an average of only 0.09 probability disparity with the teacher on SVHN as steps are taken from class i to j on the student. This is inline with the observations made in Fig. 3.5. Note that we used the values $K = 100$ and $\xi = 1$ since they gave enough time for most transition curves to converge in practice. Other values of K and ξ give the same trend but different MTE magnitudes, and must be reported clearly when using this metric.

Table 3.2: Mean Transition errors (MTE) for SVHN and CIFAR-10, between our zero-shot student and the teacher, and between a student distilled with KD+AT and the teacher. Our student matches the transition curves of the teacher to a much greater extent on both datasets.

	Zero-shot (Ours)	KD+AT
SVHN	0.09	0.64
CIFAR-10	0.22	0.68

3.6 Conclusion

In this work we demonstrate that zero-shot knowledge transfer can be achieved in a simple adversarial fashion, by training a generator to produce images where the student does not match the teacher yet, and training that student to match the teacher at the same time. On simple datasets like SVHN, even when the training set is large, we obtain students whose performance is close to distillation with the full training set. On more

diverse datasets like CIFAR-10, we obtain compelling zero and few-shot distillation results, which significantly improve on the previous state-of-the art. We hope that this work will pave the way for more data-free knowledge transfer techniques, as private datasets likely become increasingly common in the future.

3.7 Appendices

Appendix A: Extra loss terms

Here we describe a number of loss terms that have been tried with our method in order to encourage some behaviour from the generator, but have all resulted in a decrease of performance from the student. This happens despite finding the optimal scaling for each loss term, denoted here by γ. In general, we believe that this is due to the generator already achieving the desired behaviours due to the nature of the adversarial dynamics, and so extra losses simply create an imbalance between the two adversaries. Extra loss terms added to $\mathcal{L}_G$ relate to:

1. **The entropy of the teacher**: $\mathcal{L}_G \mathrel{+}= \gamma \times \left(-\sum_i \boldsymbol{t}_p^{(i)} \log \boldsymbol{t}_p^{(i)}\right)$ where $\boldsymbol{t}$ are the teacher's probability outputs on pseudo data $\boldsymbol{x}_p$, and i is the class index. When positive, this encourages the generator to search regions of the input space where the teacher is confident, which could correlate with regions close to the real data manifold.
2. **The entropy of the student**: $\mathcal{L}_G \mathrel{+}= \gamma \times \left(-\sum_i \boldsymbol{s}_p^{(i)} \log \boldsymbol{s}_p^{(i)}\right)$. This encourages the generator to take more risks and look for images that the student is confidently wrong about.
3. **The consistency of the images generated**: $\mathcal{L}_G \mathrel{+}= \gamma \times D_{KL}(T(\boldsymbol{x}_p) \;||\; T(A(\boldsymbol{x}_p)))$ where A is some augmentation operation, such as Gaussian noise or Gaussian blurring. Here the idea is to constrain the generator to search images for which being augmented does not change the output of the teacher. Again this is an attempt to drive the search closer to real data and away from adversarial images.

4. **The diversity of the images generated**: $\mathcal{L}_G \mathrel{+}= -\gamma \times \left\|\boldsymbol{\nu}\boldsymbol{\nu}^T\right\|$, where $\boldsymbol{\nu} \in \mathbb{R}^{N \times D}$ corresponds to the representation of size D for a batch of N images in the penultimate layer of the teacher. Here the loss encourages each batch to be diverse in the space spanned by the teacher's last layer. So at any one time, the generator is penalized if all of its samples look too similar according to the teacher.

Appendix B: Network predictions on noise

On CIFAR-10 we observe that the volume each class occupies in the input space of common neural networks does not seem to be equal. Here, we produce uniform noise by sampling each pixel $x_i \sim U(0, 255)$ discretely, and normalizing the resulting images by the mean and standard deviations of CIFAR-10, as used during training time. The distribution of the predictions made by common neural networks (pretrained on CIFAR-10) are shown in Fig. 3.6. The predictions are mostly birds or frogs, which suggests that decision boundaries have a higher density close to the real images. Another way to reason about this is that adding uniform noise to a real image is much more likely to change its class than adding uniform noise to uniform noise.

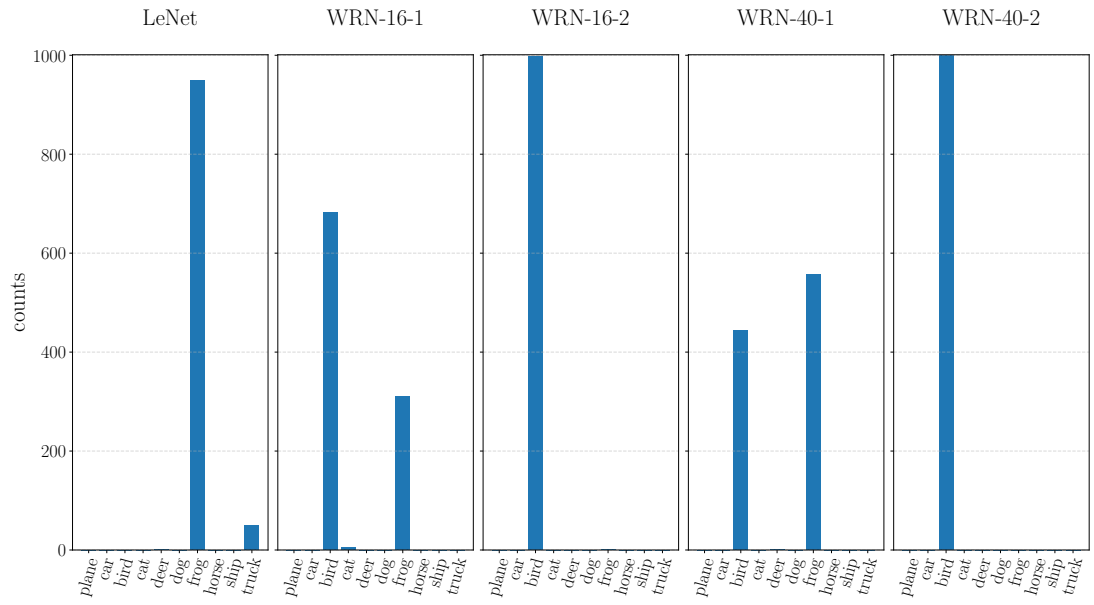

Figure 3.6: Distribution of predictions across different architectures when given 1000 images of uniform noise. Interestingly, the predictions are largely focused on two classes.

Chapter 4

Gradient-based Hyperparameter Optimization Over Long Horizons

This chapter is about my paper "Gradient-based Hyperparameter Optimization Over Long Horizons", which was published in NeurIPS 2021. I start by providing a background section on the application domains relevant to this paper, which is more exhaustive that the background material included in the original publication. Following this, the paper itself is included with minor changes.

4.1 Background

4.1.1 Traditional hyperparameter optimization

Traditionally, hyperparameters refer to parameters that cannot be updated during training, and are instead set by the user for a given model before training. Examples include architecture design parameters like width and depth of a neural network, data augmentation parameters like how much image blurring to include, or optimizer parameters like learning rate and weight decay values. The field of hyperparameter optimization (HPO) aims to find hyperparameters that maximize a performance metric (*e.g.* validation accuracy) automatically, *i.e.* without the need for human expertise and intervention. The goal is to make model design and debugging hyperparameters faster, but also to make comparison between different models more fair in the deep learning

community, since one model that receives more manual hyperparameter tuning can look artificially better than another. We refer to the performance metric to be maximized as the *target function*, which inputs a hyperparameter configuration and outputs a scalar metric.

HPO has long been of focus of the machine learning community, with origins dating back to the 1990s (King et al., 1995; Kohavi and John, 1995). The main challenge in HPO is the size and complexity of the search space for hyperparameters. Indeed, if we were to naively try every combination of plausible hyperparameter values for several hyperparameters, a procedure known as *grid search*, the number of settings to evaluate grows exponentially with the dimensionality of the configuration space. Additionally, the range of acceptable values for each hyperparameter can be a mix of continuous, categorical and conditional hyperparameters. Finally, since the advent of Deep Learning, an additional challenge has emerged: the cost to evaluate the target function can be very expensive. A central theme in HPO is to effectively trade off exploration and exploitation of this target function.

There are various ways to perform modern HPO, including Bayesian optimization (BO) (Snoek et al., 2015), reinforcement learning (Zoph and Le, 2017), evolutionary algorithms (Jaderberg et al., 2017) and gradient-based methods (Bengio, 2000). The state-of-the-art in HPO depends on the problem setting, but black-box methods like Hyperband (HB) (Li et al., 2017b), and its combination with BO into a method called BOHB (Falkner et al., 2018) has been the most popular. Contrary to the above, the HPO method developed in this chapter is gradient-based, making it closer in spirit to many meta-learning approaches. Nonetheless, some of the methods above make for strong baselines, and so a more technical description of them is provided below. A more comprehensive overview of all hyperparameter optimization methods can be found in (Feurer and Hutter, 2019; Yu and Zhu, 2020).

Random search (RS). This is the simplest alternative to grid search, and consists in evaluating random hyperparameter configurations. In most real world applications, some hyperparameters matter less than others, and random search has been shown to significantly outperform grid search (Bergstra and Bengio, 2012). In this chapter I include random search as a baseline because of its simplicity and popularity in deep learning.

Bayesian Optimization (BO). This is a framework based on two components: a *surrogate model* which is fitted to match the target function, and an *acquisition function*, which is used to choose the next hyperparameter setting to evaluate. The surrogate model usually relies on a Gaussian Process, so that it can provide a measure of uncertainty to the acquisition function, and its posterior, given new observations of the target function, is computed using Bayes' rule. The acquisition function can take many forms (usually *expected improvement* or *probability of improvement*) and encodes the exploration vs. exploitation trade-off. BO has had success in hyperparameter tuning for various deep learning applications (Snoek et al., 2015; Dahl et al., 2013). A more in-depth overview of BO methods can be found in (Shahriari et al., 2016).

Bandit-based algorithms. In the multi-armed bandit problem (Katehakis and Veinott, 1987; Slivkins, 2019), the task is to allocate resources to competing options in a way that maximizes their expected reward. Each option is partially known/understood at the time of allocation, and becomes better known as an increasing amount of resource is spent on it over time. A successful bandit-based strategy understands poor options quickly from little resources and eliminates them. It then allocates more resources to promising options to find the best option amongst several good ones. In the case of HPO for deep neural networks hyperparameters, the hyperparameter configurations represent the arms. If the final objective is validation accuracy after training for 100 epochs on ImageNet, it's often the case that many poor hyperparameter configurations can be ruled out from much fewer epochs or training samples per epoch (Petrak, 2000; Komer et al., 2014). In this chapter we use the algorithm HyperBand (HB) (Li et al., 2017b) as a baseline, which extends the bandit-based algorithm SuccessiveHalving (Jamieson and

Talwalkar, 2016). SuccessiveHalving iteratively evaluates all hyperparameter configurations under consideration for a given budget, removes the half that performed worse, and doubles the budget for the next iteration. This is repeated until there is only one hyperparameter configuration left. The main limitation of SuccessiveHalving is the fact that the user must decide whether to initially try many configurations with a small budget or fewer configurations with a larger budget. HyperBand addresses this issue by using several random combinations of number of configurations vs. configuration budget, and has shown a strong performance in DL.

Bayesian optimized bandit-based hybrids. The recent algorithm BOHB (Falkner et al., 2018) proposes to combine the benefits of Bayesian optimization and HyperBand. This is done by replacing the randomness of HyperBand when proposing configurations with proposals from Bayesian optimization. In practice this means that BOHB enjoys both the strong anytime performance of HB as well as the strong final performance of BO. This method was shown to obtain state-of-the-art in a variety of HPO settings, and is thus a valuable baseline to compare our algorithm against.

4.1.2 Reverse vs. forward mode differentiation

The method proposed in this chapter relies on forward mode differentiation. While exact derivations in the context of hyperparameter optimization are provided later, it is worth clarifying the high level differences between reverse and forward mode differentiation. Note that both reverse and forward mode differentiation are typically faster than the HPO methods mentioned in the previous section, but they are only applicable to differentiable hyperparameters.

When differentiating neural networks, we are typically interested in computing gradients through the composition of several functions, since each layer's output is the input to the following layer. As an example, consider the composition $z(g(f(\boldsymbol{x})))$, where $z(\cdot)$ outputs a vector $\boldsymbol{z} \in \mathbb{R}^Z$, $g(\cdot)$ outputs a vector $\boldsymbol{g} \in \mathbb{R}^G$ and $f(\cdot)$ outputs a vector $\boldsymbol{f} \in \mathbb{R}^F$. Say that we would like to compute $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$, whose expression is given by the chain

rule as the product of 3 Jacobians:

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{g}} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{f}} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \tag{4.1}$$

Reverse mode differentiation, aka *backpropagation*, is the workhorse of Deep Learning. It operates by *tracing* operations (nodes) in the forward pass, retaining intermediate information in memory for each computation, which are used to compute gradients in a backward pass. In Eq. (4.1) the RHS is computed left to right, starting from the final node: first $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{f}} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{g}} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{f}}$ is computed, and then $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{f}} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ is computed, resulting in $ZGF+ZFX$ operations. Note how this order of operations relies on storing intermediate values for $\boldsymbol{g}$ and $\boldsymbol{f}$ in the forward pass, to be used in the backward pass.

In contrast, forward mode differentiation does not use a backward pass, and does not need to store intermediate values. Intermediate gradient terms are computed during the forward pass, and this corresponds to computing the RHS of Eq. (4.1) from right to left: first $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{f}} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ is computed, and then $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{g}} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}$ is computed, resulting in $XGF + ZGX$ operations.

Assuming that $G \simeq F$ for the sake of the argument, it can be seen that reverse-mode differentiation is cheaper than forward-mode differentiation when the dimension of the output is smaller than the dimension of the input: $Z << X$. Conversely, forward mode differentiation is cheaper when there are more outputs than inputs: $X << Z$. In Deep Learning, the reverse mode procedure has two substantial advantages. First, it is usually the case that $\boldsymbol{z}$ represents a loss function, i.e. a scalar, and $\boldsymbol{x}$ represents millions of parameters or activation values, which means that $Z << X$. Secondly, the intermediate values calculated in the backward pass for reverse mode are also wanted, while the intermediate gradient values in the forward pass have little relevance. In the example above, reverse-mode provides $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{f}}$ *for free* while computing $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$, which usually corresponds to the gradient of the loss w.r.t. the parameters/activations of some other layer.

While the two reasons given above make forward mode differentiation irrelevant for most deep learning applications, the case considered in this chapter tackles the differentiation of hyperparameters through the whole training process of a neural network. That is, the gradient descent updates on the neural network weights themselves constitute the forward pass, in the sense that they involve meta parameters (like learning rate) which must be differentiated subsequently. In this case, reverse-mode differentiation would need to store intermediate values for each layer for T gradient steps, which is intractable for modern networks for $T \gtrsim 10$. While forward-mode differentiation incurs a larger computational cost, especially as the number of hyperparameters grow, it has the benefit of being tractable in memory, because no intermediate values need to be stored.

4.2 Introduction

Deep neural networks have shown tremendous success on a wide range of applications, including image classification (He et al., 2016), generative models (Brock et al., 2019), natural language processing (Devlin et al., 2018) and speech recognition (Oord et al., 2016). This success is in part due to effective optimizers such as SGD with momentum or Adam (Kingma and Ba, 2015), which require carefully tuned hyperparameters for each application. In recent years, a long list of heuristics to tune such hyperparameters has been compiled by the deep learning community, including things like: how to best decay the learning rate (Loshchilov and Hutter, 2017), how to scale hyperparameters with the budget available (Li et al., 2020a), and how to scale learning rate with batch size (Goyal et al., 2017). Unfortunately these heuristics are often dataset specific and architecture dependent (Dong et al., 2020). They also don't apply well to new optimizers (Loshchilov and Hutter, 2019), or new tools, like batch normalization which allows for larger learning rates (Ioffe and Szegedy, 2015).

With so many ways to choose hyperparameters, the deep learning community is at risk of adopting models based on how much effort went into tuning them, rather than their methodological insight. The field of hyperparameter optimization (HPO) aims to find hyperparameters that provide a good generalization performance automatically. Unfortunately, existing tools are rather unpopular for deep learning, largely owing to their computational cost. The method developed here is a gradient-based HPO approach; that is, it calculates hypergradients, i.e. the gradient of some validation loss with respect to each hyperparameter. Gradient-based HPO should be more efficient than black-box methods as the dimensionality of the hyperparameter space increases, since it is able to utilize gradient information rather than rely on trial and error. In practice however, learning hyperparameters with gradients has only been popular in few-shot learning tasks where the horizon is short, i.e. where the underlying task is solved with a few gradient steps. This is because long horizons cause hypergradient degradation, and incur a memory cost that makes reverse-mode differentiation prohibitive.

Greedy gradient-based methods alleviate both of these issues by calculating local hypergradients based on intermediate validation losses. Unfortunately, this introduces some bias (Wu et al., 2018b) and results in a significant performance drop, which we are able to quantify in this work. We make use of forward-mode differentiation, which has been shown to offer a memory cost constant with horizon size. However, previous forward-mode methods don't address gradient degradation explicitly and are thus limited to the greedy setting (Franceschi et al., 2017; Donini et al., 2019).

We introduce FDS (Foward-mode Differentiation with hyperparameter Sharing), which to the best of our knowledge demonstrates for the first time that hyperparameters can be differentiated non-greedily over long horizons. Specifically, we make the following contributions: **(1)** we propose to share hyperparameters through time, both motivating it theoretically and with various experiments, **(2)** we combine the above in a forward-mode differentiation algorithm, and **(3)** we show that our method can significantly outperform various HPO algorithms, for instance when learning the hyperparameters of the SGD-momentum optimizer.

4.3 Related work

There are many ways to perform hyperparameter optimization (HPO) (Feurer and Hutter, 2019), including Bayesian optimization (BO) (Snoek et al., 2015), reinforcement learning (Zoph and Le, 2017), evolutionary algorithms (Jaderberg et al., 2017) and gradient-based methods (Bengio, 2000). The state-of-the-art in HPO depends on the problem setting, but black-box methods like Hyperband (HB) (Li et al., 2017b), and its combination with BO into a method called BOHB (Falkner et al., 2018) have been the most popular. Modern work in meta-learning deals with various forms of gradient-based HPO (Hospedales et al., 2021), but usually focuses on the few-shot regime (Finn et al., 2017) where horizons are conveniently short (~ 10 steps) while we focus on long horizons ($\sim 10^4$ steps).

Gradient-based HPO. Using the gradient of some validation loss with respect to the hyperparameters is typically the preferred choice when the underlying optimization is differentiable. This is a type of constrained optimization (Franceschi et al., 2018) which stems from earlier work on backpropagation through time (Werbos, 1990) and real-time recurrent learning (Williams and Zipser, 1989). Unfortunately, differentiating optimization is an expensive procedure in both time and memory, and most proposed methods are limited to small models and toy datasets (Domke, 2012; Maclaurin et al., 2015; Pedregosa, 2016). Efforts to make the problem more tractable include optimization shortcuts (Fu et al., 2016), truncation (Shaban et al., 2019) and implicit gradients (Larsen et al., 1996; Rajeswaran et al., 2019; Lorraine et al., 2019). Truncation can be combined with our approach but produces biased gradients (Metz et al., 2019), while implicit differentiation is only applicable to hyperparameters that define the training loss (e.g. augmentation) but not to hyperparameters that define how the training loss is minimized (e.g. optimizer hyperparameters). Forward-mode differentiation (Williams and Zipser, 1989) boasts a memory cost constant with horizon size, but gradient degradation has restricted its use to the greedy setting (Franceschi et al., 2017; Donini

et al., 2019). Finite difference approaches and related SPSA methods (Bhatnagar et al., 2013) aren't usually applied to the HPO setting, possibly due to the difficulty of choosing a scale for the perturbation they rely on, which is robust to both hypergradient explosion and hypergradient vanishing.

Greedy methods. One trick that prevents gradient degradation and significantly reduces compute and memory cost is to solve the bilevel optimization greedily. This has become the default trick in various long-horizon problems, including HPO over optimizers (Luketina et al., 2016; Franceschi et al., 2017; Baydin et al., 2018; Donini et al., 2019), architecture search (Liu et al., 2019), dataset distillation (Wang et al., 2018) or curriculum learning (Ren et al., 2018). Greediness refers to the heuristic-based problem solving approach of making the locally optimal choice at each step of an algorithm/strategy. In our context this translates to finding the best hyperparameters locally rather than globally. In practice, it involves splitting the inner optimization problem into smaller chunks (often just one batch), and solving for hyperparameters over these smaller horizons instead; often in an online fashion. We formalize greediness in Sec. 4.4.2. In this paper we expand upon previous observations (Wu et al., 2018b) and take the view that greediness fundamentally solves for the wrong objective. Instead, the focus of our paper is to extend forward-mode differentiation methods to the non-greedy setting.

Gradient degradation. Gradient degradation of some scalar w.r.t a parameter is a broad issue that arises when that parameter influences the scalar in a chaotic fashion, such as through long chains of nonlinear mappings. This manifests itself in HPO as vanishing or exploding hypergradients, due to low or high curvature components of the validation loss surface. This leads to hypergradients that have a large variance, making differentiation through long horizons impractical. This is usually observed in the context of recurrent neural networks (Bengio et al., 1993; Bengio et al., 1994), but also in reinforcement learning (Parmas et al., 2018) and HPO (Maclaurin et al., 2015). Solutions like LSTMs (Hochreiter and Schmidhuber, 1997) and gradient clipping (Pascanu et al., 2013) have been proposed, but are respectively inapplicable and insufficient to long-horizon HPO.

Variational optimization (Metz et al., 2019) and preconditioning warp-layers (Flennerhag et al., 2020) can mitigate gradient degradation, but these methods are expensive in memory and therefore are limited to small architectures and/or a few hundred inner steps. In comparison, we differentiate over $\sim 10^4$ inner steps for WideResNets (Zagoruyko and Komodakis, 2016b).

4.4 Setting the scene

4.4.1 Problem statement

Consider a neural network with weights $\boldsymbol{\theta}$, trained to minimize a loss $\mathcal{L}_{\text{train}}$ over a dataset $\mathcal{D}_{\text{train}}$. This is done by taking T steps with a gradient-based optimizer Φ, which uses a collection of hyperparameters $\boldsymbol{\lambda} \in \mathbb{R}^T$. For clarity of notation, consider that Φ uses one hyperparameter per step, written $\boldsymbol{\lambda}_{[t]}$, where indices are shown in brackets to differentiate them from a variable evaluated at time t. We can explicitly write out this optimizer as $\Phi : \boldsymbol{\theta}_{t+1} = \Phi(\mathcal{L}_{\text{train}}(\boldsymbol{\theta}_t(\boldsymbol{\lambda}_{[1:t]}), \mathcal{D}_{\text{train}}), \boldsymbol{\lambda}_{[t+1]}) \quad \forall t \in \{1, 2, \dots, T\}$. Note that $\boldsymbol{\theta}_t$ is a function of $\boldsymbol{\lambda}_{[1:t]}$, and it follows that $\boldsymbol{\theta}_T = \boldsymbol{\theta}_T(\boldsymbol{\lambda}_{[1:T]}) = \boldsymbol{\theta}_T(\boldsymbol{\lambda})$.

We would like to find the hyperparameters $\boldsymbol{\lambda}^*$ such that the result, at time T, of the gradient process minimizing objective $\mathcal{L}_{\text{train}}$, also minimizes some generalization loss $\mathcal{L}_{\text{val}}$ on a validation dataset $\mathcal{D}_{\text{val}}$. This can be cast as the following constrained optimization:

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}} \mathcal{L}_{\text{val}}(\boldsymbol{\theta}_T(\boldsymbol{\lambda}), \mathcal{D}_{\text{val}}) \tag{4.2}$$

$$s.t. \quad \boldsymbol{\theta}_{t+1} = \Phi(\mathcal{L}_{\text{train}}(\boldsymbol{\theta}_t(\boldsymbol{\lambda}_{[1:t]}), \mathcal{D}_{\text{train}}), \boldsymbol{\lambda}_{[t+1]}) \tag{4.3}$$

The inner loop in Eq. (4.3) expresses a constraint on the outer loop in Eq. (4.2). In gradient-based HPO, our task is to compute the hypergradient $d\mathcal{L}_{\text{val}}/d\boldsymbol{\lambda}$ and update $\boldsymbol{\lambda}$ accordingly. Note that some methods require $\boldsymbol{\theta}_T = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{train}}(\boldsymbol{\theta}, \mathcal{D}_{\text{train}}, \boldsymbol{\lambda})$ with $\mathcal{L}_{\text{train}}$ being a function of $\boldsymbol{\lambda}$ explicitly (Larsen et al., 1996; Rajeswaran et al., 2019; Lorraine et al., 2019), in which case the above becomes a bilevel optimization (Stackelberg, 1952). Our algorithm waives these requirements.

4.4.2 Greediness

Let H be the horizon, which corresponds to the number of gradient steps taken in the inner loop (to optimize $\boldsymbol{\theta}$) before one gradient step is taken in the outer loop (to optimize $\boldsymbol{\lambda}$). When solving Eq. (4.2) non-greedily we have $H = T$. However, most modern approaches are greedy (Luketina et al., 2016; Franceschi et al., 2017; Baydin et al., 2018; Liu et al., 2019; Wang et al., 2018), in that they rephrase the above problem into a sequence of several independent problems of smaller horizons, where $\boldsymbol{\lambda}^*_{[t:t+H]}$ is learned in the outer loop subject to an inner loop optimization from $\boldsymbol{\theta}_t$ to $\boldsymbol{\theta}_{t+H}$ with $H \ll T$. Online methods such as Hypergradient Descent (HD) (Baydin et al., 2018) are an example of greedy differentiation, where $H = 1$ and $\boldsymbol{\lambda}_{[t]}$ is updated at time t. Instead, non-greedy algorithms like FDS only update $\boldsymbol{\lambda}_{[t]}$ at time T.

Greediness has many advantages: it mitigates gradient degradation, makes hypergradients cheaper to compute, and it requires less memory when used with reverse-mode differentiation. However, greedy methods look for $\boldsymbol{\lambda}^*$ that does well locally rather than globally, i.e. they constrain $\boldsymbol{\lambda}^*$ to a subspace of solutions such that $\boldsymbol{\theta}_H, \boldsymbol{\theta}_{2H}, ..., \boldsymbol{\theta}_T$ all yield good validation performances. In our experiments, we found that getting competitive hyperparameters with greediness often revolves around tricks like online learning with a very low outer learning rate combined with hand-tuned initial hyperparameter values, to manually prevent convergence to small values. But solving

the greedy objective correctly leads to poor solutions, a special case of which was previously described as the "short-horizon bias" (Wu et al., 2018b). In FDS, we learn global hyperparameters in a non-greedy fashion, which means that the hypergradient $d\mathcal{L}_{\text{val}}/d\boldsymbol{\lambda}$ is only ever calculated at $\boldsymbol{\theta}_T$.

4.4.3 Forward-mode differentiation of modern optimizers

The vast majority of meta-learning applications use reverse-mode differentiation in the inner optimization problem (Eq. (4.3)) to optimize $\boldsymbol{\theta}$. However, the memory cost of using reverse-mode differentiation for the outer optimization (Eq. (4.2)) is $\mathcal{O}(FH)$, where F is the memory used by one forward pass through the network (weights plus activations). This is often referred to as backpropagation through time (BPTT). In the non-greedy setting where $H = T$, BPTT is extremely limiting: for large networks, only $T \sim 10$ gradient steps could be solved with modern GPUs, while problems like CIFAR-10 require $T \sim 10^4$. Instead, we make use of forward-mode differentiation, which scales in memory as $\mathcal{O}(DN)$, where D is the number of weights and N is the number of learnable hyperparameters. The additional scaling with N is a limitation if we learn one hyperparameter per inner step ($N = T$). Sharing hyperparameters (Sec. 4.5.1) mitigates gradient degradation, but also conveniently allows for smaller values of N.

For clarity of notation, we consider forward-mode hypergradients for the general case of using one hyperparameter per step, i.e. $\boldsymbol{\lambda} \in \mathbb{R}^T$. First, we use the chain rule to write $d\mathcal{L}_{\text{val}}/d\boldsymbol{\lambda} = (\partial\mathcal{L}_{\text{val}}/\partial\boldsymbol{\theta}_T)(d\boldsymbol{\theta}_T/d\boldsymbol{\lambda})$ where the direct gradient has been dropped since $\partial\mathcal{L}_{\text{val}}/\partial\boldsymbol{\lambda} = 0$ for optimizer hyperparameters. The first term on the RHS is trivial and can be obtained with reverse-mode differentiation as usual. The second term is more problematic because $\boldsymbol{\theta}_T = \boldsymbol{\theta}_T(\boldsymbol{\theta}_{T-1}(\boldsymbol{\theta}_{T-2}(...), \boldsymbol{\lambda}_{[T-1]}), \boldsymbol{\lambda}_{[T]})$. We use the chain rule again to calculate this term recursively:

$$\frac{d\boldsymbol{\theta}_t}{d\boldsymbol{\lambda}} = \frac{\partial\boldsymbol{\theta}_t}{\partial\boldsymbol{\theta}_{t-1}}\bigg|_{\boldsymbol{\lambda}} \frac{d\boldsymbol{\theta}_{t-1}}{d\boldsymbol{\lambda}} + \frac{\partial\boldsymbol{\theta}_t}{\partial\boldsymbol{\lambda}}\bigg|_{\boldsymbol{\theta}_{t-1}} \quad \text{which we write as} \quad \boldsymbol{Z}_t = \boldsymbol{A}_t\boldsymbol{Z}_{t-1} + \boldsymbol{B}_t \tag{4.4}$$

where $\boldsymbol{\theta}_t \in \mathbb{R}^D$, $\boldsymbol{\lambda} \in \mathbb{R}^T$, $\boldsymbol{Z}_t \in \mathbb{R}^{D\times T}$, $\boldsymbol{A}_t \in \mathbb{R}^{D\times D}$ and $\boldsymbol{B}_t \in \mathbb{R}^{D\times T}$. Note that the columns of $\boldsymbol{Z}$ at step t are zeros for indices $t+1, t+2, ..., T$, since the hyperparameters at those steps haven't been used yet.

The expressions for $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ depend on the specific hyperparameters we are differentiating. In this work, we consider the most popular optimizer, namely SGD with momentum and weight decay. To the best of our knowledge, previous work focuses on simpler versions of this optimizer, usually by removing momentum and weight decay, and only learns the learning rate, greedily. We use Pytorch's update rule for SGD (Paszke et al., 2019), namely $\boldsymbol{\theta}_t = \Phi(\boldsymbol{\theta}_{t-1}) = \boldsymbol{\theta}_{t-1} - \alpha_t \boldsymbol{v}_t$ with learning rate α_t, momentum β_t, weight decay ξ_t and velocity $\boldsymbol{v}_t = \beta_t \boldsymbol{v}_{t-1} + (\partial \mathcal{L}_{\text{train}}/\partial \boldsymbol{\theta}_{t-1}) + \xi_t \boldsymbol{\theta}_{t-1}$. Consider the case when we learn the learning rate schedule, namely $\boldsymbol{\lambda} = \boldsymbol{\alpha}$. If we use the update rule without momentum (Donini et al., 2019), $\boldsymbol{B}_t$ is conveniently sparse: it is a $D \times T$ matrix that only has one non-zero column at index t corresponding to $\partial \boldsymbol{\theta}_t / \partial \alpha_t$. However, we include terms like momentum and therefore the velocity depends on the hyperparameters of previous steps. In that case, a further recursive term $\boldsymbol{C}_t = (\partial \boldsymbol{v}_t / \partial \boldsymbol{\lambda})$ must be considered to get exact hypergradients. Putting it together (see Appendix A) we obtain:

$$\begin{cases} \boldsymbol{A}_t^{\boldsymbol{\alpha}} = \mathbf{1} - \alpha_t \left(\dfrac{\partial^2 \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}^2} + \xi_t \mathbf{1} \right) \\ \boldsymbol{B}_t^{\boldsymbol{\alpha}} = -\beta_t \alpha_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} - \delta_t^{\otimes} \left(\beta_t \boldsymbol{v}_{t-1} + \dfrac{\partial \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}} + \xi_t \boldsymbol{\theta}_{t-1} \right) \\ \boldsymbol{C}_t^{\boldsymbol{\alpha}} = \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} + \left(\xi_t \mathbf{1} + \dfrac{\partial^2 \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}^2} \right) \boldsymbol{Z}_{t-1}^{\boldsymbol{\alpha}} \end{cases} \tag{4.5}$$

where $\mathbf{1}$ is a $D \times D$ identity matrix, and $\delta_t^{\otimes}(\boldsymbol{q})$ turns a vector $\boldsymbol{q}$ of size D into a matrix of size $D \times T$, whose t-th column is set to $\boldsymbol{q}$ and other columns to $\mathbf{0}$s. While the matrices in Eq. (4.5) are updated online, the hyperparameters aren't. This is because in the non-greedy setting we don't have access to the hypergradients until we have computed $\boldsymbol{Z}_T^{\boldsymbol{\alpha}}$. A similar technique can be applied to momentum and weight decay to get $\boldsymbol{Z}_T^{\boldsymbol{\beta}}$ and $\boldsymbol{Z}_T^{\boldsymbol{\xi}}$ (see Appendix A). All hypergradient derivations in this paper were checked with finite differences. Note that we focus on learning the hyperparameters of

SGD with momentum because it is the most common optimizer in the deep learning literature. However, just like reverse-mode differentiation, forward-mode differentiation can be applied to any differentiable hyperparameter, albeit with appropriate $\boldsymbol{A_t}$ and $\boldsymbol{B_t}$ matrices.

4.5 Non-greedy differentiation over long horizons

4.5.1 Hyperparameter sharing: trading off noise reduction with bias increase

The main challenge of doing non-greedy meta-learning over long horizons is gradient degradation. In HPO this arises because a small change in $\mathcal{L}_{\text{train}}(\boldsymbol{\theta}_t)$ can cascade forward into a completely different $\boldsymbol{\theta_T}$, resulting in large fluctuations of the hypergradients. This noise (which we show in Fig. 4.1) makes the generalization loss hard to minimize (Eq. (4.2)). We find that the two main causes for this noise are the ordering of the training minibatches, and the weight initialization $\boldsymbol{\theta}_0$. Ideally, the hyperparameters we learn should be agnostic to both of these factors, and so we would like to average out their effect on hypergradients.

Ensemble averaging The most obvious way to address the above is to compute all hypergradients across several random seeds, where each seed corresponds to a different dataset ordering and weight initialization. We could then obtain an average hypergradient μ_t for each inner step t. Here, $\boldsymbol{\mu} \in \mathbb{R}^T$ is often called an ensemble average in statistical mechanics. The issue with ensemble averaging in our setting is its computational and memory cost, since each random seed requires differentiating through T unrolled inner steps for T hyperparameters. This makes both reverse-mode and forward-mode differentiation intractable. We consider the ensemble average as optimal in our analysis, which allows us to derive an expression for the mean square error between our hypergradients estimate and $\boldsymbol{\mu}$.

Time averaging In FDS, we use the long horizon to our advantage and propose to do time averaging instead of ensemble averaging, i.e. we *average out hypergradients across the inner loop (Eq.* (4.3)*) rather than the outer loop (Eq.* (4.2)*)*. More specifically, we sum hypergradients from W neighbouring time steps in the inner loop, which is exactly equivalent to sharing one hyperparameter over all these steps. The average is then obtained trivially by dividing by W. For instance, when learning the learning rate schedule $\boldsymbol{\alpha}$ for $H = 10^4$ inner steps, we can learn $\boldsymbol{\alpha} \in \mathbb{R}^{10}$ where each learning rate is used for a window of $W = 10^3$ contiguous gradient steps. A stochastic system where the time average is equivalent to the ensemble average is called *ergodic* (Walters, 1982) and has been the subject of much research in thermodynamics (Boltzmann, 1896) and finance (Peters, 2019). In our case, hypergradients aren't generally ergodic, and so using a single average hypergradient for W contiguous steps can introduce a bias. Informally, time averaging contiguous hypergradients leads to both noise reduction and bias increase, and we flesh out the nature of this trade-off in Theorem 4.5.1.

Theorem 4.5.1. Let each time step $t \in \{1, 2, ..., T\}$ have a corresponding hyperparameter λ_t and non-greedy hypergradient $g_t = \partial L_{\text{val}}(\boldsymbol{\theta}_T)/\partial \lambda_t$. Each g_t is a random variable due to the sampling process of the weight initialization $\boldsymbol{\theta}_0$ and the inner loop minibatch selection. Let $\boldsymbol{g} = [g_1, g_2, \ldots, g_T]$ be approximated by a Gaussian distribution $\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with mean $\boldsymbol{\mu} = [\mu_1, \mu_2, ..., \mu_T]$ and covariance matrix $\boldsymbol{\Sigma}$, where $\boldsymbol{\mu}$ corresponds to the optimal hypergradients. Assume that the changes in the values of $\boldsymbol{\mu}$ over time are bounded, $\mu_{t+1} = \mu_t + \epsilon_t$, where $\epsilon_t \in [-\epsilon, \epsilon]$. Finally, let $c \in [0, 1]$ denote the maximum absolute correlation between the values of $\boldsymbol{g}$, i.e. $c \geq |\Sigma_{tt'}|/\sqrt{\Sigma_{tt}\Sigma_{t't'}} \;\; \forall t \neq t'$. Then, we show that the mean squared error of the hypergradients with respect to $\boldsymbol{\mu}$ when sharing W contiguous hyperparameters has an upper bound:

$$\text{MSE}_W \leq \frac{(1 + c(W-1))}{W}\text{MSE}_1 + \epsilon^2\frac{(W^2-1)}{12} \tag{4.6}$$

$$\text{where} \qquad \text{MSE}_1 = \frac{1}{T}\sum_t \boldsymbol{\Sigma}_{tt} \tag{4.7}$$

and so for sufficiently small ϵ and c we have with certainty, $MSE_W < MSE_1$ for some $W > 1$, where MSE_1 is the hypergradient error without any sharing (See Appendix B for a proof).

Discussion Theorem 4.5.1 demonstrates how sharing W contiguous hyperparameters has two effects: 1) it reduces the hypergradient noise by a factor $\mathcal{O}(W/(1 + cW))$ due to the averaging of noisy hypergradients, and 2) it increases the error up to an amount $\mathcal{O}(\epsilon^2 W^2)$ due to an induced bias. Intuitively, averaging contiguous hypergradients maximally reduces noise when they aren't correlated ($c = 0$), and minimally increases bias when they are drawn from distributions of similar means ($\epsilon = 0$). In the simpler case where hypergradients are iid and have the same variance at each step, namely $\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \mathbf{1})$, the expressions above become $\text{MSE}_1 = \sigma^2$ and $\text{MSE}_W \leq \text{MSE}_1/W + \epsilon^2(W^2 - 1)/12$. Note that in all cases, the upper bound on MSE_W has a single minimum W^* corresponding to the optimal trade-off between noise reduction and bias increase.

4.5.2 The FDS algorithm

As it is presented in Sec. 4.4.3, forward-mode differentiation would still have a memory cost that scales as $\mathcal{O}(DT)$ since we are learning one hyperparameter per step. However, addressing gradient degradation by sharing hyperparameters also conveniently reduces that memory cost by a factor W down to $\mathcal{O}(DN)$, where $N = T/W$ is the number of unique hyperparameter values we learn. This is because we can safely average hypergradients without calculating them individually, by reusing the same column of $\boldsymbol{Z}$ for W contiguous steps. This is shown in Algorithm 3, when meta-learning a schedule of $N^{\boldsymbol{\alpha}}$ learning rates with FDS. Here, $\boldsymbol{Z}^{\boldsymbol{\alpha}}_{[i]}$ refers to the i-th column of matrix $\boldsymbol{Z}^{\boldsymbol{\alpha}}$. We don't need to store $\boldsymbol{\mathcal{H}}$ or $\boldsymbol{A}^{\boldsymbol{\alpha}}$ in memory since we calculate the Hessian matrix product $\boldsymbol{\mathcal{H}}\boldsymbol{Z}^{\boldsymbol{\alpha}}$ directly. Most importantly, note that hyperparameters aren't updated greedily or online, but are updated once per outer step, which corresponds to differentiating through the entire unrolled inner loop optimization and getting the exact hypergradients $\partial \mathcal{L}_{\text{val}}(\boldsymbol{\theta}_T)/\partial \boldsymbol{\alpha}$.

Algorithm 3: Simplified FDS algorithm when learning N^α learning rates for the SGD optimizer with momentum. Each learning rate is shared over W contiguous time steps

initialize: $N^\alpha, W = T/N^\alpha, \boldsymbol{\alpha} = \mathbf{0}^{N^\alpha}$

for *o in* $1, 2, ...$ **do**

  initialize: $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{val}}$, $\boldsymbol{\theta}_0 \in \mathbb{R}^D$, $\boldsymbol{Z}^{\boldsymbol{\alpha}} = \mathbf{0}^{D \times N^\alpha}$, $C^{\boldsymbol{\alpha}} = \mathbf{0}^{D \times N^\alpha}$

  for *t in* $1, 2, ..., T$ **do**

    $\boldsymbol{x}_{\text{train}}, \boldsymbol{y}_{\text{train}} \sim \mathcal{D}_{\text{train}}$

    $\boldsymbol{g}_{\text{train}} = \partial \mathcal{L}_{\text{train}}(\boldsymbol{x}_{\text{train}}, \boldsymbol{y}_{\text{train}}) / \partial \boldsymbol{\theta}$

    $i = \lceil t/W \rceil$

    $\boldsymbol{\mathcal{H}} \boldsymbol{Z}^{\boldsymbol{\alpha}}_{[1:i]} = \partial(\boldsymbol{g}_{\text{train}} \boldsymbol{Z}^{\boldsymbol{\alpha}}_{[1:i]}) / \partial \boldsymbol{\theta}$

    $\boldsymbol{Z}^{\boldsymbol{\alpha}}_{[1:i]} = \boldsymbol{A}^{\boldsymbol{\alpha}} \boldsymbol{Z}^{\boldsymbol{\alpha}}_{[1:i]} + \boldsymbol{B}^{\boldsymbol{\alpha}}_{[1:i]}$

    update $C^{\boldsymbol{\alpha}}$ (Eq. (4.5))

    $\boldsymbol{\theta}_{t+1} = \Phi(\boldsymbol{\theta}_t, \boldsymbol{g}_{\text{train}})$

  end

  $\boldsymbol{g}_{\text{val}} = \partial \mathcal{L}_{\text{val}}(\mathcal{D}_{\text{val}}) / \partial \boldsymbol{\theta}$

  $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - 0.1 \times \boldsymbol{g}_{\text{val}} \boldsymbol{Z}^{\boldsymbol{\alpha}} / W$

end

The main cost of Algorithm 3 comes from calculating $\boldsymbol{\mathcal{H}} \boldsymbol{Z}^{\boldsymbol{\alpha}}$. There exists several methods to approximate this product, but we found them too crude for long horizons. This includes first-order approximations or truncation (Shaban et al., 2019), which can be adapted to FDS trivially. One could also use functional forms for more complex schedules to be learned in terms of fewer hyperparameters, but this typically makes stronger assumptions about the shape of each hyperparameter schedule, which can easily cloud the true performance of HPO algorithms. In practice, we calculate $\boldsymbol{\mathcal{H}} \boldsymbol{Z}^{\boldsymbol{\alpha}}$ exactly, and use a similar form to Algorithm 3 to learn $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\xi}$.

4.6 Experiments

Our experiments show how FDS mitigates gradient degradation and outperforms competing HPO methods for tasks with a long horizon. In Sec. 4.6.1 and Sec. 4.6.2 we consider small datasets (MNIST and SVHN) and a small network (LeNet) to make reverse-mode differentiation tractable, so that the effect of hyperparameter sharing can

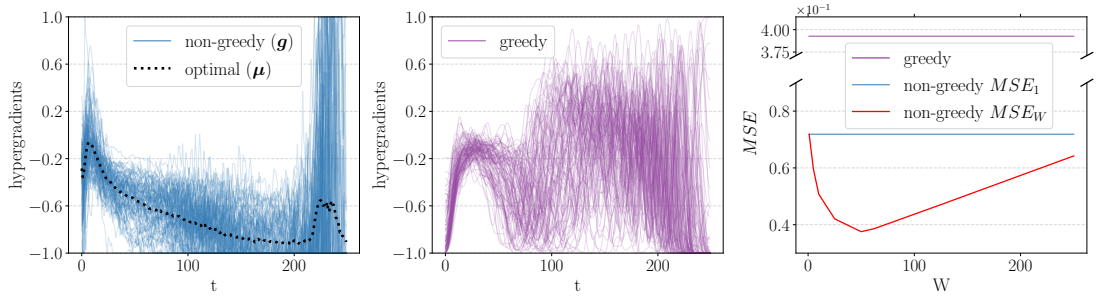

Figure 4.1: Hypergradients of $\boldsymbol{\alpha}$ on SVHN for 100 seeds in the non-greedy (left) and greedy (middle) setting. The mean squared error is also shown (right), and is calculated with respect to the optimal hypergradient ($\boldsymbol{\mu}$), i.e. the ensemble average of non-greedy hypergradients (dotted line in left figure). We can see that sharing hyperparameters over W steps lowers the MSE even for the small value of $T = 250$ used here. The best trade-off between noise reduction and bias increase is $W = 50$.

be directly measured. In Sec. 4.6.3 and Sec. 4.6.4, we then showcase FDS on CIFAR-10 where only forward-mode differentiation is tractable. All experiments are carried out on a single GTX 2080 GPU. More implementation details can be found in Appendix C.

4.6.1 The effect of hyperparameter sharing on hypergradient noise

We consider a LeNet network trained on an inner loop of $T = 250$ gradient steps, and calculate the hypergradients of the learning rate $\boldsymbol{\alpha}$ across 100 seeds. Each seed is evaluated at the same value of $\boldsymbol{\alpha}$, but corresponds to a different training dataset ordering and weight initialization. We can calculate the hypergradients greedily ($H = 1$) and non-greedily ($H = T$). The optimal hypergradients are considered to be the ensemble average of the non-greedy seeds as per Sec. 4.5.1, which allows an MSE to be calculated for each method. These results are shown in Fig. 4.1 for a learning rate schedule initialized to small values. We observe that greediness is a poor approximation to the optimal hypergradients, and that time averaging contiguous hypergradients in the non-greedy case can significantly reduce the MSE even when averaging over only $W = 50$ steps.

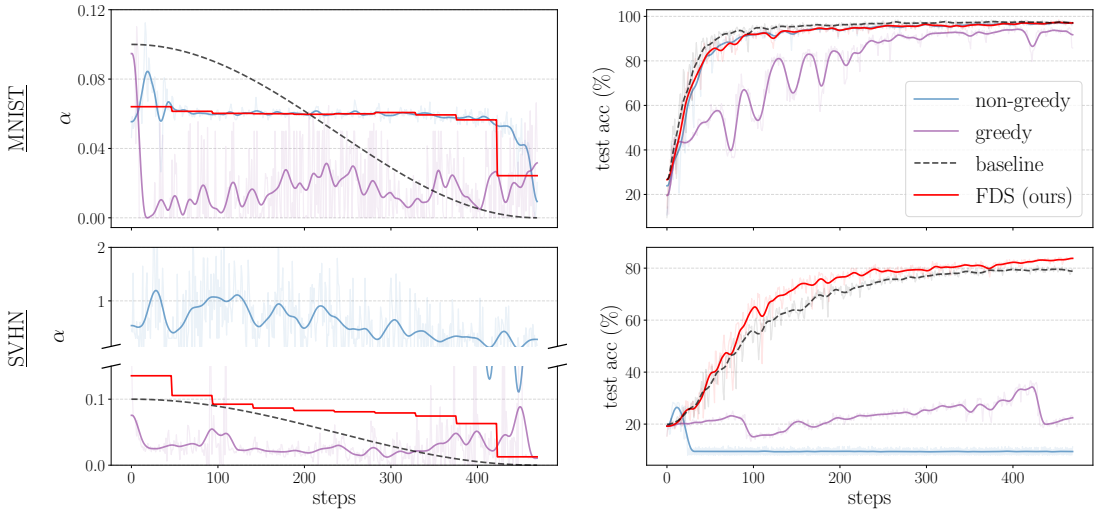

Figure 4.2: The learning rate schedule $\boldsymbol{\alpha}$ learned for the MNIST and SVHN datasets using a LeNet architecture over one epoch. We observe that on real-world datasets like SVHN, both greedy and non-greedy hyperparameter optimizations fail to learn decent learning rate schedules when using one hypergradient per inner step. However, sharing learning rates over contiguous steps stabilizes non-greedy hypergradients and allows us to find learning rates that can even outperform the baseline, which is a common off-the-shelf schedule in this setting (cosine annealing).

The small size of this toy problem allows us to use reverse-mode differentiation and obtain a hypergradient value for each step t, which allows us to calculate $\boldsymbol{\Sigma}$, c and ϵ as defined in Theorem 4.5.1, to verify that our upper bound holds and that its shape as a function of W is realistic. In the setting shown in Fig. 4.1 we find $\epsilon = \max |\mu_{t+1} - \mu_t| \sim 0.08$ and $(1/T)\sum_t \boldsymbol{\Sigma}_{tt} \sim 0.25$. The value of the maximum correlation between any two steps, c, can be quite high which makes the upper bound loose. In practice however, the shape of the upper bound in Theorem 4.5.1 as a function of W (as plotted in Appendix B) closely matches that of the measured MSE shown in Fig. 4.1. Note that the values of ϵ, $\boldsymbol{\Sigma}$, and c can vary depending on the value of $\boldsymbol{\alpha}$. As illustrated in Theorem 4.5.1, we find that settings that have a smaller values of ϵ_t benefit from a larger W and reduce the MSE more (see Appendix D for more examples).

4.6.2 The effect of hyperparameter sharing on HPO

In the section above, we considered hypergradient noise around a fixed hyperparameter setting. In this section, we consider how that noise and its mitigation translate to meta-learning hyperparameters over several outer steps.

In Fig. 4.2, we initialize $\boldsymbol{\alpha} = \mathbf{0}$ and train a LeNet network over 1 epoch ($T \sim 500$ gradient steps) on MNIST and SVHN. We compare the maximally greedy setting ($H = 1$), the non-greedy setting ($H = T$), and FDS (also $H = T$ but with sharing of $W \sim 50$ contiguous hyperparameters). In all cases we take 50 outer steps per hyperparameter. We make greediness more transparent by not using tricks as in Hypergradient Descent (Baydin et al., 2018), where α_t is set to α_{t-1} before its hypergradients are calculated. As previously observed by (Wu et al., 2018b), greedy optimization leads to poor solutions with learning rates that are always too small. While the non-greedy setting without sharing works well for simple datasets like MNIST, it fails for real-world datasets like SVHN, converging to much higher learning rates than reasonable. This is due to gradient degradation, whose negative effect can compound during outer optimization, as a single large learning rate can increase the hypergradient noise for neighbouring steps. As we share hyperparameters in FDS, we reduce and stabilize the outer optimization. This allows us to learn a much more sensible learning rate schedule, which can even outperform a reasonable cosine annealing off-the-shelf schedule on SVHN.

4.6.3 FDS on CIFAR-10

We demonstrate that our algorithm can be used to learn the learning rate, momentum and weight decay over 50 epochs of CIFAR-10 ($H \sim 10^4$) for a WideResNet of 16 layers (WRN-16-1). We choose not to use larger architectures or more epochs to enable compute time for more extensive comparisons and because hyperparameters matter most for fewer epochs. Note also that we are not interested in finding the architecture with the best performance, but rather in finding the best hyperparameters given an architecture. For the learning rate, we choose W such that the ratio of T/W is similar to the optimal one found in Sec. 4.6.1, and we set $W = T$ for the momentum and

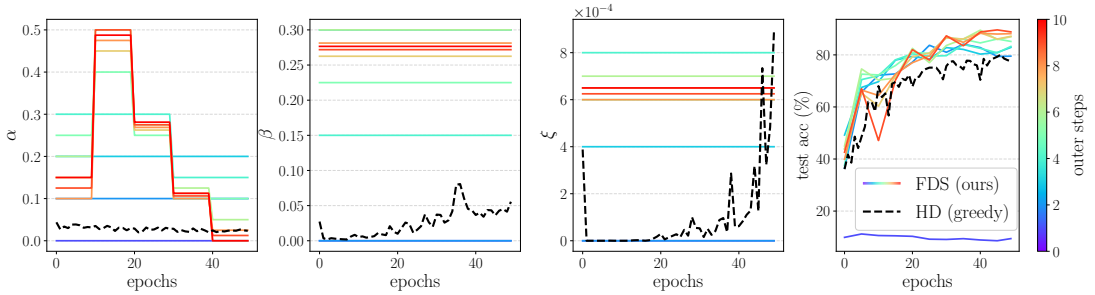

Figure 4.3: FDS applied to the SGD optimizer to learn (from left to right) the learning rate schedule $\boldsymbol{\alpha}$, the momentum $\boldsymbol{\beta}$, and weight decay $\boldsymbol{\xi}$ for a WRN-16-1 on CIFAR-10. For each outer step (color) we solve CIFAR-10 from scratch for 50 epochs, and update all hyperparameters such that the final weights minimize some validation loss. We use hyperparameter sharing over $W = 10$, 50 and 50 epochs for $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\xi}$ respectively. All hyperparameters are initialized to zero and converge within just 10 outer steps to values that significantly outperform Hypergradient Descent (HD) (Baydin et al., 2018), the greedy alternative. We match the performance of the best known hyperparameters in that setting and do so much faster than state-of-the-art black-box methods (see Sec. 4.6.4).

weight decay since only a single value is commonly used for these hyperparameters. The schedules learned are shown in Fig. 4.3, which demonstrates that FDS converges in just 10 outer steps to hyperparameters that are very different to online greedy differentiation (Baydin et al., 2018), and correspond to significantly better test accuracy performances. Note that having the maximum learning rate be large and occur half way during training is reminiscent of the one-cycle schedule (Smith and Topin, 2017).

4.6.4 FDS compared to other HPO methods

A common theme in meta-learning research has been the lack of appropriate baselines, with researchers often finding that random search (RS) can outperform complex search algorithms, for instance in NAS (Li and Talwalkar, 2019) or automatic augmentation (Cubuk et al., 2020). In this section we compare FDS to several competing HPO methods on CIFAR-10, in both the performance of the hyperparameters found and the time it takes to find them. We consider Random Search (RS), Bayesian Optimization (BO), Hypergradient Descent (HD) (Baydin et al., 2018), HyperBand (HB) (Li et al., 2017b)

and the combination of BO and HB, namely BOHB (Falkner et al., 2018). The latter is typically regarded as state-of-the-art in HPO. We use the official HpBandster (Klein, 2019) implementation of these algorithms, except for HD which we re-implemented ourselves.

The performance of HPO methods is often clouded by very small search ranges. For instance, in (Falkner et al., 2018) the learning rate is searched over the range $[10^{-6}, 10^{-1}]$. In the case of DARTS (Liu et al., 2019), expanding the search space to include many poor architectures has helped diagnose issues with its search algorithm (Zela et al., 2020). For these reasons, we assume only weak priors and consider large search ranges: $\alpha \in [-1, 1]$, $\beta \in [-1.5, 1.5]$, and $\xi \in [-4 \times 10^{-3}, 4 \times 10^{-3}]$, which includes many poor hyperparameter values. In FDS we can specify search ranges by using a fixed update size $\boldsymbol{\gamma}$ to the hyperparameters, which decays by 2 every time the hypergradient flips sign, which is common in sign based optimizers (Jacobs, 1988; Riedmiller and Braun, 1993; Bernstein et al., 2018; Safaryan and Richtárik, 2019). We found that black-box HPO methods did not scale well to more than $\sim$ 10 hyperparameters for such large search ranges, and so considered learning 7 learning rates, 1 momentum and 1 weight decay over 50 epochs. Note that increasing these numbers doesn't affect the accuracy of FDS but significantly reduces that of RS, BO, HB and BOHB. The performance over time of the hyperparameters found by each method is shown in Fig. 4.4. As is common in HPO, we plot regret (in test accuracy) with respect to the best hyperparameters known in this setting, which we obtained from an expensive grid search around the most common hyperparameters used in the literature (more details in Appendix F). To squeeze the optimal performance out of FDS in this experiment, we match the process used in HB and BOHB, namely using smaller budgets for some runs, in particular early ones. We can see that our method reaches zero regret in just 3 hours, while the next best method (BOHB) reaches zero regret in 60 hours. Other methods did not reach zero regret in the maximum of 100 hours they were run for. Note also that we retain the convergence speed of online greedy differentiation (Baydin et al., 2018) while outperforming its regret by $\sim 10\%$ test accuracy.

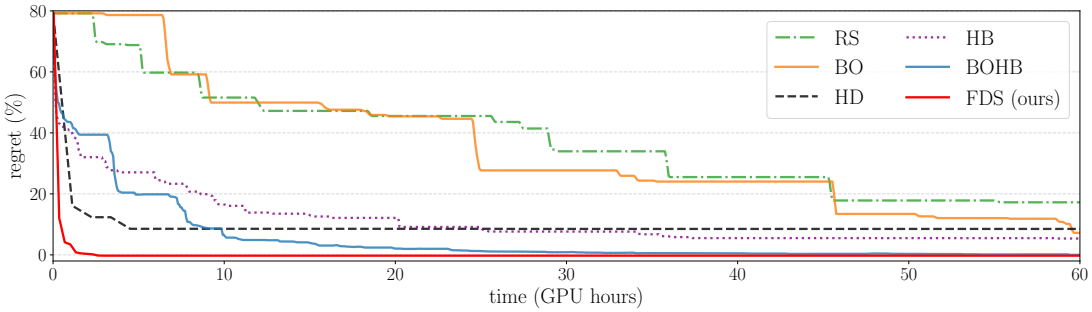

Figure 4.4: Performance of the most popular hyperparameter optimization methods when learning the learning rate, momentum and weight decay on 50 epochs of CIFAR-10, for a WideResNet of 16 layers. Non-greedy methods (RS, BO, HB (Li et al., 2017b), BOHB (Falkner et al., 2018)) solve for global hyperparameters but rely on trial-and-error which makes them slow. Greedy gradient-based methods (e.g. HD (Baydin et al., 2018)) are faster but solve for local hyperparameters which makes them suboptimal. Our method combines the strengths of these two paradigms and outperforms the next best method while converging 20 times faster. Each curve is the average of 8 seeds.

4.7 Discussion

FDS is significantly better than modern HPO alternatives in the case of learning differentiable hyperparameters over long horizons. Its main advantage over black-box methods is its convergence speed, and its main advantage over other gradient-based methods is that it's non-greedy. Furthermore, forward-mode differentiation can provide exact hypergradients for any differentiable hyperparameter, contrary to some other approaches like implicit differentiation which approximates hypergradients, and can do so for certain types of differentiable hyperparameters only.

Nonetheless, it is worth pointing out the main limitations of FDS. First, black-box methods are slower but can readily tackle non-differentiable hyperparameters, while FDS would need to use relaxation techniques to differentiate through, say, discrete variables. Then, the computational and memory cost of FDS scales linearly with the number of hyperparameters. For instance, for a WideResNet of 16 layers we are limited in memory to $\sim 10^3$ hyperparameters on a single 12 GB GPU, which falls short of reverse-mode-based greedy methods which scale to millions of hyperparameters. Another assumption of FDS is that contiguous hyperparameters in time have close optimal values, and can thus share a hypergradient. This is true for most hyperparameters

relevant to deep learning today, but may not always hold in more niche HPO domains. Finally, while automatic forward-mode differentiation is becoming available for some toolboxes like JAX (Bradbury et al., 2018), it remains unavailable for the most popular deep learning libraries, which means that some pen-and-paper work is still required to derive hypergradients for given hyperparameters (Eq. (4.5)).

The focus of our future work will be on adapting FDS to the many meta-learning applications that rely on greedy optimization, such as differentiable architecture search, to improve their performance by making them non-greedy.

4.8 Conclusion

This work makes an important step towards gradient-based HPO for long horizons by introducing FDS, which enables non-greediness through the mitigation of gradient degradation. More specifically, we theorize and demonstrate that sharing hyperparameters over contiguous time steps is a simple yet efficient way to reduce the error in their hypergradients; a setting which naturally lands itself to forward-mode differentiation. Finally, we show that FDS outperform greedy gradient-based alternatives in the quality of hyperparameters found, while being significantly faster than all state-of-the-art black-box methods. We hope that our work encourages the community to reconsider gradient-based HPO in terms of non-greediness, and pave the way towards a universal hyperparameter solver.

Additionally, it is worth reflecting on the state of hyperparameter optimization in the field. The hyperparameters learned in this chapter have effectively been grid searched by the community over years on the datasets considered, and therefore little to no performance can be gained by using HPO. Generally speaking, it is often the case that deep learning practitioners or researchers only have a few new hyperparameters to

search related to the method they introduce, but can reuse most other hyperparameters from previous research work. As such, grid search remains a favorite due to its simplicity and the insight it gives about the sensitivity of each hyperparameter. Research in HPO should therefore focus on the long-horizon many-hyperparameter setting.

4.9 Appendices

Appendix A: Forward-mode hypergradient derivations

Recall that we are interested in calculating

$$\boldsymbol{Z}_t = \boldsymbol{A}_t \boldsymbol{Z}_{t-1} + \boldsymbol{B}_t$$

recursively during the inner loop, where

$$\boldsymbol{Z}_t = \frac{d\boldsymbol{\theta}_t}{d\boldsymbol{\lambda}} \qquad \boldsymbol{A}_t = \left.\frac{\partial \boldsymbol{\theta}_t}{\partial \boldsymbol{\theta}_{t-1}}\right|_{\boldsymbol{\lambda}} \qquad \boldsymbol{B}_t = \left.\frac{\partial \boldsymbol{\theta}_t}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\theta}_{t-1}}$$

so that we can calculate the hypergradients on the final step using

$$\frac{d\mathcal{L}_{\text{val}}}{d\boldsymbol{\lambda}} = \frac{\partial \mathcal{L}_{\text{val}}}{\partial \boldsymbol{\theta}_T} \boldsymbol{Z}_T$$

Each type of hyperparameter needs its own matrix $\boldsymbol{Z}_t$, and therefore its own matrices $\boldsymbol{A}_t$, and $\boldsymbol{B}_t$. Consider first the derivation of these matrices for the learning rate, namely $\boldsymbol{\lambda} = \boldsymbol{\alpha}$. Recall that the update rule of SGD with momentum and weight decay after substituting the velocity $\boldsymbol{v}_t$ in is

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha_t \left(\beta_t \boldsymbol{v}_{t-1} + \frac{\partial \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}} + \xi_t \boldsymbol{\theta}_{t-1} \right)$$

and therefore it follows directly that

$$\boldsymbol{A}_t^{\boldsymbol{\alpha}} = \mathbf{1} - \alpha_t \left(\frac{\partial^2 \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}^2} + \xi_t \mathbf{1} \right)$$

The calculation of $\boldsymbol{B}_t^{\boldsymbol{\alpha}}$ is a bit more involved in our work because when using momentum, $\boldsymbol{v}_{t-1}$ is now itself a function of $\boldsymbol{\alpha}$. First we write

$$\begin{aligned} \boldsymbol{B}_t^{\boldsymbol{\alpha}} &= -\beta_t \left(\frac{\partial \alpha_t}{\partial \boldsymbol{\alpha}} \boldsymbol{v}_{t-1} + \alpha_t \frac{\partial \boldsymbol{v}_{t-1}}{\partial \boldsymbol{\alpha}} \right) - \frac{\partial \alpha_t}{\partial \boldsymbol{\alpha}} \left(\frac{\partial \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}} + \xi_t \boldsymbol{\theta}_{t-1} \right) \\ &= -\beta_t \alpha_t \frac{\partial \boldsymbol{v}_{t-1}}{\partial \boldsymbol{\alpha}} - \delta_t^{\otimes} \left(\beta_t \boldsymbol{v}_{t-1} + \frac{\partial \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}} + \xi_t \boldsymbol{\theta}_{t-1} \right) \end{aligned}$$

Now since

$$\boldsymbol{v}_t = \beta_t \boldsymbol{v}_{t-1} + \frac{\partial \mathcal{L}_{\text{train}}}{\boldsymbol{\theta}_{t-1}} + \xi_t \boldsymbol{\theta}_{t-1}$$

we can write the partial derivative of the velocity as an another recursive rule:

$$\begin{aligned} \boldsymbol{C}_t^{\boldsymbol{\alpha}} &= \frac{\partial \boldsymbol{v}_t}{\partial \boldsymbol{\alpha}} \\ &= \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} + \frac{\partial^2 \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\alpha} \partial \boldsymbol{\theta}_{t-1}} + \xi_t \frac{\partial \boldsymbol{\theta}_{t-1}}{\partial \boldsymbol{\alpha}} \\ &= \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} + \left(\xi_t \mathbf{1} + \frac{\partial^2 \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}^2} \right) \frac{\partial \boldsymbol{\theta}_{t-1}}{\partial \boldsymbol{\alpha}} \end{aligned}$$

And putting all together recovers the system:

$$\begin{cases} \boldsymbol{A}_t^{\boldsymbol{\alpha}} = \mathbf{1} - \alpha_t \left(\dfrac{\partial^2 \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}^2} + \xi_t \mathbf{1} \right) \\ \boldsymbol{B}_t^{\boldsymbol{\alpha}} = -\beta_t \alpha_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} - \delta_t^{\otimes} \left(\beta_t \boldsymbol{v}_{t-1} + \dfrac{\partial \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}} + \xi_t \boldsymbol{\theta}_{t-1} \right) \\ \boldsymbol{C}_t^{\boldsymbol{\alpha}} = \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\alpha}} + \left(\xi_t \mathbf{1} + \dfrac{\partial^2 \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}^2} \right) \boldsymbol{Z}_{t-1}^{\boldsymbol{\alpha}} \end{cases}$$

For learning the momentum and weight decay, a very similar approach yields

$$\begin{cases} \boldsymbol{A}_t^{\boldsymbol{\beta}} = \mathbf{1} - \alpha_t \left(\dfrac{\partial^2 \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}^2} + \xi_t \mathbf{1} \right) \\ \boldsymbol{B}_t^{\boldsymbol{\beta}} = -\beta_t \alpha_t \boldsymbol{C}_{t-1}^{\boldsymbol{\beta}} - \delta_t^{\otimes}(\alpha_t \boldsymbol{v}_{t-1}) \\ \boldsymbol{C}_t^{\boldsymbol{\beta}} = \delta_t^{\otimes}(\boldsymbol{v}_t) + \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\beta}} + \left(\xi_t \mathbf{1} + \dfrac{\partial^2 \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}^2} \right) \boldsymbol{Z}_{t-1}^{\boldsymbol{\beta}} \end{cases}$$

and

$$\begin{cases} \boldsymbol{A}_t^{\boldsymbol{\xi}} = \mathbf{1} - \alpha_t \left(\dfrac{\partial^2 \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}^2} + \xi_t \mathbf{1} \right) \\ \boldsymbol{B}_t^{\boldsymbol{\xi}} = -\beta_t \alpha_t \boldsymbol{C}_{t-1}^{\boldsymbol{\xi}} - \delta_t^{\otimes}(\alpha_t \boldsymbol{\theta}_{t-1}) \\ \boldsymbol{C}_t^{\boldsymbol{\xi}} = \delta_t^{\otimes}(\boldsymbol{\theta}_{t-1}) + \beta_t \boldsymbol{C}_{t-1}^{\boldsymbol{\xi}} + \left(\xi_t \mathbf{1} + \dfrac{\partial^2 \mathcal{L}_{\text{train}}}{\partial \boldsymbol{\theta}_{t-1}^2} \right) \boldsymbol{Z}_{t-1}^{\boldsymbol{\xi}} \end{cases}$$

Appendix B: Theorem 4.5.1 Proof

Preamble Consider that each time step $t \in \{1, 2, ..., T\}$ has a corresponding hyperparameter λ_t and hypergradient $g_t = \partial L_{\text{val}}(\boldsymbol{\theta}_T)/\partial \lambda_t$. Each g_t is viewed as a random variable due to the sampling process of the weight initialization $\boldsymbol{\theta}_0$ and the inner loop minibatch selection. Assume that $\boldsymbol{g} = [g_1, g_2, \ldots, g_T]$ is sufficiently well approximated by a Gaussian distribution, where $\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with mean $\boldsymbol{\mu} = [\mu_1, \mu_2, ..., \mu_T]$ and covariance matrix $\boldsymbol{\Sigma}$. Assume that the values of $\boldsymbol{\mu}$ can be written as the ϵ-Lipschitz function $\mu_{t+1} = \mu_t + \epsilon_t$, where $\epsilon_t \in [-\epsilon, \epsilon]$. Note that in general, the gradients at different time steps may be correlated. Let the magnitude of the correlation be bounded by $c \in [0, 1]$:

$$\frac{|\Sigma_{tt'}|}{\sqrt{\Sigma_{tt}\Sigma_{t't'}}} \leq c \qquad \forall \;\; t \neq t' \tag{4.8}$$

Let W define the size of a non-overlapping window over which hypergradients are averaged. This produces K windows, where each window $k \in \{1, 2, ..., K\}$ contains the time steps $S^{(k)}$ i.e. $S^{(1)} = \{1, 2, \ldots W\}$, $S^{(2)} = \{W + 1, W + 2, \ldots 2W\}$, etc. For simplicity of analysis [1] we assume the chosen window sizes are divisors of T such that

1. This assumption is unnecessary and can be relaxed but would result in a more cumbersome theorem statement, as the final window of size $< W$ would need to be considered.

$K = T/W$. Sharing hyperparameters over W contiguous time steps amounts to using the average hypergradient $\bar{g}^{(k)}$ for each step in that window, where

$$\bar{g}^{(k)} \coloneqq \frac{1}{W} \sum_{t \in S^{(k)}} g_t \tag{4.9}$$

We can now consider the mean squared error across all time steps when averaging contiguous hypergradients in non-overlapping windows of size W:

$$\text{MSE}_W = \frac{1}{K} \sum_k \frac{1}{W} \sum_{t \in S^{(k)}} \mathbb{E}\left[\left(\bar{g}^{(k)} - \mu_t\right)^2\right] \tag{4.10}$$

where all expectations in our proof are over $\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Note the case MSE_1 gives the standard case where no averaging occurs ($K = T$).

Theorem Then

$$\text{MSE}_1 = \frac{1}{T} \sum_t \boldsymbol{\Sigma}_{tt} \tag{4.11}$$

$$\text{MSE}_W \leq \frac{(1 + c(W-1))}{W} \text{MSE}_1 + \epsilon^2 \frac{(W^2 - 1)}{12} \tag{4.12}$$

Proof The case for MSE_1 follows trivially from the definition of variance:

$$\text{MSE}_1 = \frac{1}{T} \sum_t \mathbb{E}[(g_t - \mu_t)^2] = \frac{1}{T} \sum_t \boldsymbol{\Sigma}_{tt} \tag{4.13}$$

and so the mean squared error is the average of the variances. We now focus on the $W > 1$ case. Consider a window enumerated by k, and the vector of gradients within that window $\boldsymbol{g}^{(k)} = (g_t | t \in S^{(k)})$. Under the Gaussian assumption, that vector is Gaussian distributed with covariance $\boldsymbol{\Sigma}^{(k)}$, which is a block of the covariance matrix $\boldsymbol{\Sigma}$ corresponding to the variables in $\boldsymbol{g}^{(k)}$. Let $\bar{\mu}^{(k)} = (1/W) \sum_{t \in S^{(k)}} \mu_t$ be the average of means in window k. We consider the mean squared error from window k as :

$$\mathrm{MSE}^{(k)} = \frac{1}{W}\sum_{t\in S^{(k)}} \mathbb{E}\left[\left(\bar{g}^{(k)} - \mu_t\right)^2\right] \tag{4.14}$$

$$= \frac{1}{W}\sum_{t\in S^{(k)}} \mathbb{E}\left[\left(\bar{g}^{(k)} - \bar{\mu}^{(k)}\right)^2 + \left(\mu_t - \bar{\mu}^{(k)}\right)^2 - 2\left(\bar{g}^{(k)} - \bar{\mu}^{(k)}\right)\left(\mu_t - \bar{\mu}^{(k)}\right)\right] \tag{4.15}$$

$$= \mathbb{E}\left[\left(\bar{g}^{(k)} - \bar{\mu}^{(k)}\right)^2\right] + \frac{1}{W}\sum_{t\in S^{(k)}}\left(\mu_t - \bar{\mu}^{(k)}\right)^2 \tag{4.16}$$

Now $\bar{g}^{(k)} = (1/W)\mathbf{1}^T\boldsymbol{g}^{(k)}$, and $\bar{\mu}^{(k)} = (1/W)\mathbf{1}^T\boldsymbol{\mu}^{(k)}$, and so $\bar{g}^{(k)} - \bar{\mu}^{(k)} = (1/W)\mathbf{1}^T(\boldsymbol{g}^{(k)} - \boldsymbol{\mu}^{(k)})$. Hence

$$\mathbb{E}\left[\left(\bar{g}^{(k)} - \bar{\mu}^{(k)}\right)^2\right] = \frac{1}{W^2}\mathbb{E}\left[\mathbf{1}^T\left(\boldsymbol{g}^{(k)} - \boldsymbol{\mu}^{(k)}\right)\left(\boldsymbol{g}^{(k)} - \boldsymbol{\mu}^{(k)}\right)^T\mathbf{1}\right] \tag{4.17}$$

$$= \frac{1}{W^2}\mathbf{1}^T\boldsymbol{\Sigma}^{(k)}\mathbf{1} \tag{4.18}$$

Now let $\boldsymbol{D}$ be the diagonal matrix of variances, i.e. $\boldsymbol{D}_{ii} = \boldsymbol{\Sigma}^{(k)}_{ii} \quad \forall i$ and $\boldsymbol{D}_{ij} = 0 \quad \forall i \neq j$. We use the correlation bound (Eq. (4.8)), which can be written as $|\boldsymbol{\Sigma}^{(k)}_{ij}| < c\left[\boldsymbol{D}^{\frac{1}{2}}\mathbf{1}\mathbf{1}^T\boldsymbol{D}^{\frac{1}{2}}\right]_{ij} \quad \forall i \neq j$, and this allows us to write an upper bound for the expression above:

$$\mathbb{E}\left[(\bar{g}^{(k)} - \bar{\mu}^{(k)})^2\right] \leq \frac{1}{W^2}\mathbf{1}^T[(1-c)\boldsymbol{D} + c\boldsymbol{D}^{\frac{1}{2}}\mathbf{1}\mathbf{1}^T\boldsymbol{D}^{\frac{1}{2}}]\mathbf{1} \tag{4.19}$$

$$= \frac{(1-c)}{W^2}\sum_i \boldsymbol{D}_{ii} + c\left[\frac{1}{W}\mathbf{1}^T\boldsymbol{D}^{\frac{1}{2}}\mathbf{1}\right]\left[\frac{1}{W}\mathbf{1}^T\boldsymbol{D}^{\frac{1}{2}}\mathbf{1}\right] \tag{4.20}$$

$$= \frac{(1-c)}{W^2}\sum_i \boldsymbol{D}_{ii} + c\left[\frac{1}{W}\sum_i \sqrt{\boldsymbol{D}_{ii}}\right]^2 \tag{4.21}$$

$$= \frac{(1-c)}{W^2}\sum_i \boldsymbol{\Sigma}^{(k)}_{ii} + c\left[\frac{1}{W}\sum_i \sqrt{\boldsymbol{\Sigma}^{(k)}_{ii}}\right]^2 \tag{4.22}$$

This expression can be simplified further using Jensen's inequality for square roots:

$$\mathbb{E}\left[(\bar{g}_k - \bar{\mu}_k)^2\right] \leq \frac{(1-c)}{W^2}\sum_i \mathbf{\Sigma}_{ii}^{(k)} + \frac{cW}{W^2}\sum_i \mathbf{\Sigma}_{ii}^{(k)} \tag{4.23}$$

$$= \frac{1+c(W-1)}{W^2}\sum_i \mathbf{\Sigma}_{ii}^{(k)} \tag{4.24}$$

Now we return to the second part of (Eq. (4.16)). This second term can be bounded using the Lipschitz constraints. In particular for window size W, the maximum error is given when there is maximum deviation from the mean, which occurs when $\mu_t = \mu_{t-1} + \epsilon$. If we write the first mean in window k as $\mu_1^{(k)}$ we have $\mu_t = \mu_1^{(k)} + (t-1)\epsilon \ \forall t \in S^{(k)}$ and in that case $\bar{\mu}^{(k)} = \frac{1}{W}(\mu_1^{(k)} + (\mu_1^{(k)} + \epsilon) + (\mu_1^{(k)} + 2\epsilon) + \ldots + (\mu_1^{(k)} + (W-1)\epsilon) = \mu_1^{(k)} + \frac{(W-1)\epsilon}{2}$ and so $\mu_t - \bar{\mu}_k = (t-1)\epsilon + \frac{(W-1)\epsilon}{2}$. Note that this quantity is the same for all windows k. We can use it to write an upper bound as follows:

$$\frac{1}{W}\sum_{t\in S^{(k)}}\left(\mu_t - \bar{\mu}^{(k)}\right)^2 \leq \frac{1}{W}\sum_{j=1}^{W}\epsilon^2\left((j-1) - \frac{W-1}{2}\right)^2 \tag{4.25}$$

$$= \frac{\epsilon^2}{W}\sum_{j=0}^{W-1}\left(j - \frac{W-1}{2}\right)^2 \tag{4.26}$$

$$= \frac{\epsilon^2}{W}\sum_{j=0}^{W-1} j^2 - (W-1)j + \frac{(W-1)^2}{4} \tag{4.27}$$

$$= \frac{\epsilon^2}{W}\left(\frac{W(W-1)(2W-1)}{6} - (W-1)\frac{W(W-1)}{2} + \frac{W(W-1)^2}{4}\right) \tag{4.28}$$

$$= \epsilon^2\frac{(W^2-1)}{12} \tag{4.29}$$

Hence combining (Eq. (4.24)) and (Eq. (4.29)) together into (Eq. (4.16)) we have

$$\text{MSE}^{(k)} \leq \frac{1+c(W-1)}{W^2}\sum_i \mathbf{\Sigma}_{ii}^{(k)} + \epsilon^2\frac{(W^2-1)}{12} \tag{4.30}$$

and so incorporating it into (Eq. (4.10)) we get

$$\text{MSE}_W = \frac{1}{K}\sum_{k=1}^{K}\text{MSE}^{(k)} \tag{4.31}$$

$$\leq \frac{W}{T}\sum_{k=1}^{K}\left(\frac{(1+c(W-1))}{W^2}\sum_i \mathbf{\Sigma}_{ii}^{(k)} + \epsilon^2\frac{(W^2-1)}{12}\right) \tag{4.32}$$

$$= \frac{(1+c(W-1))}{WT}\sum_i \mathbf{\Sigma}_{ii} + \epsilon^2\frac{(W^2-1)}{12} \tag{4.33}$$

$$= \frac{(1+c(W-1))}{W}\text{MSE}_1 + \epsilon^2\frac{(W^2-1)}{12} \qquad \square$$

For sufficiently small ϵ and c we have with certainty, $MSE_W < MSE_1$ for some $W > 1$.

Discussion We assume that means $\boldsymbol{\mu}$ can be written as the ϵ-Lipschitz function $\mu_{t+1} = \mu_t + \epsilon_t$, where $\epsilon_t \in [-\epsilon, \epsilon]$. Generally speaking, contiguous hyperparameters have optimal values which are close, and therefore have close hypergradients during outer optimization. This assumption breaks if hyperparameters are initialized randomly, and so we initialize all of our hyperparameters to zero in our experiments. Ideally, we would solve for whole inner loop several times so that we can use the mean hypergradient $[\mu_0, \mu_1, ..., \mu_H]$ for each individual step, without doing any sharing. While we consider this to be the optimal hypergradients, this is too expensive in practice, and instead we consider averaging hypergradients from neighbouring inner steps. The result above indicates that when contiguous hypergradients are sufficiently de-correlated (small c), we can reduce the mean squared error by a factor W compared to not averaging. However, if means μ_t drift over time by an amount $\epsilon_t \leq \epsilon$ this introduces some bias which increases the error and eventually results in $MSE_W > MSE_1$.

Finally, it is worth considering the simpler scenario when each hypergradient is considered to be drawn independently, i.e. $\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2\mathbf{1})$. In that case, $c = 0$ and the mean squared errors become:

$$\text{MSE}_1 = \sigma^2 \tag{4.34}$$

$$\text{MSE}_W \leq \frac{\text{MSE}_1}{W} + \epsilon^2\frac{(W^2-1)}{12} \tag{4.35}$$

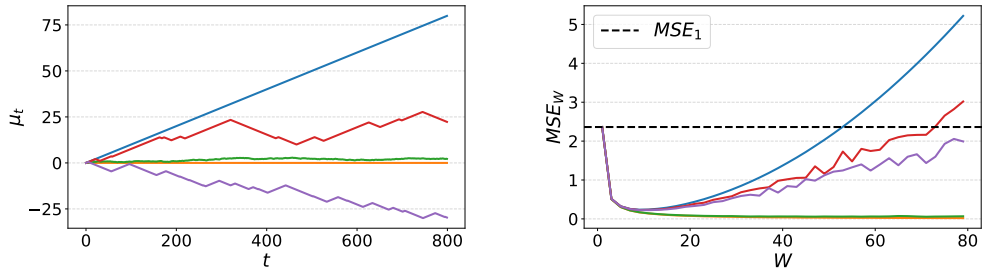

Figure 4.5: Several μ_t profiles and their corresponding mean squared error when sharing over W contiguous steps, as a function of W. The blue and yellow curve correspond to the maximal and minimal drifts scenarios respectively.

Visualizing the MSE for various μ_t profiles. Since the mean squared error depends on the specific shape of μ_t, we sample random μ_t profiles and show how their MSE evolves as a function of W. This illustrates how tight the upper bound is.

Appendix C: Implementation Details

We use a GeForce RTX 2080 Ti GPU for all experiments. We found that much of the literature on greedy methods uses the test set as the validation set, which creates a risk of meta-overfitting to the test set. Instead, we always carve out a validation set from our training set.

Figure 1 Here we used very similar settings as Figure 4 for FDS, except we learned 7 learning rates to make the search space a bit more challenging. We use the HyperBanster HPO package for RS, BO, HB and BOHB. For HB and BOHB, the minimum budget argument is set to 1 epoch to allow for lots of fast evaluations, and the maximum budget is set to 50 epochs. We also use this technique to bring down our convergence time from $\sim$ 10 hours to $\sim$ 3 hours, namely we calculate hypergradients based on 10 epochs for some outer steps, rather than calculate all hypergradients on 50 epochs. Since HD needs the user to specify initial hyperparameter values, we random search over those for several consecutive HD runs.

Figure 2 We calculate the hypergradient with respect to some learning rate schedule over 100 seeds, where each seed corresponds to a different training set ordering and network initialization. The learning rate schedule is fixed, and initialized to be a cosine decay over the full 250 batches, starting at 0.01. The batch size is set to 128, and 1000 fixed images are used for the validation data.

Figure 3 Here we used a batch size of 128 for both datasets to allow 1 epoch worth of inner optimization in about 500 inner steps. Clipping was restricted to ± 3 to show the effect of noisy hypergradients more clearly. Since MNIST and SVHN are cheap datasets to run on a LeNet architecture, we can afford 50 outer steps and early stopping based on validation accuracy. All learning rates were initialized to zero.

Figure 4 We learn 5 values for the learning rates, 1 for the momentum and 1 for the weight decay, to make it comparable to the hyperparameters used in the literature for CIFAR-10. A batch size 256 is used, with 5% of the training set of each epoch set aside for validation. We found larger validation sizes not to be helpful. Hypergradient descent uses hyperparameters initialized at zero as well, and trains all hyperparameters online with an SGD outer optimizer with learning rate 0.2 and ± 1 clipping of the hypergradients. As described in appendix G, we used a sign based outer optimizer with adaptive step sizes rather than some hand-tuned outer learning rate schedule. We used initial values $\gamma_\alpha = 0.1, \gamma_\beta = 0.15$ and $\gamma_\xi = 4 \times 10^{-4}$ but the performance barely changed when these values were multiplied or divided by 2. Since we take 10 outer steps and initialize all hyperparameters at zero, this defines a search ranges: $\alpha \in [-1, 1]$, $\beta \in [-1.5, 1.5]$, and $\gamma \in [-4 \times 10^{-3}, 4 \times 10^{-3}]$. The Hessian matrix product is clipped to ± 10 to prevent one batch from having a dominating contribution to hypergradients.

Appendix D: Other hypergradient examples

Fig. 4.1 depends on the value of $\boldsymbol{\alpha}$ at which hypergradients are calculated. For some learning rate schedules, contiguous hypergradients are sampled from closer distribution (ϵ small) and so sharing over larger windows is beneficial, as shown in the figure below.

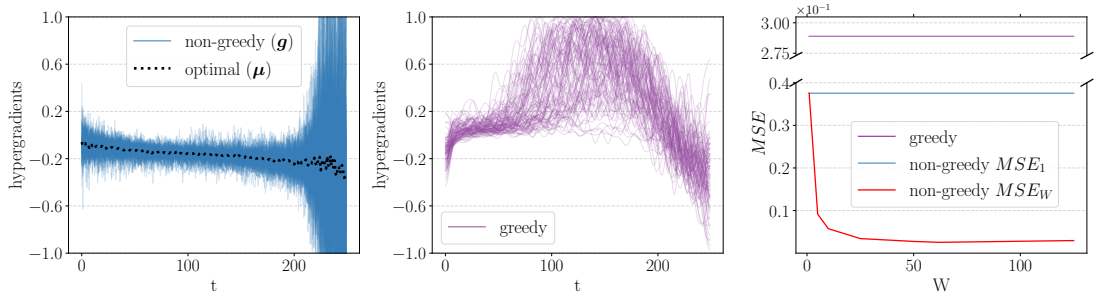

Figure 4.6: Similar to Fig. 4.1 but for a smaller ϵ. We can see that averaging hypergradients helps even more.

Appendix E: Hypergradients

Here we provide the raw hypergradients corresponding to the outer optimization shown in Appendices: Figure 1. Note that the range of these hypergradients is made reasonable by the averaging of gradients coming from contiguous hyperparameters.

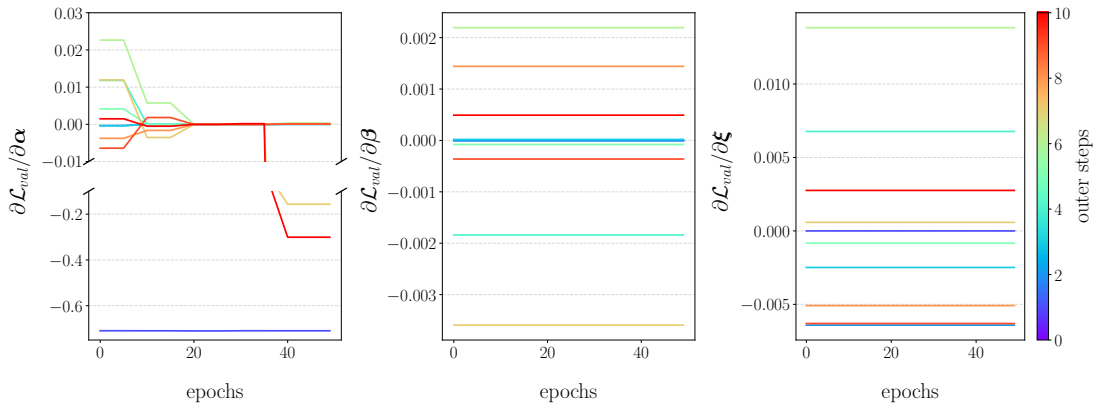

Figure 4.7: Hypergradients have a reasonable range but fail to always converge to zero when the validation performance stops improving.

Appendix F: Baselines

The objective here is to select the best hyperparameter setting that a deep learning practitioner would reasonably be expected to use in our experimental setting, based on the hyperparameters used by the community for the datasets at hand. For CIFAR-10, the most common hyperparameter setting is the following: α is initialized at $\alpha_0 = 0.2$ (for batch size 256, as used in our experiments) and decayed by a factor $\eta = 0.2$ at $30\%, 60\%$ and 80% of the run (`MultiStep` in Pytorch); the momentum β is constant at 0.9, and the weight decay ξ is constant at 5×10^{-4}. We search for combinations of hyperparameters around this setting. More specifically, we search over all combinations of $\alpha_0 = \{0.05, 0.1, 0.2, 0.4, 0.6\}$, $\eta = \{0.1, 0.2, 0.4\}$, $\beta = \{0.45, 0.9, 0.99\}$, and $\xi = \{2.5 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}$. This makes up a total of 135 hyperparameter settings, which we each run 3 times to get a mean and standard deviation. The distribution of those means are provided in Fig. 4.8, and the best hyperparameter setting is picked based on validation performance, which corresponds to $89.2 \pm 0.2\%$. Preliminary experiments showed that using schedules for the momentum and weight decay did not improve test accuracy.

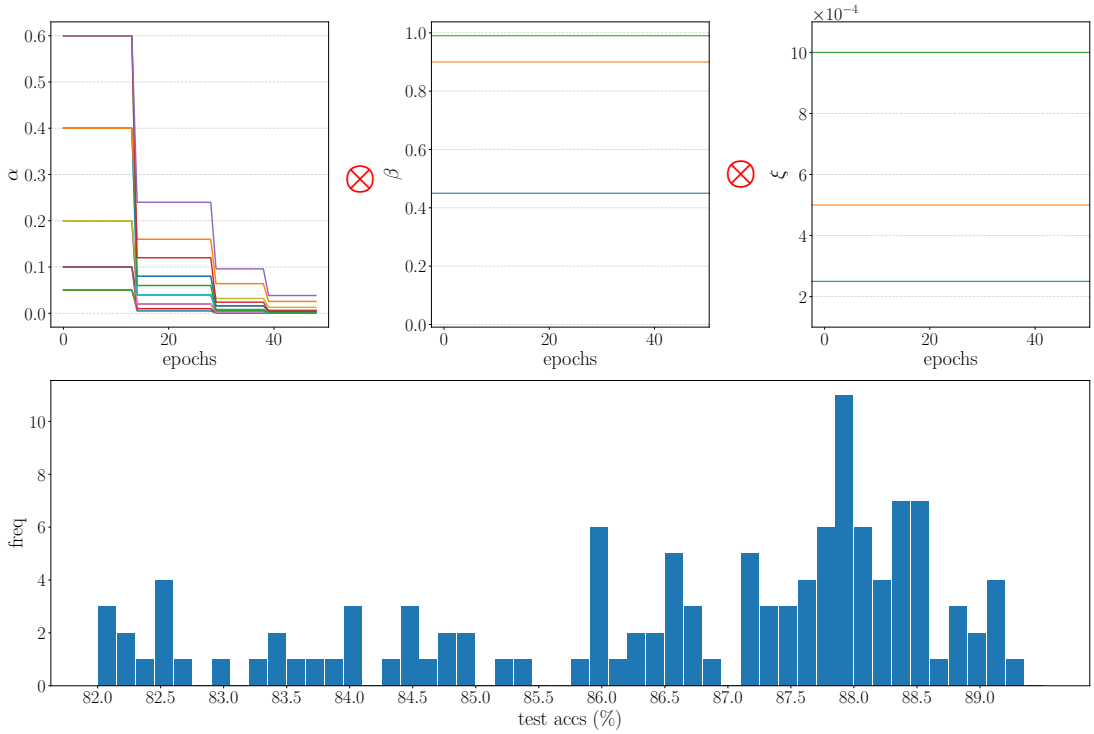

Figure 4.8: The combination of hyperparameters searched over for CIFAR-10 (top row) and the corresponding distribution of test accuracies (bottom row).

Appendix G: Using Hypergradient Signs

While hyperparameter sharing produces stable hypergradients with a reasonable range (see Appendix E), tuning the outer learning rate schedule can be tedious and unintuitive, and doesn't allow the user to specify a range to search hyperparameters over. A simple and fairly common learning rate schedule consists in decaying the learning rate (e.g. by a factor of 2) every time the gradient changes sign (Jacobs, 1988). The idea of using the sign of gradients to improve the efficiency of gradient descent dates back to the RPROP optimizer (Riedmiller and Braun, 1993). More recently, gradient signs have also been used to improve efficiency in the context of distributed learning (Bernstein et al., 2018), which has led to the discovery of robust convergence properties of sign-based SGD even in the case of biased gradients (Safaryan and Richtárik, 2019). Using such a learning rate schedule for the outer optimizer frees us from having to tune the outer learning rate, but fails to define a search range for HPO. We can achieve

this by updating each hyperparameter by an amount $\text{sgn}(g) \times \gamma$, and letting $\gamma \leftarrow \gamma/2$ when the hypergradient g changes sign across 2 consecutive outer steps. This allows for convergence after hypergradients have changed signs a few times. Being able to define the range of hyperparameter search more explicitly is especially useful to compare FDS to other HPO algorithms which also use a fixed search range (Sec. 4.6.4).

Chapter 5

Recurrence without Recurrence: Stable Video Landmark Detection with Deep Equilibrium Models

This chapter is about my paper "Recurrence without Recurrence: Stable Video Landmark Detection with Deep Equilibrium Models", which is currently under review. The background information needed to understand this chapter is already given in Sec. 2.4.2. Basic knowledge of landmark detection is assumed.

5.1 Introduction

The field of facial landmark detection has been fueled by important applications such as face recognition (Masi et al., 2018; Wang and Deng, 2018), facial expression recognition (Li and Deng, 2020; Jung et al., 2015; Kim et al., 2017; Yan et al., 2016; Hasani and Mahoor, 2017), and face alignment (Zhu et al., 2015; R et al., 2015; Zhang et al., 2013). Typically, facial landmarks are considered a valuable summary statistics of the image because of their compactness and robustness to natural image variations. Early approaches to landmark detection relied on a statistical model of the global face appearance and shape (Edwards et al., 1998; Cootes et al., 2001; Cristinacce and Cootes, 2006), but this was then superseded by deep learning regression models (Wu et al., 2018a; Feng et al., 2018; Xia et al., 2022). Both traditional and modern approaches have relied upon cascaded computation, an approach which starts with an initial guess of the landmarks and iteratively produces corrected landmarks which match the input

face more finely. These iterations typically increase the training memory cost linearly, and do not have an obvious stopping criteria. To solve these issues, we adapt the recently proposed Deep Equilibrium Model (Bai et al., 2019; Bai et al., 2020; Bai et al., 2021) to the setting of landmark detection. We show that our Landmark DEQ (LDEQ) achieves state-of-the-art performance on the WFLW dataset, while enjoying a constant memory with the number of cascaded iterations, and a natural stopping criteria.

Furthermore, we explore the benefits of DEQs in landmark detection from facial videos. Since obtaining landmark annotation for videos is notoriously expensive, models are virtually always trained on still images and applied frame-wise on videos at inference time. When a sequence of frames have ambiguous landmarks (*e.g.*, occluded faces or motion blur), it often leads to flickering landmarks, which rapidly oscillate between different possible configurations across consecutive frames. This poor temporal coherence is particularly problematic in applications where high precision is required, which is typically the case for facial landmarks. These applications include face transforms (MediaPipe, 2020; MediaPipe, 2021), face reenactment (Zhang et al., 2019), video emotion recognition (Jung et al., 2015; Kim et al., 2017; Yan et al., 2016; Hasani and Mahoor, 2017), movie dubbing (Garrido et al., 2015) or tiredness monitoring (Jabbar et al., 2020). Since DEQs solve a root finding problem in the forward pass, we propose to modify their objective at inference time to include a new recurrent loss term that encourages temporal coherence. We measure this improvement on our new WFLW-Video dataset (WFLW-V), demonstrating superiority over traditional filters, which typically reduce flickering at the cost of reducing landmark accuracy.

5.2 Related work

Deep Equilibrium Models (Bai et al., 2019) are part of the family of implicit models, which learn implicit functions such as the solution to an ODE (Chen et al., 2018; Dupont et al., 2019), or the solution to an optimization problem (Amos and Kolter, 2017; Djolonga and Krause, 2017; Wang et al., 2019a). These functions are called implicit

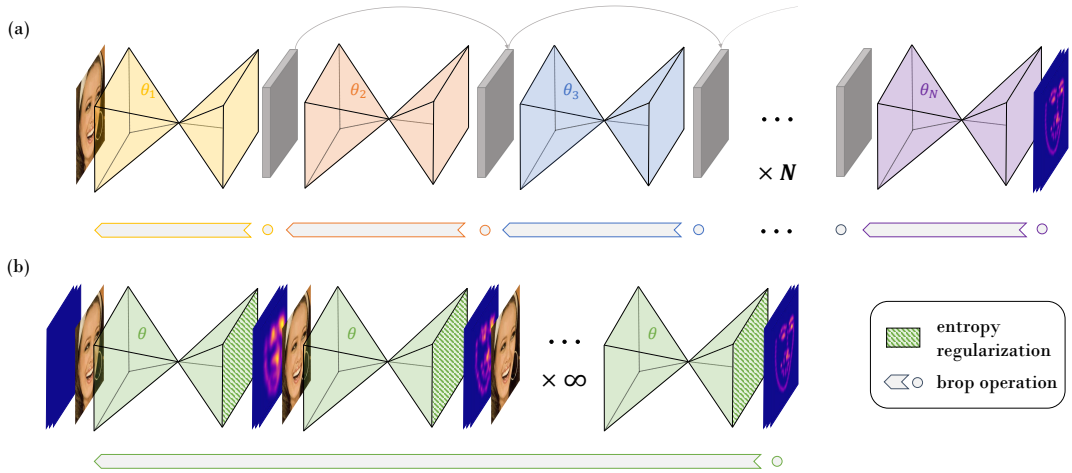

Figure 5.1: (a) Common Stacked-Hourglass cascaded architecture (Newell et al., 2016), whereby each refining stage increases the memory cost and the number of backpropagation operations. (b) Our LDEQ model, which adapts an equilibrium model (Bai et al., 2019) to the landmark detection setting, enjoys a constant memory cost as we increase the number of refining stages. At each stage, we compute landmark probability heatmaps, and encourage convergence to an equilibrium by lowering their entropy.

in the sense that the output cannot be written as a function of the input explicitly. In the case of DEQs, the output is the root of a function, and the model learned is agnostic to the root solver used. Early DEQ models were too slow to be competitive, and much work since has focused on better architecture design (Bai et al., 2020) and faster root solving (Bai et al., 2021; Fung et al., 2021; Geng et al., 2021). Since the vanilla formulation of DEQs does not guarantee convergence or uniqueness of an equilibrium, another branch of research has focused on providing convergence guarantees (Winston and Kolter, 2020; Revay et al., 2020; Pabbaraju et al., 2021a), which usually comes at the cost of a performance drop. Most similar to our work is the recent use of DEQs for videos in the context of optical flow estimation (Bai et al., 2022), where slightly better performance than LSTM-based models was observed.

A common theme throughout the development of landmark detection models has been the idea of cascaded compute, which has repeatedly enjoyed a better performance compared to single stage models (Wang and Deng, 2018). This is true in traditional models like Cascaded Pose Regression (CPR) (Dollár et al., 2010; Cao et al., 2012; Xiong and De la Torre, 2013; Sun et al., 2013; Asthana et al., 2014), but also in most

modern landmark detection models (Yang et al., 2017; Wu et al., 2018a; Wan et al., 2021; Kumar et al., 2020; Wang et al., 2019b; Huang et al., 2021; Lan et al., 2021), which usually rely on the Stacked-Hourglass backbone (Newell et al., 2016), or an RNN structure (Trigeorgis et al., 2016; Lai et al., 2018). In contrast to these methods, our DEQ-based model can directly solve for an infinite number of refinement stages at a constant memory cost and without gradient degradation. This is done by relying on the implicit function theorem (Krantz and Parks, 2003), as opposed to tracking each forward operation in autograd. Furthermore, our model naturally supports adaptive compute, in the sense that the number of cascaded stages will be determined by how hard finding an equilibrium is for a specific input, while the number of stages in, say, the Stacked-Hourglass backbone, must be constant at all times.

Our recurrence without recurrence approach is most closely related to test time adaptation methods. These have been most commonly developed in the context of domain adaptation (Sun et al., 2020; Wang et al., 2021), reinforcement learning (Hansen et al., 2021), meta-learning (Zhang et al., 2021b), generative models (Jiang et al., 2021; Mu et al., 2021), pose estimation (Li et al., 2021) or super resolution (Assaf Shocher, 2018). Typically, the methods above finetune a trained model at test time using a form of self-supervision. In contrast, our model doesn't need to be fine-tuned at test time: since DEQs solve for an objective function in the forward pass, this objective is simply modified at test time.

5.3 DEQs for landmark detection

Consider learning a landmark detection model F parameterized by $\boldsymbol{\theta}$, which maps an input image $\boldsymbol{x}$ to landmarks $\boldsymbol{z}$. Instead of directly having $\boldsymbol{z}$ as 2D landmarks, it is common for $\boldsymbol{z}$ to represent L heatmaps of size $D \times D$, where D is a hyperparameter (usually 64) and L is the number of landmarks to learn. While typical machine learning models can explicitly write down the function $\boldsymbol{z} = F(\boldsymbol{x}; \boldsymbol{\theta})$, in the DEQ approach this function is expressed implicitly by requiring its output to be a fixed point of another

function $f(\boldsymbol{z}, \boldsymbol{x}; \boldsymbol{\theta})$:

$$F : \boldsymbol{x} \rightarrow \boldsymbol{z}^* \quad \text{s.t.} \quad \boldsymbol{z}^* = f(\boldsymbol{z}^*, \boldsymbol{x}; \boldsymbol{\theta}) \tag{5.1}$$

where $\boldsymbol{z}^*$ denotes the fixed point, or equivalently the root of $g(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) = f(\boldsymbol{z}, \boldsymbol{x}; \boldsymbol{\theta}) - \boldsymbol{z}$. The function f must have inputs and outputs of similar shape, but beyond this restriction there is still limited understanding of its desired properties in machine learning. For simplicity, we build f from the ubiquitous hourglass module h (similar to a Unet):

$$f(\boldsymbol{z}, \boldsymbol{x}; \boldsymbol{\theta}) = \sigma(h([\boldsymbol{x}, \boldsymbol{z}]; \boldsymbol{\theta})) \tag{5.2}$$

where h inputs the concatenation of $\boldsymbol{x}$ and $\boldsymbol{z}$, and σ is a normalization function. For clarity, we omit from our notation that image x is downsampled with a few convolutions to match the shape of $\boldsymbol{z}$.

To evaluate f in the forward pass, we must solve for its fixed point $\boldsymbol{z}^*$. When f is a contraction mapping, $f \circ f \circ f \circ \cdots \circ f(\boldsymbol{z}^{(0)})$ converges to a unique $\boldsymbol{z}^*$ for any initial heatmap $\boldsymbol{z}^{(0)}$. In practice, it is neither tractable or helpful to directly take an infinite number of fixed point iteration steps. Instead, it is common to achieve the same result by leveraging quasi Newtonian solvers like Broyden's method (Broyden, 1965) or Anderson acceleration (Anderson, 1965), which find $\boldsymbol{z}^*$ in fewer iterations. Similarly to the original DEQ model, we use $\boldsymbol{z}^{(0)} = \mathbf{0}$ when training our LDEQ on still images.

Guaranteeing the existence of a unique fixed point by enforcing contraction restrictions on f is cumbersome, and better performance can often be obtained by relying on regularization heuristics that are conducive to convergence, such as weight normalization and variational dropout (Bai et al., 2020). In our landmark model, we did not find these tricks helpful, and instead used a simple normalization of the heatmaps to $[0, 1]$ at each refining stage:

$$\sigma(\boldsymbol{z}) = \exp\left(\frac{\boldsymbol{z} - \max(\boldsymbol{z})}{T}\right) \tag{5.3}$$

where T is a temperature hyperparameter. This layer also acts as an entropy regularizer, since it induces low-entropy ("peaked") heatmaps, which we found to improve convergence of root solvers.

We contrast our model in Fig. 5.1 to the popular Stacked-Hourglass backbone (Newell et al., 2016). Contrary to this model, our DEQ-based model uses a single hourglass module which updates $\boldsymbol{z}$ until an equilibrium is found. The last predicted heatmaps, $\boldsymbol{z}^*$, are converted into 2D landmark points $\hat{\boldsymbol{p}} = \Phi(\boldsymbol{z}^*)$ using the softargmax function $\Phi(\boldsymbol{z})$ proposed in (Luvizon et al., 2017). These points are trained to match the ground truth $\boldsymbol{p}$, and so the DEQ landmark training problem can be seen as a constrained optimization:

$$\begin{aligned} \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \quad & \mathcal{L}_{\text{MSE}}\left(\Phi(\boldsymbol{z}^*), \boldsymbol{p}\right) \\ \text{s.t.} \quad & \boldsymbol{z}^* = f(\boldsymbol{z}^*, \boldsymbol{x}; \boldsymbol{\theta}) \end{aligned} \tag{5.4}$$

To differentiate our loss function $\mathcal{L}_{\text{MSE}}$ through this root solving process, the implicit function theorem is used (Bai et al., 2019) to derive

$$\frac{\partial \mathcal{L}_{\text{MSE}}}{\partial \boldsymbol{\theta}} = -\frac{\partial \mathcal{L}_{\text{MSE}}}{\partial \boldsymbol{z}^*} \mathcal{J}_{\boldsymbol{g}\boldsymbol{z}^*}^{-1} \frac{\partial f(\boldsymbol{z}^*, \boldsymbol{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{5.5}$$

where the first two terms on the RHS can be expressed as the solution to a fixed point problem as well. Solving for this root in the backward pass means that we do not need to compute or store the expensive inverse Jacobian term $\mathcal{J}_{\boldsymbol{g}\boldsymbol{z}^*}^{-1}$ directly (Bai et al., 2019; Zico Kolter, 2019). Importantly, this backward-pass computation only depends on $\boldsymbol{z}^*$ and doesn't depend on the operations done in the forward pass to reach an equilibrium. This means that these operations do not need to be tracked by autograd, and therefore that training requires a memory cost of O(1), despite differentiating through a potentially infinite recurrence.

5.4 Recurrence without recurrence

Low temporal coherence (i.e. a large amount of flicker) is illustrated in Fig. 5.2. This can be a nuisance for many applications that require consistently precise landmarks for video. In this section, we describe how our LDEQ model can address this challenge by enabling recurrence at test time without recurrence at training time (RwR).

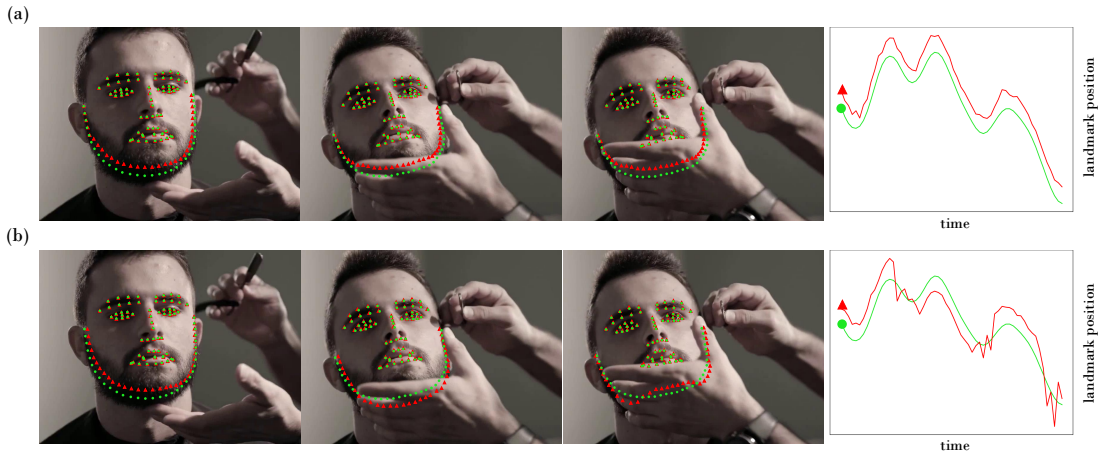

Figure 5.2: Model predictions (▲) and ground truth landmarks (●) for (a) a model with high temporal coherence and (b) a model with the same accuracy but exhibiting a worse temporal coherence, due to ambiguity at the chin. This causes flickering around the ground truth, as illustrated in the landmark trajectory (right). This flickering is common when video frames are evaluated individually, as opposed to recurrently.

Recall that in the DEQ formulation of Sec. 5.3, there is no guarantee that a unique fixed point solution exists. This can be a limitation for some applications, and DEQ variants have been proposed to allow provably unique solutions at the cost of additional model complexity (Winston and Kolter, 2020; Pabbaraju et al., 2021b). In this work, we instead propose a new paradigm: we leverage the potentially large solution space of DEQs after training to allow for some additional objective at inference time. This new objective is used to disambiguate which fixed point of $f(\boldsymbol{z}, \boldsymbol{x}; \boldsymbol{\theta}^*)$ is found, in light of extra information present at test time. We demonstrate this approach for the specific application of training a landmark model on face images and evaluating it on videos. In this case, DEQs allow us to include a recurrent loss term at inference time, which isn't achievable with conventional architectures.

Let $f(\boldsymbol{z}, \boldsymbol{x}; \boldsymbol{\theta}^*)$ be our DEQ model trained as per the formulation in Eq. (5.4). We would now like to do inference on a video of N frames $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N$. Consider that a given frame $\boldsymbol{x}_n$ has a corresponding set of fixed points $\mathcal{Z}_n^* = \{\boldsymbol{z} \quad \text{s.t.} \quad f(\boldsymbol{z}, \boldsymbol{x}_n; \boldsymbol{\theta}^*) = \boldsymbol{z}\}$, representing plausible landmark heatmaps. If we select some $\boldsymbol{z}_n^* \in \mathcal{Z}_n^*$ at random for each frame n, the corresponding heatmaps $\boldsymbol{z}_1^*, \boldsymbol{z}_2^*, \cdots, \boldsymbol{z}_N^*$ often exhibit some flickering artefacts, whereby landmarks rapidly change across contiguous frames (see Fig. 5.2).

We propose to address this issue by choosing the fixed point at frame n that is closest to the fixed point at frame $n-1$. This can be expressed by the following constrained optimization:

$$\boldsymbol{z}_n^* = \arg\min_{\boldsymbol{z}} \left\| \boldsymbol{z} - \boldsymbol{z}_{n-1}^* \right\|_2^2 \tag{5.6a}$$

$$\text{s.t.} \quad f(\boldsymbol{z}, \boldsymbol{x}_n; \boldsymbol{\theta}^*) = \boldsymbol{z} \tag{5.6b}$$

The problem above is equivalent to solving for the saddle point of a Lagrangian as follows:

$$\min_{\boldsymbol{z}} \max_{\boldsymbol{\lambda}} \left\| \boldsymbol{z} - \boldsymbol{z}_{n-1}^* \right\|_2^2 + \boldsymbol{\lambda}^T (f(\boldsymbol{z}, \boldsymbol{x}_n; \boldsymbol{\theta}^*) - \boldsymbol{z}) \tag{5.7}$$

where $\boldsymbol{\lambda}$ are Lagrange multipliers. Effectively, we are using the set of fixed points $\mathcal{Z}_n^*$ in Eq. (5.6b) as the trust region for the objective in Eq. (5.6a). In practice, adversarial optimization is notoriously unstable, as is typically observed in the context of GANs (Goodfellow et al., 2014; Arjovsky et al., 2017; Srivastava et al., 2017). Furthermore, this objective breaks down if $\mathcal{Z}_n^* = \emptyset$ for any $\boldsymbol{x}_n$. We can remedy both of these problems by relaxing the inference time optimization problem to:

$$\min_{\boldsymbol{z}} \| f(\boldsymbol{z}, \boldsymbol{x}_n; \boldsymbol{\theta}^*) - \boldsymbol{z} \|_2^2 + \frac{\alpha}{2} \left\| \boldsymbol{z} - \boldsymbol{z}_{n-1}^* \right\|_2^2 \tag{5.8}$$

where α is a hyperparameter that trades off fixed-point solver error vs. the shift in heatmaps across two consecutive frames. This objective can be more readily tackled with Newtonian optimizers like L-BFGS (Liu and Nocedal, 1989). When doing so, our DEQ at inference time can be described as a form of OptNet (Amos and Kolter, 2017), albeit without any of the practical limitations (*e.g.*, quadratic programs) related to making gradient calculations cheap.

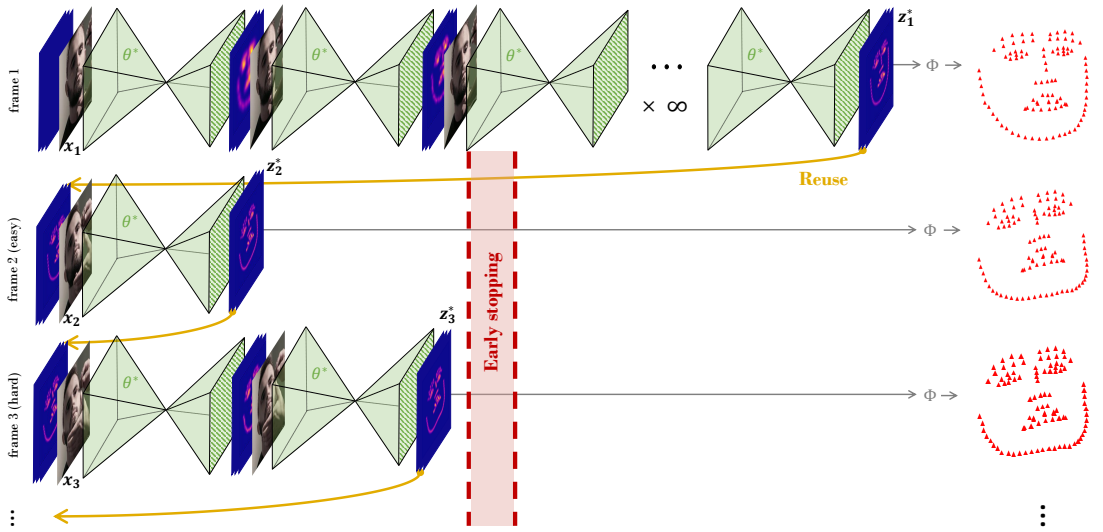

Figure 5.3: Recurrence without recurrence (RwR) on video data, from an LDEQ that was trained on still images. We use the initialization $\boldsymbol{z}_1^{(0)} = \mathbf{0}$ for the first frame, and then reuse $\boldsymbol{z}_n^{(0)} = \boldsymbol{z}_{n-1}^*$ for $n > 1$. Combined with early stopping, this is equivalent to regularizing the fixed point $\boldsymbol{z}_n$ so that it is more temporally coherent with all its predecessors.

Converting our root solving problem into an optimization problem during the forward pass of each frame can significantly increases inference time. Thankfully, the objective in Eq. (5.8) can also be solved by using root solvers. First, note that it is equivalent to finding the MAP estimate given a log likelihood and prior:

$$\log p(\boldsymbol{x}_n|\boldsymbol{z};\boldsymbol{\theta}^*) \propto -\|f(\boldsymbol{z},\boldsymbol{x}_n;\boldsymbol{\theta}^*) - \boldsymbol{z}\|_2^2 \tag{5.9a}$$

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z};\boldsymbol{z}_{n-1},\alpha^{-1}\mathbb{1}) \tag{5.9b}$$

It has been demonstrated in various settings that this prior, when centered on the initialization to a search algorithm, can be implemented by early stopping this algorithm (Sjöberg and Ljung, 1992; Bishop, 1995; Santos, 1996; Grant et al., 2018). As such, we can approximate the solution to Eq. (5.6a-5.6b) by simply initializing the root solver with $\boldsymbol{z}_n^{(0)} = \boldsymbol{z}_{n-1}^*$ ("reuse") and imposing a hard limit on the number of steps that it can take ("early stopping"). We call this approach Recurrence without Recurrence (RwR), and illustrate it in Fig. 5.3.

5.5 Video landmark coherence

In this section we describe the metric (NMF) and the dataset (WFLW-V) that we contribute to measure the amount of temporal coherence in landmark videos. These are later used to benchmark the performance of our RwR paradigm against alternatives in Sec. 5.6.2.

5.5.1 NMF: a metric to track temporal coherence

The performance of a landmark detection model is typically measured with a single metric called the Normalized Mean Error (NME). Consider a video sequence of N frames, each containing L ground truth landmarks. A single landmark point is a 2D vector denoted $\boldsymbol{p}_{n,l} \in \mathcal{R}^2$, where n and l are the frame and landmark index respectively. Let $\boldsymbol{r}_{n,l} = \boldsymbol{p}_{n,l} - \hat{\boldsymbol{p}}_{n,l}$ be the residual vector between ground truth landmarks and predicted landmarks $\hat{\boldsymbol{p}}_{n,l}$. The NME simply averages the ℓ_2 norm of this residual across all landmarks and all frames:

$$\text{NME}_n = \frac{1}{L} \sum_{l=1}^{L} \frac{\|\boldsymbol{r}_{n,l}\|}{d_0} \tag{5.10a}$$

$$\text{NME} = \frac{1}{N} \sum_{n=1}^{N} \text{NME}_n \tag{5.10b}$$

Here d_0 is usually the inter-ocular distance, and aims to make the NME better correlate with the human perception of landmark error. We argue that this metric alone is insufficient to measure the performance of a landmark detector in videos. In Fig. 5.2 we show two models of equal NME but different coherence in time, with one model exhibiting *flickering* between plausible hypotheses when uncertain. This flickering is a nuisance for many applications, and yet is not captured by the NME. This is in contrast to random noise (*jitter*) which is unstructured and already reflected in the NME metric.

To measure temporal coherence, we propose a new metric called the Normalized Mean Flicker (NMF). We design this metric to be orthogonal to the NME, by making it agnostic to the magnitude of $\boldsymbol{r}_{n,l}$, and only focusing on the change of $\boldsymbol{r}_{n,l}$ across consecutive frames:

$$\text{NMF}_n = \sqrt{\frac{1}{L}\sum_{l=1}^{L}\frac{\|\boldsymbol{r}_{n,l} - \boldsymbol{r}_{n-1,l}\|^2}{d_1^2}} \tag{5.11a}$$

$$\text{NMF} = \sqrt{\frac{1}{N}\sum_{n=2}^{F}\text{NMF}_n^2} \tag{5.11b}$$

We replace the means in the NME with a root mean square to better represent the human perception of flicker. Indeed, this penalizes a short sudden changes in $\boldsymbol{r}_{n,l}$ compared to the same change smoothed out in time and space. The value d_1^2 is chosen to be the face area. This prevents a long term issue with the NME score, namely the fact that large poses can have an artificially large NME due to having a small d_0.

5.5.2 A new landmark video dataset: WFLW-V

Figure 5.4: Example of ground truth labels (●) obtained semi-automatically from an ensemble of 45 models (▲). Ensembling these diverse models provides ground truth labels that do not flicker and can thus be used to measure flickering against. Note how landmark points with the most uncertainty are the most prone to having flickering predictions across consecutive frames.

Due to the tedious nature of producing landmark annotations for video data, there are few existing datasets for face landmark detection in videos. Shen *et al*. have proposed 300-VW (Shen et al., 2015), a dataset made up of $\sim$ 100 videos using the 68-point landmark scheme from the 300-W dataset (Sagonas et al., 2013). Unfortunately, two major issues make this dataset unpopular: 1) it was labelled using fairly weak models from (Chrysos et al., 2015) and (Tzimiropoulos, 2015) which results in many labelling errors and high flicker (see Appendix B), and 2) it only contains 100 videos of little diversity, many of which being from the same speaker, or from different speakers in the same environment. Taken together, these two issues mean that the performance of modern models on 300-WV barely correlates with performance on real-world face videos.

We propose a new video dataset for facial landmarks: WFLW-Video (WFLW-V). It consists of 1000 Youtube creative-commons videos (*i.e.*, an order of magnitude more than its predecessor) and covers a wide range of people, expressions, poses, activities and background environments. Each video is 5s in length. These videos were collected by targeting challenging faces, where ground truth landmarks are subject to uncertainty. The dataset contains two subsets, *hard* and *easy*, made up of 500 videos each. This allows debugging temporal filters and smoothing techniques, whose optimal hyperparameters are often different for videos with little or a lot of flicker. This split was obtained by scraping 2000 videos, and selecting the top and bottom 500 videos based on the variance of predictions in the ensemble. To evaluate a landmark model on the WFLW-V dataset, we train it on the WFLW training set, and evaluate it on all WFLW-V videos. This pipeline best reflects the way large landmark models are trained in the industry, where labelled video data is scarce.

Contrary to the 68-landmark scheme of the 300-VW dataset, we label videos semi-automatically using the more challenging 98-landmark scheme from the WFLW dataset (Wu et al., 2018a), as it is considered the most relevant dataset for future research in face landmark detection (Khabarlak and Koriashkina, 2022). To produce ground truth labels, we train an ensemble of 45 state-of-the-art models using a wide range of data

Method		Params (M)	Full	Large poses	Expressions	Illumination	Makeup	Occlusion	Blur
LAB (Wu et al., 2018a)	CVPR	12.3	5.27	10.24	5.51	5.23	5.15	6.79	6.32
Wing (Feng et al., 2018)	CVPR	25	4.99	8.43	5.21	4.88	5.26	6.21	5.81
MHHN (Wan et al., 2021)	TIP	-	4.77	9.31	4.79	4.72	4.59	6.17	5.82
DecaFA (Dapogny et al., 2019)	ICCV	10	4.62	8.11	4.65	4.41	4.63	5.74	5.38
HRNet (Sun et al., 2019)	TPAMI	9.7	4.60	7.94	4.85	4.55	4.29	5.44	5.42
AS (Qian et al., 2019)	ICCV	35	4.39	8.42	4.68	4.24	4.37	5.60	4.86
LUVLI (Kumar et al., 2020)	CVPR	-	4.37	7.56	4.77	4.30	4.33	5.29	4.94
AWing (Wang et al., 2019b)	ICCV	24.2	4.36	7.38	4.58	4.32	4.27	5.19	4.96
SDFL (Lin et al., 2021)	TIP	-	4.35	7.42	4.63	4.29	4.22	5.19	5.08
SDL (Li et al., 2020b)	ECCV	-	4.21	7.36	4.49	4.12	4.05	4.98	4.82
ADNet (Huang et al., 2021)	ICCV	13.4	4.14	6.96	4.38	4.09	4.05	5.06	4.79
SLPT (Xia et al., 2022)	CVPR	19.5	4.12	6.99	4.37	**4.02**	4.03	5.01	4.79
HIH (Lan et al., 2021)	-	22.7	4.08	6.87	4.06	4.34	3.85	4.85	4.66
LDEQ (ours)	-	21.8	**3.92**	**6.86**	**3.94**	4.17	**3.75**	**4.77**	**4.59**

Table 5.1: Performance of our model and previous state-of-the-art on the various WFLW subsets, using the NME metric (↓) for comparison. Models using pre-training on other datasets have been excluded for fair comparison (Bulat et al., 2021; Zheng et al., 2021; Yu and Tao, 2021).

augmentations of both the test and train set of WFLW (amounting to 10,000 images). We use a mix of large Unets, HRNets (Sun et al., 2019) and HRFormers (Yuan et al., 2021) to promote landmark diversity. The heatmaps of these models are averaged to produce the ground truth heatmap, allowing uncertain models to weight less in the aggregated output. Ensembling models provides temporal coherence without the need for using hand-tuned filtering or smoothing, which are susceptible to misinterpreting signal for noise (*e.g.* closing eye landmarks mimic high frequency noise).

We provide examples of annotated videos in Fig. 5.4. Note that regions of ambiguity, such as occluded parts of the face, correspond to a higher variance in landmark predictions. While the ground truth for some frames may be subjective (*e.g.* occluded mouth), having a temporally stable "best guess" is sufficient to measure flicker of individual models. We found that 45 models in the ensemble was enough to provide a low error on the mean for all videos in WFLW-V. While we manually checked that our Oracle was highly accurate and coherent in time, note that it is completely impractical to use it for real-world applications due to its computational cost ($\sim 2B$ parameters).

We checked each frame manually for errors, which were rare. When we found an error in annotation, we corrected it by removing inaccurate models from the ensemble for the frames affected, rather than re-labeling the frame manually. We found this approach to be faster and less prone to human subjectivity when dealing with ambiguous faces,

such as occluded ones. Occlusion has been characterized as one of the main remaining challenges in modern landmark detection (Wu and Ji, 2019), being most difficult in video data (Shen et al., 2015). Finally, the stability of our oracle also depends on the stability of the face bounding box detector. We found the most popular detector, MTCNN (Zhang et al., 2016) to be too jittery, and instead obtained stable detection by bootstrapping our oracle landmarks into the detector. More details about scraping and curating our dataset can be found in Appendix A.

5.6 Experiments

The aim of these experiments is to demonstrate that: 1) LDEQ is a state-of-the-art landmark detection model for high precision settings like faces, and 2) the LDEQ objective can be modified at inference time on videos to include a recurrence loss. This increases temporal coherence without decreasing accuracy, a common pitfall of popular filters.

Metric	Method	Full	Large poses	Expressions	Illumination	Makeup	Occlusion	Blur
FR_{10} (↓)	LAB	7.56	28.83	6.37	6.73	7.77	13.72	10.74
	HRNet	4.64	23.01	3.50	4.72	2.43	8.29	6.34
	AS	4.08	18.10	4.46	2.72	4.37	7.74	4.40
	LUVLi	3.12	15.95	3.18	2.15	3.40	6.39	3.23
	AWing	2.84	13.50	2.23	2.58	2.91	5.98	3.75
	SDFL	2.72	12.88	1.59	2.58	2.43	5.71	3.62
	SDL	3.04	15.95	2.86	2.72	1.45	5.29	4.01
	ADNet	2.72	12.72	2.15	2.44	1.94	5.79	3.54
	SLPT	2.72	11.96	1.59	**2.15**	1.94	5.70	3.88
	HIH	2.60	12.88	**1.27**	2.43	**1.45**	**5.16**	3.10
	LDEQ	**2.48**	**12.58**	1.59	2.29	1.94	5.36	**2.84**
AUC_{10} (↑)	LAB	0.532	0.235	0.495	0.543	0.539	0.449	0.463
	HRNet	0.524	0.251	0.510	0.533	0.545	0.459	0.452
	AS	0.591	0.311	0.549	0.609	0.581	0.516	0.551
	LUVLi	0.557	0.310	0.549	0.584	0.588	0.505	0.525
	AWing	0.572	0.312	0.515	0.578	0.572	0.502	0.512
	SDFL	0.576	0.315	0.550	0.585	0.583	0.504	0.515
	SDL	0.589	0.315	0.566	0.595	0.604	0.524	0.533
	ADNet	0.602	0.344	0.523	0.580	0.601	0.530	0.548
	SLPT	0.596	0.349	0.573	0.603	0.608	0.520	0.537
	HIH	0.605	0.358	0.601	0.613	0.618	0.539	0.561
	LDEQ	**0.624**	**0.373**	**0.614**	**0.631**	**0.631**	**0.552**	**0.574**

Table 5.2: AUC_{10} and FR_{10} on the WFLW test set.

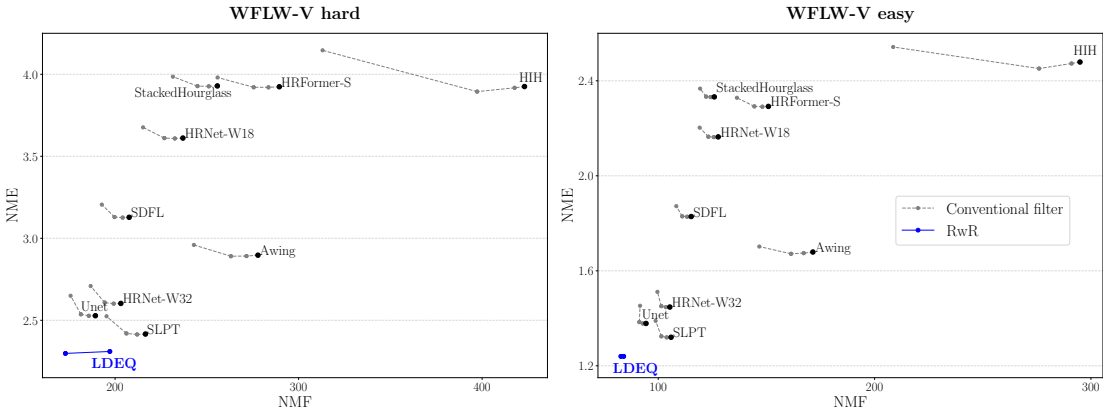

Figure 5.5: NME (accuracy) vs NMF (temporal coherence) for various models, on the hard (left) and easy (right) WFLW-V subsets. We compare our RwR scheme (blue) to a conventional filter (exponential moving average) using three different smoothing coefficients (gray). For hard videos susceptible to flicker, RwR on LDEQ decreases NMF by 12% without increasing NME, contrary to the conventional filter alternative. For easy videos that contain little to no flicker, conventional filters can increase both NME and NMF, while our model converges to the same fixed point with or without RwR. These results are given in tabular form in Appendix C.

5.6.1 Landmark accuracy

We compare the performance of LDEQ to state-of-the-art models on the WFLW dataset (Wu et al., 2018a), which is based on the WIDER Face dataset (Yang et al., 2016) and is made up of 7500 train images and 2500 test images. Each image is annotated with a face bounding box and 98 2D landmarks. Compared to previous datasets, WFLW uses denser facial landmarks, and introduces much more diversity in poses, expressions, occlusions and image quality (the test set is further divided into subsets reflecting these factors). This makes it uniquely appropriate for training landmark detectors meant to be deployed on real-world data, like the videos of WFLW-V.

The common evaluation metrics for the WFLW test set are the Normalized Mean Error (see Eq. (5.10b)), the Area Under the Curve (AUC), and the Failure Rate (FR). The AUC is computed on the cumulative error distribution curve (CED), which plots the fraction of images with NME less or equal to some cutoff, vs. increasing cutoff values. We report AUC_{10}, referring to a maximum NME cutoff of 10 for the CED curve. Higher AUC is better. The FR_X metric is equal to the percentage of test images whose NME is larger than X. We report FR_{10}; lower is better.

We train our LDEQ for 60 epochs using the pre-cropped WFLW dataset as per (Lan et al., 2021), and the common data augmentations for face landmarks: rotations, flips, translations, blurs and occlusions. We found the Anderson and fixed-point-iteration solvers to work best over the Broyden solver used in the original DEQ model (Bai et al., 2019; Bai et al., 2020). By setting a normalization temperature of $T = 2.5$, convergence to fixed points only takes around 5 solver iterations. The NME, AUC and FR performance of LDEQ can be found in tables 5.1 and 5.2. We outperform all existing models on all three evaluation metrics, usually dominating individual subsets as well, which is important when applying LDEQ to video data.

5.6.2 Landmark temporal coherence

We evaluate the performance of LDEQ on the WFLW-V dataset, for both landmark accuracy (NME) and temporal coherence (NMF). We use RwR with early stopping after 2 solver iterations, allowing some adaptive compute (1 solver iteration) in cases where two consecutive frames are almost identical. Our baselines include previous state-of-the-art models that have publicly available weights (Lan et al., 2021; Xia et al., 2022; Lin et al., 2021; Wang et al., 2019b), as well as common architectures of comparable parameter count, which we trained with our own augmentation pipeline. We apply a conventional filtering algorithm, the exponential moving average, to our baselines. This was found to be more robust than more sophisticated filters like Savitzky-Golay filter (Savitzky and Golay, 1964) and One Euro filter (Casiez et al., 2012).

The NME vs. NMF results are shown in Fig. 5.5 for all models, for the hard and easy WFLW-V subsets. For videos that contain little uncertainty in landmarks (WFLW-V easy), there is little flickering and conventional filtering methods can mistakenly smooth out high frequency signal (*e.g.* eyes and mouth moving fast). For videos subject to more flickering (WFLW-hard), these same filtering techniques do indeed improve the NMJ metric, but beyond a certain smoothing factor this comes at the cost of increasing the NME. In contrast, LDEQ + RwR correctly smoothes out flicker for WFLW-W

hard without compromising performance on WFLW-V easy. This improvement comes from the fact that the RwR loss in Eq. (5.8) contains both a smoothing loss plus a log likelihood loss that constrains output landmarks to be plausible solutions, while conventional filters only optimize for the former.

5.7 Conclusion

We adapt Deep Equilibrium Models to landmark detection, and demonstrate that our LDEQ model can reach state-of-the-art accuracy on the challenging WFLW facial dataset. We then bring the attention of the landmark community to a common problem in video applications, whereby landmarks flicker across consecutive frames. We contribute a new dataset and metric to effectively benchmark solutions to that problem. Since DEQs solve for an objective in the forward pass, we propose to change this objective at test time to take into account new information. This new paradigm can be used to tackle the flickering problem, by adding a recurrent loss term at inference that wasn't present at training time (RwR). We show how to solve for this objective cheaply in a way that leads to state-of-the-art video temporal coherence. We hope that our work brings attention to the potential of deep equilibrium models for computer vision applications, and in particular, the ability to add loss terms to the forward pass process, to leverage new information at inference time.

5.8 Appendices

Appendix A: More details on making our WFLW-V dataset

In this section we detail the procedure used to collect, label and curate the 1000 videos that make up the WFLW-V dataset.

Step 1: Video search. We start by producing a list of 100 YouTube search strings, that we think would be correlated with videos conducive to landmark uncertainty. These search strings fall within 7 categories: "Skin care & Makeup" (*e.g. how to put lipstick*), "Hair & Beard care" (*e.g. how to cut your own hair*), "Singing & Podcasts" (*e.g. how to setup your mic*), "Brass instruments" (*e.g. learn to play the French horn*"), "Eating" (*e.g. how to eat fast*), "Smoking" (*e.g. how to smoke the cigar*), and "Miscellaneous" (*e.g. how to brush your teeth*"). Each English search string is translated to 10 languages, to produce more diverse videos: French, German, Spanish, Italian, Portuguese, Catalan, Czech, Danish, Estonian, Dutch.

We use YouTube filters to search for videos less than 4 minutes long, and with a CC BY licence. This licence is the most permissive creator licence. It allows reusers to distribute, remix, adapt the video, and even to use it for commercial use. We only consider videos that have a frame rate between 24 and 31 fps inclusive. This is mostly to exclude all videos like kid cartoons that have very low fps. In total, this step produces around $15,000$ videos.

Step 2: Video cleaning. Our task is now to find 5s of contiguous *clean* face for each video. A *clean* face is a real human face, at least 20% visible, from a single person, without video or camera filters (*e.g.* face filters, jump cuts). We also limit the number of videos that come from the same youtuber, so as not to lower diversity. We use the most popular face detector, the Multi-task Cascaded Convolutional Networks (MTCNN) (Zhang et al., 2016) to help with video cleaning. In total, this leaves around $2,000$ videos.

Step 3: Video annotation. We use an oracle made up of 45 pretrained models, including 15 large Unets (larger than our LDEQ backbone), 15 HRNets-W48, and 15 HRFormer-B. We average the final heatmap of each model to create a mean heatmap, from which we extract our oracle predictions. We found that the bounding box from the MTCNN model are jittery, which in turns facilitates jitter and flicker for landmarks. To fix this we bootstrap our oracle to the bounding box detection. This is done by using the original MTCNN bounding box, finding landmarks, defining a new bounding box based

on the smallest/largest landmark coordinates and a scaling margin factor of 1.2, finding landmarks in this new bounding box, and so on. This is repeated for 3 iterations, after which the bounding box values have have converged. We use the landmark predictions on the final bounding box as our oracle predictions.

Step 4: Video verification. We verify each video frame by frame for issues. In some cases, videos are discarded because the degree of uncertainty or occlusion is too high that even a "human best guess" wouldn't be good. This includes videos with very large poses as well. In other cases, particularly for hard videos, some models in the ensemble are visibly mistaken. These models are singled out and removed for problematic frames. These cases are rare (~ 25) but worth correcting so we don't bias the dataset by only including videos where our oracle does best.

Step 5: Subset creation. Since we have access to all 45 model predictions in the ensemble, it is easy to see the average variance of these models for each video. This score correlates well with uncertainty, and we use it to rank all videos from hard to easy. We used the top 500 videos for WFLW-hard, and the bottom 500 for WFLW-easy.

Appendix B: Errors in 300-VW

The 300-VW dataset (Shen et al., 2015) was labelled using the now obsolete models from (Chrysos et al., 2015) and (Tzimiropoulos, 2015). This results in several labelling errors (Fig. 5.6) that have gone unnoticed. Errors in the ground truth of datasets lead to misleading insights and models that generalize poorly to real-world settings.

We also note that many (perhaps all) of the videos in 300-VW do not have a creative commons licence, and so the legality of their use for industrial research labs may be more ambiguous.

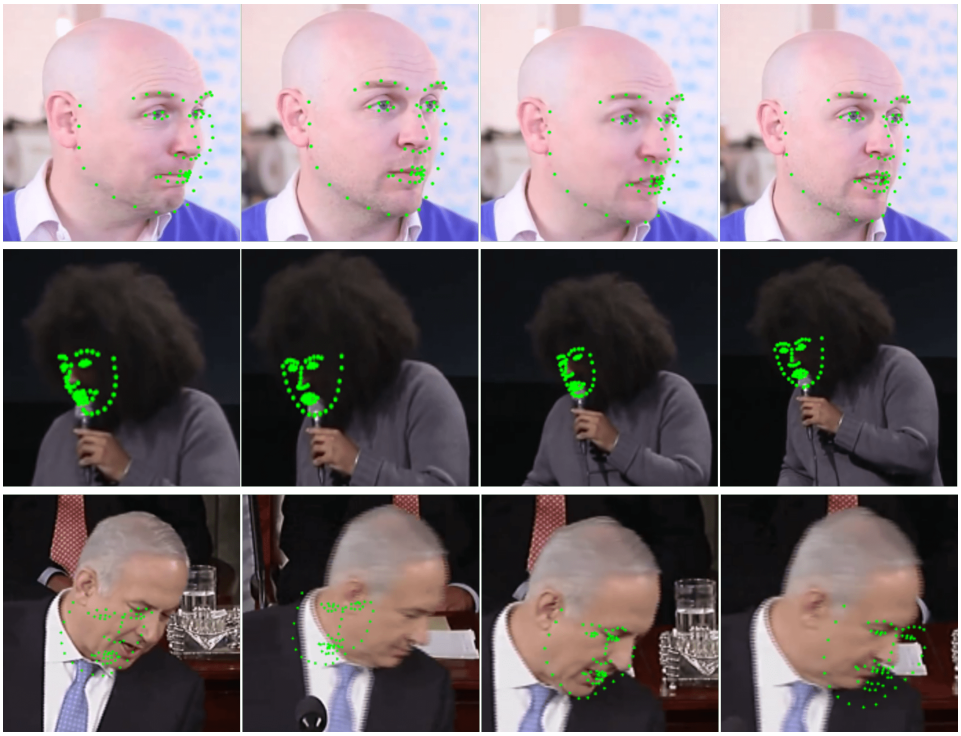

Figure 5.6: Examples of poorly labelled videos in the 300-VW dataset. We show three levels of labelling errors from top to bottom: medium, bad, very bad. Our new WFLW-V dataset uses much stronger labellers and was checked frame by frame to avoid such errors.

Appendix C: WFLW-V Results

We show the results of Figure 5 in tabular form in Tab. 5.3. We compare our RwR scheme to the exponential moving average (ema), and show that contrary to ema, our method can improve temporal coherence without lowering accuracy. We tried the following ema weights: [0.005, 0.01, 0.02, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]. When considering all baselines at once, we found that a weight 0.15 struck the best balance between lowering NMF without increasing NME too much. This was also better than the grid searched Savitzky-Golay filter (Savitzky and Golay, 1964) and One Euro filter (Casiez et al., 2012). The only exception was the HIH model which is both very jittery and flickery, and for which we used an ema weight of 0.3.

Method	WFLW-V hard		WFLW-V easy		WFLW-V FULL	
	NME	NMF	NME	NMF	NME	NMF
HIH	3.93	423.11	2.48	294.94	3.20	359.03
+ ema	4.15	313.07	2.54	208.60	3.34	260.84
StackedHourglass	3.93	255.74	2.33	125.91	3.13	190.82
+ ema	3.99	231.52	2.37	119.35	3.18	175.43
HRFormer-S	3.92	289.50	2.29	150.91	3.11	220.21
+ ema	3.98	255.85	2.33	136.37	3.15	196.11
HRNet-W18	3.61	236.96	2.16	127.72	2.89	182.34
+ ema	3.68	215.26	2.20	119.13	2.94	167.20
SDFL	3.13	207.68	1.83	115.22	2.48	161.45
+ ema	3.21	192.77	1.87	108.35	2.54	150.56
Awing	2.90	277.86	1.68	171.48	2.29	224.67
+ ema	2.96	242.95	1.70	146.71	2.33	194.83
HRNet-W32	2.60	203.15	1.45	105.35	2.03	154.25
+ ema	2.71	186.69	1.51	99.62	2.11	143.15
Unet	2.53	189.28	1.38	94.35	1.95	141.81
+ ema	2.65	175.76	1.45	91.58	2.05	133.67
SLPT	2.42	216.56	1.32	105.93	1.87	161.25
+ ema	2.52	195.35	1.39	98.79	1.96	147.07
DEQ	2.31	197.16	1.24	84.03	1.77	140.59
+ RwR	2.30	172.95	1.24	82.74	**1.77**	**127.85**

Table 5.3: NME and NMF on the WFLW-V dataset, comparing the effect of an exponential moving average smoothing (ema) with our recurrence without recurrence scheme.

Finally, we found that more augmentations can help performance on WFLW-V while reducing performance on the WFLW test set. This is likely because the WFLW-V dataset is more diverse than the WFLW test set, and unecessary augmentations on WFLW can reduce performance. We therefore retrained LDEQ with more augmentations to get the best performance on WFLW-V.

Chapter 6

Conclusion

Nesting optimization is a key component of a vast number of deep learning methods and applications. As such, developing efficient nested optimization tools is key to the development of the field of AI in general. In this thesis, I considered three forms of nested problems: adversarial games, meta-learning, and deep equilibrium models. These settings were respectively applied to model compression, hyperparameter optimization and video landmark detection. In each case, I pushed the understanding of previous work to obtain state-of-the-art performance.

Nested adversarial games like the minimax problem have a leader-follower structure that allows training the leader by jointly improving a nested adversarial follower. In Chapter 3, I used this paradigm to perform knowledge distillation from a large teacher network to a small student network, in the case when the dataset used to pretrain the teacher network isn't available. Remarkably, I showed that nesting the optimization of the student with that of an adversarial generator leads to the student learning to match the teacher for real-world data, despite never using any during training.

Meta-learning was also a large focus of this thesis. The paradigm of "learning to learn" can be most naturally formulated as a nested optimization problem, and in Chapter 4 I proposed a novel forward-mode differentiation algorithm that enables learning hyperparameters for optimization problems that are solved in many gradient steps. This algorithm is faster and more accurate than previous state-of-the-art methods in the case of differentiable hyperparameters. While greedy approximations work well in the context of adversarial games, I demonstrated that they are largely inappropriate in the context of meta-learning.

Finally, I considered a variant of deep equilibrium models in Chapter 5 that can be cast as a nested optimization problem. I used it to improve temporal coherence in the context of landmark detection for video, showing how traditional fixed point solvers can be used along with early stopping to simulate MAP estimation. In this work, I also created a dataset and metric for benchmarking future models interested in reducing landmark flicker.

My thesis benefited very little from the traditional bilevel optimization literature, despite its long history. A perhaps cynical lesson learned is that scaling up these traditional algorithms, which are often developed on toy problems with strict assumptions, is in a sense just as hard as developing new nested optimization algorithms independently. It is worth noting as well that nested optimization in the context of deep learning is itself just barely starting to leave the toy problem setting: arguably, only Chapter 3 and Chapter 5 have demonstrated performance on real-world tasks, since in Chapter 4 I learned hyperparameters whose optimal values where already known on a relatively small dataset.

A main limitation of all the models proposed in this thesis is their efficiency. Indeed, whether it be the forward-mode algorithm in Chapter 4 or the DEQ model in Chapter 5, the methods I have built upon are still too costly to be popular among deep learning practitioners. This is due to the nature of nested optimization: solving the upper level problem with an iterative algorithm like gradient descent often requires solving the lower level problem several times. Solving both problems simultaneously can be an efficient alternative, as I've shown in the adversarial setting of Chapter 3, but it can sometimes approximate the nested structure poorly, as I've shown in the meta-learning setting of Chapter 4. Generally speaking, future work should focus on formalizing this efficiency vs. performance trade-off for the field of nested optimization more generally, as most progress on that front remains empirical.

The challenges pointed out above should also be considered as research questions produced by this thesis. The nested optimization tools I used and built upon have endless applications, as demonstrated by the diversity of problems tackled in this thesis alone. Furthermore, much of the contributions made are only relevant to the particular application domains tackled, independently from nested optimization. However, having reached state-of-the-art performance in these popular deep learning applications will, I hope, continue to drive more interest to the field of scalable nested optimization as a whole.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Ahn, S., Hu, S. X., Damianou, A. C., Lawrence, N. D., and Dai, Z. (2019). Variational information distillation for knowledge transfer. *CoRR*, abs/1904.05835.

Albrecht, S., Ramírez-Amaro, K., Ruiz-Ugalde, F., Weikersdorfer, D., Leibold, M., Ulbrich, M., and Beetz, M. (2011). Imitating human reaching motions using physically inspired optimization principles. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 602–607.

Amos, B. and Kolter, J. Z. (2017). OptNet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR.

Amouzegar, M. A. and Moshirvaziri, K. (1999). Determining optimal pollution control policies: An application of bilevel programming. *European Journal of Operational Research*, 119(1):100–120.

An, B., Ordóñez, F., Tambe, M., Shieh, E., Yang, R., Baldwin, C., DiRenzo, J., Moretti, K., Maule, B., and Meyer, G. (2013). A deployed quantal response-based patrol planning system for the u.s. coast guard. *Interfaces*, 43(5):400–420.

Anderson, D. G. (1965). Iterative procedures for nonlinear integral equations. *J. ACM*, 12(4):547–560.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR.

Assaf Shocher, Nadav Cohen, M. I. (2018). "zero-shot" super-resolution using deep internal learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Asthana, A., Zafeiriou, S., Cheng, S., and Pantic, M. (2014). Incremental face alignment in the wild. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1859–1866.

Ba, J. and Caruana, R. (2014). Do deep nets really need to be deep? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2654–2662. Curran Associates, Inc.

Bai, S., Geng, Z., Savani, Y., and Kolter, J. Z. (2022). Deep equilibrium optical flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Bai, S., Kolter, J. Z., and Koltun, V. (2019). Deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Bai, S., Koltun, V., and Kolter, J. Z. (2020). Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Bai, S., Koltun, V., and Kolter, J. Z. (2021). Stabilizing equilibrium models by jacobian regularization. In *International Conference on Machine Learning (ICML)*.

Bard, J. F. and Moore, J. T. (1992). An algorithm for the discrete bilevel programming problem. *Naval Research Logistics (NRL)*, 39(3):419–435.

Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M., and Wood, F. (2018). Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations*.

Bendsøe, M. P. (1995). Optimization of structural topology, shape, and material.

Bengio, Y. (2000). Gradient-based optimization of hyperparameters. *Neural Comput.*

Bengio, Y., Frasconi, P., and Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. In *IEEE International Conference on Neural Networks*.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305.

Bernstein, J., Wang, Y., Azizzadenesheli, K., and Anandkumar, A. (2018). SIGNSGD: compressed optimisation for non-convex problems. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 559–568. PMLR.

Bertinetto, L., Henriques, J. F., Torr, P. H., and Vedaldi, A. (2019). Meta-learning With Differentiable Closed-form Solvers. In *ICLR*.

Bhatnagar, S., Prasad, H., and L.A., P. (2013). *Stochastic Recursive Algorithms for Optimization: Simultaneous Perturbation Methods*, volume 434.

Bishop, C. (1995). Regularization and complexity control in feed-forward networks. In *Proceedings International Conference on Artificial Neural Networks ICANN'95*, volume 1, pages 141–148. EC2 et Cie.

Boltzmann, L. (1896). *Vorlesungen uber Gastheorie*. J.A. Barth.

Bracken, J. and McGill, J. T. (1973). Mathematical programs with optimization problems in the constraints. *Operations Research*, 21(1):37–44.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

Brock, A., Donahue, J., and Simonyan, K. (2019). Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*.

Brown, G., Carlyle, M., Diehl, D., Kline, J., and Wood, K. (2005). A two-sided optimization for theater ballistic missile defense. *Operations Research*, 53(5):745–763.

Brown, G. G., Carlyle, W. M., Harney, R. C., Skroch, E. M., and Wood, R. K. (2009). Interdicting a nuclear-weapons project. *Operations Research*, 57(4):866–877.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Broyden, C. G. (1965). A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19:577–593.

Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 535–541, New York, NY, USA. ACM.

Bulat, A., Sanchez, E., and Tzimiropoulos, G. (2021). Subpixel heatmap regression for facial landmark localization. In *Proceedings of the British Machine Vision Conference (BMVC)*.

Burton, P. R., Murtagh, M. J., Boyd, A., Williams, J. B., Dove, E. S., Wallace, S. E., Tassé, A.-M., Little, J., Chisholm, R. L., Gaye, A., Hveem, K., Brookes, A. J., Goodwin, P., Fistein, J., Bobrow, M., and Knoppers, B. M. (2015). Data Safe Havens in health research and healthcare. *Bioinformatics*, 31(20):3241–3248.

Candela, J. Q. and Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959.

Candler, W. and Norton, R. (1977). Multi-level programming and development policy.

Cao, X., Wei, Y., Wen, F., and Sun, J. (2012). Face alignment by explicit shape regression. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2887–2894.

Casiez, G., Roussel, N., and Vogel, D. (2012). 1 € filter: A simple speed-based low-pass filter for noisy input in interactive systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, page 2527–2530, New York, NY, USA. Association for Computing Machinery.

Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., and Mukhopadhyay, D. (2021). A survey on adversarial attacks and defences. *CAAI Transactions on Intelligence Technology*, 6(1):25–45.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Christiansen, S., Patriksson, M., and Wynter, L. (2001). Stochastic bilevel programming in structural optimization. *Structural and Multidisciplinary Optimization*, 21:361–371.

Chrysos, G. G., Antonakos, E., Zafeiriou, S., and Snape, P. (2015). Offline deformable face tracking in arbitrary videos. In *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pages 954–962.

Clark, P. and Westerberg, A. (1990). Bilevel programming for steady-state chemical process design—i. fundamentals and algorithms. *Computers & Chemical Engineering*, 14(1):87–97.

Colson, B., Marcotte, P., and Savard, G. (2007). An overview of bilevel optimization. *Annals OR*, 153:235–256.

Cootes, T., Edwards, G., and Taylor, C. (2001). Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):681–685.

Cristinacce, D. and Cootes, T. F. (2006). Feature detection and tracking with constrained local models. In *Proceedings of the British Machine Vision Conference*, pages 95.1–95.10. BMVA Press. doi:10.5244/C.20.95.

Crowley, E. J., Gray, G., and Storkey, A. (2018). Moonshine: Distilling with cheap convolutions. In *Advances in Neural Information Processing Systems*.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. (2020). Randaugment: Practical automated data augmentation with a reduced search space. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3008–3017.

Dahl, G. E., Sainath, T. N., and Hinton, G. E. (2013). Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8609–8613.

Dapogny, A., Cord, M., and Bailly, K. (2019). Decafa: Deep convolutional cascade for face alignment in the wild. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6892–6900.

Daskalakis, C. and Panageas, I. (2018). The limit points of (optimistic) gradient descent in min-max optimization. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Dehghani, M., Mehrjou, A., Gouws, S., Kamps, J., and Schölkopf, B. (2017). Fidelity-weighted learning. *CoRR*, abs/1711.02799.

Dempe, S. (2018). Bilevel optimization: theory, algorithms and applications.

Dempe, S. and Zemkoho, A. (2021). *Bilevel Optimization: Advances and Next Challenges*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Diggle, P. J. and Gratton, R. J. (1984). Monte carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(2):193–212.

Djolonga, J. and Krause, A. (2017). Differentiable learning of submodular models. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Dollár, P., Welinder, P., and Perona, P. (2010). Cascaded pose regression. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1078–1085.

Domke, J. (2012). Generic methods for optimization-based modeling. In Lawrence, N. D. and Girolami, M., editors, *International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, La Palma, Canary Islands. PMLR.

Dong, X., Tan, M., Yu, A. W., Peng, D., Gabrys, B., and Le, Q. V. (2020). Autohas: Differentiable hyper-parameter and architecture search.

Donini, M., Franceschi, L., Pontil, M., Majumder, O., and Frasconi, P. (2019). Scheduling the Learning Rate via Hypergradients: New Insights and a New Algorithm. *arXiv e-prints*.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2017). Rl2: Fast reinforcement learning via slow reinforcement learning. In *International Conference on Learning Representations*.

Dupont, E., Doucet, A., and Teh, Y. W. (2019). Augmented neural odes. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Edwards, G., Taylor, C., and Cootes, T. (1998). Interpreting face images using active appearance models. In *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*, pages 300–305.

Eichfelder, G. (2007). Solving nonlinear multiobjective bilevel optimization problems with coupled upper level constraints.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1436–1445.

Feng, Z.-H., Kittler, J., Awais, M., Huber, P., and Wu, X.-J. (2018). Wing loss for robust facial landmark localisation with convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*, pages 2235–2245. IEEE.

Feurer, M. and Hutter, F. (2019). *Chapter 1: Hyperparameter Optimization*. Springer International Publishing, Cham.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR.

Flennerhag, S., Rusu, A. A., Pascanu, R., Visin, F., Yin, H., and Hadsell, R. (2020). Meta-learning with warped gradient descent. In *International Conference on Learning Representations*.

Fortuny-Amat, J. and McCarl, B. (1981). A representation and economic interpretation of a two-level programming problem. *The Journal of the Operational Research Society*, 32(9):783–792.

Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. (2017). Forward and reverse gradient-based hyperparameter optimization. In Precup, D. and Teh, Y. W., editors, *International conference on Machine Learning*, Proceedings of Machine Learning Research, International Convention Centre, Sydney, Australia. PMLR.

Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., and Pontil, M. (2018). Bilevel programming for hyperparameter optimization and meta-learning. In *International conference on Machine Learning*.

Fredrikson, M., Jha, S., and Ristenpart, T. (2015). Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1322–1333, New York, NY, USA. ACM.

Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D., and Ristenpart, T. (2014). Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 17–32, Berkeley, CA, USA. USENIX Association.

Fu, J., Luo, H., Feng, J., Low, K. H., and Chua, T.-S. (2016). Drmad: Distilling reverse-mode automatic differentiation for optimizing hyperparameters of deep neural networks. In *IJCAI*.

Fung, S. W., Heaton, H., Li, Q., McKenzie, D., Osher, S. J., and Yin, W. (2021). Fixed point networks: Implicit depth models with jacobian-free backprop. *CoRR*, abs/2103.12803.

Gadhi, N. and Dempe, S. (2012). Necessary optimality conditions and a new approach to multiobjective bilevel optimization problems. *Journal of Optimization Theory and Applications*, 155.

Garrido, P., Valgaerts, L., Sarmadi, H., Steiner, I., Varanasi, K., Pérez, P., and Theobalt, C. (2015). Vdub: Modifying face video of actors for plausible visual alignment to a dubbed audio track. *Comput. Graph. Forum*, 34(2):193–204.

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. (2018). Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *CoRR*, abs/1811.12231.

Geng, Z., Zhang, X.-Y., Bai, S., Wang, Y., and Lin, Z. (2021). On training implicit models. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 24247–24260. Curran Associates, Inc.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

Goodfellow, I. J. (2017). Nips 2016 tutorial: Generative adversarial networks. *ArXiv*, abs/1701.00160.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. (2018). Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). Deep learning with limited numerical precision. *CoRR*, abs/1502.02551.

Gutmann, M. and Corander, J. (2016). Bayesian optimization for likelihood-free inference of simulator-based statistical models. *Journal of Machine Learning Research*, 17(125):1–47.

Halter, W. and Mostaghim, S. (2006). Bilevel optimization of multi-component chemical systems using particle swarm optimization. In *2006 IEEE International Conference on Evolutionary Computation*, pages 1240–1247.

Han, S., Mao, H., and Dally, W. J. (2016). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149.

Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1135–1143, Cambridge, MA, USA. MIT Press.

Hansen, N., Jangir, R., Sun, Y., Alenyà, G., Abbeel, P., Efros, A. A., Pinto, L., and Wang, X. (2021). Self-supervised policy adaptation during deployment. In *International Conference on Learning Representations*.

Hansen, P., Jaumard, B., and Savard, G. (1992). New branch-and-bound rules for linear bilevel programming. *SIAM Journal on Scientific and Statistical Computing*, 13(5):1194–1217.

Hasani, B. and Mahoor, M. H. (2017). Facial expression recognition using enhanced deep 3d convolutional neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2278–2288.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Herskovits, J., Leontiev, A., Dias, G., and Santos, G. (2000). Contact shape optimization: A bilevel programming approach. *Structural and Multidisciplinary Optimization*, 20:214–221.

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

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*.

Hospedales, T. M., Antoniou, A., Micaelli, P., and Storkey, A. J. (2021). Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1.

Hsu, K., Levine, S., and Finn, C. (2019). Unsupervised learning via meta-learning. In *International Conference on Learning Representations*.

Huang, Y., Yang, H., Li, C., Kim, J., and Wei, F. (2021). Adnet: Leveraging error-bias towards normal direction in face alignment. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3060–3070.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. (2019). Adversarial Examples Are Not Bugs, They Are Features. *arXiv e-prints*, page arXiv:1905.02175.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. and Blei, D., editors, *International conference on Machine Learning*, Proceedings of Machine Learning Research, Lille, France. PMLR.

Jabbar, R., Shinoy, M., Kharbeche, M., Al-Khalifa, K., Krichen, M., and Barkaoui, K. (2020). Driver drowsiness detection model using convolutional neural networks techniques for android application. In *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, pages 237–242.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A. G., Adam, H., and Kalenichenko, D. (2017). Quantization and training of neural networks for efficient integer-arithmetic-only inference. *CoRR*, abs/1712.05877.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4):295–307.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. (2017). Population based training of neural networks. *arXiv preprint arXiv:1711.09846*.

Jamieson, K. and Talwalkar, A. (2016). Non-stochastic best arm identification and hyperparameter optimization. In Gretton, A. and Robert, C. C., editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 240–248, Cadiz, Spain. PMLR.

Jiang, H., Liu, S., Wang, J., and Wang, X. (2021). Hand-object contact consistency reasoning for human grasps generation. In *Proceedings of the International Conference on Computer Vision*.

Jin, C., Netrapalli, P., and Jordan, M. (2020). What is local optimality in nonconvex-nonconcave minimax optimization? In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4880–4889. PMLR.

Jung, H., Lee, S., Yim, J., Park, S., and Kim, J. (2015). Joint fine-tuning in deep neural networks for facial expression recognition. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2983–2991.

Katehakis, M. N. and Veinott, A. F. (1987). The multi-armed bandit problem: Decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268.

Khabarlak, K. and Koriashkina, L. (2022). Fast facial landmark detection and applications: A survey. *Journal of Computer Science and Technology*, 22.

Kim, D. H., Lee, M. K., Choi, D. Y., and Song, B. C. (2017). Multi-modal emotion recognition using semi-supervised learning and multiple neural networks in the wild. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, ICMI '17, page 529–535, New York, NY, USA. Association for Computing Machinery.

Kimura, A., Ghahramani, Z., Takeuchi, K., Iwata, T., and Ueda, N. (2018). Few-shot learning of neural networks from scratch by pseudo example optimization. *arXiv e-prints*, page arXiv:1802.03039.

King, R. D., Feng, C., and Sutherland, A. (1995). Statlog: comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence*, 9(3):289–333.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Klein, A. (2019). `https://automl.github.io/HpBandSter/build/html/quickstart.html`.

Kohavi, R. and John, G. H. (1995). Automatic parameter selection by minimizing estimated error. In Prieditis, A. and Russell, S., editors, *Machine Learning Proceedings 1995*, pages 304–312. Morgan Kaufmann, San Francisco (CA).

Komer, B., Bergstra, J., and Eliasmith, C. (2014). Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. pages 32–37.

Krantz, S. and Parks, H. (2003). The implicit function theorem : History, theory, and applications / s.g. krantz, h.r. parks.

Krizhevsky, A. (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. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Kumar, A., Marks, T. K., Mou, W., Wang, Y., Jones, M., Cherian, A., Koike-Akino, T., Liu, X., and Feng, C. (2020). Luvli face alignment: Estimating landmarks' location, uncertainty, and visibility likelihood. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kurakin, A., Goodfellow, I. J., and Bengio, S. (2017). Adversarial machine learning at scale. In *International Conference on Learning Representations*.

Lai, H., Xiao, S., Pan, Y., Cui, Z., Feng, J., Xu, C., Yin, J., and Yan, S. (2018). Deep recurrent regression for facial landmark detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(5):1144–1157.

Lan, X., Hu, Q., and Cheng, J. (2021). HIH: towards more accurate face alignment via heatmap in heatmap. *CoRR*, abs/2104.03100.

Larochelle, H., Erhan, D., and Bengio, Y. (2008). Zero-data learning of new tasks. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 646–651.

Larsen, J., Hansen, L. K., Svarer, C., and Ohlsson, M. (1996). Design and regularization of neural networks: the optimal use of a validation set. In *Neural Networks for Signal Processing VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop*, pages 62–71.

LeCun, Y., Denker, J., and Solla, S. (1990). Optimal brain damage. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

Ledig, C., Theis, L., Huszar, F., Caballero, J., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2016). Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. (2019). Meta-Learning With Differentiable Convex Optimization. In *CVPR*.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2017a). Pruning filters for efficient convnets.

Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017b). Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52.

Li, L. and Talwalkar, A. (2019). Random search and reproducibility for neural architecture search. In *UAI*.

Li, M., Yumer, E., and Ramanan, D. (2020a). Budgeted training: Rethinking deep neural network training under resource constraints. In *International Conference on Learning Representations*.

Li, S. and Deng, W. (2020). Deep facial expression recognition: A survey. *IEEE Transactions on Affective Computing*, pages 1–1.

Li, T., Li, J., Liu, Z., and Zhang, C. (2018). Knowledge distillation from few samples. *CoRR*, abs/1812.01839.

Li, W., Lu, Y., Zheng, K., Liao, H., Lin, C., Luo, J., Cheng, C.-T., Xiao, J., Lu, L., Kuo, C.-F., and Miao, S. (2020b). Structured landmark detection via topology-adapting deep graph learning. In *ECCV*.

Li, Y., Hao, M., Di, Z., Gundavarapu, N. B., and Wang, X. (2021). Test-time personalization with a transformer for human pose estimation. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

Li, Y., Yang, Y., Zhou, W., and Hospedales, T. M. (2019). Feature-Critic Networks For Heterogeneous Domain Generalization. In *ICML*.

Lin, C., Zhu, B., Wang, Q., Liao, R., Qian, C., Lu, J., and Zhou, J. (2021). Structure-coherent deep feature learning for robust face alignment. *IEEE Transactions on Image Processing*, 30:5313–5326.

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *MATHEMATICAL PROGRAMMING*, 45:503–528.

Liu, H., Simonyan, K., and Yang, Y. (2019). DARTS: Differentiable architecture search. In *International Conference on Learning Representations*.

Lopes, R. G., Fenu, S., and Starner, T. (2017). Data-free knowledge distillation for deep neural networks. *CoRR*, abs/1710.07535.

Lorraine, J., Vicol, P., and Duvenaud, D. (2019). Optimizing Millions of Hyperparameters by Implicit Differentiation. *arXiv e-prints*.

Loshchilov, I. and Hutter, F. (2017). Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*.

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

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110.

Luketina, J., Berglund, M., Greff, K., and Raiko, T. (2016). Scalable gradient-based tuning of continuous regularization hyperparameters. In *International conference on Machine Learning*.

Luvizon, D. C., Tabia, H., and Picard, D. (2017). Human pose regression by combining indirect part detection and contextual information. *CoRR*, abs/1710.02322.

Lyu, C., Huang, K., and Liang, H.-N. (2015). A unified gradient regularization family for adversarial examples. In *2015 IEEE International Conference on Data Mining*, pages 301–309.

Maclaurin, D., Duvenaud, D., and Adams, R. P. (2015). Gradient-based Hyperparameter Optimization through Reversible Learning. *arXiv e-prints*.

Martins, A. F. T. and Astudillo, R. F. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. *CoRR*, abs/1602.02068.

Masi, I., Wu, Y., Hassner, T., and Natarajan, P. (2018). Deep face recognition: A survey. In *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 471–478.

Mathieu, R., Pittard, L., and Anandalingam, G. (1994). Genetic algorithm based approach to bi-level linear programming. *RAIRO - Operations Research - Recherche Opérationnelle*, 28(1):1–21.

MediaPipe (2020). Mediapipe face mesh. `https://google.github.io/mediapipe/solutions/face_mesh`. Accessed: 2022-11-09.

MediaPipe (2021). Mediapipe 3d face transform. `https://developers.googleblog.com/2020/09/mediapipe-3d-face-transform.html`. Accessed: 2022-11-09.

Menghani, G. (2021). Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *CoRR*, abs/2106.08962.

Mescheder, L., Nowozin, S., and Geiger, A. (2017). Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. In *International Conference on Machine Learning (ICML)*.

Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., and Sohl-Dickstein, J. (2019). Understanding and correcting pathologies in the training of learned optimizers. 97:4556–4565.

Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. (2017). Unrolled generative adversarial networks. In *International Conference on Learning Representations*.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2017). Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Mombaur, K., Truong, A., and Laumond, J.-P. (2010). From human to humanoid locomotion-an inverse optimal control approach. *Auton. Robots*, 28:369–383.

Moore, J. T. and Bard, J. F. (1990). The mixed integer linear bilevel programming problem. *Operations Research*, 38(5):911–921.

Mu, J., Qiu, W., Kortylewski, A., Yuille, A. L., Vasconcelos, N., and Wang, X. (2021). A-SDF: learning disentangled signed distance functions for articulated shape representation. In *ICCV*, pages 12981–12991.

Nayak, G. K., Mopuri, K. R., Shaj, V., Babu, R. V., and Chakraborty, A. (2019). Zero-shot knowledge distillation in deep networks. In *International Conference on Machine Learning (ICML)*.

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 2011*.

Newell, A., Yang, K., and Deng, J. (2016). Stacked hourglass networks for human pose estimation. *CoRR*, abs/1603.06937.

Oliver, A., Odena, A., Raffel, C., Cubuk, E. D., and Goodfellow, I. J. (2018). Realistic evaluation of deep semi-supervised learning algorithms. *CoRR*, abs/1804.09170.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

OpenAI (2023). Gpt-4 technical report.

Pabbaraju, C., Winston, E., and Kolter, J. Z. (2021a). Estimating lipschitz constants of monotone deep equilibrium models. In *International Conference on Learning Representations*.

Pabbaraju, C., Winston, E., and Kolter, J. Z. (2021b). Estimating lipschitz constants of monotone deep equilibrium models. In *International Conference on Learning Representations*.

Papernot, N., Mcdaniel, P., and Goodfellow, I. J. (2016). Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *ArXiv*, abs/1605.07277.

Papernot, N. and McDaniel, P. D. (2017). Extending defensive distillation. *CoRR*, abs/1705.05264.

Parmas, P., Rasmussen, C. E., Peters, J., and Doya, K. (2018). PIPPS: Flexible model-based policy search robust to the curse of chaos. volume 80 of *Proceedings of Machine Learning Research*, pages 4065–4074, Stockholmsmässan, Stockholm Sweden. PMLR.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on Machine Learning*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. In *International conference on Machine Learning*.

Peters, O. (2019). The ergodicity problem in economics. *Nature Physics*, 15(12):1216–1221.

Petrak, J. (2000). Fast subsampling performance estimates for classification algorithm selection.

Pontryagin, L. S., Boltyanskii, V. G., Gamkrelidze, R. V., and Mishchenko, E. F. (1962). The mathematical theory of optimal processes.

Qian, S., Sun, K., Wu, W., Qian, C., and Jia, J. (2019). Aggregation via separation: Boosting facial landmark detector with semi-supervised style translation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10152–10162.

R, M. B., Chari, V., and Jawahar, C. (2015). Efficient face frontalization in unconstrained images. In *2015 Fifth National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG)*, pages 1–4.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Rajeswaran, A., Finn, C., Kakade, S. M., and Levine, S. (2019). Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Ren, M., Zeng, W., Yang, B., and Urtasun, R. (2018). Learning to reweight examples for robust deep learning. In *International conference on Machine Learning*.

Revay, M., Wang, R., and Manchester, I. R. (2020). Lipschitz bounded equilibrium networks. *CoRR*, abs/2010.01732.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591 vol.1.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2015). Fitnets: Hints for thin deep nets. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. (2019). Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907.

Rusu, A. A., Colmenarejo, S. G., Gülçehre, Ç., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2016). Policy distillation. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Safaryan, M. and Richtárik, P. (2019). On Stochastic Sign Descent Methods. *arXiv e-prints*, page arXiv:1905.12938.

Sagonas, C., Tzimiropoulos, G., Zafeiriou, S., and Pantic, M. (2013). 300 faces in-the-wild challenge: The first facial landmark localization challenge. *2013 IEEE International Conference on Computer Vision Workshops*, pages 397–403.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Santos, R. J. (1996). Equivalence of regularization and truncated iteration for general ill-posed problems. *Linear Algebra and its Applications*, 236:25–33.

Savitzky, A. and Golay, M. J. E. (1964). Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639.

Seider, W. D. and White III, C. W. (1985). Chemical reaction equilibrium analysis: Theory and algorithms by william r. smith and ronald w. missen, 364 pp., john wiley, 1983, $42.95. *AIChE Journal*, 31(1):176–176.

Shaban, A., Cheng, C.-A., Hatch, N., and Boots, B. (2019). Truncated back-propagation for bilevel optimization. In *AISTATS*.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Shen, J., Zafeiriou, S., Chrysos, G. G., Kossaifi, J., Tzimiropoulos, G., and Pantic, M. (2015). The first facial landmark tracking in-the-wild challenge: Benchmark and results. In *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pages 1003–1011.

Sinha, A., Malo, P., and Deb, K. (2018). A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295.

Sinha, A., Malo, P., Frantsev, A., and Deb, K. (2013). Multi-objective stackelberg game between a regulating authority and a mining company: A case study in environmental economics. In *2013 IEEE Congress on Evolutionary Computation*, pages 478–485.

Sinha, A., Malo, P., Frantsev, A., and Deb, K. (2014). Finding optimal strategies in a multi-period multi-leader–follower stackelberg game using an evolutionary algorithm. *Computers & Operations Research*, 41:374–385.

Sjöberg, J. and Ljung, L. (1992). Overtraining, regularization, and searching for minimum in neural networks. *IFAC Proceedings Volumes*, 25(14):73–78. 4th IFAC Symposium on Adaptive Systems in Control and Signal Processing 1992, Grenoble, France, 1-3 July.

Slivkins, A. (2019). Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286.

Smith, L. N. and Topin, N. (2017). Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Snelson, E. and Ghahramani, Z. (2005). Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, pages 1257–1264.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. (2015). Scalable bayesian optimization using deep neural networks. In *International conference on Machine Learning*.

Socher, R., Ganjoo, M., Manning, C. D., and Ng, A. Y. (2013). Zero-shot learning through cross-modal transfer. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 935–943.

Srivastava, A., Valkov, L., Russell, C., Gutmann, M. U., and Sutton, C. (2017). Veegan: Reducing mode collapse in gans using implicit variational learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Stackelberg, H. (1952). *The Theory Of Market Economy*. Oxford University Press.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. *CoRR*, abs/1707.02968.

Sun, K., Xiao, B., Liu, D., and Wang, J. (2019). Deep high-resolution representation learning for human pose estimation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5686–5696.

Sun, Y., Wang, X., Liu, Z., Miller, J., Efros, A., and Hardt, M. (2020). Test-time training with self-supervision for generalization under distribution shifts. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9229–9248. PMLR.

Sun, Y., Wang, X., and Tang, X. (2013). Deep convolutional network cascade for facial point detection. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3476–3483.

Suryan, V., Sinha, A., Malo, P., and Deb, K. (2016). Handling inverse optimal control problems using evolutionary bilevel optimization. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 1893–1900.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2014). Intriguing properties of neural networks. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708.

Titsias, M. K. (2009). Variational learning of inducing variables in sparse Gaussian processes. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, pages 567–574.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jegou, H. (2021). Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, volume 139, pages 10347–10357.

Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. (2016). Stealing machine learning models via prediction apis. *CoRR*, abs/1609.02943.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. (2018). Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*.

Trigeorgis, G., Snape, P., Nicolaou, M. A., Antonakos, E., and Zafeiriou, S. (2016). Mnemonic descent method: A recurrent process applied for end-to-end face alignment. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4177–4187.

Tzimiropoulos, G. (2015). Project-out cascaded regression with an application to face alignment. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3659–3667.

Vanhoucke, V., Senior, A., and Mao, M. Z. (2011). Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*.

Vicente, L., Savard, G., and Júdice, J. (1996). Discrete linear bilevel programming problem. *J. Optim. Theory Appl.*, 89(3):597–614.

Vicente, L. N., Savard, G., and Júdice, J. (1994). Descent approaches for quadratic bilevel programming. *Journal of Optimization Theory and Applications*, 81:379–399.

von Stackelberg, H., Bazin, D., Hill, R., and Urch, L. (1934). *Market Structure and Equilibrium*. Springer Berlin Heidelberg.

Walters, P. (1982). *An Introduction to Ergodic Theory*. Graduate texts in mathematics. Springer-Verlag.

Wan, J., Lai, Z., Liu, J., Zhou, J., and Gao, C. (2021). Robust face alignment by multi-order high-precision hourglass network. *IEEE Transactions on Image Processing*, 30:121–133.

Wang, D., Shelhamer, E., Liu, S., Olshausen, B., and Darrell, T. (2021). Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*.

Wang, M. and Deng, W. (2018). Deep face recognition: A survey. *CoRR*, abs/1804.06655.

Wang, P., Donti, P. L., Wilder, B., and Kolter, J. Z. (2019a). Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR.

Wang, T., Zhu, J., Torralba, A., and Efros, A. A. (2018). Dataset distillation. *CoRR*, abs/1811.10959.

Wang, X., Bo, L., and Fuxin, L. (2019b). Adaptive wing loss for robust face alignment via heatmap regression. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6970–6980.

Wein, L. M. (2009). Homeland security: From mathematical models to policy implementation: The 2008 philip mccord morse lecture. *Operations Research*, 57(4):801–811.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*.

Whittaker, G., Färe, R., Grosskopf, S., Barnhart, B., Bostian, M., Mueller-Warrant, G., and Griffith, S. (2017). Spatial targeting of agri-environmental policy using bilevel evolutionary optimization. *Omega*, 66:15–27.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*.

Winston, E. and Kolter, J. Z. (2020). Monotone operator equilibrium networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10718–10728. Curran Associates, Inc.

Wu, W., Qian, C., Yang, S., Wang, Q., Cai, Y., and Zhou, Q. (2018a). Look at boundary: A boundary-aware face alignment algorithm. In *CVPR*.

Wu, Y. and Ji, Q. (2019). Facial landmark detection: A literature survey. *Int. J. Comput. Vision*, 127(2):115–142.

Wu, Y., Ren, M., Liao, R., and Grosse., R. (2018b). Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Xia, J., Qu, W., Huang, W., Zhang, J., Wang, X., and Xu, M. (2022). Sparse local patch transformer for robust face alignment and landmarks inherent relation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4052–4061.

Xiong, X. and De la Torre, F. (2013). Supervised descent method and its applications to face alignment. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 532–539.

Yan, J., Zheng, W., Cui, Z., Tang, C., Zhang, T., Zong, Y., and Sun, N. (2016). Multi-clue fusion for emotion recognition in the wild. In *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, ICMI '16, page 458–463, New York, NY, USA. Association for Computing Machinery.

Yang, J., Liu, Q., and Zhang, K. (2017). Stacked hourglass network for robust facial landmark localisation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2025–2033.

Yang, S., Luo, P., Loy, C. C., and Tang, X. (2016). Wider face: A face detection benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ye, J. J. (2011). Necessary optimality conditions for multiobjective bilevel programs. *Mathematics of Operations Research*, 36(1):165–184.

Yildiz, C., Heinonen, M., and Lahdesmaki, H. (2019). Ode2vae: Deep generative second order odes with bayesian neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Yin, Y. (2000). Genetic-algorithms-based approach for bilevel programming models. *Journal of Transportation Engineering*, 126(2):115–120.

Yu, B. and Tao, D. (2021). Heatmap regression via randomized rounding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Yu, T. and Zhu, H. (2020). Hyper-parameter optimization: A review of algorithms and applications. *CoRR*, abs/2003.05689.

Yuan, Y., Fu, R., Huang, L., Lin, W., Zhang, C., Chen, X., and Wang, J. (2021). Hrformer: High-resolution transformer for dense prediction. In *NeurIPS*.

Zagoruyko, S. and Komodakis, N. (2016a). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *CoRR*, abs/1612.03928.

Zagoruyko, S. and Komodakis, N. (2016b). Wide residual networks. In *BMVC*.

Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., and Hutter, F. (2020). Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*.

Zhang, H., Tian, C., Li, Y., Su, L., Yang, N., Zhao, W. X., and Gao, J. (2021a). Data poisoning attack against recommender system using incomplete and perturbed data. KDD '21, page 2154–2164, New York, NY, USA. Association for Computing Machinery.

Zhang, J., Zeng, X., Pan, Y., Liu, Y., Ding, Y., and Fan, C. (2019). Freenet: Multi-identity face reenactment. *CoRR*, abs/1905.11805.

Zhang, K., Zhang, Z., Li, Z., and Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503.

Zhang, M. M., Marklund, H., Dhawan, N., Gupta, A., Levine, S., and Finn, C. (2021b). Adaptive risk minimization: Learning to adapt to domain shift. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

Zhang, Y., Shao, M., Wong, E. K., and Fu, Y. (2013). Random faces guided sparse many-to-one encoder for pose-invariant face recognition. In *2013 IEEE International Conference on Computer Vision*, pages 2416–2423.

Zheng, Y., Yang, H., Zhang, T., Bao, J., Chen, D., Huang, Y., Yuan, L., Chen, D., Zeng, M., and Wen, F. (2021). General facial representation learning in a visual-linguistic manner. *CoRR*, abs/2112.03109.

Zhong, Y. D., Dey, B., and Chakraborty, A. (2020). Symplectic ode-net: Learning hamiltonian dynamics with control. In *International Conference on Learning Representations*.

Zhu, X., Lei, Z., Yan, J., Yi, D., and Li, S. Z. (2015). High-fidelity pose and expression normalization for face recognition in the wild. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 787–796.

Zico Kolter, David Duvenaud, M. J. (2019). `http://implicit-layers-tutorial.org/deep_equilibrium_models/`.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.