Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

On Trainability and Generalization of Deep Neural Networks

Ph.D. candidate
Luca Zancato

Advisor
Prof. Alessandro Chiuso

Director & Coordinator
Prof. Andrea Neviani

Ph.D. School in
Information Engineering

Department of
Information Engineering
University of Padova
2021

University of Padova
Department of Information Engineering

Università
degli Studi
di Padova

Ph.D. course in: Information Engineering
Curriculum: Information Science and Technology
Series: 34^{th} (2018-2021)

On Trainability and Generalization of Deep Neural Networks

Director: Prof. Andrea Neviani
Advisor: Prof. Alessandro Chiuso

Ph.D. candidate: Luca Zancato

Year: 2021

To Sonia

Abstract

The last few years have witnessed the rise of Deep Neural Networks. Since the introduction of AlexNet in 2012, the community of researchers and industries employing Deep Learning has exploded. This surge in attention led to the development of State of The Art algorithms in many different fields such as Computer Vision, Natural Language Processing and Time Series modeling. The empirical success of Deep Learning posed new methodological challenges for academia and allowed industry to deploy world-wide large scale web services unthinkable ten years ago.

Despite such incontrovertible success, Deep Learning does not come free of issues: model design is highly costly, model interpretability is not easy, deployment often requires very specialized experts and, not least, *any* Deep Neural Network requires a large amount of data for training. Moreover, from a theoretical standpoint many important guarantees on optimization convergence and generalization are still lacking.

In this thesis we address trainability and generalization of Deep Neural Network models: we analyze the optimization trajectories and the generalization of typical over-parametrized models; moreover, we design a specialized inductive bias and regularization scheme to foster interpretability and generalization of Deep Neural Networks.

The starting point in our analysis is a recently proposed tool: the Neural Tangent Kernel for over-parametrized models. Building on this fundamental result, we investigate the number of optimization steps that a pre-trained Deep Neural Network needs to converge to a given value of the loss function (*Training Time*). Moreover, we exploit the Neural Tangent Kernel theory to solve the problem of choosing the best pre-trained Deep Neural Network within a "model zoo" when only the target dataset is known and without training any model (*Model Selection*).

Our analysis started to unblock the adoption of real-world Computer Vision AutoML systems: Users fine-tune models selected from a large "model zoo" testing hundreds of combinations of different architectures, pre-training sets and hyper-parameters, but are reluctant to do so without an estimate of the expected training cost. Our results are a step towards better understanding of transfer learning through a novel study on the interplay between generalization and highly over-parametrized Deep Neural Networks.

We then build a specialized Deep architecture equipped with a strong inductive bias and explicit regularization, that are designed both to constrain the representational power of our architecture and to allow Bayesian automatic complexity selection. Then, we show our novel method can be successfully applied both for non-linear System Identification and for Anomaly Detection of large scale Time Series.

Sommario

Negli ultimi anni abbiamo assistito all'ascesa delle Reti Neurali Profonde. A partire dall'introduzione di AlexNet, nel 2012, la comunità di ricercatori e industrie che sfruttano l'Apprendimento Profondo è cresciuta a dismisura. Tale aumento di visibilità ha portato all'avanzamento dello stato dell'arte in diversi campi: visione artificiale, elaborazione del linguaggio naturale e modellazione di serie temporali. Il successo dell'apprendimento profondo ha posto nuove sfide metodologiche per l'accademia e al tempo stesso ha reso possibile all'industria il dispiego di servizi web su larga scala impensabili pochi anni fa.

Nonostante tale indiscutibile successo, l'apprendimento automatico non è privo di limiti: sviluppare nuovi modelli è costoso, la loro interpretabilità è scarsa, il loro impiego richiede esperti altamente specializzati e, non per ultimo, ogni rete neurale profonda richiede un grande quantitativo di dati per essere allenata. In più, in letteratura mancano ancora risultati teorici fondamentali a garantire la convergenza dell'ottimizzazione e la generalizzazione dei modelli profondi.

In questa tesi studiamo l'addestrabilità e la capacità di generalizzazione delle reti neurali profonde: in particolare, analizziamo le traiettorie di ottimizzazione e di generalizzazione di modelli sovra parametrizzati; in più, proponiamo un bias induttivo specializzato e una regolarizzazione che favoriscono sia l'interpretabilità che la generalizzazione delle reti neurali profonde. Il punto di partenza della nostra analisi è un risultato recentemente proposto in letteratura: il "Neural Tangent Kernel" per modelli sovra parametrizzati.

Basandoci su questo strumento, studiamo il numero di passi di ottimizzazione necessari ad una rete neurale profonda pre-allenata per convergere ad un dato valore della funzione di costo (*Tempo di Allenamento*). In più, sfruttando la teoria sul "Neural Tangent Kernel", risolviamo il problema di scegliere il miglior modello pre-allenato all'interno di un "model zoo" quando solamente i dati su cui allenare la rete neurale sono noti e senza ottimizzare alcun modello.

La nostra analisi è spinta dalla necessità di sbloccare l'adozione di sistemi per la visione artificiale su larga scala. In questi sistemi gli utenti allenano modelli selezionandoli all'interno di un "model zoo" ottenuto combinando svariate architetture pre-allenate ed iper-parametri, ma sono riluttanti a farlo senza una stima del costo. I nostri risultati, basati su una nuova analisi dell'interazione tra generalizzazione e sovra-parametrizzazione, sono un passo avanti nello studio della capacità di adattamento delle reti neurali profonde.

Sfruttando questi risultati proponiamo quindi una nuova architettura profonda basata su un forte bias induttivo e regolarizzazione esplicita, entrambi sono pensati ed usati per limitare la capacità espressiva dell'architettura e permettono di applicare tecniche Bayesiane di selezione automatica della complessità. Per concludere, applichiamo con successo il nostro metodo per l'identificazione di sistemi non-lineari e per l'individuazione di anomalie su serie temporali di grandi dimensioni.

Acknowledgments

The first big *thank you* goes to my advisor Prof. Alessandro Chiuso, who has been kind, wise and has inspired me since *Day 1*[1]. This work would not exist without his unweary support, his brilliant suggestions and the countless hours of discussion we spent together.

Furthermore, I would like to thank Stefano Soatto, from which I learnt the importance of being relentless and *Thinking Big*. I will never forget our car conversations heading to UCLA and most importantly the fiery passion that drove us.

My sincere thanks go to Alessandro Achille, which I consider not only a mentor but also a friend. Working together has been one of the most exciting experiences I had, I owe him so much! If I had recorded and written down all the information he shared with me during the last two years this thesis would have been at least twice its size[2]. I am eager to work hard and have fun together once again!

In the last years I have been so fortunate to collaborate with many gifted people: Giovanni Paolini, Avinash Ravichandran, Guneet Singh Dhillon, Hao Li, Aditya Deshpande, Charless Fowlkes and Pietro Perona from AmazonAI. I want to thank them all for sharing with me their unique perspectives and experiences.

Since the beginning of 2020 our lives have radically changed, at that time, when the entire world faced for the first time COVID-19, I had just moved to Pasadena to start an internship within Amazon Web Services. Lockdown is hard for anyone, but it is especially harsh when you are very far from home[3], nonetheless I must confess I have never truly suffered from loneliness. For that, I must thank: Alessandro, Giovanni, Aleksandra, Luis, Guneet and Jöel[4]. It has been a great pleasure and a breath of fresh air to spend time with them playing Kubb at Caltech (keeping safe distance), exploring the D&D world (remotely) and biking uphill (on Marengo).

Another huge *thank you* goes to Andrea, we met more than 15 years ago, since then I have always been inspired by his endless knowledge and integrity. I want to thank him for being such a prominent example of the person I would like to be.

Special thanks go to all the friends with which I spent countless hours of "unproductive" time[5]: Carlo, Edoardo, Clara, Stefano, Sergej, Francisco, Alvise, Pierfrancesco, Federico, Stefano, Luca, Alessandro, Tommaso, Gaia, Jacopo and Tihol.

I started this three year long journey laser focused on my research, I would have never expected to meet so many interesting and diverse colleagues. I owe them so many happy moments: Nicola, Alberto, Luca, Giulia, Daniele, Irene, Marco, Nicola, Giulia, Anna,

[1] I decided which Master's Degree to pursue while attending one his lectures.

[2] And, there is no doubt that a huge chapter would have been devoted to home made automatic watering systems!

[3] I measure distance by time zones in this case, since even 100 meters could feel like thousands kilometers during lockdown.

[4] The most hospitable man on earth!

[5] The older I get the more I can confidently say that spending "unproductive" time in the "right" way is what makes life interesting and unpredictable $\implies$ Fun.

Fabio, Francesco, Luca, Matteo, Mattias, Nicola, Enrico, Enrica, Enrico[6]. Thank you all for our interesting daily interactions and our fun social activities.

It is now time to thank my parents, they always help and support me in each and every decision I take[7]. I am extremely grateful to them and proud to be their son. I want to thank from the bottom of the heart Lisa, my sister. I have always admired her attitude towards work-life harmony and I feel honored to be her older brother. Thank you "Nonna Anna", "Nonno Sergio", "Nonna Paola", "Nonno Benito", "Zio Toni" and "Zia Luciana", you have always been a pillar of strength and wisdom for me! I have plenty of other important people to personally thank for the many years we spent together but I think the best way to convey what I feel is to refer to them as Family. So, thank you all, you all are and will always be my Family, regardless of the distance[8] between us.

Last but definitely not least, I want to thank my wife Sonia, for everything she has done in the past years, for the moments we shared and for the future we are building. There is not doubt we will continue learning and improving ourselves together!

[6]Note how many overlapping names on such a small number of people!

[7]It's worth mentioning that arguing is part of the supporting process.

[8]Measured in kilometers, in time zones, in light-years or in any other imaginable way.

Contents

List of Symbols

Symbol	Description
$:=$	Left side is defined by the right side
$=:$	Right side is defined by the left side
$\mathbb{N}$	Set of natural numbers
$\mathbb{R}$	Set of real numbers
$\mathbb{R}_+$	Set of non-negative real numbers
$\mathbb{R}^n$	Euclidean n-dimensional space
$\mathbb{R}^{m\times n}$	Space of real matrices with m rows and n columns
$[n]$	$\{1, ..., n\}$ set of integers from 1 to n
a	Lower case letters denote vectors
$(a_i)_{i\in[n]}$	Entry-wise vector definition
$(a)_i$	Entry at i-th position of a vector a
$\mathbb{1}$	Vector of ones
A	Upper case letters denote matrices
I_n	$n \times n$ identity matrix (subscript is omitted if the dimension n is clear from the context)
$(A)_i$	i-th row of a matrix A
$(A)_{i,j}$	Entry at (i, j) of a matrix A
A^T	Transpose of matrix A
A^{-1}	Inverse of matrix A
$A^\dagger$	Moore-Penrose pseudoinverse of A
$\mathrm{diag}(a_1, ..., a_n)$	$n \times n$ diagonal matrix with entries $a_1, ..., a_n$
$\mathrm{Tr}[A]$	Trace of the matrix A
$\det(A)$	Determinant of the matrix A
$\mathrm{vec}(A)$	Vectorization (column-wise) of matrix A
$\langle a, b\rangle$	$a^T b$ standard inner product
$\|a\|_2$	Euclidean norm of vector a
$\|a\|_Q$	$a^T Q a$ with Q symmetric positive definite weighting matrix
$\|a\|_{\ell^2}$	Euclidean norm of the infinite sequence a
$\|a\|_{L^2}$	Euclidean norm of the function $a(x)$
$\|A\|_F$	Frobenius norm of a matrix
$a \otimes b$	Outer product ab^T between vectors a and b
$\frac{\partial f_w(x)}{\partial w}$ or $\nabla_w f$	Partial derivative of the function f w.r.t. w if w is scalar otherwise gradient of f w.r.t. w (a column vector if f is scalar valued)
$\arg\min_w f(w)$	Value of w which minimizes $f(w)$ (similar for $\arg\max$)

Symbol	Description
$p(x)$	Probability Density Function (PDF) of a random vector x
$x \sim p(x)$	x is a random vector distributed according to the PDF $p(x)$
$\mathcal{N}(\mu, \Sigma)$	Gaussian distribution with mean μ and covariance Σ
$\mathbb{E}[x]$	Expected value of the random vector x
$\mathrm{Cov}(x)$	Covariance of the random vector x
$\delta_{i,j}$	Kronecker delta
$\xrightarrow{p}$	convergence in probability

List of Acronyms

Acronym	Description
ML	Machine Learning
DL	Deep Learning
SOTA	State Of The Art
DNN	Deep Neural Network
CNN	Convolutional Neural Network
LSTM	Long short-term memory
GRU	Gated recurrent unit
TCN	Temporal Convolutional Network
CPU	Central Processing Unit
GPU	Graphics Processing Unit
ODE	Ordinary Differential Equation
SDE	Stochastic Differential Equation
GD	Gradient Descent
GF	Gradient Flow
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Unit
ERM	Empirical Risk Minimization
NTK	Neural Tangent Kernel
RKHS	Reproducing Kernel Hilbert Space
BN	Batch normalization
CV	Cross-Validation
KA	Kernel Alignment
CKA	Centered Kernel Alignment
TT	Training Time
ELR	Effective Learning Rate
LGC	Label-Gradient Correlation
LFC	Label-Feature Correlation
HPO	Hyper-parameter optimization
NARX	Non-linear AutoRegressive with eXogenous inputs
NARMAX	Non-linear AutoRegressive Moving Average with eXogenous inputs
PEM	Prediction Error Method

Acronym	Description
EMD	Earth mover distance
GP	Gaussian Process
GPR	Gaussian Process Regression
MAP	Maximum A Posteriori
CE	Cross-Entropy
SO	Soft orthogonality
TRIAD	Time series Reliable and Interpretable Anomaly Detection
AD	Anomaly Detection
HP	Hodrick Prescott
AUC	Area Under the Curve
CUMSUM	Cumulative Sum
PDF	Probability Density Function
w.r.t.	with respect to
i.e.	id est
e.g.	exempli gratia
w.l.o.g.	without loss of generality

List of Figures

List of Tables

Introduction

Almost a decade has passed since the introduction of AlexNet, the first Convolutional Deep Neural Network (DNN) which received attention of the general public by achieving groundbreaking results on the ImageNet competition. Since then, the community of researchers and industries employing Deep Learning (DL) has exploded. This surge in attention led to the development of State Of The Art (SOTA) algorithms in many different applications and fields: Computer Vision has been revolutionized by Convolutional DNNs [Krizhevsky et al., 2012, He et al., 2016], Natural Language Processing by Transformers [Devlin et al., 2019, Vaswani et al., 2017] and Time Series modeling by Recurrent DNNs/Long Short Term Memory (LSTM) networks/Gated Recurrent Units (GRU) networks [Goodfellow et al., 2016, Hochreiter and Schmidhuber, 1997, Chung et al., 2014].

Despite such incontrovertible success, Deep Learning does not come free of issues: designing DL models is highly costly, interpreting DNNs is not easy, deploying SOTA DL models often requires very specialized experts and, not least, *any* DNN requires a large amount of data for training. Moreover, from a theoretical standpoint many important guarantees on optimization convergence and generalization are still lacking.

Efforts spent to analyze and better understand the empirical success of DNNs followed mainly two lines of work: *representation learning* [He et al., 2016, Achille and Soatto, 2017, Zhang et al., 2017] and *optimization* [Li et al., 2018, Achille et al., 2019, Chaudhari and Soatto, 2018, Li et al., 2020].

Typically, DNNs are designed so that no hand crafted set of features is imposed, rather, features are learnt automatically from data. So that, studying Deep Learning generalization capabilities, in turn, corresponds to the study of hidden representations built within DNN models [Achille and Soatto, 2017, Zhang et al., 2017]. In general, one is interested in extracting a *sufficient representation* of available data which is also *minimal* to solve the given task. These opposing requirements guarantee DNNs' predictions to be less sensitive from detrimental nuisance factors and therefore better generalize to "unseen" data [Achille and Soatto, 2017, Achille et al., 2019].

On the other hand, theories of generalization of DNNs which focus on optimization usually consider DNNs as black-box models for which the optimal set of parameters (*weights*) is found according to some optimality criteria. The existance of the optimal set of weights, under the simplifying assumption of shallow Neural Networks, has been proved in [Cybenko, 1989, Hornik et al., 1989]. Nonetheless, no results on the learnability of the best set of parameters of shallow networks is known. So that, even if an optimal set of parameters is known to exist, there is no guarantee that it can be found by means of standard optimization methods performed on finite samples of data. Despite this daunting observation which seems to jeopardize Neural Networks learning, recent years have been a clear proof of the gap between theory and practice: vastly over-parametrized models do generalize well even if statistical learning theories [Vapnik, 1998] (bias-variance trade-off) would predict severe overfitting. Recent results suggest this is mainly due to non-trivial interaction between the loss landscape of over-

parametrized models and optimization schemes, such as Stochastic Gradient Descent (SGD) [Chaudhari and Soatto, 2018, Goodfellow et al., 2016, Kingma and Ba, 2014], that are used to find the best set of weights [Achille and Soatto, 2017, Zhang et al., 2017]: e.g. from [He et al., 2016, Li et al., 2018, Jacot et al., 2018] we know that the more a DNN model is over-parametrized the smoother its loss landscape is. Hence over-parametrization makes optimization "easier": over-parametrized DNNs models suffer less from typical first-order optimization pitfalls.

As a further layer of complexity, prior knowledge has proven to be fundamental in Deep Learning: specialized architectures (endowed with a strong inductive bias) outperform general purpose fully-connected DNNs. For example, many of the most remarkable successes in recent years in Computer Vision have been achieved by means of specialized architectures such as Convolutional Neural Networks [Krizhevsky et al., 2012, He et al., 2016] or Transformers [Dosovitskiy et al., 2021]. A similar observation holds for Time Series prediction [Bai et al., 2018, Oreshkin et al., 2019, Zancato et al., 2020] and Natural Language Processing [Vaswani et al., 2017]. So that it is clear that building highly specialized architectures is key to obtain state of the art results when applying Deep Learning in practice. Together with the choice of good inductive biases for a given target domain the design of suitable regularization schemes proved to be essential for DNNs generalization too. Regularization can be considered either explicitly, by adding suitable penalty terms on a standard training loss [Bansal et al., 2018, Golatkar et al., 2019], or implicitly, by the choice of optimization algorithm (e.g. SGD [Chaudhari and Soatto, 2018, Achille and Soatto, 2017]).

All the elements described so far have proven to be essential in enabling the huge success experienced by Deep Learning based models in recent years. Overall, the aim of this thesis is two-fold:

- Analyze the optimization trajectories and the generalization in the typical over-parametrized regime;
- Design a specialized inductive bias and regularization scheme which foster interpretability and generalization.

Outline of the Thesis

The goal of the thesis is to provide answers to the following open questions: what is the connection between learning dynamics and generalization of over-parametrized DNNs? How does modern deep learning theory compares with standard practice? Can we exploit domain prior knowledge and explicit regularization to improve DNNs learning and interpretablity?

In particular, the thesis is divided into three main parts. Part I describes both the general setup of learning with DNNs (Chapter 1) and the theory of learning in the over-parametrized regime (Chapter 2). Part II analyzes the learning dynamics of DNNs models under typical large scale learning problems. The theory we develop in Chapter 3

and Chapter 4 does not depend on a particular application domain (e.g. image processing or time series analysis). On the other hand Part III exploits domain knowledge and prior information to design DNN model structures and explicit regularization schemes (Chapter 5, Chapter 6 and Chapter 7).

In Chapter 3 we tackle the problem of predicting the number of optimization steps that a pre-trained deep network needs to converge to a given value of the loss function. To do so, we leverage the fact that the training dynamics of a deep network during fine-tuning are well approximated by the Neural Tangent Kernel (NTK) dynamics [Jacot et al., 2018, Lee et al., 2019, Arora et al., 2019b]. This allows us to approximate the training loss and accuracy at any point during training by solving a low-dimensional Stochastic Differential Equation (SDE) in function space. Using this result, we are able to predict the time it takes for Stochastic Gradient Descent to fine-tune a model to a given loss without having to perform any training.

The results in this chapter do not address a particular task, or a particular dataset, but rather address the technical issue of how to predict convergence time and hence compute resources. Nonetheless the driving force of the material developed in Chapter 3 started to unblock the adoption of real-world *Computer Vision* AutoML systems: Users fine-tune models selected from a large model zoo testing hundreds of combinations of different architectures, pre-training sets and hyper-parameters, but are reluctant to do so without visibility of the expected rough order of magnitude cost of the training. As such, we expect our method benefits individual researchers with limited computational resources, by allowing them to optimize for maximum impact by estimating the time and therefore the cost of optimizing DNN models.

In our experiments, we are able to predict training time of ResNets applied to image classification tasks within a 20% error margin on a variety of datasets and hyper-parameters, at a 30 to 45-fold reduction in cost compared to actual training. We also discuss how to further reduce the computational and memory cost of our method, and in particular we show that by exploiting the spectral properties of the gradients' matrix it is possible to predict training time on a large dataset while processing only a subset of the samples.

This chapter is based on the results presented on [Zancato et al., 2020]:

Zancato, L., Achille A., Ravichandran A., Bhotika R. and Soatto, S. *Predicting Training Time Without Training*, in Advances in Neural Information Processing Systems 2020.

In Chapter 4 we consider an image classification task and address the problem of choosing the best pre-trained DNN within a collection of DNN models ("model zoo") when only the target dataset is known and without training any model.

In particular, fine-tuning from a collection of models pre-trained on different image domains is emerging as a technique to improve test accuracy in the low-data regime. However, model selection, i.e. how to pre-select the right model to fine-tune from a model zoo without performing any training, remains an open topic. The results in Chapter 4 directly help towards better understanding of transfer learning through

a novel study on the interplay between generalization and highly over-parametrized DNNs. We use a linearized framework (based on NTK theory) to approximate fine-tuning, and introduce two new baselines for model selection – Label-Gradient and Label-Feature Correlation. Since all model selection algorithms in the literature have been tested on different use-cases and never compared directly, we introduce a new comprehensive benchmark for model selection comprising of many target tasks and a model zoo of single and multi-domain models. Our benchmark highlights accuracy gain with model zoo compared to fine-tuning Imagenet models. We show our model selection baseline can select optimal models to fine-tune in few selections and has the highest ranking correlation to fine-tuning accuracy compared to existing algorithms.

This chapter is based on the results presented on [Deshpande et al., 2021]:

Deshpande A., Achille A., Ravichandran A., Hao L., Zancato L., Fowlkes C., Bhotika R., Soatto S. and Perona P. *A linearized framework and a new benchmark for model selection for fine-tuning*, ArXiv CoRR, vol. abs/2102.00084, 202, 2021.

In Chapter 5 we consider the task of incorporating inductive bias in general DNN architectures. Differently from previous chapters the results we present in this chapter are restricted to time series and do not straightforwardly generalize to images.

In particular, we foster generalization of temporal DNNs, fully-connected or Temporal Convolutional Networks (TCN), by constraining their representational power by means of a proper inductive bias (on the architecture) and an explicit regularization scheme (on the loss function). Our inductive bias hinges on the *Fading Memory* property of certain class of dynamical systems and allows for automatic complexity selection based solely on available data, in this way the number of hyper-parameters that must be chosen by the user is reduced. Our automatic complexity selection criterion is based on the empirical Bayes procedure (Type II Maximum Likelihood) so that the usual trade-off between model fitting and model complexity is automatically solved.

In Chapter 6 we extend the results of Chapter 5 to a classic non-linear system identification problem. Exploiting the highly parallelizable DNN framework (based on stochastic optimization) we successfully apply our method to large scale datasets where typical SOTA methods for non-linear system identification cannot be applied without approximations.

This chapter is based on the results presented on [Zancato and Chiuso, 2021]:

Zancato L. and Chiuso A. *A novel Deep Neural Network architecture for non-linear system identification*, 19th IFAC Symposium on System Identification SYSID 2021.

In Chapter 7 we present STRIC, a residual-style architecture for interpretable forecasting and anomaly detection in multivariate time series. Our architecture is composed of stacked residual blocks designed to separate components of the signal such as trend, seasonality and linear dynamics. These are followed by a Temporal Convolutional Net-

work (TCN) that can freely model the remaining components and can aggregate global statistics from different time series as context for the local predictions of each time series. The architecture can be trained end-to-end and automatically adapts to the time scale of the time series. Then, we use an anomaly detection system based on the classic CUMSUM algorithm and a variational approximation of the f-divergence to detect both isolated point anomalies and change-points in statistics of the signals.

Anomaly detection in time series is crucial for reliable and fair machine learning systems. Nowadays, complex machine learning methods are deployed on many systems that make critical data driven decisions in a broad class of domains such as: factory manufacturing, autonomous driving, surveillance, inventory management, etc. We specifically design STRIC to be interpretable and robust, so that it can be "safely" applied outside of the context in which it is trained. Moreover, we expose interpretable parameters to the user so that it is possible to calibrate our method depending on the desired operating point.

STRIC outperforms both state-of-the-art robust statistical and Deep Learning based methods on typical time series benchmarks. To further illustrate the general applicability of STRIC, we show that it can be successfully employed on complex data such as text embeddings of newspaper articles.

This chapter is based on the results presented on [Zancato et al., 2022]:

Zancato L., Achille A., Paolini G., Chiuso A. and Soatto S. *STRIC: Stacked Residuals of Interpretable Components for Time Series Anomaly Detection.*, ArXiv (under review)

Chapter 8 summarizes the main contributions of the thesis and describes interesting avenues for future research.

I

Background

1

Deep Learning Methods

The goal of Machine Learning (ML) is to build algorithms that can "learn" from data. From here on, the term *learning* will be referred to the process of converting experience into expertise. Despite their intuitive meaning, how should we interpret experience and expertise in the ML parlance?

The Cambridge dictionary defines experience as follows: "knowledge or skill you get from doing, seeing, or feeling something", clearly this definition does not easily apply to an algorithm (computer program). What does it mean for an algorithm to see or to feel something? The simplest way to rephrase this "human-based" definition into ML parlance is referring to experience as the information that a human being or an algorithm can gather interacting with his/her or its own environment. In this sense every person acquires information interacting with the world around him/her as well as a computer program acquires information (in the form of data) from its own accessible world.

How should an algorithm convert data into expertise? Defining expertise can be even more puzzling from a ML perspective than defining experience. Expertise is intimately connected with generalization which in turn is related to the ability of abstract over experience to extract semantic knowledge. The problem of testing whether a program is capable of semantic reasoning (behaviour) has been faced since Alan Turing proposed his famous test in 1950. In this chapter we shall describe the statistical learning framework typically used to describe ML in general and in particular Deep Learning. We shall describe both the process of converting experience into expertise (so called *model training*) and the generalization index used to assess algorithm's expertise (so called *model assessment*). Moreover we shall describe typical DNNs architectures and optimization algorithms used in practice.

1.1 Machine Learning vs Optimization

In this section we shall describe the main ingredients of a Machine Learning problem. We shall consider a generic learner and describe what its experience is and how to exploit such experience to choose the best learner among a family of learners. The best learner is chosen according to a performance index designed to foster its generalization.

1.1.1 Learning ingredients

The learning model. Throughout the thesis we shall consider the learner as a generic algorithm/model which is thought as a function mapping from an input space to an output space. We denote a model with $f \in \mathcal{F}$ where $\mathcal{F}$ is the model class (hypothesis set). In general the model class might contain either parametric models (e.g. DNNs) or non-parametric ones (e.g. kernel models). Let the input space be $\mathcal{I}$ (e.g. images, text documents, ...) and the output space be $\mathcal{O}$ (e.g. labels), so that we can write the model as the following map: $f : \mathcal{I} \to \mathcal{O}$, $x \mapsto f(x)$.

The data generation model. We assume all data are generated by some probability distribution $\mathcal{P}$ defined over the input $\mathcal{I}$ and output $\mathcal{O}$ spaces. In particular, we shall consider $(x, y) \sim \mathcal{P}$, where $x \in \mathcal{I}$ is the input of the learner f while $y \in \mathcal{O}$ is the target value associated to the input x.

The experience. We call the data used to learn the model f *training data*, and denote this set with $\mathcal{D} := \{(x_i, y_i)\}_{i=1}^N$ where $(x_i, y_i) \in \mathcal{I} \times \mathcal{O}$ for $i \in [N]$. We denote the set containing only the input training data as $\mathcal{X} := \{x_i\}_{i=1}^N$ and the set containing only the output training data as $\mathcal{Y} := \{y_i\}_{i=1}^N$.

The measure of success. To measure the approximation capability of a given learner f it is customary to introduce a loss function. The loss function is designed to measure the difference between target data y and the learner's prediction $\hat{y}$: $l : \mathcal{O} \times \mathcal{O} \to \mathbb{R}_+$, $y \times \hat{y} \mapsto l(y, \hat{y})$, where $\mathbb{R}_+$ is the set of non-negative real numbers. Depending on the task to be solved (e.g. classification or regression) the choice of the loss function is different. For classification problems (e.g. image/text multiclass classification) the Cross-Entropy loss is used, while for regression problems the squared loss is typically chosen.

1.1.2 Training vs Generalization

Once the loss function is chosen, the overall measure of goodness (*generalization error/loss*) of a model f is obtained by averaging its performance over the data generating

distribution $\mathcal{P}$:

$$\mathcal{L}_{\mathcal{P}}(f) := \mathbb{E}_{(x,y)\sim\mathcal{P}}\left[l(y, f(x))\right] \tag{1.1}$$

unfortunately we do not have access to this quantity since we do not know $\mathcal{P}$. Nonetheless we can observe a finite number of samples drawn from $\mathcal{P}$, so that in practice the following average measure of performance (*training error/loss*) is used:

$$\mathcal{L}_{\mathcal{D}}(f) := \frac{1}{N}\sum_{i=1}^{N} l(y_i, f(x_i)) \tag{1.2}$$

when clear from the context we shall refer to the training error using the shorthand notation $\mathcal{L}(f)$. Note that the training error is an empirical estimator of the generalization error.

Ideally, one should look for the optimal learner by minimizing its generalization error Eq. (1.1):

$$f^* := \arg\min_{f\in\mathcal{F}} \mathcal{L}_{\mathcal{P}}(f) \tag{1.3}$$

Since the generalization error is typically not known, it is customary to find a good model by solving the so-called *Empirical Risk Minimization* (ERM) which is defined as:

$$f_{\text{ERM}} := \arg\min_{f\in\mathcal{F}} \mathcal{L}_{\mathcal{D}}(f) \tag{1.4}$$

Under suitable normalization, it holds that $\mathcal{L}_{\mathcal{D}}(f) \xrightarrow{p} \mathcal{L}_{\mathcal{P}}(f)\ \forall f$ as $N \to \infty$ [Van der Vaart, 2000], despite the convergence of the training error to generalization error we are not guarantee that $f_{\text{ERM}} \xrightarrow{p} f^*\ \forall x \in \mathcal{I}$. Such convergence is more delicate to be proved, nonetheless some results exist, we refer the interested reader to [Van der Vaart, 2000] for further details.

In practice, one has only access to a finite (limited) number of data so that, in general, no convergence of the solution of ERM to the optimal learner f^* is expected. Hence, solving the ERM does not necessarily minimize the generalization error, the gap between generalization and training error is known as *generalization gap*. A model learnt by ERM with high generalization gap is said to overfit the training data: the overfitting phenomenon occurs when a model describes well the observed data $\mathcal{D}$ but its prediction capability on other data from $\mathcal{P}$ is poor. Overfitting is typically associated with complex model structures which can easily capture spurious fluctuations (due to "noise") of the training data. The standard argument in the choice of the optimal complexity follows the bias-variance trade-off: a large model class $\mathcal{F}$ has low bias and

high variance, while a relatively rigid model class has high bias and low variance. The bias should be considered as the extent to which the average prediction, computed over all possible datasets of a given size extracted from $\mathcal{P}$, differs from the optimal regression function f^*. On the other hand, the variance measures the extent to which the ERM solutions for individual datasets vary around their average.

In practice, solving the bias-variance trade-off entails non trivial difficulties. Many successful approaches exploit either prior knowledge to design a suitable model class (Chapter 5) or techniques specifically built to perform model selection (Chapter 4).

In the following sections we shall briefly describe the model class of Deep Neural Networks, in particular we shall specify two model classes: fully-connected DNNs and convolutional DNNs. The first is a general purpose model class while the second one is built with a strong inductive bias towards input data that lie in a grid-like topology. We shall describe in great detail other inductive biases and their respective architecture design choices in Chapter 5. We shall devote Chapter 4 and Section 2.3.6 to the model selection problem.

1.2 DNN architectures

In this section we shall describe widely adopted DNNs architectures: fully connected and convolutional Neural Networks. These two simple types of DNNs are the basic building blocks of more specialized architectures which we shall consider in Chapter 5.

1.2.1 Fully connected Neural Networks

Fully connected Neural Networks are general purpose Neural Networks and are considered to be the corner stone of Deep Learning. Typically, deep fully connected networks are represented by the composition of many non-linear layers connected in a chain. Each stage in the chain is called hidden layer while the last layer is called output layer, the number of hidden layers defines the depth of the deep fully connected network. The outputs of each hidden layer are to be considered as hidden representations of the input data and are typically learnt at training time (during optimization). The dimensions of the hidden representations determine the width of the hidden layers (which need not to be uniform across the network).

We shall denote the output of a generic DNN with $f_w(x) \in \mathbb{R}^C$ where $x \in \mathbb{R}^d$ is a datum. Let $g^{(0)}(x) := x \in \mathbb{R}^d$ and $d_0 = d$, we define an L-hidden-layer fully-connected

neural network as:

$$f^{(h)}(x) = W^{(h)} g^{(h-1)}(x) \in \mathbb{R}^{d_h}, \quad g^{(h)}(x) = \sigma(f^{(h)}(x)) \in \mathbb{R}^{d_h}, \quad h = 1, ..., L \tag{1.5}$$

where $W^{(h)} \in \mathbb{R}^{d_h \times d_{h-1}}$ is the weight matrix of the h-th layer, $\sigma : \mathbb{R} \to \mathbb{R}$ is a coordinate-wise activation function. The last layer of the neural network is $f_w(x) = f^{(L+1)}(x) = W^{(L+1)} g^{(L)}(x)$, where $W^{(L+1)} \in \mathbb{R}^{C \times d_L}$ and we stack all the parameters of the DNN in a single vector $w = \left(\text{vec}(W^{(1)})^T, ..., \text{vec}(W^{(L+1)})^T\right)^T$.

From the user's perspective the hyper-parameters of the fully connected DNN are: the activation functions σ (e.g. ReLU, logistic sigmoid, hyperbolic tangent), the number of hidden layers L and dimension of each hidden layer d_h (the number of hidden units/neurons).

Remark 1.1 (Non-linear hidden features). Typically the user has not control over the hidden representations of the network: $g^{(h)}(x)$. These are automatically learnt by the network during optimization. Therefore prior knowledge should be encoded in the design of the architecture rather than in features design (see Chapter 5).

1.2.2 Universal approximation property and depth

We now consider the approximation/representation capabilities of fully connected Neural Networks. We are interested in answering the following: what class of functions can be approximated using a general fully connected Neural Network?

At first, one might think the structure of a fully connected Neural Network should be different depending on the kind of function to be learnt. Interestingly, this is not true. Authors of [Cybenko, 1989, Hornik et al., 1989] proved that fully connected single hidden layer Neural Networks possess the universal approximation property: any fully connected Neural Network with a linear output layer and at least one hidden layer with any "sigmoid" activation function can approximate any Borel measurable function with any desired nonzero amount of error on compact sets, provided enough neurons are used. A similar result holds true if ReLU activations are used [Goodfellow et al., 2016].

The universal approximation theorem guarantees that no matter how complicated the optimal regression function (the goal of learning) is, a sufficiently large Neural Network is enough to approximate the unknown function with any desired amount of precision. Despite such a reassuring result, learning a Neural Network from data can fail for different reasons: the optimization algorithm used while training might not be able to find the correct value of the parameters to represent the regression function (e.g.

first order optimization methods can be trapped in local minima of the cost function) or the training algorithm might overfit the training data.

Moreover the complexity (measured by the number of neurons) of the Neural Networks required for approximating a given regression function might be humongously large and not obtainable in practice. In these circumstances deeper Neural Networks are preferred to shallow ones since deeper models can reduce the number of neurons required to represent the unknown function [Montufar et al., 2014] (see Chapter 5).

1.2.3 Convolutional Networks

Convolutional Neural Networks (CNNs), are specialized types of Neural Networks to process data with a known grid-like topology. Some prototypical examples of data on grids are time series (1-D grid, index by time) and images (2-D grid, indexed by width and height). In the following we shall be interested only on regular grids, i.e. equally spaced grids, which in the previous examples simply mean: time series sampled at regular time instants and regular images with squared pixels. The main operation implemented on CNNs are *convolutions*: a "specialized" kind of linear operation. While fully-connected DNNs are characterized by multiplication with dense matrices, CNNs are characterized by the application of linear convolutions (which can be characterized by Toepliz matrices).

Discrete convolutions

We now briefly introduce the notation used to describe discrete convolutions which are the basic building blocks of CNNs.

Definition 1.1 (Discrete convolution)**.** Given two discrete (infinite) sequences $x(t) \in \mathbb{R}$ and $\omega(t) \in \mathbb{R}$ we denote the convolution between the sequence x and ω with $(x * \omega)(t) \in \mathbb{R}$ and define it as:

$$g(t) := (x * \omega)(t) := \sum_{i=-\infty}^{\infty} \omega(i)x(t-i) \tag{1.6}$$

Remark 1.2. Note the convolution is a linear operator and can be trivially applied to finite sequences by simply considering a fine sum.

Typically, the convolved signals have very specific meanings, if we consider the signal $x(t)$ as the *input* to the convolutional filter then $\omega(t)$ is called *kernel* (and it characterizes the linear convolutional filter), moreover the output signal $g(t)$, in ML parlance, is called *feature*.

Definition 1.2 (2-D convolution)**.** Given two two-dimensional images (represented as matrices) $I \in \mathbb{R}^{w\times h}$ and $K \in \mathbb{R}^{w\times h}$, we denote the 2-D convolution between the image I and K with $(I * K)(i,j) \in \mathbb{R}^{w\times h}$ and define it as:

$$G(i,j) := (I * K)(i,j) := \sum_{m,n} I(i-m, j-n)K(m,n)$$

One of the most important property of discrete convolutions is that they can be represented by Toepliz matrices for 1-D convolutions and doubly block circulant matrices for 2-D convolutions. This makes them extremely computational and memory efficient, in fact, a matrix product can be easily parallelized with GPUs and storing a $N \times N$ Toepliz matrix only costs $O(N)$. Moreover, typically one wants the kernel to be local so that the number of non-zero coefficient of convolutional filters is much smaller than N, in turn this implies a sparse matrix representation which highly reduces both computational and memory costs.

Remark 1.3 (Any fully-connected NN can represent convolutions)*.* Any fully-connected Neural Network can represent convolutions since it is built with dense matrices which admit Toepliz matrices or doubly block circulant matrices as special cases. Despite this, the cost of storing a dense matrix in memory is $O(N^2)$ and there is no general way to improve GPU computations for dense matrices (for which the structure is not known a priori).

Despite their computational advantages, convolutions are grounded on three important ideas: sparse interactions, parameter sharing and equivariant representations. We refer to Section 5.1.1 for a more in depth discussion on these important properties which are known to greatly help DNNs optimization.

1.3 Gradient Descent and Gradient Flow

In this section we introduce standard Gradient Descent (GD) and Gradient Flow (GF) algorithms. Typically we solve the ERM using iterative first order optimization algorithms so that GD (or its variant SGD) is the standard optimization algorithm used in practice. Nonetheless GF is a stepping stone toward understanding discrete algorithms and has been thoroughly studied by many authors [Du et al., 2018a, Du et al., 2019b, Allen-Zhu et al., 2019a, Nitanda and Suzuki, 2020, Chen et al., 2020]. Note that GF approximates GD as we take smaller discretization steps (equivalently reduce the step sizes).

Gradient Descent

The gradient descent algorithm belongs to the family of optimization algorithms called iterative algorithms. It is typically applied to solve the ERM problem in Eq. (1.4) when the loss function is differentiable w.r.t. to the parameters of the model $f_w \in \mathcal{F}$. The main idea underlying GD is to generate a relaxation sequence $\{\mathcal{L}(w_k)\}_{k=0}^{\infty}$, i.e. a sequence for which it holds:

$$\mathcal{L}(w_{k+1}) \leq \mathcal{L}(w_k) \qquad k = 0, 1, ...$$

Since $\mathcal{L}(w) \geq 0 \ \forall w$ and $\mathcal{L}(w_0) < \infty$ the relaxation sequence converges (not necessarily to a global minimum if the problem is not convex). The standard way to build such a sequence is by applying updates on the current parameters following the steepest descent direction: the negative gradient $\nabla_w \mathcal{L}(w_k)$.

So that the update equation of GD optimization algorithm is:

$$w_{k+1} = w_k - \eta \nabla_w \mathcal{L}(w_k) \tag{1.7}$$

where $\eta \in \mathbb{R}_+$ is known as the learning rate (or step size). Hence, given an initial parameter w_0 (initial condition) by applying Eq. (1.7) we are guaranteed to converge to a local minimizer of the ERM problem (provided the step size is chosen sufficiently small).

Gradient Flow

It is possible to approximate Eq. (1.7) by an Ordinary Differential Equation (ODE) if we consider the step size infinitesimal. In particular if we denote with dt the discretization step of the GD algorithm we have the following:

$$w(t + dt) = w(t) - \eta dt \nabla_w \mathcal{L}(w(t)) \rightarrow \dot{w}(t) = -\eta \nabla_w \mathcal{L}(w(t)) \qquad dt \rightarrow 0$$

where we decouple the infinitesimally small learning rate as the product of two contributions: dt which is infinitesimally small and η which can be interpreted as the rate at which the infinitesimal step size goes to zero.

Remark 1.4 (Step-size interpretation). The meaning of η in the two update equations is different. For GD it represents the magnitude of the step taken in the direction of the negative gradient, while for GF η parametrizes the time evolution of the optimization trajectories (typically it is assumed to be 1).

1.4 Stochastic Gradient Descent

Once inductive bias and regularization (i.e. complexity criterion) are fixed, learning a DNN boils down to solving Eq. (1.4). In this section we briefly describe the most widely employed optimization algorithm when optimizing DNNs: Stochastic Gradient Descent.

To begin with, SGD is a first order optimization algorithm so that it only requires gradient computations to solve the ERM problem. The main difference w.r.t. to classical first order optimization schemes, such as GD, is that SGD only requires an unbiased estimate of the gradients. The standard way to compute an unbiased estimator of the training loss is by approximating it with random mini-batches whose size B is much smaller than the number of training data N. Let $\mathcal{B} = \{x_{j_i}\}_{i=1}^{B}$ be a set of size B indices, where the indices $\{j_i\}_{i=1}^{B}$ are extracted randomly from $[N]$. The unbiased estimate of the full gradient is:

$$\nabla_w \mathcal{L}_{\mathcal{D}}(w) := \frac{1}{N}\sum_{i=1}^{N} \nabla_w l(y_i, f_w(x_i)) \approx \frac{1}{B}\sum_{i=1}^{B} \nabla_w l(y_{j_i}, f_w(x_{j_i})) =: \nabla_w \mathcal{L}_{\mathcal{B}}(w)$$

where $\nabla_w \mathcal{L}_{\mathcal{D}}(w)$ is the full gradient on the training data and $\nabla_w \mathcal{L}_{\mathcal{B}}(w)$ is the mini-batch stochastic gradient.

Remark 1.5 (Computational Complexity). The cost of computing the complete gradient or the stochastic one is linear w.r.t. the number of data used. Hence when N is large, we can choose B small to reduce the computational complexity: computations over a batch can be parallelized and easily stored in modern hardware (GPUs). Nonetheless as we reduce the number of data in the mini-batch the variance of the stochastic gradient increases (i.e. the informativity of the stochastic gradient in optimizing the training loss decreases).

Remark 1.6 (Other Stochastic Optimization methods). Other stochastic optimization methods used in practice are: Adagrad [Duchi et al., 2011], RMSProp [Tieleman et al., 2012] and Adam [Kingma and Ba, 2014]. All of which are variations of the SGD update rule.

Remark 1.7 (SGD convergence). To ensure convergence of the SGD algorithm it is often required some kind of learning rate schedule. Typical results on SGD convergence require the learning rates to satisfy the following: $\sum_{k=1}^{\infty} \eta_k = \infty$ and $\sum_{k=1}^{\infty} \eta_k^2 < \infty$, where η_k is the learning rate at the k-th iteration [Bottou, 1998].

1.4.1 Optimization Challenges

In this section we report some well-known optimization challenges in DNNs training.

Ill-conditioning

The learning trajectory induced by a first order method is highly sensitive to local curvature. If the Hessian at current iteration is ill-conditioned (i.e. it has both high and low curvature directions) it might happen that the convergence speed of gradient based algorithms decreases. Intuitively, this phenomenon can be described as follows: at each gradient update GD (or SGD) bounces back and forth in high curvature directions while making slow progress in low curvature directions. Notably, along the directions with high curvature the learning dynamics could become unstable if the step size is not chosen "sufficiently" small (depending on the condition number of the Hessian).

Local Minima

Every local minimum that is not a global one might badly affect the learning dynamics. These minima might trap first order optimization methods within a basin of attraction whose loss is higher than the optimal one. It is still an open question whether these minima are present in over-parametrized real world DNNs. Nonetheless it is widely believed that for a sufficiently large DNN most of the local minima have low value of the loss function. Hence, even if the training algorithm is trapped within a non-optimal basin of attraction its training loss is not too far from the optimal one [Goodfellow et al., 2016, Li et al., 2018, Keskar et al., 2016a].

Moreover SGD is known to have a reduced sensitivity to local minima w.r.t. GD. In fact, thanks to the gradient approximation used in SGD (which introduces "noise" during optimization) it is known that SGD could escape local minima.

Plateaus and Saddle points

Gradients near a saddle point can become very small, therefore an initialization or a learning trajectory near a saddle point might require a lot of iterations to escape from it (along the directions of negative eigenvalues). Interestingly, the noisy update of SGD allows the learning trajectories to increase the probability of avoiding saddle points (similar argument used for local minima). So that the effects of saddles is only to increase the training time (i.e. number of iterations required to solve ERM).

As the number of saddle points in the loss landscape of DNNs increases, it is more likely for optimization trajectories to slow down so that the training time increases.

Unfortunately, no result in known to count or measure the proximity of saddle points for DNNs: proliferation of saddle points in the loss landscape of DNNs is an active area of research [Kawaguchi, 2016, Goodfellow et al., 2016].

1.5 Batch normalization

We now synthesize Batch Normalization (BN) [Ioffe and Szegedy, 2015]: a common technique used to improve learning of DNNs. The training of general deep fully-connected or convolutional Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during optimization. Usually, this phenomenon (called internal covariate shift) decreases convergence speed since smaller learning rates are used (see Section 1.4) and increases the sensitivity w.r.t. parameters initialization.

The main idea of Batch Normalization is to normalize each layer (activations) of a DNN model for each mini-batch during training, in this way the internal covariate shift is reduced and the aforementioned problems are attenuated. Moreover, BN has a beneficial effect on the gradients propagation through the network [Ioffe and Szegedy, 2015] since it reduces the dependence of gradients on the scale of the parameters or of their initial values.

For the sake of simplicity we now consider the activations of a fully-connected DNNs $g^{(h)}(x)$ (see Eq. (1.5) and note that BN can be applied also to CNNs).

Consider a mini-batch $\mathcal{B} = \{x_{j_i}\}_{i=1}^{B}$ of size B, where the indices $\{j_i\}_{i=1}^{B}$ are randomly chosen from $[N]$. BN normalizes the activations of each hidden layer by applying the following equations:

$$
\begin{aligned}
\mu_{\mathcal{B}}^{(h)} &:= \frac{1}{B}\sum_{i=1}^{B} g^{(h)}(x_{j_i}) \\
{\sigma^2}_{\mathcal{B}}^{(h)} &:= \frac{1}{B}\sum_{i=1}^{B} \left(g^{(h)}(x_{j_i}) - \mu_{\mathcal{B}}^{(h)}\right)^2 \\
\bar{g}^{(h)}(x_{j_i}) &:= \gamma \frac{g^{(h)}(x_{j_i}) - \mu_{\mathcal{B}}^{(h)}}{\sqrt{{\sigma^2}_{\mathcal{B}}^{(h)} + \epsilon}} + \beta \qquad \forall j_i \in \{j_i\}_{i=1}^{B}
\end{aligned}
$$

where the division is performed component-wise, $\bar{g}^{(h)}(x_{j_i}) \in \mathbb{R}^{d_h}$ is the output of the BN layer and γ and β are parameters to be learnt during optimization [Ioffe and Szegedy, 2015].

Note BN normalizes each component of the h-th activation layer independently.

2

Neural Tangent Kernel

Learning dynamics of DNNs is widely believed to be one of the most important factors influencing DNNs generalization, which at the time of writing has not been fully understood yet. Many conundrum permeate empirical successes of DNNs. A typical example which collides with decades of Learning theory successes [Vapnik, 1998, Friedman et al., 2001] is synthesized in [Zhang et al., 2017]: SOTA over-parametrized DNNs possess strong generalization performance while having enough complexity to fit random labeling of the training data. These findings have been empirically demonstrated to hold also when explicit regularization is added and even when true data is replaced by random noise. In such cases standard Learning Theory results would not predict good generalization due to data overfitting. The direct connection with the *learning dynamics* is the empirical observation that the random lables/random data regimes are characterized by a slower convergence w.r.t. the "clean" data regime. Following this observation many authors developed theories and algorithms to directly analyze the highly-complex dynamics of Deep Learning [Jacot et al., 2018, Li et al., 2018, Golatkar et al., 2019, Achille et al., 2019, Arora et al., 2019a, Zancato et al., 2020, Deshpande et al., 2021]. In this chapter we shall present one of these theories: the theory of Neural Tangent Kernel (NTK) of DNNs. By studying the NTK it is possible to analyze the learning dynamics of over-parametrized DNNs (i.e. the number of Network parameters larger than the number of data) and develop algorithms well suited to make predictions on the convergence speed and generalization of the trained models [Zancato et al., 2020, Deshpande et al., 2021].

The main idea in the NTK literature is to map the learning dynamics of Neural Networks into function space and exploit an infinite width limit (in the number of parameters) to convexify the learning problem [Jacot et al., 2018, Lee et al., 2019, Arora

et al., 2019a, Nitanda and Suzuki, 2020].

We define the NTK for scalar output Neural Networks trained under the squared loss in Section 2.1. Then, in Section 2.2 we describe its main properties: the convergence to a finite limit and the stability w.r.t. the GD or GF dynamics as the width of the DNN goes to infinity. In Section 2.3 we show that the gradient flow dynamics in the infinite width limit of a Neural Network is equivalent to that of a kernel regression under gradient flow. We then exploit linearized DNNs (Section 2.4) to study the trainability and generalization of infinitely wide DNNs in Section 2.5 (which we shall expand further in Chapter 3 and Chapter 4). We conclude this chapter by extending the results of Section 2.1 and Section 2.4 to vector valued output Neural Networks trained with the Cross-Entropy loss (Section 2.6) and SGD (Section 2.7).

2.1 Neural Tangent Kernel definition

To make the notation easier, in this section we shall consider the output of a DNN to be a scalar, everything still hold when the output is vector-valued which is the main case of interest for DNN models applied in practice (Section 2.6 and Chapter 3).

In the following we shall consider a DNN model with scalar output: $f_w(x) \in \mathbb{R}$, where $w \in \mathbb{R}^D$ and $x \in \mathbb{R}^d$ is the input datum. As defined in Chapter 1, we shall denote with $\mathcal{D} := \{(x_i, y_i)\}_{i=1}^N$ the training dataset where $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$, with $\mathcal{X} := \{x_i\}_{i=1}^N$ the set of input data and with $\mathcal{Y} := \{y_i\}_{i=1}^N$ the set of target labels. We shall now introduce the theory of NTK in the case of squared loss, since the most interesting quantities describing the learning dynamics can be written explicitly. Despite this simplifying assumption it is possible to analyze the learning dynamics under different loss functions, e.g. the Cross-Entropy loss (Section 2.6).

Consider a Neural Network model optimized for a regression task by minimizing the squared loss over the training data: $\mathcal{L}(w) := \frac{1}{2}\sum_{i=1}^N (f_w(x_i) - y_i)^2$. Then it is straightforward to prove the following proposition, which establishes the optimization dynamics in function space (i.e. on the DNN's outputs) is characterized by the training residuals and the matrix of inner products of gradients.

Proposition 2.1 (Neural Tangent Kernel Dynamics [Arora et al., 2019b])**.** *Consider minimizing the squared loss $\mathcal{L}(w)$ by gradient flow:*

$$\frac{dw(t)}{dt} = -\eta \nabla_w \mathcal{L}(w(t)) \tag{2.1}$$

Now let $f_{w(t)}(\mathcal{X}) := (f_{w(t)}(x_i))_{i\in[N]} \in \mathbb{R}^N$ be the vector of stacked outputs of the Neu-

ral Network with parameters $w(t)$ *and let* $y := (y_i)_{i\in[N]}$ *be the vector containing the stacked target outputs. With this definition we can write* $\mathcal{L}(w(t)) = \frac{1}{2}\left\|f_{w(t)}(\mathcal{X}) - y\right\|_2^2$. *Moreover, the evolution over time (i.e. during optimization) of* $f_{w(t)}(\mathcal{X})$ *is described by:*

$$\frac{df_{w(t)}(\mathcal{X})}{dt} = -\eta\Theta(t)(f_{w(t)}(\mathcal{X}) - y) \tag{2.2}$$

where $\Theta(t) \in \mathbb{R}^{N\times N}$ *is a positive semi-definite matrix defined as:*

$$(\Theta(t))_{i,j} := \left\langle \frac{\partial f_{w(t)}(x_i)}{\partial w}, \frac{\partial f_{w(t)}(x_j)}{\partial w} \right\rangle$$

Proof. The parameters w evolve according to the following differential equation [Arora et al., 2019b, Du et al., 2018a]:

$$\frac{dw(t)}{dt} = -\eta\nabla_w\mathcal{L}(w(t)) = -\eta\sum_{j=1}^{N}(f_{w(t)}(x_j) - y_j)\frac{\partial f_{w(t)}(x_j)}{\partial w}$$

We can then express the evolution of the network output on the i-th datum as:

$$\frac{df_{w(t)}(x_i)}{dt} = \left\langle \frac{\partial f_{w(t)}(x_i)}{\partial w(t)}, \frac{\partial w(t)}{\partial t} \right\rangle = -\eta\sum_{j=1}^{N}(f_{w(t)}(x_j) - y_j)\left\langle \frac{\partial f_{w(t)}(x_i)}{\partial w}, \frac{\partial f_{w(t)}(x_j)}{\partial w} \right\rangle$$

Since $f_{w(t)}(\mathcal{X}) = (f_{w(t)}(x_i))_{i\in[n]} \in \mathbb{R}^N$ is the vector containing the network's stacked outputs at each time instant t we get Eq. (2.2).

■

Remark 2.1 (Gradient Flow). The analysis of *gradient flow* is a stepping stone towards understanding discrete algorithms, and has been thoroughly studied by many authors [Du et al., 2018a, Du et al., 2019b, Allen-Zhu et al., 2019a, Nitanda and Suzuki, 2020, Chen et al., 2020]. See the connection between Gradient Flow and Gradient Descent in Section 1.3).

Proposition 2.1 characterizes the evolution in function space of any function $f_{w(t)}$ (in particular any Neural Network) under gradient flow dynamics. We now follow [Arora et al., 2019b] and define a Deep Network architecture whose width is allowed to go to infinity, while fixing the training data as above. As the width of the Neural Network increases, it can be shown that the matrix $\Theta(t)$ remains *constant* along optimization trajectory (i.e. $\Theta(t) = \Theta(0)$). Moreover, under random initialization of w, the random matrix $\Theta(0)$ converges in probability to a deterministic matrix Θ^* as the width of the

DNN goes to infinity. So that the gradient flow is described by the following ODE:

$$\frac{df_{w(t)}(\mathcal{X})}{dt} = -\eta\Theta^*(f_{w(t)}(\mathcal{X}) - y) \tag{2.3}$$

The limit object Θ^* is called *Neural Tangent Kernel* since it can be equivalently defined by the following kernel evaluated on training data:

$$k_{NTK}(x, x') = \mathbb{E}_{w\sim\mathcal{W}} \left\langle \frac{\partial f_w(x)}{\partial w}, \frac{\partial f_w(x')}{\partial w} \right\rangle \tag{2.4}$$

where $\mathcal{W}$ is the initial weights distribution.

2.2 Properties of the Neural Tangent Kernel

In this section we shall study the behaviour of the finite width NTK $\Theta(t)$ (*empirical NTK*) at initialization and during optimization for fully connected Neural Networks. We prove that as the width of the DNN goes to infinity the kernel matrix remains constant. To prevent degeneracies as the width of a DNN increases, we shall introduce a layer-wise normalization which is commonly known as NTK parametrization [Jacot et al., 2018, Arora et al., 2019a, Arora et al., 2019b, Allen-Zhu et al., 2019a, Nitanda and Suzuki, 2020]. We conclude the section by extending these results to infinitely wide Convolutional Neural Networks [Garriga-Alonso et al., 2018, Arora et al., 2019b].

2.2.1 Infinite width limit of fully-connected Neural Networks

Let $g^{(0)}(x) := x \in \mathbb{R}^d$ and $d_0 = d$, we define a scalar output L-hidden-layer fully-connected neural network parametrized according to the NTK parametrization as:

$$f^{(h)}(x) = W^{(h)}g^{(h-1)}(x) \in \mathbb{R}^{d_h}, \quad g^{(h)}(x) = \sqrt{\frac{c_\sigma}{d_h}}\sigma(f^{(h)}(x)) \in \mathbb{R}^{d_h}, \quad h = 1, ..., L \tag{2.5}$$

where $W^{(h)} \in \mathbb{R}^{d_h \times d_{h-1}}$ is the weight matrix in the h-th layer, $\sigma : \mathbb{R} \to \mathbb{R}$ is a coordinate-wise activation function, and $c_\sigma = (\mathbb{E}_{z\sim\mathcal{N}(0,1)}[\sigma(z)^2])^{-1}$. The last layer of the neural network is $f_w(x) = f^{(L+1)}(x) = W^{(L+1)}g^{(L)}(x)$, where $W^{(L+1)} \in \mathbb{R}^{1\times d_L}$ and $w = \left(\text{vec}(W^{(1)})^T, ..., \text{vec}(W^{(L+1)})^T\right)^T$ represents all the parameters of the DNN.

By random initialization we usually mean that all the weights are initialized to be i.i.d. $\mathcal{N}(0, 1)$ random variables, and by infinite width we mean $d_i \to \infty\ \forall i \in [L]$. The scaling factor after the activation function $\sqrt{c_\sigma/d_h}$ ensures that the norm of the input $g^{(h)}(x)$ for each layer $h \in [L]$ is approximately preserved at initialization [Arora et al.,

2019b].

Infinite width limit: It is not hard to prove that a fully-connected randomly initialized Neural Network converges to a Gaussian Process as its width goes to infinity. We refer to [Lee et al., 2017] for the proof, we recall here that the proof hinges on an application of the Central Limit Theorem applied layer-wise. In particular each coordinate of the pre-activation vectors $f^{(h)}(x) \in \mathbb{R}^{d_h}\ \forall h \in [L]$ tends to a zero mean Gaussian Processes (independent w.r.t. any other coordinate) with covariance $\Sigma^{(h-1)} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. It is straightforward to show each covariance matrix can be recursively defined as:

$$\Sigma^{(0)}(x, x') = x^T x', \tag{2.6}$$

$$\Lambda^{(h)}(x, x') = \begin{pmatrix} \Sigma^{(h-1)}(x, x) & \Sigma^{(h-1)}(x, x') \\ \Sigma^{(h-1)}(x', x) & \Sigma^{(h-1)}(x', x') \end{pmatrix} \in \mathbb{R}^{2\times 2}, \tag{2.7}$$

$$\Sigma^{(h)}(x, x') = c_\sigma \mathbb{E}_{(u,v)\sim\mathcal{N}(0,\Lambda^{(h)})}[\sigma(u)\sigma(v)]. \tag{2.8}$$

We can now define the Neural Tangent Kernel matrix for a randomly initialized and infinitely wide fully connected Neural Network $(\Theta^{(L)}(x, x'))$ as in [Arora et al., 2019b, Lee et al., 2017]. It is sufficient to apply Eqs. (2.6) to (2.8) to the NTK definition Eq. (2.4) to prove the following [Arora et al., 2019b]:

$$\Theta^{(L)}(x, x') = \sum_{h=1}^{L+1} \left(\Sigma^{(h-1)}(x, x') \prod_{h'}^{L+1} \dot{\Sigma}^{(h')}(x, x') \right) \tag{2.9}$$

where $\dot{\Sigma}^{(h)}(x, x') = c_\sigma \mathbb{E}_{(u,v)\sim\mathcal{N}(0,\Lambda^{(h)})}[\dot{\sigma}(u)\dot{\sigma}(v)]$ and we let $\dot{\Sigma}^{(L+1)}(x, x') = 1$.

Remark 2.2. Eq. (2.9) holds for any activation function σ.

2.2.2 Convergence of Empirical Neural Tangent Kernel

The natural question now is: does the finite width tangent kernel (for random initialization) converge to the infinite width NTK as the width of the DNN grows (i.e. $\Theta(0) \to \Theta^{(L)}$ as $d_h \to \infty\ \forall h$)?

Different results which guarantee such a convergence are known in literature [Jacot et al., 2018, Arora et al., 2019b], [Arora et al., 2019b] assumes the activation functions are ReLU activations (one of the most widely used activations in Deep Learning practice), while [Jacot et al., 2018] only assumes the activation function σ is a Lipschitz nonlinearity. The proof in [Jacot et al., 2018] requires a sequential limit of the hidden widths so that one at the time $d_1, ..., d_L$ are taken to infinity. We now report the result

given in [Arora et al., 2019b] which guarantees a non-asymptotic bound on the difference of the finite width kernel w.r.t. the infinite width one and only requires $\min_{h\in[L]} d_h$ to be sufficiently large (which is a strictly weaker condition than the one assumed in [Jacot et al., 2018]).

Theorem 2.1 (Convergence to the NTK at initialization [Arora et al., 2019b])**.** *Fix $\epsilon > 0$ and $\delta \in (0,1)$. Suppose $\sigma(z) = \max(0,z)$ (ReLU activation function) and $\min_{h\in[L]} d_h \geq \Omega\Big(\frac{L^6}{\epsilon^4}\log(L/\delta)\Big)$. Then for any inputs x, $x' \in \mathbb{R}^d$ such that $\|x\| \leq 1$, $\|x'\| \leq 1$, with probability at least $1-\delta$ we have:*

$$\left|\left\langle \frac{\partial f_w(x)}{\partial w}, \frac{\partial f_w(x')}{\partial w}\right\rangle - \Theta^{(L)}(x,x')\right| \leq (L+1)\epsilon \tag{2.10}$$

Remark 2.3 (Random initialization is essential). The assumption on random initialization of the network weights is necessary to prove the convergence under the infinite width limit. This guarantees the NTK matrix at initialization $\Theta(0)$ converges to the limiting matrix Θ^* which we called $\Theta^{(L)}$ for a L layer fully-connected Neural Network.

2.2.3 Stability of Empirical Neural Tangent Kernel

In this section we show the second remarkable property of the empirical NTK matrix: $\Theta(t) \to \Theta(0)$ $\forall t$ as the minimum width increases. We shall call this property: *stability of NTK under gradient flow*. The following theorems are taken from [Lee et al., 2019].

We begin by stating the working assumptions:

Assumption 2.1. *The widths of the hidden layers are identical $d_1 = ... = d_L$ (the proof naturally extends to the setting with different widths and $n := \min\{d_1, ..., d_L\} \to \infty$.*

Assumption 2.2. *The analytic NTK Θ^* is full-rank, i.e. $0 < \lambda_{min} \leq \lambda_{max} < \infty$ where $\lambda_{min} := \min(\Lambda(\Theta^*))$ and $\max(\Lambda(\Theta^*)) := \lambda_{max}$.*

Assumption 2.3. *The training set $\mathcal{D}$ is contained in some compact set and $x \neq x'$ $\forall x, x' \in \mathcal{X}$.*

Assumption 2.4. *The activation function σ satisfies $|\sigma(0)| \leq \infty$, $\|\sigma\|_\infty \leq \infty$ and*

$$\sup_{x\neq x'} |\sigma'(x) - \sigma'(x')|/|x - x'| < \infty$$

Remark 2.4 (Kernel positivity). Assumption 2.2 holds when $\mathcal{X} \subseteq \{x \in \mathbb{R}^d : \|x\|_2 = 1\}$ and $\sigma(x)$ grows non-polynomially for large x [Jacot et al., 2018].

Proposition 2.2 (Local Lipschitzness of the Jacobian [Lee et al., 2019]). *There is a constant $k_J > 0$ such that for every $c > 0$, with high probability over random initialization the following holds $\forall w, \tilde{w} \in \mathcal{B}(w_0, c) := \{w : \|w - w_0\|_2 < c\}$:*

$$\|\nabla_w f_w(\mathcal{X}) - \nabla_w f_{\tilde{w}}(\mathcal{X})\|_F \leq k_J \|w - \tilde{w}\|_2 \tag{2.11}$$

$$\|\nabla_w f_w(\mathcal{X})\|_F \leq k_J \tag{2.12}$$

where $\nabla_w f_w(\mathcal{X}) := \left(\frac{\partial f_w(x_1)}{\partial w} \middle| \frac{\partial f_w(x_2)}{\partial w} \middle| \dots \middle| \frac{\partial f_w(x_N)}{\partial w}\right)^T \in \mathbb{R}^{N \times D}$ is the Neural Network Jacobian evaluated on the dataset $\mathcal{X}$ and parameter w.

Remark 2.5 (Extension to other parametrizations). Proposition 2.2 does not require the NTK parametrization (normalization) and can be extended to other parametrizations as well: it is sufficient to consider $\mathcal{B}\left(w_0, \frac{c}{\sqrt{n}}\right)$ and multiply $\frac{1}{\sqrt{n}}$ on the left side of Eq. (2.11) and Eq. (2.12).

As implied by Eq. (2.9) $f_{w(0)}(x)$ converges to a zero mean Gaussian process as the width increases. Under the assumption of bounded inputs it is possible to guarantee with arbitrarily high probability that the prediction error at initialization $\left\|f_{w(0)}(x) - y\right\|_2$ is bounded.

Proposition 2.3 (Bounded input bounded errors). *For $\delta_0 > 0$ and $\|y\|_2 < \infty$, there exists a positive constant $0 < r_0 < \infty$ and $n_0 \in \mathbb{N}$ (both possibly depending on δ_0, the number of data N and the asymptotic covariance of the limiting Gaussian Process Eq. (2.9)). Such that $\forall n \geq n_0$, with probability at least $(1-\delta_0)$ over random initialization it holds: $\left\|f_{w(0)}(x) - y\right\|_2 < r_0$.*

Theorem 2.2 (Gradient Flow, NTK parametrization [Lee et al., 2019]). *Under assumptions 2.1 to 2.4, for $\delta_0 > 0$, there exists $r_0 > 0$, $\bar{n} \in \mathbb{N}$ and $k_J > 1$, such that for every $n \geq \bar{n}$, the following holds with probability at least $(1 - \delta_0)$ over random initialization when applying gradient flow with learning rate η (i.e. $\frac{dw}{dt} = -\eta \nabla_w \mathcal{L}(w(t))$):*

$$\left\|f_{w(t)}(\mathcal{X}) - y\right\|_2 \leq \exp^{-\frac{\eta \lambda_{min}}{3} t} r_0 \tag{2.13}$$

$$\|w(t) - w(0)\|_2 \leq \frac{3 k_J r_0}{\lambda_{min}} \left(1 - \exp^{-\frac{\eta \lambda_{min}}{3} t}\right) \tag{2.14}$$

We are now ready to prove the final result of this section:

Theorem 2.3 (Stability of NTK under gradient flow [Lee et al., 2019]). *Under assumptions 2.1 to 2.4, for $\delta_0 > 0$, there exists $r_0 > 0$, $\bar{n} \in \mathbb{N}$ and $k_J > 1$, such that for every*

$n \geq \bar{n}$, the following holds with probability at least $(1 - \delta_0)$ over random initialization when applying gradient flow with learning rate η (i.e. $\frac{dw}{dt} = -\eta \nabla_w \mathcal{L}(w(t))$):

$$\sup_t \|\Theta(0) - \Theta(t)\|_F \leq \frac{6k_f^3 r_0}{\lambda_{min}} \frac{1}{\sqrt{n}} \tag{2.15}$$

Remark 2.6 (Extensions to other parametrizations). It is possible to extend both Theorem 2.2 and Theorem 2.3 to other architectures as long as: Proposition 2.2 holds (i.e. the Jacobian is locally Lipschitz), the empirical NTK converges in probability and the limit is positive definite.

2.2.4 Convolutional Neural Networks

The results we presented so far can be extended to convolutional Neural Networks too. In particular, [Arora et al., 2019b, Garriga-Alonso et al., 2018] provide a closed form expression of a kernel obtained from convolutional Neural Networks at random initialization as the number of convolutional filters tends to infinity (this result is analogous to the one reported in Section 2.2.1). It is worth to point it out that the closed form applies to vanilla CNNs and CNNs with global average pooling while it is not applicable to more complex architectures (e.g. where batch normalization is used). In particular it is worth noting the closed form computations required to compute the Neural Tangent Kernel for this class of architectures can be performed efficiently by means of dynamic programming. With this efficient implementation it has been possible to apply the convolutional NTK to large datasets such as CIFAR-10 [Arora et al., 2019b] achieving classification accuracy within 6-7% of that of the corresponding CNN architecture (SOTA result of a kernel based method on CIFAR10).

2.3 Connection with Kernel methods

The goal of this section is to show the gradient flow dynamics in the infinite width limit Eq. (2.3) is equivalent to that of a kernel regression under gradient flow. This establishes a direct connection between Neural Networks training and the class of kernel methods defined by the Neural Tangent Kernel. We shall proceed introducing regression over a Reproducing Kernel Hilbert Spaces (Section 2.3.1) and prove its equivalence with Gaussian Process Regression (Section 2.3.2). Then we will proceed by showing that both these approaches are equivalent to linear models in a function space defined implicitly by their characterizing kernel (Section 2.3.4). Then, we conclude by connecting kernel regression under gradient flow to the training dynamics of infinite DNNs under the NTK regime (Section 2.3.5).

2.3.1 Kernel methods and RKHS

The theory of Reproducing Kernel Hilbert Spaces (RKHS) [Aronszajn, 1950] provides a flexible and general framework to solve function estimation problems. Differently from parametric models such as DNNs, kernel methods find the optimal function through an infinite dimensional optimization problem whose searching space (a space of functions) is implicitly defined by the kernel which characterizes the RKHS [Scholkopf and Smola, 2001]. Typically these methods are known to posses favourable numerical properties and are some of the most utilized methods to solve ill-posedness of inverse problems [Tikhonov and Arsenin, 1977].

A typical regression problem with squared loss in a RKHS can be framed as follows: given a finite dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N \subset \mathbb{R}^d \times \mathbb{R}$ of samples generated from an unknown distribution, the optimal regression function $\hat{g}$ is estimated solving the following regularized infinite dimensional problem:

$$\hat{g} := \arg\min_{g \in \mathcal{H}} \sum_{i=1}^{N} (y_i - g(x_i))^2 + \lambda \|g\|_{\mathcal{H}}^2 \tag{2.16}$$

where $\mathcal{H}$ is the RKHS in which we look for the optimal regression function, $\|\cdot\|_{\mathcal{H}}$ is the associated RKHS norm and λ is the regularization parameter which regulates the relative weight of the fit (reconstruction loss) and penalty (RKHS norm) terms.

To highlight the connection with the Neural Tangent Kernel we point out that *any* RKHS is associated with a positive semi-definite kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, called reproducing kernel [Aronszajn, 1950]. This kernel completely characterizes the RKHS and therefore any function in $\mathcal{H}$ can be characterized through such k [Scholkopf and Smola, 2001, Aronszajn, 1950].

A remarkable result which is fundamental in practice is the so called *representer theorem* [Wahba, 1990, Scholkopf and Smola, 2001] which allows us to express the optimal solution of Eq. (2.16) as a linear combination of a finite number of basis functions (*kernel sections*). Interestingly the number of basis functions is given by the number of data N so that we can express $\hat{g}$ as:

$$\hat{g}(x) = \sum_{i=1}^{N} \hat{\alpha}_i k_{x_i}(x) = k(x, \mathcal{X})\hat{\alpha} \tag{2.17}$$

where α_i are the recombination coefficients and $k_{x_i}(x) : \mathbb{R}^d \to \mathbb{R}$ is the kernel section evaluated on the input location x_i: a function from the input space to the output one parametrized by the input location x_i. Most importantly any kernel section is simply

the kernel evaluated (on x_i) on one of its entries. To make the notation more compact we define $\hat{\alpha} \in \mathbb{R}^N$ as the concatenation of $\hat{\alpha}_i$ and we define $k(x, \mathcal{X}) \in \mathbb{R}^{1\times N}$ as the row vector defined as $(k(x, \mathcal{X}))_{1,i} := k(x, x_i)$ with $i \in [N]$.

The representer theorem allows us to express the infinite dimensional problem in Eq. (2.16) as a finite dimensional one (whose dimension is the number of available data N). The optimal solution is found by solving the following convex optimization problem:

$$\begin{aligned}\hat{\alpha} :&= \arg\min_{\alpha\in\mathbb{R}^N} \sum_{i=1}^{N} \left(y_i - \sum_{i=1}^{N} \hat{\alpha}_i k_{x_i}(x) \right)^2 + \lambda \sum_{i,j}^{N} \alpha_i \alpha_j \left\langle k_{x_i}(\cdot), k_{x_j}(\cdot) \right\rangle_{\mathcal{H}} \\ &= \arg\min_{\alpha\in\mathbb{R}^N} \|y - K(\mathcal{X}, \mathcal{X})\alpha\|_2^2 + \lambda \alpha^T K(\mathcal{X}, \mathcal{X})\alpha \end{aligned} \tag{2.18}$$

where $K(\mathcal{X}, \mathcal{X}) \in \mathbb{R}^{N\times N}$ is the kernel matrix evaluated on all pairs of data from the dataset $\mathcal{X}$. This is a quadratic convex optimization problem whose optimal solution can be computed in closed form as follows:

$$\hat{\alpha} = (K(\mathcal{X}, \mathcal{X}) + \lambda I_N)^{-1} y \tag{2.19}$$

which always exists since $K(\mathcal{X}, \mathcal{X}) + \lambda I_N$ is full rank (sum of a positive semi definite matrix and a positive definite).

Remark 2.7 (On the solution of a RKHS problem). Note Eq. (2.18) might be solved with any optimization algorithm, for example we can apply gradient descent (or gradient flow) on the loss function Eq. (2.16) so that we avoid computing the inverse. This perspective proves to be extremely valuable in practice since the cost in computing explicitly Eq. (2.19) scales as $O(N^3)$. It is easy to prove that gradient descent or gradient flow provide a solution which converges to Eq. (2.19) as the number of iterations goes to infinity (so long as the learning rate is chosen sufficiently small to avoid numerical instability).

Remark 2.8 (How to choose initialization?). When using gradient descent or gradient flow to solve Eq. (2.18) an initial condition for α is necessary. Typically this is initialized as the zero vector. This implies the initial residuals are obtained by comparing the target labels y with the null function (Eq. (2.17) with null recombination coefficients).

Finally, we can express the optimal function $\hat{g}$ on a generic input location (not

necessarily contained in the training dataset $\mathcal{X}$) as:

$$\hat{g}(x) = k(x, \mathcal{X})\hat{\alpha} = k(x, \mathcal{X})(K(\mathcal{X}, \mathcal{X}) + \lambda I_N)^{-1} y \tag{2.20}$$

2.3.2 RKHS and Gaussian Processes

The connection between RKHS and Gaussian Processes (GP) [Wahba, 1990, Rasmussen and Williams, 2006, Scholkopf and Smola, 2001] allows to think at a regularized function estimation in RKHS as an optimization problem over a space of functions in which a prior is considered. The optimal solution of Gaussian Process Regression (GPR) is defined as the optimal trade-off between performance (i.e. the likelihood) and model complexity (i.e. prior).

In this section we shall see that the MAP solution to a GP problem is the regression function obtained by solving Eq. (2.16) in a suitable RKHS [Wahba, 1990, Rasmussen and Williams, 2006]. One of the main advantages of GP w.r.t. RKHS is that it allows to consider both uncertainty of the optimal predictor and model selection (known as *kernel selection*).

The main idea is that a Gaussian Process can be used to describe a distribution over functions.

Definition 2.1 (Gaussian Process)**.** A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.

A Gaussian process $f(x)$ is completely specified by its mean function and covariance function. We call $m(x) : \mathbb{R}^d \to \mathbb{R}$ the GP mean function and $k_{\text{GP}}(x, x') : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ its covariance function and define them as:

$$\begin{aligned} m(x) :=& \mathbb{E}[f(x)] \\ k_{\text{GP}}(x) :=& \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] \end{aligned}$$

Note we assume the GP models a scalar function over the input domain, it is not difficult to extend GPs to multivariate functions of the input domain (e.g. by considering independent stacked GPs for each dimension). We will write a Gaussian Process as:

$$f(x) \sim \mathcal{GP}(m(x), k_{\text{GP}}(x, x'))$$

to simplify notation it is common to assume the mean function to be identically zero: this is by no means necessary so that the main results we shall show still hold when the mean function is not zero.

Typically we are not interested in drawing samples from the prior GP, we rather want to incorporate the knowledge that training data (observations of the GP) provide. The goal is to approximate $f(x_*) \in \mathbb{R}$ given observations of f on some input locations $\{f(x_i)\}_i$ with $i \in [N]$. We shall denote $f(\mathcal{X}) := (f(x_i))_i \in \mathbb{R}^N$ as the vector containing the stacked observations of the GP f. The joint distribution of $f(\mathcal{X})$ and $f(x_*)$, which is Gaussian by definition, is:

$$\begin{bmatrix} f(\mathcal{X}) \\ f(x_*) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0_N \\ 0 \end{bmatrix}, \begin{bmatrix} K_{\mathrm{GP}}(\mathcal{X},\mathcal{X}), K_{\mathrm{GP}}(\mathcal{X},x_*) \\ K_{\mathrm{GP}}(x_*,\mathcal{X}), k_{\mathrm{GP}}(x_*,x_*) \end{bmatrix} \right)$$

where $(K_{\mathrm{GP}}(\mathcal{X},\mathcal{X}))_{i,j} := k_{\mathrm{GP}}(x_i,x_j)$ is a $N \times N$ matrix and $(K_{\mathrm{GP}}(\mathcal{X},x_*))_i := k_{\mathrm{GP}}(x_i,x_*)$ is a N-dim column vector.

It is now trivial to predict $f(x_*)$ given data $f(\mathcal{X})$ by applying standard results of conditioned Gaussian random variables [Rasmussen and Williams, 2006, Friedman et al., 2001]:

$$f(x_*) \mid f(\mathcal{X}) \sim \mathcal{N}\Big(K_{\mathrm{GP}}(x_*,\mathcal{X}) K_{\mathrm{GP}}(\mathcal{X},\mathcal{X})^{-1} f(\mathcal{X}), \\ K_{\mathrm{GP}}(x_*,x_*) - K_{\mathrm{GP}}(x_*,\mathcal{X}) K_{\mathrm{GP}}(\mathcal{X},\mathcal{X})^{-1} K_{\mathrm{GP}}(\mathcal{X},x_*) \Big)$$

The closed form of the posterior distribution given training data $f(\mathcal{X})$ (which is again Gaussian) allows us to directly compute the posterior mean and covariance (which does not depend on the measured data $f(\mathcal{X})$) for each test input location. Note the MAP estimator of $f(x_*)$ is the posterior mean (since the posterior is Gaussian) and is given by $K_{\mathrm{GP}}(x_*,\mathcal{X}) K_{\mathrm{GP}}(\mathcal{X},\mathcal{X})^{-1} f(\mathcal{X})$. This expression is almost identical to Eq. (2.20) (identical if $\lambda = 0$). The full equivalence of the MAP predictor of GPR and the RKHS predictor $\hat{g}$ can be obtained by assuming the actual measurements of the unknown function $f(x_i)$ are corrupted by i.i.d. Gaussian noise $\epsilon_i \sim \mathcal{N}(0,\sigma^2)$: $y = f(\mathcal{X}) + e$, where $(e)_i = \epsilon_i$ with $i \in [N]$. This preserves Gaussianity of the joint distribution of y and $f(x_*)$. So that the MAP estimator can be written as:

$$f_{\mathrm{MAP}}(x_*) = \mathbb{E}[f(x_*) \mid y] = K_{\mathrm{GP}}(x_*,\mathcal{X})(K_{\mathrm{GP}}(\mathcal{X},\mathcal{X}) + \sigma^2 I_N)^{-1} y \tag{2.21}$$

Remark 2.9 (Linear predictor and kernel sections). Eq. (2.21) shows the MAP predictor is a linear combination of observations y. Moreover, dropping the dependency on the test datum x_* it is clear the optimal predictor can be represented as linear combination of kernel sections as already shown with the representer theorem for RKHS problems Eq. (2.17): $f_{\mathrm{MAP}}(\cdot) = \sum_{i=1}^{N} \alpha_i k_{\mathrm{GP}}(x_i,\cdot)$.

We now follow a different route to effectively show that computing the MAP predictor with GP is actually solving the same RKHS optimization problem in Eq. (2.16). As already observed, both y and f are jointly GPs (i.e. their evaluation on a finite number of points is a joint Gaussian distribution). Hence, by applying Bayes rule on a finite collection of data, it holds: $p(f \mid y) = \frac{p(y|f)p(f)}{p(y)}$, where $p(f \mid y)$ is the posterior density of the unknown function f evaluated on the training input locations, $p(y \mid f)$ is the *likelihood function*, $p(f)$ is the *prior* over the space of functions and $p(y)$ is the so-called *marginal likelihood*. The maximum a posteriori estimate of f on the finite sample $\mathcal{X}$ is obtained by maximizing $p(f \mid y)$. To ease computations it is standard practice to consider the log-posterior $\log p(f \mid y)$ which does not alter the optimal solution due to monotonicity of the logarithm. Therefore, the MAP estimator for GPs with Gaussian likelihood can be obtained as:

$$\begin{aligned} f_{\text{MAP}}(\mathcal{X}) &:= \arg\max_{f \in \mathbb{R}^N} \log p(y \mid f) + \log p(f) \\ &= \arg\min_{f \in \mathbb{R}^N} \|y - f(\mathcal{X})\|_2^2 + \sigma^2 f^T K_{\text{GP}}(\mathcal{X}, \mathcal{X})^{-1} f \end{aligned} \tag{2.22}$$

Which is a finite dimensional convex optimization problem whose optimal solution is $f_{\text{MAP}}(\mathcal{X}) = K_{\text{GP}}(\mathcal{X}, \mathcal{X})(K_{\text{GP}}(\mathcal{X}, \mathcal{X}) + \sigma^2 I_N)^{-1} y$. Note this is equivalent to Eq. (2.16) if considering both $\sigma^2 = \lambda$ and the GP covariance k_{GP} identical to the RKHS kernel k. Either σ or λ are not known a priori, and are typically estimated by means of Cross Validation or similar techniques for which the optimal trade-off between the fitting (likelihood) and the complexity (hyper-parameters prior) is optimized.

Once again, the optimal solution MAP solution is a linear combination of kernel sections (centered on the data $\{x_i\}$ with $i \in [N]$) weighted by some data dependent optimal parameters: $(K_{\text{GP}}(\mathcal{X}, \mathcal{X}) + \sigma^2 I_N)^{-1} y$.

2.3.3 Hyper-parameters optimization

Up to now we considered hyper-parameters as fixed values, though, in practice this is never the case and one needs to estimate them from data too. Changing hyper-parameters means changing kernel and therefore changing the RKHS over which the optimal estimator is built. In this section we shall consider the GP covariance function dependent on some hyper-parameters jointly denoted as κ and we shall describe some prototypical methods used to find optimal hyper-parameters in practice. As standard practice in Bayesian methods we consider κ as random vector whose prior distribution

is given by: $p(\kappa)$. The standard Bayesian predictor of f given the data y is given by:

$$p(f \mid y) = \int p(f \mid y, \kappa) p(\kappa \mid y) d\kappa \tag{2.23}$$

Note the posterior of the regression functions (w.r.t. the observed data y) is a weighted average of $p(f \mid y, \kappa)$ (the posterior of f w.r.t. data with fixed hyper-parameters).

We can further express $p(f \mid y, \kappa)$ by applying once again Bayes rule:

$$p(f \mid y, \kappa) = \frac{p(y \mid f, \kappa) p(f \mid \kappa)}{p(y \mid \kappa)} \tag{2.24}$$

where $p(y \mid f, \kappa)$ is the likelihood function and $p(f \mid \kappa)$ is the prior density once the kernel (covariance) hyper-parameters are fixed. The normalizing constant $p(y \mid \kappa)$ is known as marginal likelihood and does not depend on the unknown function (i.e. f is marginalized out).

In general solving Eq. (2.23) (*Full Bayes*) is intractable, since usually the posterior on the hyper-parameters does not allow closed form integrals. So that Monte-Carlo approximations are often required: $p(f \mid y) \approx \frac{1}{n_{\text{MC}}} \sum_{i=1}^{n_{\text{MC}}} p(f \mid y, \kappa_i) =: \hat{p}(f \mid y)$, where κ_i are fixed kernel-hyperparameters sampled according to the posterior $p(\kappa \mid y)$. Since in general it might not be trivial to sample from such distribution, Markov Chain Monte Carlo techniques might be necessary [Rasmussen and Williams, 2006]. If samples from the posterior $p(f \mid y)$ are required, they can be obtained as samples from its approximation $\hat{p}(f \mid y)$; on the other hand, whenever the MAP estimator is required it is sufficient to maximize $\hat{p}(f \mid y)$.

Empirical Bayes

Since approximating the posterior over the hyper-parameters might not be trivial, a possible solution is to approximate it with a delta distribution and then optimize the *marginal likelihood* to find the delta center (call it κ^*). Under this assumption Eq. (2.23) becomes: $p(f \mid y) = p(f \mid y, \kappa^*)$ so that the Full Bayesian posterior is simply the posterior obtained with a specific set of hyper-parameters.

Now the question is: how to choose the optimal delta center κ^*? And which optimality criterion should be used? We start with this observation: when a non-informative hyper-parameters prior is chosen (i.e. $p(\kappa) = \text{constant}$), it holds:

$$p(\kappa \mid y) = \frac{p(y \mid \kappa) p(\kappa)}{p(y)} \propto p(y \mid \kappa)$$

Hence, we can choose the best hyper-parameter κ^* to be the MAP estimate:

$$\kappa^* = \arg\max_{\kappa} p(y \mid \kappa)$$

This approximation is known as Type II maximum likelihood (ML-II) or evidence procedure [Rasmussen and Williams, 2006] and $p(y \mid \kappa)$ is also called Type II likelihood. Note that approximating the posterior $p(\kappa \mid y)$ with a single hyper-parameter κ^* might lead to overfitting, especially if many hyper-parameters are present [Rasmussen and Williams, 2006].

It is now worth to have a closer look at the marginal likelihood:

$$p(y \mid \kappa) = \int p(y \mid f, \kappa) p(f \mid \kappa) df \tag{2.25}$$

This equation is the main difference between Bayesian schemes of inference from other schemes based on optimization: this is due to the integration over the unknown f. In particular it is a property of the marginal likelihood that it automatically implements a trade-off between model fit (likelihood) and model complexity (prior) [Rasmussen and Williams, 2006].

Unfortunately, once again, integral in Eq. (2.25) is not always solvable in closed form solution, nonetheless it can be solved in closed form on some interesting scenarios, such as for Gaussian Processes:

$$\log p(y \mid \kappa) = -\frac{1}{2} y^T K_{\mathcal{X},\kappa}^{-1} y - \frac{1}{2} \log \det(K_{\mathcal{X},\kappa}) - \frac{n}{2} \log 2\pi \tag{2.26}$$

where $K_{\mathcal{X},\kappa} := K_{\mathrm{GP}\kappa}(\mathcal{X}, \mathcal{X}) + \sigma^2 I_N$ is the covariance function of the data (we are assuming σ is not part of the kernel hyper-parameters, nonetheless it is possible to consider it as an hyper-parameter too). The interpretation of the explicit form of the log marginal likelihood is readily available: $y^T K_{\mathcal{X},\kappa}^{-1} y$ is the data fit term, $\log \det(K_{\mathcal{X},\kappa})$ on the other hand is a complexity measure of the functions described by k_{GP}.

Remark 2.10 (Marginal likelihood for GPs is not convex). Note that the dependency on the hyper-parameters κ of the (log) marginal likelihood is non-convex in general (multiple non-connected stationary points might exist).

Remark 2.11 (GPs and RKHS main differences). Despite RKHS optimization can be connected to GP [Rasmussen and Williams, 2006] here are some of its remarkable shortcomings:

- RKHS does not characterize uncertainty in the predictions, nor does it handle

well multimodality in the posterior

- RKHS approximates the first level of Bayesian inference (i.e. function approximation $\hat{g}$) while it does not usually extend to the next level: hyper-parameters optimization. This is straightforward for GP (closed form solutions exist for Gaussian Likelihoods) and is usually solved by means of the marginal (log) likelihood. In particular marginal (log) likelihood is used a model selection criterion (e.g. it used to select the optimal parameters of the covariance function, we shall explore in more details kernel selection methods in Section 2.3.6).

2.3.4 Kernel methods as linear models

We now show how kernel methods can be interpreted as linear models in a (potentially infinite) feature space, these features are implicitly defined by the kernel choice k and the user does not need to explicitly work with them [Scholkopf and Smola, 2001] (in the Machine Learning literature this is known as *kernel trick*). In particular any solution to a RKHS optimization problem Eq. (2.16) does not explicitly compute any feature since its solution only requires kernel pairwise evaluation on data. The main idea is that any valid kernel function k can be written as $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\ell^2}$ where $\langle \cdot, \cdot \rangle_{\ell^2}$ is the standard ℓ^2 inner product for sequences, while the map $\phi : \mathbb{R}^d \to \mathbb{R}^p$ is called feature map and in general might not be finite dimensional (i.e. $p = \infty$). We shall denote with $\phi_i(x)$ the i-th component of the feature vector, which is a map $\mathbb{R}^d \to \mathbb{R}$. It can be shown that features induced by the kernel k are orthonormal functions w.r.t. the inner product defined in $\mathcal{H}$ (i.e. $\langle \phi_i(x), \phi_j(x) \rangle_{\mathcal{H}} = \delta_{i,j}$) and orthogonal w.r.t. the standard inner product in L^2 (i.e. $\langle \phi_i(x), \phi_j(x) \rangle_{L^2} = 0$ if $i \neq j$).

It is straightforward to substitute each kernel evaluation on the optimal representation given by the representer theorem Eq. (2.17):

$$\hat{g}(x) = \sum_{i=1}^{N} \hat{\alpha}_i \langle \phi(x_i), \phi(x) \rangle_{\ell^2} = \left\langle \sum_{i=1}^{N} \hat{\alpha}_i \phi(x_i), \phi(x) \right\rangle_{\ell^2} = \langle c, \phi(x) \rangle_{\ell^2}$$

which, for different input locations x, is a linear function with fixed feature recombination coefficients $c \in \mathbb{R}^p$. Hence it is possible to rewrite the RKHS norm of any function in the RKHS in terms of the feature recombination coefficients c as:

$$\|g\|_{\mathcal{H}}^2 = \left\langle \sum_{i=1}^{p} c_i \phi_i(x), \sum_{j=1}^{p} c_j \phi_j(x) \right\rangle_{\mathcal{H}} = \sum_{i,j=1}^{p} c_i c_j \langle \phi_i(x), \phi_j(x) \rangle_{\mathcal{H}} = \|c\|_{\ell^2}^2 \tag{2.27}$$

where we use the linearity of inner products and the orthonormality property of features

w.r.t. inner product in $\mathcal{H}$.

So that equation Eq. (2.18) can be written as:

$$\hat{c} = \arg\min_{c\in\mathbb{R}^p} \sum_{i=1}^{N} (y_i - \langle c, \phi(x_i)\rangle_{\ell^2})^2 + \lambda \|c\|_{\ell^2}^2 = \arg\min_{c\in\mathbb{R}^p} \|y - \Phi(\mathcal{X})c\|_2^2 + \lambda c^T c \tag{2.28}$$

where $\Phi(\mathcal{X}) := (\phi(x_1) \mid \phi(x_2) \mid \ldots \mid \phi(x_N))^T \in \mathbb{R}^{N\times p}$ is the matrix containing the feature vectors of x_i in each row and the inner product in ℓ^2 is represented as the inner product for finite sequences (i.e. $\langle \phi(x_i), c\rangle_{\ell^2} = \phi(x_i)^T c$ and $c^T c = \|c\|_{\ell^2}^2$), this notation allows to describe both finite and infinite dimensional feature space cases.

When the feature dimension is finite, the optimal set of coefficients is given by:

$$\hat{c} = (\Phi(\mathcal{X})^T \Phi(\mathcal{X}) + \lambda I_p)^{-1} \Phi(\mathcal{X})^T y \tag{2.29}$$

Remark 2.12 (On the solution of a RKHS problem in feature space). As mentioned in Remark 2.7, the optimal solution can be found without computing the matrix inverse $(p \times p)$ by exploiting gradient descent or gradient flow on the loss Eq. (2.28).

To conclude, the optimal prediciton function can be computed as:

$$\hat{g}(x) = \langle \phi(x), \hat{c}\rangle_{\ell^2} = \phi(x)^T (\Phi(\mathcal{X})^T \Phi(\mathcal{X}) + \lambda I_p)^{-1} \Phi(\mathcal{X})^T y \tag{2.30}$$

Remark 2.13 (Equivalent optimal solutions). Eq. (2.20) and Eq. (2.30) are indeed equivalent, to see this (when p is finite) it is sufficient to apply the matrix inversion lemma [Rasmussen and Williams, 2006].

Remark 2.14 (From kernel to features). In general there is no explicit formula to compute the feature vector given a kernel k. Nonetheless Mercer's theorem [Rasmussen and Williams, 2006] (Theorem 4.2), which holds when the kernel is continuous on a compact metric space (i.e. it is a bounded operator), guarantees the following *spectral decomposition*: $k(x, x') = \sum_{i=1}^{p} \lambda_i e_i(x) e_i(x')$ where $p = \infty$, $\lambda_i \in \mathbb{R}$ are the eigenvalues associated with the kernel k and $e_i(x)$ are orthonormal eigenfunctions w.r.t. the L^2 norm (these function are not orthonormal in the w.r.t. inner product induced by the kernel k (i.e. $\langle\cdot,\cdot\rangle_{\mathcal{H}}$). Note that Mercer's theorem gives us the following feature map for the kernel k: $\psi : \mathbb{R}^d \to \mathbb{R}^p$ where $\psi : x \mapsto (\sqrt{\lambda_i} e_i(x))_i$ with $i \in [p]$. With this definition it holds: $k(x, x') = \sum_{i=1}^{p} \lambda_i e_i(x) e_i(x') = \langle \psi(x), \psi(x')\rangle_{\ell^2}$ which guarantees ψ is a valid feature vector. Unfortunately this constructive argument does not allow, in general, to built feature explicitly since the functional form of $e_i(\cdot)$ as a function of the kernel k might not be available.

2.3.5 Kernel regression under gradient flow

We are now ready to prove the equivalence between a fully-trained sufficiently wide neural network and a kernel regression problem on a RKHS with kernel k_{NTK}.

To make the notation easier we shall denote with f_{NTK} the optimal function obtained by solving the kernel regression problem in Eq. (2.16) using kernel k_{NTK} and $\lambda = 0$, whose evaluation on any input location is given by:

$$f_{\text{NTK}}(x) = k_{NTK}(x, \mathcal{X})(H^*)^{-1}y \tag{2.31}$$

The inverse $(H^*)^{-1}$ is guaranteed to exist by assumption 2.2, see also Remark 2.4.

As in [Arora et al., 2019b] we define $f_{\text{NN}_{(t)}}(x) := \varepsilon f_{w(t)}(x)$ where $\varepsilon > 0$ is a small perturbation that is used to make the output of the DNN at initialization arbitrarily small, this is necessary to establish the equivalence between neural network and kernel regression (Remark 2.8 and Remark 2.28). Note however such a scale factor does not play any role on trained Network (since the scale of the output is optimized during training by changing w). We define the optimal infinitely wide DNN at the end of training as: $f^*_{\text{NN}}(x) := \lim_{t\to\infty} f_{\text{NN}_{(t)}}(x)$.

Theorem 2.4 (Equivalence between trained net and kernel regression [Arora et al., 2019b]). *Suppose* $\sigma(z) = \max(0, z)$, $1/\kappa = poly(1, /\epsilon, \log(n/\delta))$ *and* $d_i = n\ \forall i \in [L]$ *with* $n \geq poly(1/\kappa, L, 1/\lambda_{min}, N, \log(1/\delta))$. *Then for any* $x \in \mathbb{R}^d$ *with* $\|x\|_2 = 1$, *with probability at least* $1 - \delta_0$ *over the random initialization, we have:*

$$|f^*_{NN}(x) - f_{NTK}(x)| \leq \epsilon$$

Remark 2.15. One of the most important consequences of Theorem 2.4 is that the Neural Network predictor is essentially a kernel predictor. Therefore to study the properties (such as generalization) of over-parametrized DNNs it is sufficient to study the corresponding NTK.

Remark 2.16. As noted in [Arora et al., 2019b], while Theorem 2.4 provides guarantees only for a single point x, it is possible to extend its validity to exponentially many (but finite) number of points by simply considering an union bound.

2.3.6 Kernel Selection Criteria

For kernel based methods such as RKHS and GPs, kernel selection and optimization is often considered as a model selection problem. Commonly used measures for model

selection are: cross validation (CV) [Friedman et al., 2001], marginal likelihood [Rasmussen and Williams, 2006], kernel alignment [Cristianini et al., 2002, Cortes et al., 2012] and centered kernel alignment [Wang et al., 2012, Cortes et al., 2012]. Each one of these algorithms is characterized by its own strengths and limitations. For example cross validation is known to require high computational costs while being general enough to be applicable to a broad set of learning problems [Wahba, 1990, Friedman et al., 2001]. Marginal likelihood is an automatic algorithm to trade-off model complexity and fit but can be efficiently applied only in the Bayesian setting (Section 2.3.3). Kernel alignment is independent of the actual learning machine used but it is known to suffer from data imbalance. In the following we shall describe each kernel selection algorithm in detail.

Cross Validation

Cross validation is one of the simplest and most popular model (kernel) selection criteria for estimating the *generalization error* of a given predictor obtained according to the Empirical Risk Minimization paradigm. The main idea is to estimate out of sample prediction error by means of data that are not used during the ERM procedure. If the available data are plenty, then one can simply split them into two groups: training and validation. By training the model on the first set and then estimating the prediction error on the validation set, one is guaranteed to get an unbiased estimator of the generalization error of the trained model. In practice the number of available data is never enough to guarantee good statistical approximation on both datasets, so that K-fold cross-validation is usually employed to overcome such limitation [Friedman et al., 2001]. The main idea is to split the available data into K-folds. Then a model can be trained on all the folds but the i-th one. Then the i-th fold is used to estimate the generalization error of the predictor. Repeating the procedure for K times, so that each fold is used as validation dataset, an estimate of the generalization error of the predictor is obtained by averaging over these K estimates.

The choice of the optimal K is non trivial in general and trades-off computational complexity and accuracy of the generalization prediction. In particular, choosing $K = N$ the CV is approximately unbiased for the true expected error but the resulting variance might be high since the optimized models possess almost the same training data. On the other hand when K is small the datasets used to build each model are not much overlapped but the bias in the final estimate might be high (this depends on the sensitivity of the optimal predictor to the dataset size). The case $K = N$ is also known as leave-one-out cross-validation and for certain model classes (linear models under the

squared loss) the computational complexity can be highly decreased.

Marginal Likelihood

For an in-depth description of the Marginal Likelihood we refer to Section 2.3.3. Here we simply point out that the marginal likelihood criterion can be efficiently applied to problems in which the marginal likelihood Eq. (2.25) can be expressed in closed form. Assuming a Gaussian likelihood and a GP prior on the regression function it is possible to write the marginal likelihood explicitly Eq. (2.26). One of the most remarkable properties of this equation is the presence of a fitting term (the quadratic form) and of a complexity term (the log det). The first measures how well the variance of the data can be approximated by the kernel, while the second measures how complex the hypothesis space is (it is a measure of volume in the space of functions induced by a particular kernel choice). The marginal likelihood directly depends on the available data so that no kernel machine needs to be optimized on a given set of kernel parameters. Note however that both the inverse and the determinant of the $N \times N$ kernel matrix $K_{\mathcal{X},\kappa}$ are required, both of which have computational complexity scaling as $O(N^3)$ (which is a similar time complexity to the one required to solve the GP Eq. (2.21)) [Rasmussen and Williams, 2006].

Kernel Alignment

The third type of kernel selection algorithm we present is: Kernel Alignment (KA). It has been first introduced by [Cristianini et al., 2002] in the case of binary classification. However it can be extended to other learning cases [Wang et al., 2012, Cristianini et al., 2002], such that multi-class classification, unbalanced class classification and regression.

Definition 2.2 (Kernel Alignment)**.** Let k_1 and k_2 be two kernel functions defined over $\mathbb{R}^d \times \mathbb{R}^d$ and denote with $\mathcal{P}$ the data distribution. If k_1 and k_2 are such that $0 < \mathbb{E}_{x,x'}[k_i^2] < +\infty$ for $i = 1, 2$. Then, the alignment between k_1 and k_2 is defined by:

$$\rho(k_1, k_2) = \frac{\mathbb{E}_{x,x'}[k_1(x, x')k_2(x, x')]}{\sqrt{\mathbb{E}_{x,x'}[k_1^2(x, x')]\mathbb{E}_{x,x'}[k_2^2(x, x')]}} \tag{2.32}$$

where $\mathbb{E}_{x,x'}$ is the expected value with samples $x, x' \sim \mathcal{P} \times \mathcal{P}$.

We now introduce its natural empirical estimate.

Definition 2.3 (Empirical Kernel Alignment)**.** Let $K_1 \in \mathbb{R}^{N\times N}$ and $K_2 \in \mathbb{R}^{N\times N}$ be two kernel matrices such that $\|K_i\|_F \neq 0$ for $i = 1, 2$. Then, the alignment between K_1

and K_2 is defined by:

$$\hat{\rho}(K_1, K_2) = \frac{\langle K_1, K_2 \rangle_F}{\|K_1\|_F \|K_2\|_F} \tag{2.33}$$

where $K_i \in \mathbb{R}^{N \times N}$ is the kernel matrix evaluated on all the data $\{x_i\}$ with $i \in [N]$ and $\langle \cdot, \cdot \rangle_F$ is the Frobenius inner product of two matrices.

Remark 2.17 (Empirical KA as similarity score). If the kernel matrices K_1 and K_2 are considered as bidimensional vectors, the empirical kernel alignment can be seen as a similarity score based on the cosine of their angle. Therefore KA is bounded between -1 and 1. Moreover, since K_i are positive semi-definite Gram matrices, KA is lower-bounded by 0.

The natural way to apply this measure of alignment to a classification problem is by defining the following two kernels: the first one measures similarity between input locations $k_x(x, x')$ and the second measures similarity between target labels $k_y(y, y')$. For both kernels many definitions exist, for example when dealing with a classification problem one might consider $k_y(y, y') = 1$ if $y = y'$ and 0 otherwise. In this case it is possible to write the following expression for the empirical kernel: $K_y = YY^T$ where $Y \in \mathbb{R}^{N \times C}$ is a matrix whose rows are the target labels of dimension C for each datum. In this case it is possible to rewrite the Empirical KA in the following form:

$$\hat{\rho}(K_x, K_y) = \frac{\langle K_x, YY^T \rangle_F}{\|K_x\|_F \|K_y\|_F} = \frac{\operatorname{Tr} Y^T K_x Y}{\|K_x\|_F \|K_y\|_F} \tag{2.34}$$

It is well known that this definition of Kernel Alignment possesses favorable properties [Cristianini et al., 2002, Wang et al., 2012]. *Computational efficiency*, the computational cost to evaluate Eq. (2.34) scales as $O(N^2)$. *Concentration*, the probability of the empirical estimate $\hat{\rho}(K_1, K_2)$ Eq. (2.33) deviating from its mean $\rho(k_1, k_2)$ Eq. (2.32) can be bounded as an exponentially decaying function, so that the empirical estimator is stable w.r.t. different split of the data. *Generalization*, KA positively correlates with generalization, high alignment values imply there exists a separation of the data with low bound on the generalization error. It is therefore expected that maximizing KA on training set foster generalization performance.

We now focus on kernel selection for a family of kernel functions defined by some hyper-parameters κ, nonetheless the following approach does not require parametrized kernels and can be applied in general to compare kernels belonging to different families.

Let k be a kernel function parametrized by the hyper-parameters κ, $(K_\kappa)_{i,j} := (k_\kappa(x_i, x_j))$ for all $i, j \in [N]$ and $K_y = YY^T$. The following optimization can be

used for kernel selection on a given dataset:

$$\kappa^* = \arg\max_{\kappa} \hat{\rho}(K_\kappa, K_y) = \arg\max_{\kappa} \frac{\operatorname{Tr} Y^T K_\kappa, Y}{\|K_x\|_F} \tag{2.35}$$

Remark 2.18 (Connection with Marginal Likelihood). Eq. (2.35) is analogous to the Marginal Likelihood formula Eq. (2.26). In particular note Eq. (2.35) can be decoupled into two separate terms: a term depending on the target labels $\operatorname{Tr} Y^T K_\kappa Y$ and a complexity term $\|K_\kappa\|_F$. The first term measures the correlation between similarity in feature space (measured by K_κ) and target similarity (measured by K_y). So that it is large if features are good to separate input data (good fitting) and low if not (i.e. no correlation between features similarity and target labels means poorly designed features). Overall, the optimal kernel needs to face a trade-off between data fit and kernel complexity, so that the kernels with high correlation with target labels (low fitting loss) and small complexity are the optimal ones.

Centered Kernel Alignment

We not briefly discuss Centered Kernel Alignment (CKA) which has been proposed in [Cortes et al., 2012] as an improved version of Kernel Alignment, this new alignment measure is considered to better correlate with generalization and improves Kernel Alignment predictions on unbalanced classification tasks.

The main idea in [Cortes et al., 2012] is to compute the Kernel Alignment measure on a centered feature space.

Definition 2.4 (Centered Kernel Function [Cortes et al., 2012])**.** Let k be a kernel function defined over $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and call its induced feature map $\phi : \mathbb{R}^d \to \mathbb{R}^p$. The feature map ϕ is centered by subtracting from it its expectation $\mathbb{E}_x[\phi]$, where $\mathbb{E}_x$ denotes the expected value of ϕ when its input x is sampled according to the data distribution $\mathcal{P}$. The centered kernel k_c associated to the kernel k is defined as:

$$k_c(x, x') = (\phi(x) - \mathbb{E}_x[\phi])^T(\phi(x') - \mathbb{E}_x[\phi]) \tag{2.36}$$

Remark 2.19 (Connection with uncentered kernel). From Eq. (2.36) it is clear the following equivalent way of writing k_c:

$$k_c(x, x') = k(x, x') - \mathbb{E}_x[k(x, x')] - \mathbb{E}_{x'}[k(x, x')] + \mathbb{E}_{x,x'}[k(x, x')]$$

This shows the definition does not depend on the choice of the feature mapping asso-

ciated to k and that centering the feature map implies centering the kernel function $\mathbb{E}_{x,x'}[k_c(x,x')] = 0$. Moreover, since k_c is defined as an inner product, it is a positive semidefinite kernel.

As done for KA it is possible to define the Empirical Centered Kernel Alignment as:

Definition 2.5 (Centered Kernel Matrix [Cortes et al., 2012])**.** The kernel matrix $K_c \in \mathbb{R}^{N\times N}$ associated to the centered kernel k_c on a given set of samples $\{x_i\}$ with $i \in [N]$ is defined as the kernel matrix obtained using features centered with their empirical expectation: $\phi(x_i) - \bar{\phi}$ where $\bar{\phi} := \frac{1}{N}\sum_{i=1}^{N}\phi(x_i)$. Therefore the corresponding kernel matrix K_c is defined as:

$$(K_c)_{i,j} = K_{i,j} - \frac{1}{N}\sum_{i=1}^{N} K_{i,j} - \frac{1}{N}\sum_{j=1}^{N} K_{i,j} + \frac{1}{N^2}K_{i,j}$$

Remark 2.20. By letting $\Phi := (\phi(x_1), ..., \phi(x_N))^T$ and $\bar{\Phi} = (\bar{\phi}, ..., \bar{\phi})^T$ the following more compact form holds:

$$K_c = (\Phi - \bar{\Phi})(\Phi - \bar{\Phi})^T$$

This shows that the kernel matrix K_c is positive semi-definite and that the empirical kernel matrix is centered: $\frac{1}{N^2}\sum_{i,j=1}^{N}(K_c)_{i,j} = 0$.

Remark 2.21 (Relation with Uncentered kernel matrix). The following expression describes the relationship between the centered and uncentered kernel matrices:

$$K_c = \left[I_N - \frac{\mathbb{1}\mathbb{1}^T}{N}\right] K \left[I_N - \frac{\mathbb{1}\mathbb{1}^T}{N}\right]$$

where $\mathbb{1}$ is the vector of ones of the proper size.

It is now straightforward to extend the definitions used for Kernel Alignment to the Centered case.

Definition 2.6 (Centered Kernel Alignment)**.** Let k_1 and k_2 be two kernel functions defined over $\mathbb{R}^d \times \mathbb{R}^d$ such that $0 < \mathbb{E}_{x,x'}[k_i^2] < +\infty$ for $i = 1, 2$. Then, the *centered alignment* between k_1 and k_2 is defined by:

$$\rho_c(k_1, k_2) = \frac{\mathbb{E}_{x,x'}[k_{c1}(x,x')k_{c2}(x,x')]}{\sqrt{\mathbb{E}_{x,x'}[k_{c1}^2(x,x')]\mathbb{E}_{x,x'}[k_{c2}^2(x,x')]}} \tag{2.37}$$

Definition 2.7 (Centered Empirical Kernel Alignment)**.** Let $K_1 \in \mathbb{R}^{N\times N}$ and $K_2 \in \mathbb{R}^{N\times N}$ be two kernel matrices such that $\|K_i\|_F \neq 0$ for $i = 1, 2$. Then, the centered alignment between K_1 and K_2 is defined by:

$$\hat{\rho}_c(K_1, K_2) = \frac{\langle K_{c1}, K_{c2}\rangle_F}{\|K_{c1}\|_F \|K_{c2}\|_F} \tag{2.38}$$

where $K_{ci} \in \mathbb{R}^{N\times N}$ is the kernel matrix evaluated on all the data $\{x_i\}$ with $i \in [N]$ and $\langle\cdot,\cdot\rangle_F$ is the Frobenius inner product of two matrices.

Remark 2.22 (Properties of KA transfer to CKA). As for KA, also CKA is lower bounded by 0 and upper bounded by 1 and its empirical estimate concentrates around its expected value [Cortes et al., 2012]. Moreover we can connect CKA to the marginal likelihood for Gaussian Processes as done for KA in Remark 2.18.

As pointed out by [Cortes et al., 2012] the centered definition of Kernel Alignment does not seem to differ much from the definition of Kernel Alignment. Nonetheless the difference between the two is more than a technicality, without the centering it can be shown that the definition of alignment does not correlate well with generalization. To see this we refer to [Cortes et al., 2012] in which a toy example, which can be solved in closed form, is analyzed.

Remark 2.23 (Feature centering or kernel centering?). The centered notion of alignment introduced so far is defined by centering the features, a natural question is: can one center the kernels directly? Despite sounding natural, centering the kernel values, as opposed to centering feature values, is not directly relevant to linear prediction in feature space. As already observed, centering features values does imply centering kernel values while the vice-versa is not true.

Remark 2.24 (KA and CKA in practice). Both KA and CKA are independent of the actual learning machine used, moreover they only require information of the complete training data and can be computed efficiently. These properties make them especially useful in practice [Deshpande et al., 2021, Cristianini et al., 2002, Wang et al., 2012].

2.4 Deep Neural Networks as linear models

We are now ready to show a very important connection between Deep Neural Networks, their first order Taylor approximation and the Neural Tangent Kernel. To begin with, recall that as the width of vanilla fully connected and convolutional Neural Networks increases we have proved that the optimal DNN found following gradient flow dynamics

under the squared loss converges to the optimal solution of a kernel regression whose fixed kernel Theorem 2.3 is the NTK. Moreover, we proved that for any kernel regression problem the kernel function induces a set of features which might be in principle infinite and that the optimal kernel predictor is a linear model in this feature space. A natural question now is: can we describe an infinitely wide DNN using a linear model? If so, what are the features induced by the NTK?

2.4.1 NTK features

In general there is no closed form result to explicitly define the features induced by Θ^*, nonetheless we can find finite sized features which approximate the unknown ones. By definition we have that the finite width neural tangent kernel is defined as $\left\langle \frac{\partial f_w(x)}{\partial w}, \frac{\partial f_w(x')}{\partial w} \right\rangle_{\ell^2}$, we therefore know that $\frac{\partial f_w(x)}{\partial w}$ is a proper feature vector for the finite width kernel. We now observe that the finite width neural tangent kernel converges to the infinite width one Θ^* as the width increases Theorem 2.1, therefore the gradient vector at initialization is a good (asymptotic) approximation of the features induced by the infinite width NTK.

Remark 2.25 (Finite width approximation). In practice due to the finite size of the DNNs the convergence of $\Theta(0) \to \Theta^*$ and the stability of NTK (i.e. $\Theta(t) \to \Theta(0)$ as the width of the DNN increases $\forall t$) are not guaranteed and therefore the gradients at initialization are not guaranteed to be good enough to describe the learning trajectory of a general finite size DNN. The finite sample effects we just described have been one of the main reasons for the large research effort that machine learning researches have devoted to the study of the Neural Tangent Kernel [Jacot et al., 2018, Arora et al., 2019b, Arora et al., 2019a, Lee et al., 2017, Lee et al., 2019, Garriga-Alonso et al., 2018, Allen-Zhu et al., 2019b, Allen-Zhu et al., 2019a, Mu et al., 2020, Du et al., 2018a, Du et al., 2018b, Goldblum et al., 2019, Zancato et al., 2020, Deshpande et al., 2021].

2.4.2 First order Taylor expansion of DNNs

We now explicitly show how to use a first order Taylor expansion of a DNN to build a linear model whose learning trajectory approximates the infinite width (or the equivalent kernel regression) one. In this section we mainly show results from [Lee et al., 2019].

To improve readability we shall use the following notation $f_t := f_{w(t)}$. Consider a

first order Taylor expansion of the output of a Network f_t with input x:

$$f_t^{\text{lin}}(x) = f_0(x) + \left.\frac{\partial f_{w(t)}(x)}{\partial w}\right|_{w=w_{(0)}} \theta_t \tag{2.39}$$

where $\theta_t := w_t - w_0$ is the change in the parameters from their initial values. The linearized DNN is composed by two terms, the first one does not change during training while the second captures the output change w.r.t. the initial value during training.

Similarly to Proposition 2.1 we can express how the prediction outputs and parameters of the linearized model change during training with Gradient Flow (or Gradient Descent). In particular for Gradient Flow it holds:

$$\begin{aligned}\frac{d\theta_t}{dt} &= -\eta \nabla_\theta \mathcal{L}(\theta(t)) = -\eta \sum_{j=1}^{N} (f_t^{\text{lin}}(x_j) - y_j) \frac{\partial f_t^{\text{lin}}(x_j)}{\partial \theta} \\ &= -\eta \nabla_\theta f_t^{\text{lin}}(\mathcal{X})^T \nabla_{f_t^{\text{lin}}(\mathcal{X})} \mathcal{L}(\theta(t))\end{aligned} \tag{2.40}$$

where $\nabla_\theta f_t^{\text{lin}}(\mathcal{X}) := \left(\frac{\partial f_t^{\text{lin}}(x_1)}{\partial \theta} \middle| \frac{\partial f_t^{\text{lin}}(x_1)}{\partial \theta} \middle| \dots \middle| \frac{\partial f_t^{\text{lin}}(x_N)}{\partial \theta}\right)^T \in \mathbb{R}^{N \times D}$ is the matrix containing on each row the gradient of the network output on the i-th sample while $\nabla_{f_t^{\text{lin}}(\mathcal{X})} \mathcal{L}(\theta(t)) = \left(\frac{\partial \mathcal{L}(\theta(t))}{\partial f_t^{\text{lin}}(x_i)}\right)_i$ for $i \in [N]$.

Remark 2.26 (Extension to multi dimensional output). Previous formulation can be extended to Networks whose output is a vector (e.g. representing the number of classes C in a classification problem) by simply considering $\nabla_\theta f_t^{\text{lin}}(\mathcal{X}) \in \mathbb{R}^{CN \times D}$ (Section 2.6).

It is trivial to verify the following: $\nabla_\theta f_t^{\text{lin}}(\mathcal{X}) = \nabla_w f_0(\mathcal{X})$ where the last quantity is obtained from gradients of the non-linear model w.r.t. its parameters evaluated at initialization. Following the same derivation in Proposition 2.1 we can write how the output of the linear network evolves during the Gradient Flow dynamics:

$$\frac{df_t^{\text{lin}}(x)}{dt} = -\eta \Theta_t^{\text{lin}}(x, \mathcal{X}) \nabla_{f_t^{\text{lin}}(\mathcal{X})} \mathcal{L}(\theta(t)) = -\eta \Theta_0(x, \mathcal{X}) \nabla_{f_t^{\text{lin}}(\mathcal{X})} \mathcal{L}(\theta(t)) \tag{2.41}$$

Note $\Theta_t^{\text{lin}}(x, \mathcal{X}) := \nabla_\theta f_t^{\text{lin}}(x) \nabla_\theta f_t^{\text{lin}}(\mathcal{X})^T = \nabla_w f_0(x) \nabla_w f_0(\mathcal{X})^T = \Theta_0(x, \mathcal{X})$ which is the Neural Tangent Kernel computed from the gradients of the non-linear model at initialization.

Remark 2.27 (NTK parametrization). The definition of the NTK matrix as the inner product of gradients implicitly assumes the gradients are normalized by the number of parameters Eq. (2.5) so that as the width increases the NTK matrix remains bounded.

When computing the empirical NTK for architecture whose gradients are not implicitly normalized by the number of parameters it is fundamental to normalize the gradients or the empirical NTK by diving it with the number of parameters.

Since Θ_0 stays constant during training, the dynamics of gradient flow are often quite simple. In particular, given a quadratic loss $\mathcal{L}(\theta(t)) = \frac{1}{2}\sum_{i=1}^{N}(f_t^{\text{lin}}(x_i) - y_i)^2$ we have the following ODEs:

$$\frac{d\theta_t}{dt} = -\eta \nabla_w f_0(\mathcal{X})^T (f_t^{\text{lin}}(\mathcal{X}) - y) \tag{2.42}$$

$$\frac{df_t^{\text{lin}}(x)}{dt} = -\eta \Theta_0(x, \mathcal{X})(f_t^{\text{lin}}(\mathcal{X}) - y) \tag{2.43}$$

Proposition 2.4 (Explicit dynamics of linearization)**.** *The solutions of previous ODEs (Eq. (2.42) and Eq. (2.43)) can be written in closed form as:*

$$\theta_t = -\nabla_w f_0(\mathcal{X})^T \Theta_0^{-1}(I_N - e^{-\eta\Theta_0 t})(f_0(\mathcal{X}) - y) \tag{2.44}$$

$$f_t^{lin}(\mathcal{X}) = (I_N - e^{-\eta\Theta_0 t})y + e^{-\eta\Theta_0 t} f_0(\mathcal{X}) \tag{2.45}$$

Moreover the evolution of the network prediction on a test point x is given by:

$$f_t^{lin}(x) = f_0(x) - \Theta_0(x, \mathcal{X})\Theta_0^{-1}(I - e^{-\eta\Theta_0 t})(f_0(\mathcal{X}) - y) \tag{2.46}$$

Remark 2.28 (Non-null network initialization). Note Eq. (2.46) coincides with the NTK regression prediction in Eq. (2.31) when the output of the network at initialization is negligible.

2.4.3 Infinite width neural networks are linearized networks

As described in Remark 2.25 the dynamics of a finite width Neural Network is intractable in general, since the empirical NTK is not constant along the optimization path. However, for the mean squared loss, [Lee et al., 2019] proves that the gradient descent (and gradient flow) dynamics of the original neural network falls into its linearized dynamics regime as long as the learning rate is small enough.

Proposition 2.5 (Infinite width DNNs are linear models [Lee et al., 2019])**.** *Under assumptions 2.1 to 2.4 and gradient flow dynamics on the squared loss with learning rate η (i.e. $\frac{dw}{dt} = -\eta\nabla_w \mathcal{L}(w(t))$). For every $x \in \mathbb{R}^d$ with $\|x\|_2 \leq 1$, for $\delta_0 > 0$ arbitrarily small, there exist $r_0 > 0$ and $\bar{n} \in \mathbb{N}$ such that for every $n \geq \bar{n}$ where $n := \min\{d_1, ..., d_L\}$,*

with probability at least $(1-\delta_0)$ *over random initialization we have:*

$$\sup_t \left\| f_t^{lin}(\mathcal{X}) - f_{w(t)}(\mathcal{X}) \right\|_2 \lesssim \frac{r_0^2}{\sqrt{n}}, \qquad \sup_t \left\| f_t^{lin}(x) - f_{w(t)}(x) \right\|_2 \lesssim \frac{r_0^2}{\sqrt{n}} \tag{2.47}$$

where $\lesssim$ *is used to hide dependence on uninteresting constants.*

Remark 2.29 (Discrepancy bounds as a function of t). Previous bounds hold uniformly in t so that they are not applicable when finer bounds, as a function of t, are required. The following can be used to bound the discrepancy of the non-linear DNN and its linearization for any t. This has been derived in [Lee et al., 2019] (see Eq. S118) and is used to prove Proposition 2.5:

$$\left\| f_t^{\text{lin}}(\mathcal{X}) - f_{w(t)}(\mathcal{X}) \right\|_2 \leq (\eta \sigma_t t e^{\eta(-\lambda_{\min}+\sigma_t)t}) \left\| f_0^{\text{lin}}(\mathcal{X}) \right\|_2 \tag{2.48}$$

where $\sigma_t := \sup_{0\leq s\leq t} \|\Theta_s - \Theta_0\|_F \leq \frac{r_0}{\sqrt{n}}$.

Remark 2.30. Previous theorem holds beyond the NTK parametrization provided gradients are normalized by their number of parameters.

2.5 Trainability and generalization of infinitely wide Neural Networks

The main goal of this section is to study the convergence trajectory of infinitely wide Networks and their relation with the simple dynamics of their linearization around initialization.

The global convergence of the gradient descent for NTK has been recently demonstrated for over-parametrized DNNs by different authors [Du et al., 2019b, Allen-Zhu et al., 2019b, Du et al., 2018a, Allen-Zhu et al., 2018]. In these works the strictly positivity of the NTK matrix at initialization (which remains constant during training) plays a crucial role in providing convergence guarantees. In particular the positivity of the NTK matrix leads to a fast (exponential) decay of the training loss to zero which in turns, allows to localize the entire training dynamics around initialization. However, the assumption on the positivity of the NTK eigenvalues might not hold in practice [Su and Yang, 2019, Nitanda and Suzuki, 2020] and, as the number of examples increases the NTK spectrum descreases to zero. It is well known that the decay rate of integral operators (in this case the NTK) is directly connected with the complexity of the hypothesis space of functions induced by it: the faster the convergence of the eigenvalues to zero the lower the complexity of the induced features. This highlights the intuitive idea that larger model classes (characterized by a slow decay to zero of the eigenvalues

of the NTK matrix) are more likely to be faster learners. Unfortunately, similar claims might not be straightforward for generalization [Hardt et al., 2015] since a larger search space (faster learners) might be associated to over-fitting.

2.5.1 Training trajectories of over-parametrized DNNs

To begin with, we write the evolution of the squared loss of the linearized model:

$$\mathcal{L}(\theta(t)) = \left\| y - f_t^{\text{lin}}(\mathcal{X}) \right\|_2^2 = (y - f_0(\mathcal{X}))^T e^{-2\eta\Theta_0 t}(y - f_0(\mathcal{X})) = \|\delta y_0\|^2_{e^{-2\eta\Theta_0 t}} \tag{2.49}$$

where we simply used the closed form solution for the outputs of the linear model in Eq. (2.45) and defined $\delta y_0 := y - f_0(\mathcal{X})$. Note that a randomly initialized Network converges to a zero mean Gaussian Process with finite variance so that in the infinite width limit δy_0 is likely to be "close" to y (more precisely we can only claim that with high probability over the random initialization δy_0 is bounded Proposition 2.3). What if the Network is pre-trained? In this case no convergence theorems exists, nonetheless linearization around initialization can still be applied to study the dynamics of pre-trained Neural Networks. In particular, the better the pre-training the closer δy_0 to 0. No theoretical result exists to prove the empirical tangent kernel of a *pre-trained model* converges to the Neural Tangent Kernel Θ^* as its width increases. Nonetheless, under the assumption the pre-training dataset is close in some sense to the target $\mathcal{X}$, we argue pre-trained models are more likely not to change much during optimization so that a first order Taylor expansion at initialization is likely to well approximate their non-linear dynamics even for finite width DNNs [Deshpande et al., 2021, Zancato et al., 2020, Achille et al., 2021].

With Eq. (2.49) it is rather easy to study the evolution of the squared loss along the gradient flow for the linearized model, but how close is the loss of the linearized model to that of the non-linear one? We provide an answer to this question with the following proposition (which is a direct consequence of Proposition 2.5).

Proposition 2.6. *Under the same assumptions of Proposition 2.5. The following holds with probability at least $(1 - \delta_0)$ over random initialization:*

$$\left| \left\| y - f_{w(t)}(\mathcal{X}) \right\|_2 - \left\| y - f_t^{lin}(\mathcal{X}) \right\|_2 \right| \lesssim \frac{r_0^2}{\sqrt{n}} \tag{2.50}$$

Proof. By the triangle's inequality we have:

$$\left\| y - f_{w(t)}(\mathcal{X}) \right\|_2 \leq \left\| y - f_t^{\text{lin}}(\mathcal{X}) \right\|_2 + \left\| f_t^{\text{lin}}(\mathcal{X}) - f_{w(t)}(\mathcal{X}) \right\|_2$$

and therefore using the result in Proposition 2.5 which holds for any time t we conclude. ■

The assumption on gradient flow can be relaxed and a similar result can be shown for gradient descent dynamics under suitable small learning rate. The gradient descent dynamics has been analyzed in [Arora et al., 2019a] in the case of a randomly initialized two layer Neural Network. So that the extension of Proposition 2.6 to gradient descent is a generalization of the results in [Arora et al., 2019a]. In particular, the main insights on the convergence rates provided in [Arora et al., 2019a] are still valid on a deeper architecture for which it holds Eq. (2.50).

As a straightforward consequence of Eq. (2.50) it is possible to prove Proposition 3.2 [Zancato et al., 2020] which provides us a simple tool to conduct a fine-grained analysis on the training dynamics: it distinguishes between different types of prediction errors at initialization and training labels. Note that this result applies both to randomly initialized models and pre-trained ones provided linearized approximation is good enough. Such a decomposition allows us to answer some very important questions:

- Why do true labels give faster convergence rate than random labels for gradient descent? [Arora et al., 2019a, Zhang et al., 2017]
- How to choose the best pre-trained model for a given task? [Deshpande et al., 2021]

2.5.2 Generalization properties of over-parametrized DNNs

In this section we study the generalization ability of infinite width Neural Networks trained by gradient descent. Most of the material here is taken from [Arora et al., 2019a], in which the generalization ability of a two layer ReLU activated Neural Network is analyzed by means of Rademacher complexity bounds. Their main result Theorem 2.5 is based on a fine-grained analysis of the mobility of Neural Network's parameters during optimization. The generalization bound proposed in [Arora et al., 2019a] depends on a *data-dependent complexity measure* which, notably, is independent of the number of hidden units in the network. Interestingly, such a bound is not vacuous for real world datasets such as MNIST and CIFAR [Arora et al., 2019a].

The main requirement for Theorem 2.5 is a *non-degeneracy* assumption on the data distribution, which is equivalent to a positive definitness constraint on the spectrum of the Empirical NTK matrix. As discussed in Section 2.2.3 and Section 2.5.1 this assumption guarantees convergence to zero training loss [Du et al., 2018a, Du et al., 2019b, Allen-Zhu et al., 2019a, Allen-Zhu et al., 2018].

Definition 2.8. A distribution $\mathcal{P}$ over $\mathbb{R}^d \times \mathbb{R}$ is $(\lambda_{\min}, \delta, N)$-*non-degenerate*, if for N i.i.d. samples $\{x_i, y_i\}_i$ with $i \in [N]$ from $\mathcal{P}$, with probability at least $1-\delta$ we have $\min \Lambda(H^*) \geq \lambda_{\min} > 0$.

Theorem 2.5 (Generalization bound for two layer NN [Arora et al., 2019a])**.** *Fix a failure probability $\delta \in (0,1)$. Suppose the dataset $\mathcal{D}$ is composed of N i.i.d. samples from a $(\lambda_{min}, \delta/3, N)$-non-degenerate distribution $\mathcal{P}$. Moreover assume $\kappa = O\left(\frac{\lambda_{min}\delta}{N}\right)$ and $m \geq \kappa^{-2} poly(N, \lambda_{min}^{-1}, \delta^{-1})$. Consider any loss function $l : \mathbb{R} \times \mathbb{R} \to [0,1]$ that is 1-Lipschitz in the first argument such that $l(y,y) = 0$. Then with probability at least $1-\delta$ over the random initialization and the training samples, a two-layer Neural Network $f_{a,w_{(k)}}$, with ReLU activations, trained by gradient descent for $k \geq \Omega\left(\frac{1}{\eta\lambda_{min}} \log \frac{N}{\delta}\right)$ iterations has population loss $\mathcal{L}_{\mathcal{P}}(f_{a,w_{(k)}}) = \mathbb{E}_{(x,y)\sim\mathcal{P}}[l(f_{a,w_{(k)}}(x), y)]$ bounded as:*

$$\mathcal{L}_{\mathcal{P}}(f_{a,w_{(k)}}) \leq \sqrt{\frac{2y^T(H^*)^{-1}y}{n}} + O\left(\sqrt{\frac{\log\left(\frac{N}{\lambda_{min}\delta}\right)}{n}}\right)$$

The main idea of the proof is to bound the distance of the trained Neural Network's parameters w.r.t. the ones at initialization. This hinges on the proximity of the non-linear dynamics of the Neural Network to the linear dynamics of its infinite width limit (which is characterized by H^*). This observation implies the class of two-layer Neural Networks implemented are those whose weights are close to initialization. So that, in this special case, it is possible to exploit a bound on the Rademacher complexity which characterizes generalization.

In Theorem 2.5 there are three sources of possible failures: failure of satisfying $\min \Lambda(H^*) \geq \lambda_0$, failure of random initialization and failure in the data sampling. Most notably, the dominating term which bounds generalization can be interpreted as a complexity measure of data (w.r.t. to the architecture). In particular, we can estimate an upper bound on generalization before even training the network by only looking at the sampled training data and the Empirical NTK.

Remark 2.31 (Connection with Model Selection). Theorem 2.5 has important consequences even if its assumptions are rather limiting in practice: being able to estimate an index of generalization even before training allows one to perform model selection without incurring in any training cost. Moreover, the subtle dependency architecture-dataset is taken into account since H^* contains both information about architecture and input locations while y directly models the target labels. Different questions such as how to perturb labels (without changing input data) to ruin or improve generaliza-

tion can be analyzed by looking at $\sqrt{\frac{2y^T(H^*)^{-1}y}{n}}$. In [Deshpande et al., 2021] a model selection criterion which employs a similar complexity criterion has been proposed to solve the hard task of choosing the best Neural Network from a plethora of models to solve a given dataset Chapter 4.

The last important question in this section is: how general is the previous result? Does it hold for a broader class of neural networks (larger than the two layer ReLU activated Neural Networks studied in [Arora et al., 2019a])? From a theoretical standpoint no such result can be found in literature, we believe this is due to the specific technique used in Theorem 2.5 to prove the Empirical Rademacher complexity bound which is not easily generalizable to more complex networks. Nonetheless, some empirical results show complex over-parametrized non-linear convolutional Neural Networks can be analyzed with an approximation that provides good empirical results on real-world datasets [Deshpande et al., 2021, Zancato et al., 2020].

2.6 Training dynamics under the NTK approximation on other losses

Up to now we described the training dynamics of DNNs under the squared loss since most of the involved computations can be carried out explicitly. We now show how to extend the NTK to the Cross-Entropy loss, which is typically used to solve multi-class classification tasks. To do so we shall follow [Lee et al., 2019, Zancato et al., 2020, Deshpande et al., 2021] and exploit the gradient descent dynamics of the DNN linearization described in Section 2.4.2.

In a general C-classes classification problem the output of a model is assumed to be of dimension C, so that in this section we shall consider $f_{w(t)}(x) \in \mathbb{R}^C$. The per-sample expression of the Cross-Entropy loss is given by:

$$l(f(x_i), y_i) = -\sum_{j=1}^{C} y_{i,j} \log \text{softmax}(f(x_i))_j$$

where $y_i \in \mathbb{R}^C$ is the label vector associated to the i-th datum and the softmax is defined as:

$$\text{softmax}(f)_j := \frac{\exp f_j}{\sum_{l=1}^{C} \exp f_l}$$

The evolution of a general non-linear model under gradient flow [Lee et al., 2019]

both on the training dataset and on a general input location x is given by:

$$\frac{df_{w(t)}(\mathcal{X})}{dt} = -\eta\Theta_t(\mathcal{X},\mathcal{X})\nabla_{f_{w(t)}(\mathcal{X})}\mathcal{L}(w(t)) = -\eta\Theta_t(\mathcal{X},\mathcal{X})(\text{softmax}(f_{w(t)}(\mathcal{X})) - y)$$
$$\frac{df_{w(t)}(x)}{dt} = -\eta\Theta_t(x,\mathcal{X})\nabla_{f_{w(t)}(\mathcal{X})}\mathcal{L}(w(t)) = -\eta\Theta_t(x,\mathcal{X})(\text{softmax}(f_{w(t)}(\mathcal{X})) - y)$$

Where the NTK matrix is now a $\mathbb{R}^{CN\times CN}$ matrix since for each datum the model outputs C scalars, $\nabla_{f_{w(t)}(\mathcal{X})}\mathcal{L}(w(t)) := \left(\frac{\partial\mathcal{L}(w(t))}{\partial f_{w(t)}(x_1)}^T \middle| \ldots \middle| \frac{\partial\mathcal{L}(w(t))}{\partial f_{w(t)}(x_N)}^T\right)^T \in \mathbb{R}^{CN}$ is obtained by stacking: $\left(\frac{\partial\mathcal{L}(w(t))}{\partial f_{w(t)}(x_i)}\right)_j := \frac{\partial\mathcal{L}(w(t))}{\partial(f_{w(t)}(x_i))_j} = \text{softmax}(f(x_i))_j - y_{i,j}$. This notation assumes $y \in \mathbb{R}^{CN}$ is obtained by stacking target labels for each datum and similarly for $\text{softmax}(f_{w(t)}(\mathcal{X})) \in \mathbb{R}^{CN}$ where the softmax is applied for each datum separately.

Remark 2.32 (The dynamics with CE loss has not closed form solutions). Previous equations can be applied to a general non-linear network $f_{w(t)}$, the main difference with the equations derived for the squared loss is that no closed form solution exists. Moreover, substituting in previous equations a general non-linear network $f_{w(t)}$ with its linearized version f_t^{lin} only changes the NTK matrix Θ_t so that it becomes constant Θ_0. Nonetheless, closed form solution is possible even for the linearized model, hence numerical integration methods are required [Lee et al., 2019, Deshpande et al., 2021].

2.7 Stochastic Gradient Descent and NTK regime

Up to now we studied the training dynamics of gradient descent or gradient flow for training DNNs and showed that for heavily over-parametrized models the non-linear dynamics can be closely approximated by the one of a linear model (or kernel method). From the linearization around initialization it is possible to gather insights on the most important factors influencing the learning dynamics (Section 2.5.1) and generalization (Section 2.5.2). Nonetheless, in practice DNNs models are trained using a Stochastic Gradient Descent so that a theory able to couple both over-parametrization and SGD is required. As described in Section 1.4, SGD has been primarily introduced to overcome practical limitations (e.g. memory requirements due to large scale models and datasets) but it gathered popularity thanks to its high performance measured on many different applications and domains. Currently no conclusive and exhaustive theory of SGD for over-parametrized DNNs exists, nonetheless some remarkable results have been recently obtained: [Chen et al., 2020, Nitanda and Suzuki, 2020]. Both works analyze the convergence of some variants of SGD for over-parametrized two-layer Neural Networks for regression problems. The first [Chen et al., 2020] provides a generalized Neural

Tangent Kernel analysis and shows that noisy gradients with weight decay still exhibit a "*kernel-like*" behaviour. [Nitanda and Suzuki, 2020] instead, provides some guarantees on the minimax optimal convergence rate of averaged SGD under the NTK regime.

II

Finite DNNs and Neural Tangent Kernel

3

Training Time Prediction

Say you are a researcher with many more ideas than available time and compute resources to test them. You are pondering to launch thousands of experiments but, as the deadline approaches, you wonder whether they will finish in time, and before your computational budget is exhausted. Could you predict the time it takes for a network to converge, before even starting to train it?

In [Zancato et al., 2020] we look to efficiently estimate the number of training steps a Deep Neural Network needs to converge to a given value of the loss function, without actually having to train the network. This problem has received little attention thus far, possibly due to the fact that the initial training dynamics of a randomly initialized DNN are highly non-trivial to characterize and analyze. However, in most practical applications, it is common to *not* start from scratch, but from a pre-trained model (*fine-tuning*). This may simplify the analysis, since the final solution obtained by fine-tuning is typically not too far from the initial solution obtained after pre-training. In fact, it is known that the dynamics of overparametrized DNNs [Du et al., 2019a, Zou et al., 2018, Allen-Zhu et al., 2019b] during fine-tuning tend to be more predictable and close to convex [Mu et al., 2020]. Our main contribution in [Zancato et al., 2020] is to introduce the problem of predicting training time in realistic use cases, in particular how training time depends on the hyper-parameters and, most importantly, on the interaction between target task and pre-training (which, to the best of our knowledge, is new).

We therefore characterize the training dynamics of a pre-trained network and provide a computationally efficient procedure to estimate the expected profile of the loss curve over time.

We use a linearized version of the DNN model around pre-trained weights to study

its actual dynamics. In [Lee et al., 2019] a similar technique is used to describe the learning trajectories of *randomly initialized* wide neural networks. Such an approach is inspired by the Neural Tangent Kernel for infinitely wide networks [Jacot et al., 2018] (Chapter 2). While we note that NTK theory may not correctly predict the dynamics of real (finite size) randomly initialized networks [Goldblum et al., 2019], we show that our linearized approach can be extended to fine-tuning of real networks in a similar vein to [Mu et al., 2020]. In order to predict fine-tuning Training Time without training we introduce a Stochastic Differential Equation (similar to [Hayou et al., 2019]) to approximate the behavior of SGD: we do so for a linearized DNN and in function space rather than in weight space. That is, rather than trying to predict the evolution of the weights of the network (a D-dimensional vector), we aim to predict the evolution of the outputs of the network on the training set (a $N \times C$-dimensional vector, where N is the size of the dataset and C the number of network's outputs). This drastically reduces the dimensionality of the problem for over-parametrized networks (that is, when $NC \ll D$).

A possible limiting factor of our approach is that the memory requirement to predict the dynamics scales as $O(DC^2N^2)$. This would rapidly become infeasible for datasets of moderate size and for real architectures (D is in the order of millions). To mitigate this, we show that we can use random projections to restrict to a much smaller D_0-dimensional subspace with only minimal loss in prediction accuracy. We also show how to estimate Training Time using a small subset of N_0 samples, which reduces the total complexity to $O(D_0\, C^2 N_0^2)$. We do this by exploiting the spectral properties of the Gram matrix of the gradients. Under mild assumptions the same tools can be used to estimate Training Time on a larger dataset without actually seeing the data.

The method we propose in this chapter does not depend on a particular application domain (say image processing or time series analysis) or loss function (e.g. squared loss or cross-entropy loss) and can be applied so long as feedforward DNNs are used. Nonetheless, the driving force of the material developed in this chapter started to unblock the adoption of real-world Computer Vision AutoML systems for classification, therefore our empirical validation has only been conducted on this application domain.

To summarize, the main contributions of this chapter are:

1. We present both a qualitative and quantitative analysis of the fine-tuning Training Time as a function of the Gram-Matrix Θ of the gradients at initialization (empirical NTK matrix).
2. We show how to reduce the cost of estimating the matrix Θ using random projections of the gradients, which makes the method efficient for common architectures

and datasets.

3. We introduce a method to estimate how much longer a network will need to train if we increase the size of the dataset without actually having to see the data (under the hypothesis that new data is sampled from the same distribution).
4. We test the accuracy of our predictions on off-the-shelf state-of-the-art models trained on real datasets. We are able to predict the correct training time within a 20% error with 95% confidence over several different datasets and hyperparameters at only a small fraction of the time it would require to actually run the training (30-45x faster in our experiments).

3.1 Bibliographical Notes

Predicting the training time of a state-of-the-art deep architecture on large scale datasets is a relatively understudied topic. In this direction, Justus et al. [Justus et al., 2018] try to estimate the wall-clock time required for a forward and backward pass on given hardware. We focus instead on a complementary aspect: estimating the number of fine-tuning steps necessary for the loss to converge below a given threshold. Once this has been estimated we can combine it with the average time for the forward and backward pass to get a final estimate of the wall clock time to fine-tune a DNN model without training it.

Hence, we are interested in predicting the learning dynamics of a pre-trained DNN trained with either Gradient Descent or Stochastic Gradient Descent.

In order to predict the learning dynamics many works are based on learning curve prediction (see [Klein et al., 2017] and references therein). These methods mainly focus on predicting the effect of different hyper-parameters for fixed task and architectures. Differently, we provide qualitative interpretation and quantitative prediction of the convergence speed of a DNN as a function of optimization hyper-parameters, network pre-training and target task.

Other results are known to describe training dynamics under a variety of assumptions (e.g. [Keskar et al., 2016b, Smith and Le, 2018, Saxe et al., 2013, Brutzkus et al., 2017]), we are mainly interested on recent developments which describe the optimization dynamics of a DNN using a linearization approach. Several works [Jacot et al., 2018, Lee et al., 2017, Du et al., 2018a] suggest that in the over-parametrized regime wide DNNs behave similar to linear models, and in particular they are fully characterized by the Gram-Matrix of the gradients, also known as empirical Neural Tangent Kernel (Section 2.4).

Under these assumptions, [Jacot et al., 2018, Arora et al., 2019a] derive a simple connection between training time and spectral decomposition of the NTK matrix. However, their results are limited to Gradient Descend dynamics and to simple architectures which are not directly applicable to real scenarios. In particular, their arguments hinge on the assumption of using a randomly initialized very wide two-layer or infinitely wide neural network [Arora et al., 2019a, Du et al., 2018b, Li and Yuan, 2017]. We take this direction a step further, providing a unified framework which allows us to describe training time for both SGD and GD on common architectures.

Again, we rely on a linear approximation of the model, but while the practical validity of such linear approximation for randomly initialized state-of-the-art architectures (such as ResNets) is still discussed [Goldblum et al., 2019], we follow Mu et al. [Mu et al., 2020] and argue that the fine-tuning dynamics of over-parametrized DNNs can be closely described by a linearization. We expect such an approximation to hold true since the network does not move much in parameters space during fine-tuning and *over-parametrization* leads to smooth and regular loss function around the pre-trained weights [Du et al., 2019a, Zou et al., 2018, Allen-Zhu et al., 2019b, Li et al., 2020]. Under this premise, to tackle both GD and SGD in an unified framework we build on [Hayou et al., 2019] modelling linearized training using a Stochastic Differential Equation in function space. We show we can use linearization to study the fine-tuning dynamics as suggested by [Mu et al., 2020] and provide accurate estimates on Training Time.

3.2 Training Time Definitions

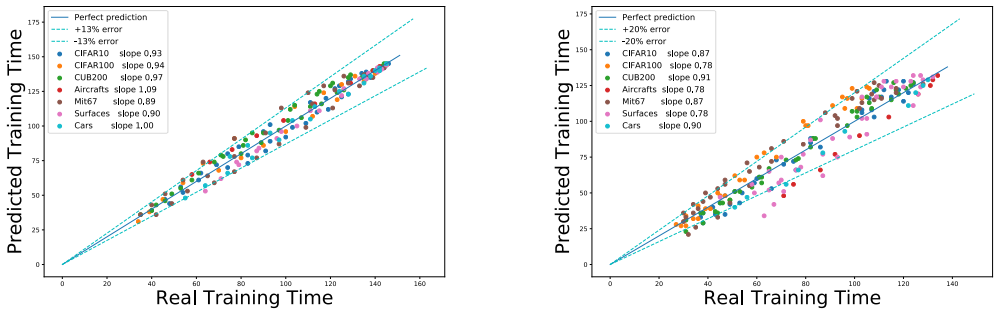
We start this chapter by proposing three possible definitions of *Training Time* (TT), we shall mainly be concerned on TT for DNNs nonetheless all definitions hold for any model optimized by iterative optimization schemes. We shall assume the DNNs are trained up to convergence and denote the training loss at each time instant t as: $\mathcal{L}(t)$.

Definition 3.1 (Absolute Training Time)**.** Given the training loss of a DNN model $\mathcal{L}(t)$ and a threshold value ϵ we define the *absolute training time* of the DNN model as the smallest time index t^* for which the training loss is consistently lower than the threshold:

$$t^* := \inf\{t : \mathcal{L}(s) < \epsilon, \forall s \geq t\}$$

If the set $\{t : \mathcal{L}(s) < \epsilon, \forall s \geq t\}$ is empty we define $t^* := \infty$.

Previous definition of Training Time is based on the absolute value of the training



(a) Training with Gradient Descent.

(b) Training with SGD.

Figure 3.1: **Training time prediction (# iterations) for several fine-tuning tasks.** Scatter plots of the predicted time vs the actual training time when fine-tuning a ResNet-18 pre-trained on ImageNet on several tasks. Each task is obtained by randomly sampling a subset of five classes with 150 images (when possible) each from one popular dataset with different hyperparameters (batch size, learning rate). The closer the scatter plots to the bisector the better the TT estimate. Our prediction is **(a)** within 13% of the real training time 95% of the times when using GD and **(b)** within 20% of the real training time when using SGD.

loss, so that if the threshold is too small the Training Time might be infinite. In practice, it is not possible to know a priori which value of ϵ leads to infinite TT values. To solve this limitation we propose the following definition, which is always well defined for typical learning trajectories (for which the loss converges to a single value or a bounded stationary distribution).

Definition 3.2 (Relative Training Time)**.** Given the training loss of a DNN model $\mathcal{L}(t)$ and a threshold value ϵ we define the *relative training time* of the DNN model as the smallest time index t^* for which the training loss is consistently within an ϵ-ball around its asymptotic value:

$$t^* := \inf\{t : |\mathcal{L}(s) - \mathcal{L}(\infty)| < \epsilon, \forall s \geq t\}$$

If the set $\{t : |\mathcal{L}(s) - \mathcal{L}(\infty)| < \epsilon, \forall s \geq t\}$ is empty we define $t^* := \infty$.

Remark 3.1. If stochastic optimization schemes are used, the limit loss might not be well defined, nonetheless it is customary to consider the asymptotic loss as a bounded stationary distribution (within a set $\mathcal{B}$) [Mandt et al., 2017]. Under such circumstances we use the following hitting time-like definition $t^* := \inf\{t : \mathcal{L}(t) \in \mathcal{B}\}$. This definition is harder to be used in practice since it requires knowledge of $\mathcal{B}$ that cannot be easily characterized. A straightforward relaxation is obtained by replacing $\mathcal{B}$ with

$\hat{\mathcal{B}} := \{|\mathcal{L}(t) - \mathbb{E}[\mathcal{L}(\infty)]| < \epsilon\}$. Where $\mathbb{E}[\mathcal{L}(\infty)]$ can be approximated by a running average of training loss values.

Remark 3.2 (Equivalence of the definitions of TT). If $\lim_{t\to\infty} \mathcal{L}(t) = 0$ then Definition 3.1 and Definition 3.2 are equivalent.

Definition 3.3. Given the training loss of a DNN model $\mathcal{L}(t)$, a threshold value ϵ and a time length T, we define the *early stopping training time* of a DNN model as the smallest time index t^* for which the training loss does not decreases by a factor ϵ over the last T iterations:

$$t^* := \inf\{t : \mathcal{L}(t) - \epsilon \leq \mathcal{L}(s), \forall s \in [t, t+T]\}$$

In the following we shall only consider Definition 3.2 as the definition of Training Time, so that by Training Time we mean the number of optimization steps – of either Gradient Descent or Stochastic Gradient Descent – needed to bring the training loss within an ϵ ball around an asymptotic value.

Remark 3.3 (Applicability of our TT estimator). Our Training Time estimator can be used on any of the above definitions of Training Time.

3.3 Predicting Training Time

In this section we look at how to efficiently approximate the training time of a DNN without actual training, to do so we use a linearized approximation of the DNN. Such an approximation can be computed without any fine-tuning and mimics the actual learning dynamics of the non-linear DNN. Therefore from the loss trajectory of the linearized model we can get an estimate of the actual Training Time of the non-linear model (see Section 3.5 for the complete algorithm).

We start by introducing our main tool. Let $f_w(x)$ denote the output of the network, where w denotes the weights of the network and $x \in \mathbb{R}^d$ denotes its input (e.g. an image). Let w_0 be the weight configuration after pre-training. We assume that when fine-tuning a pre-trained network the solution remains close to pre-trained weights w_0 [Mu et al., 2020, Du et al., 2019a, Zou et al., 2018, Allen-Zhu et al., 2019b]. Under this assumption – which we discuss further in Section 3.6.4 – we can approximate the network with its Taylor expansion around w_0 [Lee et al., 2019]. Let w_t be the fine-tuned weights at time t. Using big-O notation and $f_t := f_{w_t}$, we have:

$$f_t(x) = f_0(x) + \nabla_w f_0(x)|_{w=w_0}(w_t - w_0) + O(\|w_t - w_0\|^2)$$

We now want to use this approximation to characterize the training dynamics of the network during fine-tuning to estimate TT. In such theoretical analyses [Jacot et al., 2018, Lee et al., 2019, Arora et al., 2019a] it is common to assume that the network is trained with Gradient Descent rather than Stochastic Gradient Descent, and in the limit of a small learning rate. In this limit, the dynamics are approximated by the gradient flow differential equation $\dot{w}_t = -\eta \nabla_{w_t} \mathcal{L}$ [Jacot et al., 2018, Lee et al., 2019] where η denotes the learning rate and $\mathcal{L}(w)$ denotes the loss function $\mathcal{L}(w) = \frac{1}{2}\sum_{i=1}^{N} \ell(y_i, f_w(x_i))$, where ℓ is the per-sample loss function (e.g. Cross-Entropy). This approach however has two main drawbacks. First, it does not properly approximate Stochastic Gradient Descent, as it ignores the effect of the gradient noise on the dynamics, which affects both training time and generalization. Second, the differential equation involves the weights of the model, which live in a very high dimensional space thus making finding numerical solutions to the equation not tractable.

To address both problems, building on top of [Hayou et al., 2019] in Section A.5.1 we prove the following result.

Proposition 3.1 (Neural Tangent Kernel learning as a SDE)**.** *In the limit of small learning rate η, the output on the training set of a linearized network f_t^{lin} trained with SGD evolves according to the following Stochastic Differential Equation:*

$$df_t^{lin}(\mathcal{X}) = \underbrace{-\eta \Theta \nabla_{f_t^{lin}(\mathcal{X})} \mathcal{L}_t \, dt}_{\textit{deterministic part}} + \underbrace{\frac{\eta}{\sqrt{|B|}} \nabla_w f_0^{lin}(\mathcal{X}) \Sigma^{\frac{1}{2}}(f_t^{lin}(\mathcal{X})) dn}_{\textit{stochastic part}}, \tag{3.1}$$

where $\mathcal{X}$ is the set of training images, $|B|$ the batch-size and dn is a D-dimensional Brownian motion. We have defined the Gram gradients matrix Θ [Jacot et al., 2018, Shawe-Taylor et al., 2005] (i.e., the empirical Neural Tangent Kernel matrix) and the covariance matrix Σ of the gradients as follows:

$$\Theta = \nabla_w f_0(\mathcal{X}) \nabla_w f_0(\mathcal{X})^T \tag{3.2}$$

$$\Sigma(f_t^{lin}(\mathcal{X})) = \mathbb{E}\big[(J_i \nabla_{f_t^{lin}(x_i)} \mathcal{L}) \otimes (J_i \nabla_{f_t^{lin}(x_i)} \mathcal{L})\big] - \mathbb{E}\big[J_i \nabla_{f_t^{lin}(x_i)} \mathcal{L}\big] \otimes \mathbb{E}\big[J_i \nabla_{f_t^{lin}(x_i)} \mathcal{L}\big] \tag{3.3}$$

where $J_i := \nabla_w f_0(x_i)$. Note both Θ and Σ only require gradients w.r.t. parameters computed at initialization.

The first term of Eq. (3.1) is an ordinary differential equation describing the deterministic part of the optimization, while the second stochastic term accounts for the noise. In Figure 3.2 (left) we show the qualitative different behaviour of the solution

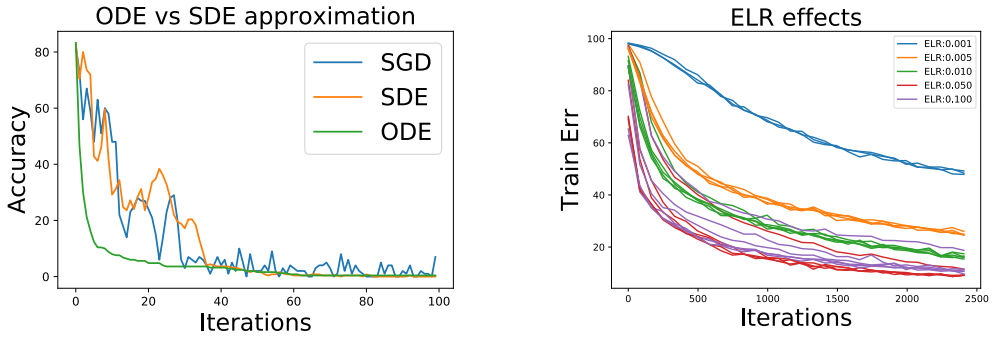


Figure 3.2: **(Left) ODE vs. SDE.** ODE approximation may not be well suited to describe the actual non-linear SGD dynamics (high learning rates regime). **(Right) Fine-tuning with the same ELR have similar curves**. We fine-tune an ImageNet pre-trained network on MIT-67 with different combinations of learning rates and momentum coefficients. We note that as long as the effective learning rate is the same, the loss curves are also similar.

to the deterministic part of Eq. (3.1) and the complete SDE Eq. (3.1). While several related results are known in the literature for the dynamics of the network in weight space [Chaudhari and Soatto, 2017], note that Eq. (3.1) completely characterizes the training dynamics of the linearized model by looking at the evolution of the output $f_t^{\text{lin}}(\mathcal{X})$ of the model on the training samples – a $N \times C$-dimensional vector – rather than looking at the evolution of the weights w_t – a D-dimensional vector. When the number of data points is much smaller than the number of weights (which are in the order of millions for ResNets), this can result in a drastic dimensionality reduction, which allows easier estimation of the solution to Eq. (3.1). Solving Eq. (3.1) still comes with some challenges, particularly in computing Θ efficiently on large datasets and architectures. We tackle these in Section 3.4. Before that, we take a look at how different hyper-parameters and different pre-trainings affect the training time of a DNN on a given task.

3.3.1 Effect of hyper-parameters

Effective learning rate. From Proposition 3.1 it is possible to gauge how hyper-parameters will affect the optimization process of the linearized model and, by proxy, of the original model it approximates. One thing that should be noted is that Proposition 3.1 assumes the network is trained with momentum $m = 0$. Using a non-zero momentum leads to a second order differential equation in weight space, that is not captured by Proposition 3.1. We can however, introduce heuristics to handle the effect

of momentum: [Smith and Le, 2018] note that the momentum acts on the stochastic part scaling it by a factor $\sqrt{1/(1-m)}$. Meanwhile, under the assumptions we used in Proposition 3.1 (small learning rate), we can show (see Appendix A.4) the main effect of momentum on the deterministic part is to re-scale the learning rates by a factor $1/(1-m)$. Given these results, we define the effective learning rate (ELR) $\hat{\eta} = \eta/(1-m)$ and claim that, in first approximation, we can simulate the effect of momentum by using $\hat{\eta}$ instead of η in Eq. (3.1). In particular, models with different learning rates and momentum coefficients will have similar (up to noise) dynamics (and hence training time) as long as the effective learning rate $\hat{\eta}$ remains the same. In Figure 3.2 we show empirically that indeed same effective learning rate implies similar loss curve. That similar effective learning rate gives similar test performance has also been observed in [Li et al., 2020, Smith and Le, 2018].

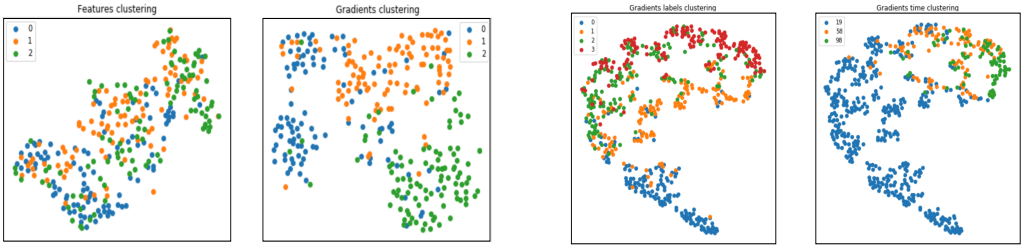
Batch size. The batch size appears only in the stochastic part of the equation, its main effect is to decrease the scale of the SDE noise term. In particular, when the batch size goes to infinity $|B| \to \infty$ we recover the deterministic gradient flow also studied by [Lee et al., 2019]. Note that we need the batch size $|B|$ to go to infinity, rather than being as large as the dataset since we assumed random batch sampling with replacement. If we assume extraction without replacement the stochasticity is annihilated as soon as $|B| = N$ (see [Chaudhari and Soatto, 2017] for a more in depth discussion).

3.3.2 Effect of pre-training

We now use the SDE in Eq. (3.1) to analyze how the combination of different pre-trainings of the model – that is, different w_0's – and different tasks affect the training time. In particular, we show that a necessary condition for fast convergence is that the gradients after pre-training cluster well with respect to the labels. We conduct this analysis for a binary classification task with $y_i = \pm 1$, but the extension is straightforward for multi-class classification, under the simplifying assumptions that we are operating in the limit of large batch size (GD) so that only the deterministic part of Eq. (3.1) remains. Note the infinite batch size assumption is used only here to derive Eq. (3.4), we make no such assumption in Eq. (3.1), which is what we use for the actual training time prediction.

Under these assumptions, Eq. (3.1) can be solved analytically and the loss of the linearized model at time t can be written in closed form as (see Section A.5.2):

$$L_t = (y - f_0(\mathcal{X}))^T e^{-2\eta\Theta t}(y - f_0(\mathcal{X})) \tag{3.4}$$



(a) Features and Gradients clustering. (b) Trajectory clustering.

Figure 3.3: **Are gradients good descriptors to cluster data by semantics and training time?** In these figures we use t-SNE [van der Maaten and Hinton, 2008] to visualize high dimensional vectors (e.g. gradients and features). (a) **Features vs Gradients clustering. (Right)** t-SNE plot of the first five principal components of the gradients of each sample in a subset of CIFAR-10 with 3 classes. Colors correspond to the sample class. We observe that the first 5 principal components are enough to separate the data by class. By Proposition 3.2 this implies faster training time. (**Left**) In the same setting as before, t-SNE plot of the features using the first 5 components of PCA. We observe that gradients separate the classes better than the features. **(b) t-SNE on predicted trajectories.** We consider a subset of 4 classes extracted from CIFAR-10 and test whether gradients are good descriptors of both semantics and training time. To do so, we use gradients to predict linearized trajectories of the output of a DNN and then we cluster the trajectories using t-SNE. In **(left)** we color each point by class and in **(right)** by training time (i.e. time to converge of the prediction residual of each datum to zero). We observe that: clusters split trajectories according both to labels **(left)** and training time **(right)**. Interestingly inside each class there are clusters of points that may converge at different speed.

where L_t is the squared loss on the binary classification and $y = (y_1, ..., y_N)^T \in \mathbb{R}^N$.

The following characterization can be obtained with an eigen-decomposition of the matrix Θ (see Section A.5.2 for the proof).

Proposition 3.2 (Loss decomposition)**.** *Let $S = \nabla_w f_w(\mathcal{X})^T \nabla_w f_w(\mathcal{X})$ be the second moment matrix of the gradients and let $S = U\Lambda^2 U^T$ be the uncentered PCA of the gradients, where $\Lambda = \text{diag}(\lambda_1^2, \dots, \lambda_n^2, 0, \dots, 0)$ is a $D \times D$ diagonal matrix, $n \leq \min(N, D)$ is the rank of S and λ_i^2 are the eigenvalues of S sorted in descending order. Then we have:*

$$L_t = \sum_{k=1}^{n} e^{-2\eta\lambda_k^2 t} (\delta y \cdot v_k)^2, \tag{3.5}$$

where $\lambda_k v_k = (\nabla_w f_0(x_i) \cdot u_k)_{i=1}^N$ is the N-dimensional vector containing the value of the k-th principal component of gradients $\nabla_w f_0(x_i)$ and $\delta y := y - f_0(\mathcal{X})$.

Training speed and gradient clustering. We can give the following intuitive interpretation: consider the gradient vector $\nabla_w f_0(x_i)$ as a representation of the sample x_i. If the first principal components of $\nabla_w f_0(x_i)$ are sufficient to separate the classes (i.e. cluster them), then convergence is faster (see Figure 3.3). Conversely, if we need to use the higher components (associated to small λ_k) to separate the data, then convergence will be exponentially slower. Authors of [Arora et al., 2019a] also use the eigen-decomposition of Θ to explain the slower convergence observed for a randomly initialized two-layer network trained with random labels. This is straightforward since the projection of a random vector will be uniform on all eigenvectors, rather than concentrated on the first few, leading to slower convergence. However, we note that the exponential dynamics predicted by [Arora et al., 2019a] do not hold for more general networks trained from scratch [Zhang et al., 2017] (see Section 3.6.4).

3.4 Efficient numerical estimation of Training Time

In Proposition 3.2 we have shown a closed form solution to the SDE in Eq. (3.1) in the limit of large batch size, and for the MSE loss. Unfortunately, in general Eq. (3.1) does not have a closed form expression when using the cross-entropy loss [Lee et al., 2019]. A numerical solution is however possible, enabled by the fact that we describe the network training in function space, which is much smaller than weight space for over-parametrized models. The main computational cost is to create the matrix Θ in Eq. (3.1) – which has cost $O(DC^2N^2)$ – and to compute the noise in the stochastic term. Here we show how to reduce the cost of Θ to $O(D_0C^2N^2)$ for $D_0 \ll D$ using

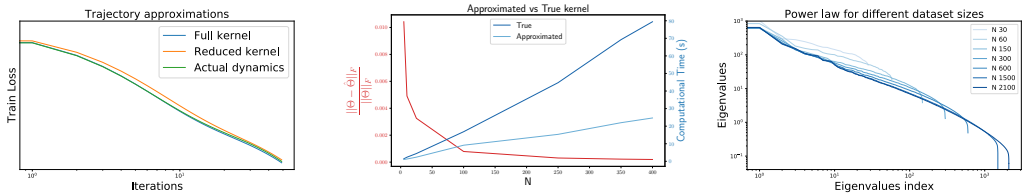


Figure 3.4: **Left** Actual fine-tuning of a DNN with GD compared to the numerical solution of Eq. (3.1) and the solution using an approximated Θ. The approximated Θ can faithfully describe fine-tuning dynamics while being twice as fast to compute and 100 times smaller to be stored. **Center** Relative difference in Frobenius norm of the real and approximated Θ as the dataset size varies (red), and their computational time (blue). **Right**: Eigen-spectrum of Θ computed on subsets of MIT-67 of increasing size. Note the convergence to a common power law (i.e. a line in log-log scale).

a random projection approximation. Then, we propose a fast approximation for the stochastic part. Finally, we describe how to reduce the cost in N by using only a subset $N' < N$ of samples to predict training time.

3.4.1 Random projection

To keep the notation uncluttered, here we assume w.l.o.g. $C = 1$. In this case the matrix Θ contains N^2 pairwise dot-products of the gradients (a D-dimensional vector) for each of the N training samples (see Eq. (3.2)). Since D can be very large (in the order of millions) storing and multiplying all gradients can be expensive as N grows. Hence, we look at a dimensionality reduction technique. The optimal dimensionality reduction that preserves the dot-product is obtained by projecting on the first principal components of SVD, which however are themselves expensive to obtain. A simpler technique is to project the gradients on a set of D' standard Gaussian random vectors: it is known that such random projections preserve (in expectation) pairwise product [Bingham and Mannila, 2001, Achlioptas, 2003] between vectors, and hence allow us to reconstruct the Gram matrix while storing only D'-dimensional vector, with $D' \ll D$. We further increase computational efficiency using multinomial random vectors {-1,0,+1} as proposed in [Achlioptas, 2003] which further reduce the computational cost by avoiding floating point multiplications. In Figure 3.4 we show that the entries of Θ and its spectrum are well approximated using this method, while the computational time becomes much smaller.

3.4.2 Computing the noise

The noise covariance matrix $\Sigma(f_t^{\text{lin}}(\mathcal{X}))$ is a $D \times D$-matrix that changes over time. Both computing it at each step and storing it is prohibitive. Estimating Σ correctly is important to describe the dynamics of SGD [Chaudhari and Soatto, 2018], however we claim that a simple approximation may suffice to describe the simpler dynamic in function space. We approximate $\nabla_w f_0^{\text{lin}}(\mathcal{X})\Sigma^{1/2}$ approximating Σ with its diagonal (so that the we only need to store a D-dimensional vector). Rather than computing the whole Σ at each step, we estimate the value of the diagonal at the beginning of the training. Then, by exploiting Eq. (3.3), we see that the only change to Σ is due to $\nabla_{f_t^{\text{lin}}}\mathcal{L}$, whose norm decreases over time. Therefore we use the easy-to-compute $\nabla_{f_t^{\text{lin}}}\mathcal{L}$ to re-scale our initial estimate of Σ.

3.4.3 Larger datasets

In the MSE case from Eq. (3.4), knowing the eigenvalues λ_k and the corresponding residual projections $p_k = (\delta y \cdot v_k)^2$ we can predict in closed form the whole training curve. Is it possible to predict λ_k and p_k using only a subset of the dataset? It is known [Shawe-Taylor et al., 2005] that the eigenvalues of the Gram matrix of Gaussian data follow a power-law distribution of the form $\lambda_k = ck^{-s}$. Moreover, by standard concentration argument, one can prove that the eigenvalues should converge to a given limit as the number of datapoints increases. We verify that a similar power-law and convergence result also holds for real data (see Figure 3.4). Exploiting this result, we can estimate c and s from the spectrum computed on a subset of the data, and then predict the remaining eigenvalues. A similar argument holds for the projections p_k, which also follow a power-law (albeit with slower convergence). We describe the complete estimation in Appendix A.3.

3.5 Training Time prediciton Algorithm

We can compute an estimate of training time exploiting the predictions of the linearized approximation (see Algorithm 1): we use the predictions $f_t^{\text{lin}}(\mathcal{X})$ to compute the prediction errors and hence the training loss along the optimization path (e.g. Fig. 3.9). Then, we estimate the TT on the linearized loss according to the given TT definition.

We now briefly describe some implementations details regarding the numerical solution of ODE and SDE. Both of them can be solved by means of standard algorithms: in the ODE case we used LSODA (which is the default integrator in *scipy.integrate.odeint*), in the SDE case we used Euler-Maruyama algorithm for Ito equations.

We observe removing batch normalization (preventing the statistics to be updated) and removing data augmentation improve linearization approximation both in the case of GD and SGD. Interestingly data augmentation only marginally alters the spectrum of the Gram matrix Θ and has little impact on the linearization approximation w.r.t. batch normalization. [Goldblum et al., 2019] observed similar effects but, differently from us, their analysis has been carried out using randomly initialized ResNets.

Algorithm 1 Estimate the Training Time on a given target dataset and hyper-parameters.

Data: Number of steps T to simulate, threshold ϵ to determine convergence, pre-trained weights w_0 of the model, a target dataset with images $\mathcal{X} = \{x_i\}_{i=1}^N$ and labels $\mathcal{Y} = \{y_i\}_{i=1}^N$, batch size B, learning rate η, momentum $m \in [0, 1)$.
Result: An estimate $\hat{T}_\epsilon$ of the number of steps necessary to converge within an ϵ-ball around the "asymptotic" value $\bar{\mathcal{L}}$: $T_\epsilon := \min\{t : |\mathcal{L}_s - \bar{\mathcal{L}}| < \epsilon, \forall s \geq t\}$.
Initialization: Compute initial network predictions $f_0(\mathcal{X})$, estimate Θ using random projections (Section 3.4), compute the ELR $\tilde{\eta} = \eta/(1-m)$ to use in Eq. (3.1) instead of η
if B = N **then**
    Get $f_t^{\text{lin}}(\mathcal{X})$ solving the ODE in Eq. (3.1) (only *deterministic part*) for T steps
else
    Get $f_t^{\text{lin}}(\mathcal{X})$ solving the SDE in Eq. (3.1) for T steps (see Section 3.4)
end if
Using $f_t^{\text{lin}}(\mathcal{X})$ and $\mathcal{Y}$ compute linearized loss $\mathcal{L}_t^{lin} \quad \forall t \in \{1, ..., T\}$
if B = N **then**
    Set $\bar{\mathcal{L}} = \mathcal{L}_T$
else
    Set $\bar{\mathcal{L}} = \sum_{i=T-T_{\text{avg}}+1}^{T} \mathcal{L}_i$, where T_{avg} is the length of the running average window used to smooth the trajectory loss
end if
return $\hat{T}_\epsilon := \min\{t : |\mathcal{L}_s^{lin} - \mathcal{L}_T^{lin}| < \epsilon, \forall s \geq t\}$

3.6 Experiments

We now empirically validate the accuracy of Proposition 3.1 in approximating the loss curve of an actual Deep Neural Network fine-tuned on a large scale dataset. We also validate the goodness of the numerical approximations described in Section 3.4. Due to the lack of a standard and well established benchmark to test Training Time estimation

algorithms we developed one with the main goal to closely resemble fine-tuning common practice for a wide spectrum of different tasks.

Table 3.1: Training Time absolute errors (number of steps) for CE loss using GD for $T = 150$ epochs at different thresholds ϵ. TT estimates when ODE assumptions do and do not hold: high η (0.005) and small η (0.0001).

TT error (# of steps)	$\epsilon = 1\%$		$\epsilon = 10\%$		$\epsilon = 40\%$	
η	low	high	low	high	low	high
Cars [Krause et al., 2013]	9	18	7	8	1	0
Surfaces [Bell et al., 2015]	6	13	6	7	6	3
Mit67 [Quattoni and Torralba, 2009]	8	10	6	8	3	1
Aircrafts [Maji et al., 2013]	5	21	5	4	9	7
CUB200 [Welinder et al., 2010]	6	6	5	8	1	1
CIFAR100 [Krizhevsky, 2009]	10	15	6	7	2	3
CIFAR10 [Krizhevsky, 2009]	9	14	8	9	3	3

3.6.1 Experimental setup

We define Training Time as the time the (smoothed) loss remains persistently below a given threshold. However, since different datasets converge at different speeds, the same threshold can be too high (it is hit immediately) for some datasets, and too low for others (it may take hundreds of epochs to be reached). To solve this, and have cleaner readings, we define a "normalized" threshold as follows: we fix the total number of fine-tuning steps T, and measure instead the first time the loss is persistently within ϵ from the final value at time T (see Definition 3.2). This measure takes into account the "asymptotic" loss reached by the DNN within the computational budget (which may not be close to zero if the budget is low), and naturally adapts the threshold to the difficulty of the dataset. We compute both the real loss curve and the predicted training curve using Proposition 3.1 and compare the ϵ-training-time measured on both. We report the *absolute prediction error*, that is $|t_{\text{predicted}} - t_{\text{real}}|$. For all the experiments we extract 5 random classes from each dataset (Table 3.1) and sample 150 images (or the maximum available for the specific dataset). Then we fine-tuned ResNet18/34 using either GD or SGD.

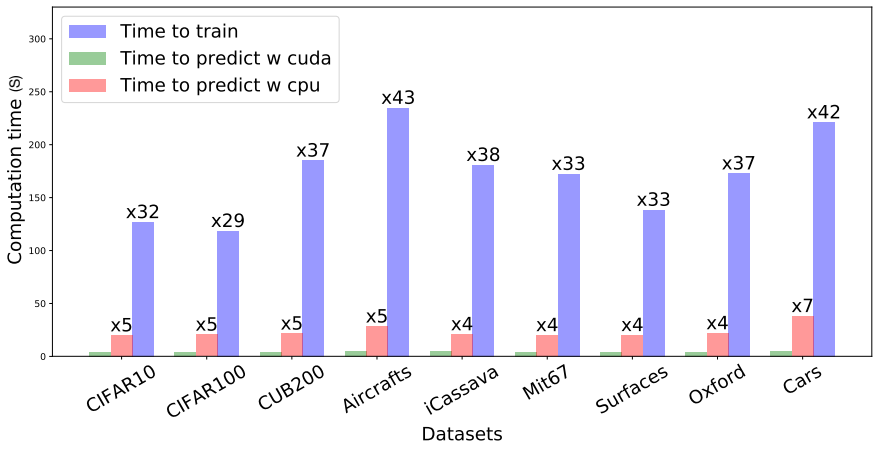


Figure 3.5: **Wall clock time (in seconds) to compute TT estimate vs fine-tuning running time.** We run the methods described in Section 3.4.1 both on GPU and CPU. Training (fine-tuning) is done on GPU. We implemented the approximation methods presented in Section 3.4.1 both in GPU and CPU (time comparison is reported).

3.6.2 Training Time prediction Benchmark

In Figure 3.1 we show TT estimates errors (for different $\epsilon \in \{1, ..., 40\}$) under a plethora of different conditions ranging from different learning rates, batch sizes, datasets and optimization methods. For all the experiments we choose a multi-class classification problem with Cross Entropy loss unless specified otherwise, and fixed computational budget of $T = 150$ steps both for GD and SGD. We note that our estimates are consistently within respectively a 13% and 20% relative error around the actual training time 95% of the times.

In Table 3.1 we describe the sensitivity of our estimates to different thresholds ϵ both when our assumptions do and do not hold (high and low learning rates regimes). Note that a larger threshold ϵ is hit during the initial convergence phase of the network, when a small number of iterations corresponds a large change in the loss. Correspondingly, the Training Time can be measured more accurately and our errors are lower. A smaller ϵ depends more on correct prediction of the slower asymptotic phase, for which exact Training Time is more difficult to estimate.

Wall-clock run-time. In Figure 3.5 we show the wall-clock runtime of our training time prediction method compared to the time to actually train the network for T steps. Our method is 30-40 times faster. Moreover, we note that it can be run completely on

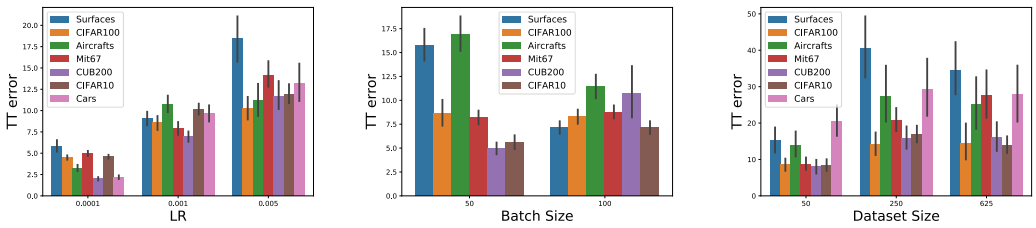


Figure 3.6: Average and 95% confidence intervals of TT estimate error for: **Left**: GD using different learning rates. **Center**: SGD using different batch sizes. **Right**: SGD using different dataset sizes. The average is taken w.r.t. random classes with different number of samples: {10, 50, 125}

CPU without a drastic drop in performance. This allows to cheaply estimate TT and allocate/manage resources even without access to a GPU.

Effect of dataset distance. We note that the average error for Surfaces (Figure 3.6) is uniformly higher than the other datasets. This may be due to the texture classification task being quite different from ImageNet, on which the network is pretrained. In this case we can expect that the linearization assumption is partially violated since the features must adjust more during fine-tuning.

3.6.3 Hyper-parameters ablation study

Effect of hyper-parameters on prediction accuracy. We derived Proposition 3.1 under several assumptions, importantly: small learning rate and w_t close to w_0. In Figure 3.6 (left) we show that increasing the learning rate decreases the accuracy of our prediction, albeit the accuracy remains good even at larger learning rates. Fine-tuning on larger dataset makes the weights move farther away from the initialization w_0. In Figure 3.6 (right) we show that this slightly increases the prediction error. Finally, we observe in Figure 3.6 (center) that using a smaller batch size, which makes the stochastic part of Proposition 3.1 larger also slightly increases the error. This can be ascribed to the approximation of the noise term (Section 3.4.1). On the other hand, in Figure 3.2 (right) we see that the effect of momentum on a fine-tuned network is very well captured by the effective learning rate (Section 3.3.1), as long as the learning rate is reasonably small, which is the case for fine-tuning. Hence the SDE approximation is robust to different values of the momentum. In general, we note that even when our assumptions are not fully met, Training Time can still be approximated with only slightly higher error. This suggests that point-wise proximity of the training trajectory of linear and real models is not necessary as long as their behavior (decay-rate) is similar

(see Section 3.6.4).

3.6.4 How good is linearization?

While we do not necessarily expect a linear approximation around a random initialization to hold during training of a real (non infinitely wide) network, we exploit the fact that when using an over-parametrized pre-trained network the weights are more likely to remain close to initialization [Mu et al., 2020], improving the quality of the approximation. In particular, in Fig. 3.7 we show that even when using a pre-trained network, the trajectories of the weights of linearized model and of the real model can differ substantially. On the other hand, we also show that the linearized model correctly predicts the *outputs* (not the weights) of the real model throughout the training, which is enough to compute the loss. We believe that this is the reason why Eq. (3.1) can accurately predict the training time using the linear approximation in function space.

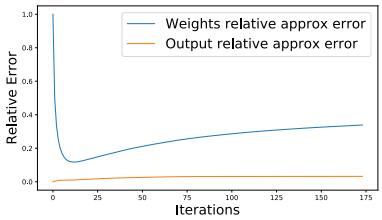


Figure 3.7: **Comparison of prediction accuracy in weight space vs. function space.** We compare the result of using the deterministic part of Eq. (3.1) to predict the weights w_t at time t and the outputs $f_t(\mathcal{X})$ of the networks under GD. The relative error in predicting the outputs is much smaller than the relative error of predicting the weights at all times. This, together with the computational advantage, motivates the decision of using Eq. (3.1) to predict the behavior in function space.

Comparison of predicted and real error curve

In Figure 3.8 we compare the error curve predicted by our method and the actual train error of the model as a function of the number of optimization steps. The model is trained on a subset of 2 classes of CIFAR-10 with 150 samples. We run the comparison for both gradient descent (left) and SGD (right), using learning rate $\eta = 0.001$, momentum $m = 0$ and (in the case of SGD) batch size 100. In both cases we observe that the predicted curve is reasonably close to the actual curve, more so at the beginning of the training (which is expected, since the linear approximation is more likely to hold). We also perform an ablation study to see the effect of different approximation of SGD noise

in the SDE in Eq. (3.1). In Figure 3.8 (right) we estimate the variance of the noise of SGD at the beginning of the training, and then assume it is constant to solve the SDE. Notice that this predicts the wrong asymptotic behavior, in particular the predicted error does not converge to zero as SGD does. In Figure 3.8 (center) we rescale the noise as we suggest in Section 3.4: once the noise is rescaled the SDE is able to predict the right asymptotic behavior of SGD.

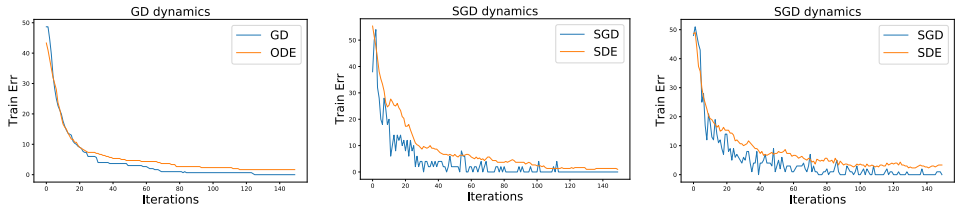


Figure 3.8: **(Left)** Comparison of the real error curve on CIFAR10 using gradient descent and the predicted curve. **(Center)** Same as before, but this time we train using SGD and compare it with the prediction using the technique described in Section 3.4 to approximate the covariance of the SGD noise that appears in the SDE in Eq. (3.1). **(Right)** Same as (center), but using constant noise. Note that in this case we do not capture the right asymptotic behavior of SGD.

Point-wise similarity of predicted and observed loss curve

In some cases, we observe that the predicted and observed loss curves can differ. This is especially the case when using Cross-Entropy loss (Figure 3.9). We hypothesize that this may be due to improper prediction of the dynamics when the softmax output saturates, as the dynamic becomes less linear [Lee et al., 2019]. However, the train error curve (which only depends on the relative order of the outputs) remains relatively correct. We should also notice that prediction of the ϵ-training-time $\hat{T}_\epsilon$ can be accurate even if the curves are not point-wise close. The ϵ-training-time seeks to find the first time after which the loss or the error is within an ϵ threshold. Hence, as long as the real and predicted loss curves have a similar asymptotic slope the prediction will be correct, as we indeed verify in Figure 3.9 (bottom).

Overestimation vs. underestimation error

Up to now we focused on prediction error rates (see Table 3.1 and Fig. 3.6), in this section we will briefly describe how the errors are distributed: are we overestimating or underestimating training time with our method? In Figure 3.9 we compare training time

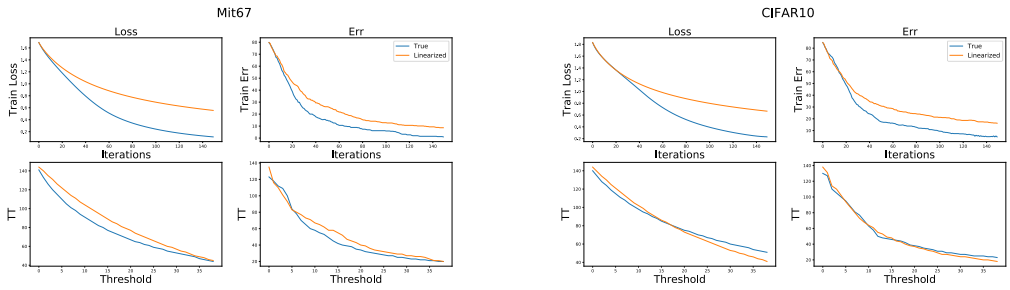


Figure 3.9: **Training time prediction is accurate even if loss curve prediction is not. (Top row)** Loss curve and error curve prediction on MIT-67 (left) and CIFAR-10 (right). **(Bottom row)** Predicted time to reach a given threshold (orange) vs real training time (blue). We note that on some datasets our loss curve prediction differs from the real curve near convergence. However this does not affect the accuracy of the prediction much since our training time definition measures the time to reach the asymptotic value rather than the time reach an absolute threshold.

predictions and actual training time values for different thresholds ϵ and datasets. While for high thresholds ϵ the errors distribution does not seem to be particularly skewed (consider both MIT-67 and CIFAR-10) the situation is different for small thresholds ϵ: our method tends to slightly overestimate training time. In particular overestimation is due to the non-linearity and high capacity of the network which might not be entirely captured by the linearization approximation (the non-linear model decreases its loss faster w.r.t. its linear approximation).

3.7 Discussions and conclusions

In this chapter we study the problem of predicting the fine-tuning Training Time of overparametrized DNNs. Our results hinge on the connection between infinitely wide feedforward DNNs and the NTK (Section 2.4). In particular, our approach is rather general and can be applied both to images, time series, and other types of data so long as infinitely wide feedforward DNNs are used. We exploit the empirical NTK matrix (the Gram-Matrix Θ of the gradients at initialization) to describe both the fine-tuning learning dynamics and estimate fine-tuning Training Time. We focus on a computer vision setup and show that it is possible to predict with a 13-20% accuracy the time that it will take for a pre-trained network to reach a given loss in only a small fraction of the time that it would require to actually train the model. We do this by studying the training dynamics of a linearized approximation of the DNN model using the SDE in Eq. (3.1), which we solve numerically. To further improve the efficiency

of our method, we propose to estimate the matrix Θ using random projections of the gradients (Section 3.4), this simple approach allows us to apply our method for common architectures and large datasets. To validate our method we measure its accuracy on off-the-shelf state-of-the-art models, we are able to predict the correct training time within a 20% error with 95% confidence over several different real datasets and hyperparameters at only a small fraction of the time it would require to actually run the training (30-45x faster in our experiments).

Moreover we study the dependency of training time from pre-training and hyperparameters (see Section 3.3.1) and show that our method yields predictions that have lower accuracy on some tasks rather than others, for instance it has lower accuracy on texture-based tasks than object classification. However, since we consider datasets as a whole, prediction inaccuracies do not impact any particular cohort or segment of the data.

In Section 3.6.4 we show that even when the trajectories of the weights of the linearized model differ from the real model ones, the linearization of a pre-trained model correctly predicts the *outputs* of the real model throughout the training. We hypothesise that this is the reason why Eq. (3.1) can accurately predict the training time with the linear approximation in function space.

To conclude, we note that the procedure described in this chapter only relies on information available at initialization and does not require any feedback from the actual training. It is straightforward to extend our method to allow training feedback (e.g. gradients updates) by simply applying our Training Time prediction at different checkpoints during fine-tuning. We expect training feedback could further improve accuracy of our method thanks to the better the linearization approximation at each checkpoint. We believe this is an interesting direction worthy of further investigation.

4

Model Selection

A "**model zoo**" is a collection of pre-trained models, obtained by training different architectures on many datasets covering a variety of tasks and domains. For instance, the zoo could comprise models (or experts) trained to classify trees, birds, fashion items, aerial images, etc. The typical use of a model zoo is to provide a good initialization which can then be fine-tuned for a new target task, for which we have few training data. This strategy is an alternative to the more common practice of starting from a model trained on a large dataset, say Imagenet [Deng et al., 2009], and is aimed at providing better domain coverage and a stronger inductive bias. Despite the growing usage of model zoos [Cui et al., 2018, Kolesnikov et al., 2020, Li et al., 2020, Triantafillou et al., 2020] there is little in the way of analysis, both theoretical and empirical, to illuminate which approach is preferable under what conditions. In Fig. 4.1 and Fig. 4.2, we show that fine-tuning with a model zoo is indeed better, especially when training data is limited. Fig. 4.1 and Fig. 4.2 also shows that using a model zoo, we can outperform Hyper-Parameter Optimization (i.e. grid search over optimization hyper-parameters) performed on fine-tuning of the Imagenet pre-trained model.

Fine-tuning with a model zoo can be done by brute-force fine-tuning of each model in the zoo, or more efficiently by using "**model selection**" to select the closest model to the target dataset (or best initialization) from which to fine-tune. The goal of model selection therefore is to find the best pre-trained model to fine-tune on the target task, without performing the actual fine-tuning. So, we seek an approximation to the fine-tuning process. In our work, we develop an analytical framework to characterize the fine-tuning process using a linearization of the model around the point of pre-training [Mu et al., 2020], drawing inspiration from the theory on the Neural Tangent Kernel [Jacot et al., 2018, Lee et al., 2019]. Inspired by NTK (Section 2.5) and kernel selection

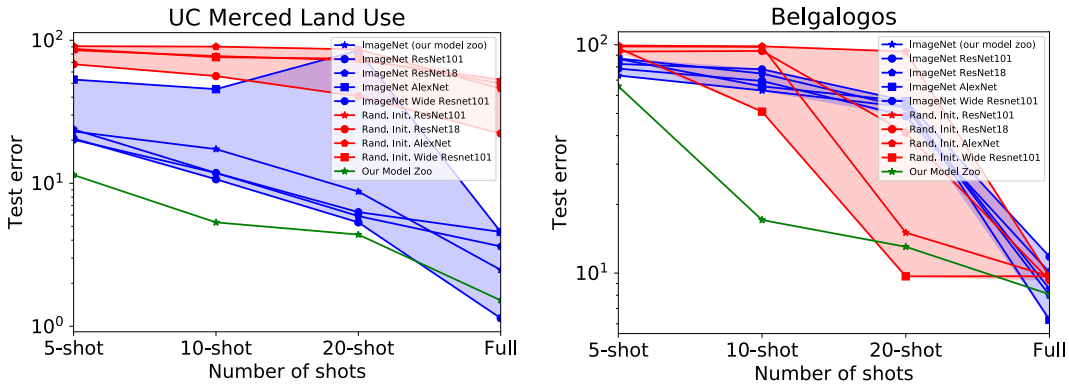


Figure 4.1: **Model zoo vs. different architectures.** Fine-tuning using our model zoo (green) is better (i.e. lower test error) than fine-tuning using different architectures with Random (red) or Imagenet pre-trained (blue) initialization. We use fine-tuning hyper-parameters of Section 4.4.3 with $\eta = .005$.

criteria (Section 2.3.6) we suggest two criteria to select the best model to fine-tune from, which we call Label-Gradient Correlation (LGC) and Label-Feature Correlation (LFC). Given its simplicity, we consider our criteria as *baselines*, rather than full-fledged methods for model selection, and compare the state-of-the-art in model selection – e.g. RSA [Dwivedi and Roig, 2019], LEEP [Nguyen et al., 2020], Domain Similarity [Cui et al., 2018], Feature Metrics [Ueno and Kondo, 2020] – against it.

The methods we propose in this chapter do not depend on a particular application domain (say image processing or time series analysis) and can be applied so long as feedforward DNNs are used. Nonetheless, the driving force of the material developed in this chapter started to unblock the adoption of real-world Computer Vision AutoML systems, therefore our empirical validation has only been conducted on this application domain.

Model selection for DNNs in computer vision being a relatively recent endeavor, there is currently no standard dataset or a common benchmark to perform such a comparison. For example, LEEP [Nguyen et al., 2020] performs its model selection experiments on transfer (or fine-tuning) from Imagenet pre-trained model to 200 randomly sampled tasks of CIFAR-100 [Krizhevsky, 2012] image classification, RSA [Dwivedi and Roig, 2019] uses the Taskonomy dataset [Zamir et al., 2018] to evaluate its prediction of task transfer (or model selection) performance. Due to these different experimental setups, the state-of-the-art in model selection is unclear. Therefore, in Section 4.4 we build a new benchmark comprising a large model zoo and many target tasks. For our

model zoo, we use 8 large image classification datasets (from different domains) to train single-domain and multi-domain experts. We use various image classification datasets as target tasks and study fine-tuning (Section 4.4.3) and model selection (Section 4.4.4) using our model zoo. To the best of our knowledge ours is the first large-scale benchmark for model selection.

By performing fine-tuning and model selection on our benchmark, we discover the following:

1. We show (Fig. 4.1 and Fig. 4.2) that fine-tuning models in the model zoo can outperform the standard method of fine-tuning with Imagenet pre-trained architectures and HPO. We obtain better fine-tuning than Imagenet expert with, both model zoo of single-domain experts (Fig. 4.3) and multi-domain experts (Fig. 4.4). While in the high-data regime using a model zoo leads to modest gains, it improves accuracy in the low-data regime.
2. For any given target task, we show that only a small subset of the models in the zoo lead to accuracy gain (Fig. 4.3). In such a scenario, brute-force fine-tuning all models to find the few that improve accuracy is wasteful. Fine-tuning with all our single-domain experts in the model zoo is 40× more compute intensive than fine-tuning an Imagenet Resnet-101 expert in Table 4.3.
3. Our LGC model selection, and particularly its approximation LFC, can find the best models from which to fine-tune without requiring an expensive brute-force search (Table 4.3). With only 3 selections, we can select models that show gain over Imagenet expert (Fig. 4.5). Compared to Domain Similarity [Cui et al., 2018], RSA [Dwivedi and Roig, 2019] and Feature Metrics [Ueno and Kondo, 2020], our LFC score can select the best model to fine-tune in fewer selections, and it shows the highest ranking correlation to the fine-tuning test accuracy (Fig. 4.7) among all model selection methods.

Fine-tuning using our model zoo can obtain lower test error compared to: Fig. 4.1 using different architectures and Fig. 4.2 hyper-parameter optimization of Imagenet expert. The standard fine-tuning approach entails picking a network architecture pre-trained on Imagenet to fine-tune and performing hyper-parameter optimization during fine-tuning. We outperform this strategy by fine-tuning using our model zoo described in Section 4.4.2. We plot test error as a function of the number of per-class samples (i.e. shots) in the dataset. In Fig. 4.1, we compare fine-tuning with our single-domain experts in the model zoo to using different architectures (AlexNet, ResNet-18, ResNet-101, Wide ResNet-101) for fine-tuning. In Fig. 4.2,

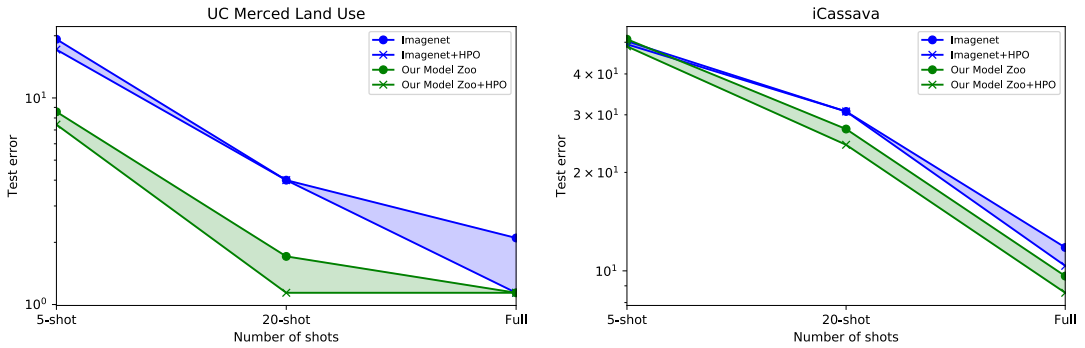


Figure 4.2: **Model zoo vs. HPO of Imagenet expert.** Fine-tuning using our model zoo (green) is better than fine-tuning with hyper-parameter optimization (HPO) on Imagenet pre-trained (blue) Resnet-101 model. We use fine-tuning hyper-parameters of Section 4.4.3 and perform HPO with $\eta = .01, .005, 0.001$.

we show fine-tuning with our model zoo obtains lower error than performing HPO on Imagenet pre-trained Resnet-101 [He et al., 2016] during fine-tuning. Model zoo lowers the test error, especially in the low-data regime (5, 10, 20-shot per class samples of target task). Since we compare to Imagenet fine-tuning, we exclude Imagenet experts from our model zoo for the above plots.

4.1 A linearized framework to analyse fine-tuning

Notation. We have a model zoo, $\mathcal{F}$, of n pre-trained models or experts: $\mathcal{F} = \{f^1, f^2, \cdots f^n\}$. Our aim is to classify a target dataset, $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, by fine-tuning models in the model zoo. Here, $x_i \in \mathcal{X}$, is the i-th input image and $y_i \in \mathcal{Y}$, is the corresponding class label . For a network $f \in \mathcal{F}$ with weights w, we denote the output of the network with $f_w(x)$. w_0 denotes the initialization (or pre-trained weights) of models in the model zoo. The goal of model selection is to predict a score $S(f_{w_0}, \mathcal{D})$ that measures the fine-tuning accuracy on the test set $\mathcal{D}^{\text{test}}$, when $\mathcal{D}$ is used to fine-tune the model f_{w_0}. Note, S does not need to exactly measure the fine-tuning accuracy, it needs to only predict a score that correlates to the ranking by fine-tuning accuracy. The model selection score for every pre-trained model, $S(f^k, \mathcal{D})$ for $k \in \{1, 2, \cdots, n\}$, can then be used as proxy to rank and select top-k models by their fine-tuning accuracy. The score S needs to estimate (a proxy for) fine-tuning accuracy without performing any fine-tuning, in the following we exploit the linearization approximation to fine-tuning we used in Chapter 3 to derive our *Label-Gradient Correlation* (S_{LG}) and *Label-Feature Correlation* (S_{LF}) (Section 4.3) scores for model selection.

4.2 Model selection as kernel selection

Given an initialization w_0 and the weights of the pre-trained model, we can define the linearized model:

$$f_t^{\text{lin}}(x) = f_0(x) + \frac{\partial f_{w(t)}(x)}{\partial w}\bigg|_{w=w_0} (w - w_0),$$

which approximates the output of the real model for w close to w_0. Mu et al. [Mu et al., 2020] observe that, while in general not accurate, a linear approximation can correctly describe the model throughout fine-tuning since the weights w tend to remain close to the initial value w_0 (see also our discussion in Section 2.4 and Chapter 3). Under the linear approximation it is possible to show (Section 2.5.1) that the squared loss evolves as:

$$\mathcal{L}(t) = (y - f_{w_0}(\mathcal{X}))^T e^{-2\eta\Theta t}(y - f_{w_0}(\mathcal{X})) \tag{4.1}$$

where $f_{w_0}(\mathcal{X})$ denotes the vector containing the output of the network on all the images in the dataset, y denotes the vectors of all training labels, and Θ is the empirical Neural Tangent Kernel (NTK) matrix Section 2.4.2.

Under the linear approximation (or infinite width limit Section 2.4.3) the DNN is equivalent to a kernel method defined on the NTK matrix Θ at initialization. In fact, from Eq. (4.1), the behavior of the network during fine-tuning is fully characterized by the kernel matrix Θ, which depends on the pre-trained model f_{w_0}, the data $\mathcal{X}$ and the task labels y. We then expect to be able to select the best model by looking at these quantities. In particular, thanks to the linearized approximation we argue it is profitable to use kernel selection criteria (Section 2.3.6) to choose the best model from the model zoo $\mathcal{F}$. As described in Section 2.3.6, the model selection problem is equivalent to a kernel selection problem, so that we can solve model selection for DNNs by using kernel selection criteria. Commonly used measures for model selection are: cross validation [Friedman et al., 2001], marginal likelihood [Rasmussen and Williams, 2006], kernel alignment [Cristianini et al., 2002, Cortes et al., 2012] and centered kernel alignment [Wang et al., 2012, Cortes et al., 2012]. Interestingly many of these criteria automatically encode a trade-off model (kernel) complexity and fit so that the best model (kernel) is the one for which the fitting term is as small as possible while not being too complex. This trade-off is typically associated to the law of parsimony (*Occam's razor principle*) for which, if two hypothesis explain reality equally well the "simpler" of the two must be the preferred one. As noted in Section 2.3.6 each kernel selection criterion is characterized by its own strengths and limitations. For example cross validation is known to require high computational costs while being general enough to be

applicable to a broad set of learning problems [Wahba, 1990, Friedman et al., 2001]. Marginal likelihood is an automatic algorithm to trade-off model complexity and fit but can be efficiently applied only in the Bayesian setting (Section 2.3.3). Kernel alignment is independent of the actual learning machine used but it is known to suffer from data imbalance. So that in the following we shall mainly focus on *Centered Kernel Alignment* which is known to be an improved version of kernel alignment and does not suffer from data imbalance and better correlates with generalization [Cortes et al., 2012, Wang et al., 2012].

We now proceed by deriving the connection between Θ and y to centered kernel alignment.

4.3 Label-Feature and Label-Gradient correlation

In this section we briefly revise the main properties of Centered Kernel Alignment (Section 2.3.6) and exploit CKA to build two model selection scores: *Label-Gradient Correlation* and *Label-Feature Correlation*.

4.3.1 Centered Kernel Alignment

Centered Kernel Alignment has been proposed in [Cortes et al., 2012] as an improved version of Kernel Alignment [Cristianini et al., 2002], this new alignment measure is considered to better correlate with generalization and improves Kernel Alignment predictions on unbalanced classification tasks [Cortes et al., 2012, Wang et al., 2012]. The main idea in [Cortes et al., 2012] is to compute the Kernel Alignment measure [Cristianini et al., 2002] on a centered feature space (see Section 2.3.6 for more details).

For the sake of completeness we now recall the definition of Empirical Centered Kernel Alignment, which measures the alignment between two given kernel matrices associated to two different kernels:

Definition 4.1 (Centered Empirical Kernel Alignment (see Definition 2.7))**.** Let $K_1 \in \mathbb{R}^{N\times N}$ and $K_2 \in \mathbb{R}^{N\times N}$ be two kernel matrices such that $\|K_i\|_F \neq 0$ for $i = 1, 2$. Then, the centered alignment between K_1 and K_2 is defined by:

$$\hat{\rho}_c(K_1, K_2) = \frac{\langle K_{c1}, K_{c2}\rangle_F}{\|K_{c1}\|_F \|K_{c2}\|_F}$$

where $K_{ci} \in \mathbb{R}^{N\times N}$ is the kernel matrix evaluated on all the data $\{x_i\}$ with $i \in [N]$

and the centered kernel matrix is defined as:

$$K_c = \left[I_N - \frac{\mathbb{1}\mathbb{1}^T}{N}\right] K \left[I_N - \frac{\mathbb{1}\mathbb{1}^T}{N}\right]$$

Remark 4.1 (Empirical CKA as similarity score). If the kernel matrices K_1 and K_2 are considered as bidimensional vectors, the empirical centered kernel alignment can be seen as a similarity score based on the cosine of their angle. Therefore CKA is bounded between -1 and 1. Moreover, since K_i are positive semi-definite Gram matrices, CKA is lower-bounded by 0.

Remark 4.2 (Properties of CKA). It is well known that Centered Kernel Alignment possesses favorable properties [Cortes et al., 2012, Wang et al., 2012]: *computational efficiency* w.r.t. other kernel selection criteria, the computational cost to evaluate $\hat{\rho}_c(K_1, K_2)$ scales as $O(N^2)$. *Concentration*, the probability of the empirical estimate $\hat{\rho}_c(K_1, K_2)$ deviating from its expected value $\rho_c(k_1, k_2)$ Eq. (2.37) can be bounded as an exponentially decaying function, so that the empirical estimator is stable w.r.t. different split of the data. *Generalization*, CKA positively correlates with generalization, high alignment values imply there exists a separation of the data with low bound on the generalization error. It is therefore expected that maximizing CKA on training set foster generalization performance.

4.3.2 Label-Gradient Correlation

We now show how to apply Centered Kernel Alignment to typical classification problems and define our Label-Gradient correlation based on the NTK matrix Θ.

The natural way to apply CKA to a classification problem is by defining the following two kernels: the first one measures similarity between input locations $k_x(x_i, x_j)$ and the second measures similarity between target labels $k_y(y_i, y_j)$. We shall use the NTK matrix evaluated on the training data Θ as our first kernel matrix while for the similarity matrix of the target labels we shall use $K_y = YY^T$ where $Y \in \mathbb{R}^{N \times C}$ is a matrix whose rows are the target labels of dimension C for each datum. K_y corresponds to the following kernel choice: $k_y(y_i, y_j) = 1$ if $y_i = y_j$ and 0 otherwise. We therefore define *Label-Gradient Correlation* as:

$$S_{\mathrm{LG}}(f_{w_0}, \mathcal{D}) = \frac{\langle \Theta, YY^T \rangle_F}{\|\Theta\|_F \|K_y\|_F} = \frac{\mathrm{Tr}\, Y^T \Theta Y}{\|\Theta\|_F \|K_y\|_F} \tag{4.2}$$

we highlight that the NTK matrix depends on the initialization w_0:

$$(\Theta)_{ij} = \nabla_w f_0(x_i)^T \nabla_w f_0(x_j) \tag{4.3}$$

Eq. (4.2) can be interpreted as giving high LG score (i.e., the model is good for the task) if the gradients are similar whenever the labels are also similar, and are different otherwise.

Remark 4.3 (Connection with Marginal Likelihood Remark 2.18). Eq. (4.2) is analogous to the Marginal Likelihood formula Eq. (2.26). In particular note Eq. (4.2) can be decoupled into two separate terms: a term depending on the target labels $\operatorname{Tr} Y^T \Theta Y$ and a complexity term $\|\Theta\|_F$. The first term measures the correlation between similarity in gradients space (measured by Θ) and target similarity (measured by K_y). So that it is large if gradients (which are to be considered as features in the linearized model) are good to separate input data (good fitting) and low if not: no correlation between gradients similarity and target labels means the model is weak for solving the task described by the input data. Overall, the optimal model needs to face a trade-off between data fit and model complexity, so that the kernels with high correlation with target labels (low fitting loss) and small complexity are the optimal ones.

Should model selection use gradients or features? Our analysis is in terms of the matrix Θ which depends on the network's gradients Eq. (4.3), not on its features (network's activations). This connections is clear when looking at the definition of linearized DNNs Eq. (2.39): the features of the linearized approximation are the gradients at initialization, not the activations. Nonetheless it is interesting to evaluate how much this fundamental difference affects our measure of generalization on unseen data. To measure such discrepancy we propose a different definition of alignment: Feature-Labels Correlation (LFC).

In Appendix B.1, we show that it suffices to use features (i.e. network activations) in Eq. (4.3) in place of the NTK matrix. Let $g^{(l)}(x_i)$ denote the feature vector (or activation) extracted from layer l of pre-trained network f after forward pass on image (see Eq. (2.5)). In analogy with the gradient similarity matrix Θ of Eq. (4.3), we define the feature similarity matrix Θ_F^l as follows:

$$\Theta_F^{(l)} := \sum_{i=l}^{L} g_w^{(i)}(\mathcal{X}) g_w^{(i)}(\mathcal{X})^T \tag{4.4}$$

4.3.3 Label-Feature Correlation

Instead of Θ, we can use the approximation $\Theta_F^{(L)}$ from Eq. (4.4) and define:

$$S_{\mathrm{LF}}(f_{w_0}, \mathcal{D}) := \frac{\left\langle \Theta_F^{(L)}, YY^T \right\rangle_F}{\left\|\Theta_F^{(l)}\right\|_F \|K_y\|_F} = \frac{\mathrm{Tr}\, Y^T \Theta_F^{(L)} Y}{\left\|\Theta_F^{(L)}\right\|_F \|K_y\|_F} \tag{4.5}$$

This score is higher if samples with the same labels have similar features extracted from the pre-trained network f_{w_0}. Indeed, it is possible to consider $\Theta_F^{(1)}$ too, which considers activations of the DNN across all hidden layers, we choose $\Theta_F^{(L)}$ for the sake of simplicity. In Appendix B.1 we proved the contribution of label-activation correlation decreases as we consider activations far from the last layer.

4.4 Experiments

Having established the problem of model selection for fine-tuning in Section 4.1, we now put our techniques to test. Section 4.4.2 describes our construction of model zoos with single-domain and multi-domain experts. In Section 4.4.3, we verify the advantage of fine-tuning using our model zoo with various target tasks. In Section 4.4.4, we compare our LFC, LGC model selection to previous work, and show that our method can select the optimal models to fine-tune from our model zoo (without performing fine-tuning).

4.4.1 Implementation

Which features and gradients to use? For LFC, we extract features from the layer before the fully-connected classification layer (for both Resnet-101 [He et al., 2016] and Densenet-169 [Huang et al., 2016] models in our model zoo of Section 4.4.2). We use these features to construct $\Theta_F^{(L)}$ and compute the LFC. For LGC, following [Mu et al., 2020], we use gradients corresponding to the last convolutional layer in the pre-trained network. For a large gradient vector, to perform fast computation of LGC, we take a random projection to $10K$ dimensions and compute the LGC score as done in Section 3.4.1 . This results in a trade-off between accuracy and computation for LGC.

Sampling of target task. Model selection is supposed to be an inexpensive pre-processing step before actual fine-tuning. To reduce its computation, following previous work of RSA [Dwivedi and Roig, 2019], we sample the training set of target dataset $\mathcal{D}$ and pick at most 25 images per class to compute our model selection scores. Note, test set is hidden from model selection. Our results show, this still allows us to select

	Pre-train	RESISC-45	Food-101	Logo 2k	G. Landmark	iNaturalist 2019	iMaterialist	ImageNet	Places-365
Densenet-169	×	93.61	82.38	64.58	82.28	71.34	66.59	76.40	55.47
	✓	96.34	87.82	76.78	84.89	73.65	67.57	-	55.58
Resnet-101	×	87.14	79.20	62.03	78.48	70.32	67.95	**77.54**	55.83
	✓	**96.53**	**87.95**	**78.52**	**85.64**	74.37	68.58	-	**56.08**
Reported Acc.	-	86.02	86.99	67.65	-	**75.40**	-	77.37	54.74

Table 4.1: **Model zoo of single-domain experts.** We train 30 models, Resnet-101 and Densenet-169, on 8 source datasets and measure the top-1 test accuracy: RESISC-45 [Cheng et al., 2017], Food-101 [Bossard et al., 2014], Logo 2k [Wang et al., 2020], G. Landmark [Noh et al., 2017], iNaturalist 2019 [Horn et al., 2017], iMaterialist [MalongTech, 2019], ImageNet [Deng et al., 2009] and Places-365 [Zhou et al., 2017]. We train our models starting with (✓) and without (×) Imagenet pre-training. For all datasets we have higher test accuracy with Resnet-101 (✓) than what is reported in the literature (last row), except for iNaturalist [Horn et al., 2017] by -1.03%. We order datasets from left to right by increasing dataset size, Nwpu-resisc45 [Cheng et al., 2017] has $25K$ training images while Places-365 [Zhou et al., 2017] has $1.8M$. We chose datasets that are publicly available and cover different domains.

Dataset	Single Domain	Shared	Multi-BN	Adapter
Nwpu-resisc45 [Cheng et al., 2017]	96.53	73.73	96.46	95.24
Food-101 [Bossard et al., 2014]	87.95	48.12	87.92	86.35
Logo 2k [Wang et al., 2020]	78.52	24.39	79.06	70.13
Goog. Land [Noh et al., 2017]	85.64	65.1	81.89	76.83
iNatural. [Horn et al., 2017]	74.37	37.6	65.2	63.04
iMaterial. [MalongTech, 2019]	68.58	42.15	63.27	57.5
Imagenet [Deng et al., 2009]	77.54	52.51	69.03	58.9
Places-365 [Zhou et al., 2017]	56.08	41.58	51.21	47.51

Table 4.2: **Multi-domain expert.** The top-1 test accuracy of multi-domain model – Multi-BN, Adapter – is comparable to single domain expert for small datasets (Nwpu-Resisc45, Food-101, Logo 2k), while the accuracy is lower on other large datasets. Multi-BN performs better than Shared, Adapter on all datasets and we use this as our multi-domain expert for fine-tuning and model selection.

models that obtain accuracy gain over Imagenet expert (Fig. 4.5), and we need few selections (< 7 for model zoo size 30) to select the optimal models (Fig. 4.7). We include additional implementation details of our model selection methods and other baselines: RSA [Dwivedi and Roig, 2019], Domain Similarity [Cui et al., 2018], LEEP [Nguyen et al., 2020], Feature Metrics [Ueno and Kondo, 2020] in Appendix B.3.

4.4.2 Model Zoo

We evaluate model selection and fine-tuning with both a model zoo of single-domain experts (i.e. models trained on single dataset) and a model zoo of multi-domain experts described below.

Source Datasets. Table 4.1 and Table B.1 lists the source datasets, i.e. the datasets

used for training our model zoo. We include publicly available large source datasets (from $25K$ to $1.8M$ training images) from different domains, e.g. Nwpu-resisc45 [Cheng et al., 2017] consists of aerial imagery, Food-101 [Bossard et al., 2014] and iNaturalist 2019 [Horn et al., 2017] consist of food, plant images, Places-365 [Zhou et al., 2017] and Google Landmark v2 [Noh et al., 2017] contain scene images. This allows us to maximize the coverage of our model zoo to different domains and enables more effective transfer when fine-tuning on different target tasks.

Model zoo of single-domain experts. We build a model zoo of a total of 30 models (Resnet-101 [He et al., 2016] and Densenet-169 [Huang et al., 2016]) trained on 8 large image classification datasets (i.e. source datasets). Since each model is trained on a single classification dataset (i.e. domain), we refer to these models as single-domain experts. This results in a model zoo, $\mathcal{F} = \{f^k\}_{k=1}^{30}$, to evaluate our model selection.

On each source dataset of Table 4.1, we train Resnet-101 and Densenet-169 models for 90 epochs, with the following hyper-parameters: initial learning rate of 0.1, with decay by 0.1× every 30 epochs, SGD with momentum of .9, weight decay of 10^{-4} and a batch size 512. We use the training script[1] from PyTorch [Paszke et al., 2019] library and ensure that our models are well-trained.

In Table 4.1, we show slightly higher top-1 test accuracy for our models trained on Imagenet [Deng et al., 2009] when compared to the PyTorch [Paszke et al., 2019] model zoo[2]. Our Resnet-101 model trained on Imagenet has +.17% top-1 test accuracy and our Densenet-169 model has +.4% top-1 test accuracy vs. PyTorch. On source datasets other than Imagenet, we train our models with (✓) and without (×) Imagenet pre-training. This allows us to study the effect of pre-training on a larger dataset when we fine-tune and perform model selection. Note that our Resnet-101 models with (✓) Imagenet pre-training have higher accuracy compared to that reported in the literature for all source datasets, except iNaturalist [Horn et al., 2017] by −1.03%.

Model zoo of multi-domain expert. We also train a Resnet-101 based multi-dataset (or multi-domain) [Rebuffi et al., 2018] model on the combination of all the 8 source datasets. Our multi-domain Resnet-101 expert, $f_{w_s,\{w_d\}_{d=1}^D}$, uses shared weights (or layers), i.e. w_s, across different domains (or datasets), and in addition it has some domain-specific parameters, i.e. $\{w_d\}_{d=1}^D$, for each domain. We have 8 source datasets or domains, so $D = 8$ in our benchmark. Note, for fine-tuning we can choose any one of the D domain-specific parameters to fine-tune. For a given multi-domain expert, this results in a model zoo of D models (one per domain) that we can fine-tune, $\mathcal{F} =$

[1] https://bit.ly/38NMvyu
[2] https://bit.ly/35vZpPE

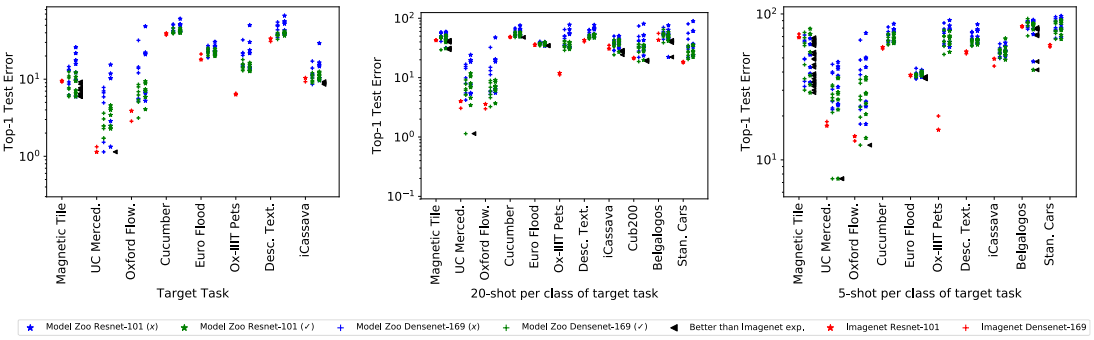


Figure 4.3: **Fine-tuning with model zoo of single-domain experts.** We plot top-1 test error (vertical axis) for fine-tuning with different single domain models in our model zoo. For every target task (on horizontal axis), we have 4 columns of markers from left to right: 1) Imagenet experts in red, 2) Densenet-169 experts with pre-train (✓) in green and without pre-train (×) in blue, 3) Resnet-101 experts with pre-train (✓) in green and without pre-train (×) in blue, 4) We use "**black** ←" to highlight models that perform better than imagenet expert (i.e. lower error than first column of Imagenet expert (red) per task). Our observations are the following: *i*) For full target task, we observe better accuracy than Imagenet expert for Magnetic Tile Defects, UC Merced Land Use and iCassava (see **black** ←). For 20 and 5-shot per class sampling of target task, with the model zoo we outperform Imagenet expert on more datasets, see Oxford Flowers 102, European Flood Depth, Belga Logos and Cub200. The accuracy gain over Imagenet expert fine-tuning is obtained only for few models, e.g. only one expert for UC Merced Land Use outperforms the Imagenet expert baseline. Hence, brute-force fine-tuning with model zoo leads to wasteful computation since many models would not beat Imagenet expert. This highlights the necessity of a reliable model selection criterion to only pick the most promising models for fine-tuning. Figure is best viewed in high-resolution.

$\{f_{w_s,w_1}, f_{w_s,w_2}, \cdots, f_{w_s,w_D}\}$.

We experiment with a few different variants of domain-specific parameters – *i*) **Shared**: The domain-specific parameters are also shared, therefore we simply train a Resnet-101 on all datasets, *ii*) **Multi-BN**: We replace each batch norm in Resnet-101 architecture with a domain-specific batch norm. Note, for a batch norm layer we replace running means, scale and bias parameters, *iii*) **Adapter:** We use the domain-specific parallel residual adapters [Rebuffi et al., 2018] within the Resnet-101 architecture. Our training hyper-parameters for the multi-domain expert are the same as our single-domain expert. The only change is that for every epoch we sample at most $100K$ training images (with replacement if $100K$ exceeds dataset size) from each dataset to balance training between different datasets and to keep the training time tractable. As

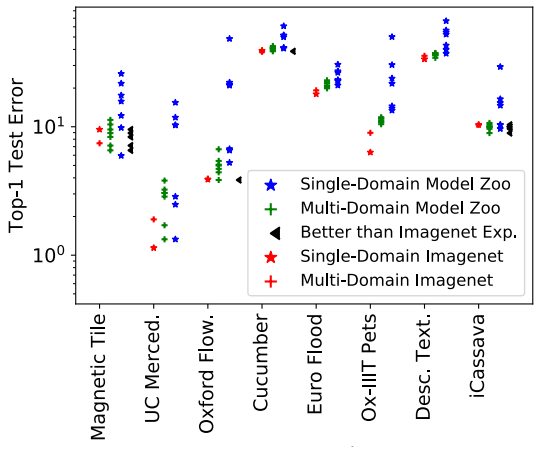
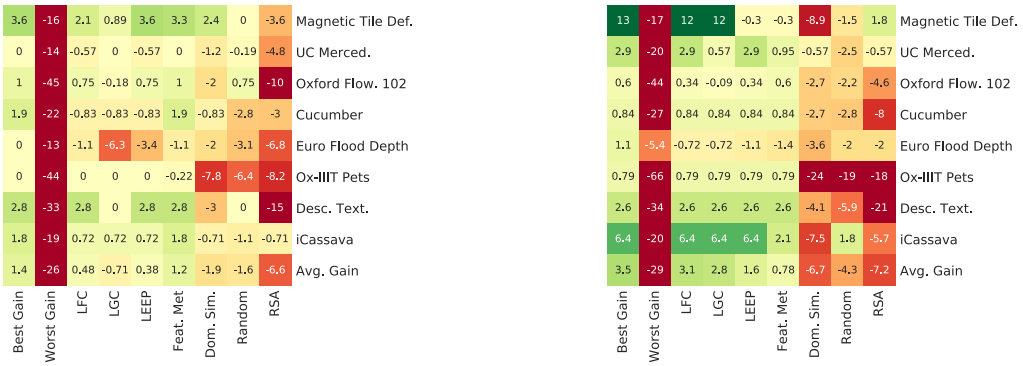


Figure 4.4: **Fine-tuning with the multi-domain expert for the full target task.** We use the same notation as Fig. 4.3. For every target task (horizontal axis), we have 4 columns corresponding to fine-tuning different models from left to right: 1) Imagenet single and multi-domain expert in red, 2) Fine-tuning with different domains of multi-domain expert in green and 3) Single-domain Resnet-101 experts in blue, 4) We highlight multi-domain experts (green) that obtain lower error than Imagenet (red) single domain with **black** ←. Note, since our multi-domain expert is Resnet-101 based, we only use all our Resnet-101 experts for for fair comparison. Our observations are: *i*) We see gains over Imagenet expert (both single and multi-domain) by fine-tuning some (not all) domains of the multi-domain expert, for Magentic Tile Defects, Oxford Flowers 102, Cucumber and iCassava target tasks. Therefore, it is important to pick the correct domain from the multi-domain expert for fine-tuning. *ii*) We observe the variance in error is smaller for fine-tuning with different domains of multi-domain experts, possibly due to shared parameters across domains, *iii*) Finally in some cases, e.g. Oxford Flowers 102 and iCassava, our multi-domain experts outperform both all single domain and Imagenet experts. Figure is best viewed in high-resolution.

we show in Table 4.2, **Multi-BN** model outperforms other multi-domain models and we use it in our subsequent fine-tuning (Section 4.4.3) and model selection (Section 4.4.4) experiments.

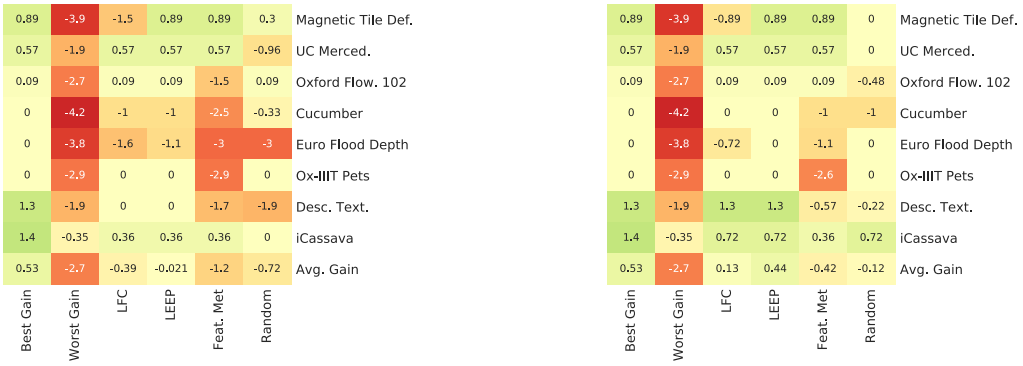
4.4.3 Fine-tuning on Target Tasks

Target Tasks. We use various target tasks (Table B.1) to study transfer learning from our model zoo (Section 4.4.2): Cucumber [dataset, 2016], Describable Textures [Cimpoi et al., 2014], Magnetic Tile Defects [Huang et al., 2018], iCassava [Mwebaze et al., 2019], Oxford Flowers 102 [Nilsback and Zisserman, 2008], Oxford-IIIT Pets [Parkhi et al., 2012], European Flood Depth [Barz et al., 2019], UC Merced Land Use [Yang



(a) Full Dataset (b) 20-Shot per class

Figure 4.5: **Model selection among single-domain experts.** The heatmap shows the accuracy gain over Resnet-101 Imagenet expert obtained by fine-tuning the top-3 selected models for different model selection methods (column) on our target tasks (row). Higher (in green) values of gain are better. Note, for every method we fine-tune all the top-3 selected models (with same hyper-parameters as Section 4.4.3) and pick the one with the highest accuracy. Model selection performs better than "Worst Gain" and random selection. On average, LFC, LGC and LEEP [Nguyen et al., 2020] outperform Domain Similarity [Cui et al., 2018], RSA [Dwivedi and Roig, 2019]. Feature Metrics [Ueno and Kondo, 2020] performs better than LFC, LEEP in high-data regime, but under-performs in the low-data regime.



(a) Top-1 Selection (b) Top-3 Selection

Figure 4.6: **Model Selection with multi-domain expert.** The heatmap shows accuracy gain obtained by fine-tuning selected domain over fine-tuning Imagenet domain from the multi-domain expert. We show results for top-1 and top-3 selections. LFC, LEEP [Nguyen et al., 2020] are close to the best gain and they outperform Feature Metrics [Ueno and Kondo, 2020] and *Random*.

and Newsam, 2010]. For few-shot, due to lesser compute needed, we use additional target tasks: CUB-200 [Welinder et al., 2010], Stanford Cars [Krause et al., 2013] and Belga Logos [Joly and Buisson, 2009]. Note, while some target tasks have domain overlap with our source datasets, e.g. aerial images of UC Merced Land Use [Yang and Newsam, 2010], other tasks do not have this overlap, e.g. defect images in Magnetic Tile Defects [Huang et al., 2018], texture images in Describable Textures [Cimpoi et al., 2014].

Fine-tuning with single-domain experts in model zoo. For fine-tuning, Imagenet pre-training is a standard technique. Note, most deep learning frameworks, e.g. PyTorch, MxNet/Gluon[3], just have the Imagenet pre-trained models for different architectures in their model zoo. Fig. 4.3 shows the top-1 test error obtained by fine-tuning single-domain experts in our model zoo vs. Imagenet expert.

Our fine-tuning hyper-parameters are: 30 epochs, weight decay of 10^{-4}, SGD with Nesterov momentum 0.9, batch size of 32 and learning rate decay by $0.1\times$ at 15 and 25 epochs. We observe that the most important hyper-parameter for test accuracy is the initial learning rate η, so for each fine-tuning we try $\eta = 0.01, 0.005, 0.001$ and report the best top-1 test accuracy.

Does fine-tuning with model zoo perform better than fine-tuning a Imagenet expert? While fine-tuning an Imagenet pre-trained model is standard and works well on most target tasks, we show that by fine-tuning models of a large model-zoo we can indeed obtain a lower test error on some target tasks (see models highlighted by **black** $\leftarrow$ in Fig. 4.3). The reduction in error is more pronounced in the low-data regime. Therefore, we establish that maintaining a model zoo of models trained on different datasets is helpful to transfer to a diverse set of target tasks with different amounts of training data.

We demonstrate gains in the low-data regime by training on a smaller subset of the target task, with only 20, 5 samples per class in Fig. 4.3 (i.e., we train in a 20-shot and 5-shot setting). In few-shot cases we still test on the full test set.

Fine-tuning with multi-domain expert. In Section 4.4.2, we show that fine-tuning can be done by choosing different domain-specific parameters within the multi-domain expert for fine-tuning. In Fig. 4.4, we fine-tune the multi-domain expert, i.e. Multi-BN of Table 4.2, on our target tasks by choosing different domain-specific parameters to fine-tune. Similar to Fig. 4.3, we show the accuracy gain obtained by fine-tuning multi-domain expert with respect to fine-tuning the standard Resnet-101 pre-trained on Imagenet. We observe that selecting the correct domain to fine-tune, i.e. the cor-

[3] https://gluon-cv.mxnet.io/api/model_zoo.html

Shots	Brute-force	Fine-tuning top-3 models				
		LFC	**LGC**	**LEEP**	**Feat. Met.**	**Dom. Sim.**
Full	48.17×	5.15×	3.89×	5.01×	6.02×	4.87×
20-shot	41.67×	4.35×	3.40×	3.85×	4.86×	4.11×

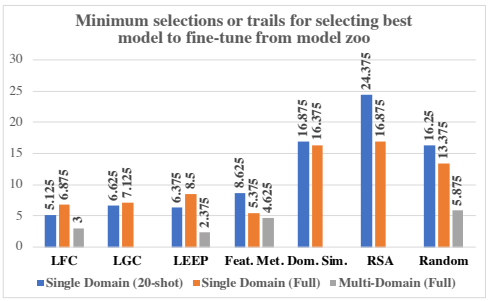
Table 4.3: **Computation cost of model selection and fine-tuning.** We measure the average run-time for all our target tasks (of Fig. 4.3) of: Brute-force fine-tuning and Fine-tuning with 3 models chosen by model selection (Fig. 4.5). We divide the run-time by the run-time of fine-tuning a Resnet-101 Imagenet expert. For the single domain model zoo, brute-force fine-tuning of all 30 experts requires 40× more computation than fine-tuning Imagenet Resnet-101 expert. Note, Densenet-169 models in our model zoo need more computation to fine-tune than Resnet-101, therefore the gain is $> 30\times$ for our model zoo of size 30. With model selection, we can fine-tune with selected models in only $3-6\times$ the computation. LFC and LEEP compute model selection scores for 30 models in our zoo with $< 1\times$ the computation of fine-tuning Imagenet Resnet-101 expert. LGC model selection is expensive due to backward passes and large dimension of the gradient vector. However, our LFC approximation to LGC is good at selecting models (Fig. 4.5) and fast.

rect w_d, where $d \in \{1, 2, \cdots, D\}$ from multi-domain model zoo $\mathcal{F} = \{f_{w_s,w_d}\}_{d=1}^{D}$, is important to obtain high fine-tuning test accuracy on the target task. In Section 4.4.4, we show that model selection algorithms help in selecting the optimal domain-specific parameters for fine-tuning our multi-domain model zoo.

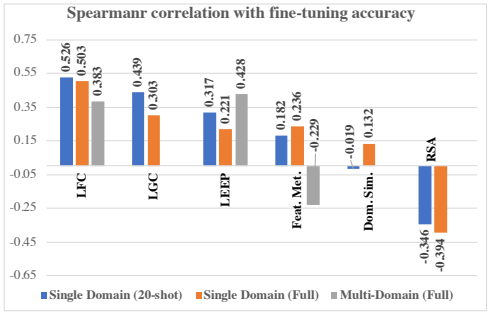
We also observe that fine-tuning with our multi-domain expert improves over the fine-tuning of single-domain model zoo for some tasks, e.g. iCassava: +1.4% accuracy gain with multi-domain expert compared to +.72% accuracy gain with single domain model expert over Imagenet expert. However, the comparison between single domain and multi-domain experts and their transfer properties is not the focus of our research and we refer the reader to [Mallya and Lazebnik, 2018, Rebuffi et al., 2017, Rebuffi et al., 2018].

4.4.4 Model Selection

In Section 4.4.3, using our benchmark we find that fine-tuning with a model zoo, both single-domain and multi-domain domain, improves the test accuracy on the target tasks. Now, we demonstrate that using a model selection algorithm we can select the best model or domain-specific parameters from our model zoos with only a few selections or trials.



(a) Selections for best model



(b) Spearman correlation of expert ranking using model selection scores to actual ranking using fine-tuning accuracy

Figure 4.7: **LFC and LGC comparison with SOTA** In Fig. 4.1, we measure the number of trials to select the best model, i.e. highest accuracy, from the model zoo. LFC, LGC and LEEP [Nguyen et al., 2020] require fewer trials than Domain Similarity [Cui et al., 2018], RSA [Dwivedi and Roig, 2019] and *Random* selection baselines. In Fig. 4.2, we show that model selection scores of LFC obtain the highest Spearman's ranking correlation to the actual fine-tuning accuracy compared to other model selection methods.

Model Selection Algorithms. We use the following scores, S, for our model selection methods: LGC (see S_{LG} defined in Eq. (4.2)), LFC (see S_{LF} defined in Eq. (4.5)), which we introduce in Section 4.3. We compare against alternative measures of model selection and/or task similarity proposed in the literature: Domain similarity [Cui et al., 2018], Feature metrics [Ueno and Kondo, 2020], LEEP [Nguyen et al., 2020] and RSA [Dwivedi and Roig, 2019]. Finally, we compare with a simple baseline: *Random* which selects models randomly for fine-tuning.

Model selection with single-domain model zoo. In Fig. 4.5, we select the top-3 experts (i.e. 3 highest model selection scores) for each model selection method for fine-tuning. We do this for all the target tasks (row) using each model selection method (column). We use the maximum of fine-tuning test accuracy obtained by 3 selected models to compute accuracy gain with respect to fine-tuning with Resnet-101 Imagenet expert. Ideally, we want the accuracy gain with the model selection method to be high and equal to the "Best Gain" possible for the target task. As seen in Fig. 4.5: LFC, LGC and LEEP obtain high accuracy gain with just 3 selections in both full dataset and 20-shot per class setting. They outperform random selection.

Model selection with multi-domain expert. For our multi-domain expert (Section 4.4.2), we use model selection to select the domain-specific parameters to fine-tune for every model selection method. We compute the accuracy gain for fine-tuning using

selected domains vs. fine-tuning Imagenet parameters in the multi-domain expert. Our results in Fig. 4.6, show that LFC and LEEP [Nguyen et al., 2020] obtain higher accuracy gain compared to Feature Metrics [Ueno and Kondo, 2020] and *Random* selection.

Is fine-tuning with model selection faster than brute-force? In Table 4.3, we show that brute-force fine-tuning is expensive. We can save computation by performing model selection using LFC and LEEP and fine-tuning only the selected top-3 models.

How many trials to select the model with best fine-tuning accuracy? In Fig. 4.7, we measure the average of selections or trials, aggregated across target tasks, required to select the best model for fine-tuning from the model zoo. LGC, LFC and LEEP [Nguyen et al., 2020] methods can select the best model in < 7 trials for our single domain model zoo of 30 experts and in < 3 trials for the multi-domain model zoo with 8 domain experts.

Are model selection scores a good proxy for fine-tuning accuracy? In Fig. 4.7, we show our LFC scores have the highest Spearman's ranking correlation to the actual fine-tuning accuracy for different experts. Note, we average the correlation for all our target tasks. Our LFC score is a good proxy for ranking by fine-tuning accuracy and it can allow us to select (or reject) models for fine-tuning.

4.5 Bibliographical Notes

Fine-tuning. The exact role of pre-training and fine-tuning in deep learning is still debated. [He et al., 2019] show that, for *object detection*, the accuracy of a pre-trained model can be matched by simply training a network from scratch but for longer. However, they notice that the pre-trained model is more robust to different hyper-parameters and outperforms training from scratch in the low-data regime. On the other hand, in *fine-grained visual classification*, [Li et al., 2020] show that even after hyper-parameter optimization and with longer training, models pre-trained on similar tasks can significantly outperform both Imagenet pre-training and training from scratch. [Achille et al., 2019], [Cui et al., 2018] study task similarity and also report improvement in performance by using the right pre-training. [Zoph et al., 2020] show that while pre-training is useful in low-data regime, self-training outperforms pre-training in high-data regime. Most of the above work, [Achille et al., 2019, Cui et al., 2018, Li et al., 2020] draws inferences of transfer learning by using Imagenet [Deng et al., 2009] or iNaturalist [Horn et al., 2017] experts.

Model Selection. Empirical evidence [Achille et al., 2019, Li et al., 2020, Zamir et al., 2018] and theory [Achille et al., 2019] suggests that effectiveness of fine-tuning

relates to a notion of distance between tasks. Taskonomy [Zamir et al., 2018] defines a distance between learning tasks *a-posteriori*, that is, by looking at the fine-tuning accuracy during transfer learning. However, for predicting the best pre-training without performing fine-tuning, an *a-priori* approach is best. [Achille et al., 2019, Achille et al., 2019] introduce a fixed-dimensional "task embedding" to encode distance between tasks. [Cui et al., 2018] propose a Domain Similarity measure, which entails using the Earth Mover Distance (EMD) between source and target features. LEEP [Nguyen et al., 2020, Tran et al., 2019] looks at the conditional cross-entropy between the output of the pre-trained model and the target labels. RSA [Dwivedi and Roig, 2019] compares representation dissimilarity matrices of features from pre-trained model and a small network trained on target task for model selection.

Few-shot. Interestingly, while pre-training has a higher impact in the few-shot regime, there is only a handful of papers that experiment with it [Dvornik et al., 2020, Goyal et al., 2019, Triantafillou et al., 2020]. This could be due to over-fitting of the current literature on standard benchmarks that have a restricted scope.

4.6 Discussions and conclusions

Fine-tuning using model zoo is a simple method to boost accuracy of a general AutoML system on different data domains (e.g. images and time series). In this chapter we propose a general method to perform model selection within a model zoo composed of DNNs, to do so we exploit the connection between fine-tuning training dynamics and the NTK for overparametrized feedforward DNNs. Our method does not depend on a particular application domain (say image processing or time series analysis) or loss function (e.g. squared loss or cross-entropy loss) and can be applied so long as feedforward DNNs are used (e.g. fully-connected or convolutional), for which the NTK theory has been developed. In particular, we focus on a computer vision application and show that while a model zoo may have modest gains in the high-data regime, it outperforms Imagenet experts networks in the low-data regime. In such a scenario, we show our model selection saves the cost of brute-force fine-tuning and makes model zoos viable in practice. Our LGC model selection criterion, and particularly its approximation LFC, can find the best models from which to fine-tune without requiring an expensive brute-force search (Table 4.3). Compared to other model selection criteria, our LFC score can select the best model to fine-tune in fewer selections, and it shows the highest ranking correlation to the fine-tuning test accuracy among all model selection methods we tested (Fig. 4.7).

III

Inductive Bias and Regularization

5

Inductive Bias and Regularization

In this chapter we shall introduce some architectural choices and regularization schemes which are specifically designed to improve learnability and generalization of DNNs. To begin with, we shall start by asking: why do we need DNNs in the first place? It is well-known that a feedforward fully-connected Neural Network with a sufficiently large single hidden layer is an universal approximator of Borel measurable functions [Cybenko, 1989, Hornik et al., 1989]. So that, virtually, there is no necessity to use DNNs in place of their "simpler" shallow counterparts. Nonetheless, even for the simpler model class of shallow networks no result on the learnability of the best set of parameters has been proved. So that there is no guarantee that a model learnt by solving ERM can generalize well on unseen data. Despite this daunting observation which seems to jeopardize Neural Networks learning, recent years have been a clear proof of the gap between theory and practice. Recently, many Neural Networks-based algorithms achieved SOTA results in different fields such as: Compute Vision [Krizhevsky et al., 2012, He et al., 2016, Dosovitskiy et al., 2021], Natural Language Processing [Devlin et al., 2019, Vaswani et al., 2017] and Time Series analysis [Bai et al., 2018, Zancato et al., 2022, Zancato and Chiuso, 2021] to name but a few. Recent trend in Neural Networks design is to exploit DNNs in place of shallow ones. The general idea is that as the network becomes deeper it becomes more efficient in representing complex functions w.r.t. to shallow networks when the same number of parameters is used [Montufar et al., 2014]. In turn, this implies the optimization problem needed to be solved with DNNs is somewhat "easier". No general theory has been proposed to justify the optimization landscape of DNNs is more amenable to be optimized by methods applied in practice (based on first order optimization e.g. SGD). Nonetheless it is well known that very deep over-parametrized models have smoother loss landscape w.r.t. to more

shallow ones [He et al., 2016, Li et al., 2018] and thus they suffer less from typical first-order optimization pitfalls (e.g. spurious stationary points [Bottou et al., 2018, Li et al., 2018]). Moreover, it is well-known that particular types of architectures are better suited to solve specific tasks than general purpose fully-connected DNNs. For example many, if not all, of the most remarkable successes in recent years in Computer Vision have been achieved by means of specialized architectures such as: Convolutional Neural Networks [Krizhevsky et al., 2012, He et al., 2016] or Transformers [Dosovitskiy et al., 2021]. A similar observation holds for other domains such as Time Series prediction [Bai et al., 2018, Oreshkin et al., 2019, Zancato et al., 2020] and Natural Language Processing [Vaswani et al., 2017]. So that it is clear that building highly specialized architecture is key to obtain state of the art results when applying Deep Learning in practice. In addition, the design of suitable regularization schemes proved to be essential for DNNs generalization. Regularization can be considered either explicitly, by adding suitable penalty terms on a standard training loss [Bansal et al., 2018, Golatkar et al., 2019], or implicitly, by the choice of optimization algorithm (e.g. SGD [Chaudhari and Soatto, 2018, Achille and Soatto, 2017]).

In the following, we shall compare fully-connected with Convolutional Neural Networks for image classification and Recurrent Neural Networks with Temporal Convolutional Neural Networks for time series prediction. Moreover we shall propose a novel inductive bias and regularization for both fully-connected DNNs and TCNs for time series data [Zancato and Chiuso, 2021]. Both the inductive bias and the regularization are specifically designed to exploit domain knowledge (i.e. time series) and allow to automatically perform model selection (automatic complexity selection) based solely on available data [Zancato and Chiuso, 2021, Zancato et al., 2022].

5.1 Architecture design principles

5.1.1 Fully vs Convolutional

We now describe the main ideas behind the huge success of convolutional layers that we introduced in Section 1.2.3: *sparse interactions*, *parameters sharing* and *equivariant representations*.

Sparse interactions. The main difference between a convolution (represented by a matrix multiplication) and a matrix multiplication between two dense matrices is *locality*. While for a fully-connected layer each input interacts with every output, for convolutional layers only a subset of the input directly interacts with the output. This is true since it is customary to choose a small convolutional kernel: smaller (several

orders of magnitude) than the input. Despite the local receptive field of each layer, by stacking several convolutional layers on top of each other it is possible to increase the receptive field even using small kernels, this idea is standard an is used in many different architectures and domains (e.g. images [Krizhevsky et al., 2012, He et al., 2016] and time series [Bai et al., 2018]). This allows the network to efficiently describe complicated interactions between many variables while maintaining the complexity of each of its building blocks small.

Parameters sharing. As a consequence of the circulant structure of convolutions (Section 1.2.3) the weights of a convolutional layer are tied together: each output is computed by means of the same kernel values. This is not true for a general fully-connected layer in which each output is computed from its inputs using different recombination weights. Moreover the memory cost of storing the kernel matrix for a fully-connected layer is $w_{\text{input}} \times h_{\text{input}}$ (where w_{input} and h_{input} are the dimension of the input and output respectively) while for convolutional layers the memory cost is p where p is the length of the kernel (user's choice, typically $p << w_{\text{input}} \times h_{\text{input}}$). This observation is crucial for Machine Learning since the number of parameters to be learnt is much smaller for convolutional layers.

Equivariance. This is one of the most important properties of convolutions. Equivariance makes convolutions very well suited for image and time series related tasks. In particular any convolutional layer is equivariant to translations, which means that if the input is translated, then the output is translated of the same amount too. Unfortunately, convolutions are nor equivariant to other typical transformations which we known are important (e.g. in image related tasks changes in scale or rotations). To overcome this limitation other layers are typically used, e.g. pooling.

Remark 5.1. Convolutions provide a natural framework to work with inputs of variable sizes.

Pooling. A typical convolutional layer of modern CNNs is built on three stages: a filter bank which performs parallel convolutions, a non-linear activation (typically ReLU) applied component-wise on the output feature of the convolutions and a *pooling function*. The main idea for a pooling function is to aggregate the statistics of the activation outputs creating a summary statistics. One of the most used pooling functions is the so-called *max pooling*, it outputs the maximum of the non-linearly activated features within a rectangular neighborhood. In this way, after the pooling layer the size of features extracted on a given input is reduced. Interestingly, max pooling can also be applied feature-wise (i.e. layer-wise) so that the optimal model can learn which transformations to become invariant to (e.g. rotations [Goodfellow et al., 2016]).

5.1.2 Recurrent vs Convolutional

We now turn our attention to specialized DNNs designed to deal with time series data. We shall describe the main differences between Recurrent Neural Networks and Temporal Convolutional Neural Networks (TCNs). TCNs are built exploiting similar design principles to the ones we described in the case of CNNs for images (Section 5.1.1) but are used to deal with time series data.

Recurrent Neural Networks

RNNs are designed to process data sequentially, in general they can be applied to sequences of variable lengths (as convolution-based models do). The main representation of RNNs closely resemble the state space representation of a dynamical non-linear system Eq. (6.1). In particular, the non linear recurrent map implemented by a RNN is described as: $h(t) = f_w(h(t-1), x(t))$, where h is a the instantaneous representation of the RNN (the so-called **state** in systems theory parlance) and $x(t)$ is the input at current time t. By unfolding the computational graph over time it is possible to compute the hidden representation at each instant (state evolution). The state h is supposed to be a non-linear summary statistics which contains the higher amount of "information" w.r.t. input signal while at the same time being minimal in the sense that all the nuisances factors not relevant for the task to be solved do not affect h. How to properly induce sufficiency and minimality of the hidden representation is an active area of research [Su et al., 2019, Allen-Zhu et al., 2018]. Then, the output of a RNN can be obtained by applying a static (non-recurrent) map on the hidden representation at each time instant t.

Note that the parametrization applied across time is not changing (w does not depend on the time index) so that, similarly to CNNs, we can consider RNNs to share parameters across inputs. Sequential parameters sharing reduces the dimension of the search space while optimizing the RNNs but, differently from CNNs, does not allow to parallelize computations. This is typical for sequential models where, to compute the representation at time $t+1$, the representation at time t must have been computed. This makes RNNs' cost of training and deployment high in general [Goodfellow et al., 2016].

One prominent challenge in learning RNNs is the so-called *long-term dependencies* limitation. The main idea is that RNNs might find difficult to model long-term dependencies so that their output sensitivity w.r.t. to far in the past inputs is small. This is mainly due to the effects of vanishing gradients which have been demonstrated to

highly impact RNNs in practice [Goodfellow et al., 2016]. In fact, assuming the non-linear model is a contractive map (stable) the relative weights of gradients associated to long-term interaction is exponentially smaller than close (in time) ones. Note that a non-stable non-linear map f_w would cause backpropagated gradients to explode (so-called exploding gradients phenomenon). Many architectures and design schemes have been introduced to solve such issues for sequential models, some remarkable examples are: Long Short-Term Memory Networks [Hochreiter and Schmidhuber, 1997], Gated Recurrent Unit Networks [Chung et al., 2014] and Echo State Networks [Jaeger, 2003].

Temporal Convolutional Neural Networks

Despite RNNs being synonymous with sequence modeling, in recent years Temporal Convolutional Networks have proved to outperform both LSTM and GRU recurrent models in a variety of conditions [Bai et al., 2018, Munir et al., 2019]. One of the main reasons for the performance improvement is the longer effective memory of TCNs than RNNs. So that, provided overfitting spurious long-term dependencies is avoided (say by means of regularization [Bai et al., 2018, Zancato et al., 2022]), TCN models can be used to model more complex time series with non trivial long term dependencies. Moreover, thanks to convolutions, TCNs can be highly parallelizable (Section 1.2.3 and Section 5.1.1), this makes them especially suited for complex time series data.

The basic building blocks of TCNs (see Fig. 5.1) are *causal 1-D convolutions*, so that there is no information "leakage" from future to past [Bai et al., 2018]. Convolutions guarantee that the input signal can be of any length (Section 1.2.3 and Section 5.1.1) as customary for convolutional networks. Moreover, it is possible to increase TCNs' receptive field by simply stacking more convolutional layers and by using *dilated convolutions* [Bai et al., 2018].

We shall now briefly describe both causal convolutions and dilated convolutions since the precise interaction of hidden representations across the layers of a deep TCN are fundamental for incorporating the novel fading memory inductive bias and automatic complexity selection we present in Section 5.2 and Section 5.3.

Causal convolutions. The main idea in causal processing for time series is the constraint of no information "leakage" of future values. More precisely, given the sequence of data $\{x(i)\}$ with $i \in [T]$ and a sequence of target values that must be predicted $\{y(i)\}$ with $i \in [T]$, the causal predictor of $y(t)$ must depend only on $\{x(i)\}$ with $i \in [t]$. Such a requirement is typical in autoregressive prediction, in which a predictor is built only using past values of the time series. 1-D causal convolutions are defined as standard convolutions Eq. (1.6) with the requirement that the 1-D kernel (impulse response) is

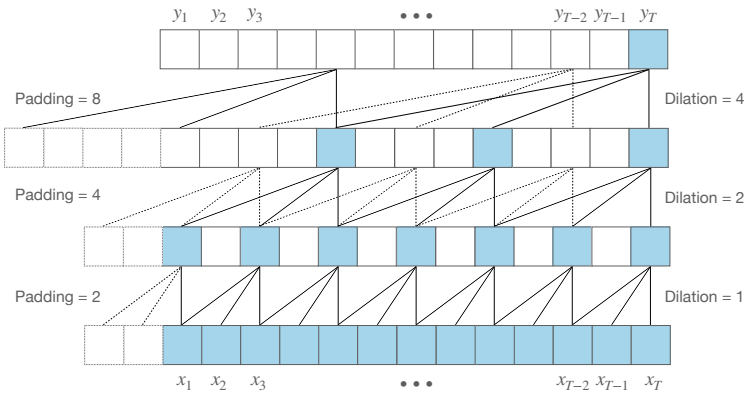


Figure 5.1: **Temporal Convolutional Network Architecture (TCN)**. Two hidden layers TCN with dilations $d = 1, 2, 4$ (see [Bai et al., 2018]).

zero for negative time indeces, this guarantees that the output of any causal convolution does not depend on future values of the input time series.

Equally sized hidden representations. TCNs are designed to receive a T dimensional input and output a T dimensional vector. Moreover, the dimension of each hidden layer is constrained to be equal to the input and output dimensions. This requires that for each 1-D convolutional layer, zero pad of length $(|\omega|-1)$ is used (where $|\omega|$ is the length of the convolutional kernel Section 1.2.3) [Bai et al., 2018]. The major disadvantage of this simple design is that to achieve a long effective memory size very deep architectures or long kernels are required (i.e. in this case the effective memory scales proportional to the number of hidden layers or kernel sizes). The solution to this issue is to use (causal) dilated convolutions.

Dilated Convolutions. Dilated convolutions are used to increase the receptive fields of TCNs, so that as the number of hidden layers increases the receptive field of the TCN increases exponentially [Bai et al., 2018].

Definition 5.1 (Dilated convolution)**.** Given a discrete (infinite) sequence $x(t) \in \mathbb{R}$ and a finite dimensional filter parametrized ω of length $|\omega|$, we denote the dilated convolution between the sequence x and ω with $(x *_d \omega)(t) \in \mathbb{R}$ and define it as:

$$g(t) := (x *_d \omega)(t) := \sum_{i=0}^{|\omega|-1} \omega(i) x(t - di)$$

where d is the dilation factor.

By using dilated convolutions it is possible to increase the receptive field either by

increasing the filter length $|\omega|$ or by increasing the dilation factor d. In standard TCNs at the j-th hidden layer the dilation factor of dilated convolutional layers is chosen as $O(2^j)$ so that as the architecture gets deeper the receptive field increases exponentially.

Residual Connections. Since large receptive fields are guaranteed only for deep models, it is important that TCNs do not suffer from typical optimization issues related to very deep architecture [He et al., 2016, Ioffe and Szegedy, 2015, Srivastava et al., 2014]. It is customary to stabilize DNNs training dynamics by means of residual connections [He et al., 2016], we refer to [Bai et al., 2018] for more details on the implementation.

5.2 Fading Memory Inductive Bias

We now describe architectural choices (inductive biases) to handle time series data both with fully-connected neural networks [Zancato and Chiuso, 2021] and with TCN [Zancato et al., 2022]. As we showed in Section 5.1.2 long-term dependencies are often very important in modeling time-series data so that models not capable to cope with far in the past input data are usually not viable models in practice. Nonetheless, too expressive models, e.g. able to handle very long-term dependencies, are often subject to overfitting. This highlights a clear design trade-off. How to choose the proper receptive field? Typically, the *relevant past* of time series is not known during the model design phase. This makes the optimal model complexity design impossible and highly prone either to overfit or underfit. In this section and in Section 5.3 we shall describe a method which enables automatic complexity selection on time series models, so that the optimal causal predictor is automatically chosen with a proper receptive field (of a similar size of the most relevant past of the time series).

5.2.1 Fading Memory Property

To begin with, we shall introduce a general property common to most non periodic time series: fading memory [Matthews and Moschytz, 1994, Pillonetto et al., 2011].

The *fading memory* property can be informally described by saying that the effect of past values $y(s)$, $s \leq t$ on $y(t)$ become negligible (tends to zero asymptotically) as $|t - s|$ goes to infinity. This property guarantees that the time series behaviour can be uniformly well approximated on compact sets. Hence the universal approximation properties of Neural Networks (see [Cybenko, 1989]) suggests that they can be seen as natural candidates to model this class of time series. For the sake of simplicity, in this section, we shall consider scalar time series that does not depend on other contextual

time series. The extension to multi-variate time series is straightforward.

We define the infinite past of the time series y at time t as $y_-^t := \{y(i)\}$ with $i \leq t$ and write the time series $y(t)$ in innovation form as:

$$y(t) = f_0(y_-^{t-1}) + e(t) \tag{5.1}$$

where $e(t)$ is the *innovation sequence* which captures all the information not captured by the infinite past of $y(t)$. The fading memory property guarantees that $f_0(y_-^{t-1}) \approx f_0(y_{t-p}^{t-1})$ where p is the *relevant past* and as p increases the approximation error decreases ($p \to \infty \implies |f_0(y_-^{t-1}) - f_0(y_{t-p}^{t-1})| \to 0$). So that we shall assume the actual time series only depends upon a finite (yet arbitrarily) long window of past data y_{t-p}^{t-1}. In the following sections we shall show how to equip both fully-connected Neural Networks and TCNs with an inductive bias based on the fading memory assumption.

5.2.2 Fading Memory for Fully-Connected DNNs

In this section we describe a fully-connected DNN specifically designed to handle time series, and in particular encode the fading memory assumption. The main idea is to exploit a block partitioned architecture so that each block attends to a specific finite window of the input time series.

We define a window of length p of past values w.r.t. t of the time series $y(t)$ as: $y_{t-p}^{t-1} \in \mathbb{R}^p$. The fading memory assumption implies the contribution of the i-th window y_{i-p}^{i-1} on the decomposition Eq. (5.1) goes to zero as $i \to -\infty$ (i.e. we look at far in the past windows of data). So that, if we want to successfully endow a non-linear model with the fading memory property we should guarantee that the sensitivity of its output w.r.t. to $y(i)$ goes to zero as i decreases. Imposing such a constraint for general fully-connected DNNs is not straightforward without further assumptions.

By exploiting an analogy with linear systems [Ramírez-Chavarría and Schoukens, 2021] shows that imposing a direct regularization on the sensitivity of the output of a non-linear model so that $\left|\frac{\partial f(y_-^{t-1})}{\partial y(i)}\right| \to 0$ as $i \to -\infty$ endows a general non-linear model structure with a prior over functions which fosters the fading memory property. We highlight that this is true up to first order approximation, in fact, even if $\left|\frac{\partial f(y_-^{t-1})}{\partial y(i)}\right| = 0$ for some indices this does not implies that the input $y(i)$ does not affect the output of the non-linear model. For example, it can happen that the paths connecting other inputs to outputs are affected by $y(i)$ (e.g. through second order interactions). To guarantee that the input $y(i)$ does not affect the outputs (or its effect is small) we argue higher

order interactions between input and outputs must be considered. For example, it is possible to take into account second order information by simply considering the hessian $\left(\frac{\partial^2 f(y_-^{t-1})}{\partial y(i)\partial y(j)}\right)_{i,j}$. In particular, one should penalize the hessian to be entry-wise as close as possible to zero. Clearly the limit case in which the hessian is zero describes linear systems for which the first order condition is sufficient to guarantee the fading memory property.

In general its is not easy to build a DNN whose hessian can be easily accessible. For that, we explicitly constraint the hessian to have a block structure by employing a block structured DNN. So that we can enforce the fading memory property by applying a similar condition to the one proposed in [Ramírez-Chavarría and Schoukens, 2021]. We consider the following non-linear model structure:

$$f_{\theta,w}\left(y_{t-p-n_B+1}^{t-1}\right) := \sum_{i=0}^{n_B-1} \theta_i f_{W_i}\left(y_{t-i-p}^{t-i-1}\right) \tag{5.2}$$

where the input $y_{t-p-n_B+1}^{t-1}$ has length $p - n_B - 1$, $w := (\operatorname{vec} {W_0}^T, ..., \operatorname{vec} {W_{n_B-1}}^T)^T$, $\theta := (\theta_0, ..., \theta_{n_B-1})^T \in \mathbb{R}^{n_B}$ and $f_{W_i} \in \mathbb{R}$ are DNNs that process a translated window of p past measurements. To prevent modeling bias, different lagged windows are not assumed to be disjoint (e.g. disjoint windows would not allow to model close in time interactions between adjacent windows). Nonetheless it is straightforward to control how much different lagged windows are overlapped by using dilated windows (similar to what is done for dilated convolutions Section 5.1.2).

Remark 5.2 (Block partitioned second order structure). Eq. (5.2) is a block partitioned non-linear model in which non-linear blocks are linearly combined by θ. Moreover each block is computed on different lagged windows of past data. This guarantees that the second order interactions between far apart time instants are zero (i.e. no information "leakage" is possible). This guarantees the fading memory property holds on the whole architecture if $\theta_i \to 0$ as $i \to \infty$ and the output of each block have roughly the same scale (i.e. they are normalized Section 5.3.2).

Overall, our architecture (depicted in Fig. 5.2) is described by the following parameters:

- n_B number of blocks
- p size of the "elementary" regressor of each DNN (lagged window of past data y_{t-p}^{t-1})
- $w = \left(\operatorname{vec} {W_0}^T, ..., \operatorname{vec} {W_{n_B-1}}^T\right)^T$ DNNs weights

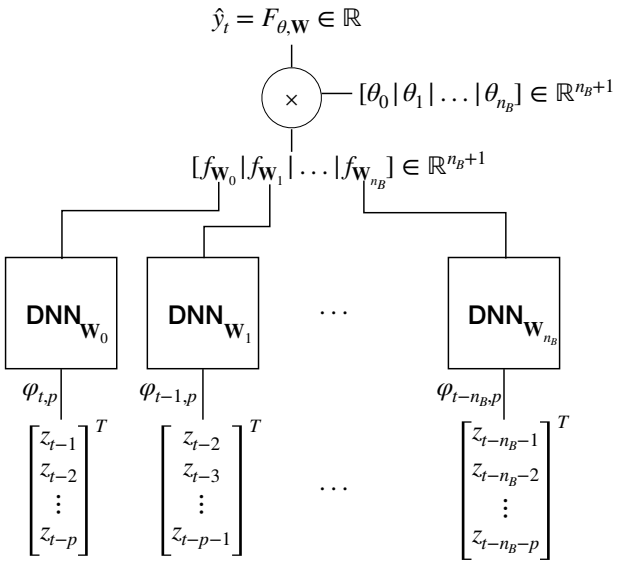


Figure 5.2: **Block-partitioned fading memory architecture.**

- $\theta = (\theta_0, ..., \theta_{n_B-1})^T$ recombination parameters

Remark 5.3 (How to choose the number of blocks?). Ideally, n_B should be large enough to capture the most relevant past, so that the architecture can approximate arbitrarily well the unknown regression function f_0 and should *not* be chosen to face a bias-variance trade-off. Regularization shall be used to control model complexity, by automatically assigning fading weights to the outputs of each block.

Remark 5.4 (How to choose p?). The past horizon of each "elementary regressor" should be chosen as small as possible without incurring in any modelling bias.

5.2.3 Fading Memory Temporal Convolutional Networks

We now extend the fading architecture in Eq. (5.2) to general TCN [Bai et al., 2018]. The key observation [Zancato et al., 2022] is that each layer in a TCN employs causal temporal convolutions, hence each output of a TCN depends only on past values of the input signal.

As described in Section 5.1.2 the output a TCN model given an input sequence of length T has dimension T ($f_{\text{TCN}}(y) \in \mathbb{R}^T$), moreover the i-th output neuron $(f_{\text{TCN}}(y))_i$, where $y \in \mathbb{R}^T$, only depends on its receptive field p which is a function of the dilation coefficient, the number of hidden layers and the convolutional kernel size used to perform 1-D convolutions. Hence we can write an equation similar to Eq. (5.2) by combining

the outputs of a TCN with the weights θ_i with $i \in [n_B - 1]$:

$$f_{\theta, w_{\text{TCN}}}\left(y_{t-p-n_B+1}^{t-1}\right) := \sum_{i=0}^{n_B-1} \theta_i \left(f_{\text{TCN}}(y)\right)_{t-i-1} \tag{5.3}$$

where $p = |\omega| 2^L$ with L the number of hidden layers of the TCN and $|\omega|$ the length of the causal convolutional filters (see Definition 5.1).

Remark 5.5. Remarks Remark 5.2, Remark 5.3 and Remark 5.4 still hold.

5.3 Fading Memory Regularization

In previous sections we described how to consider a model structure in which fading memory can be induced by simply choosing suitably decaying coefficients θ_i. The rate at which these converge to zero provides an estimate of the most relevant past (i.e. the number of time instants mostly affecting the output of the non-linear model) and is directly connected with model complexity (the faster the decay rate the lower the complexity). Nonetheless, how to choose such a decay rate is not clear. Ideally, n_B should be large enough to capture the memory of the system, so that the regression function can be approximated arbitrarily well by the "optimal" non-linear model and should *not* be chosen to face a bias-variance trade-off. In practice, too flexible feature extractors (such as TCNs or DNNs) are prone to overfitting. Therefore, some regularization is needed to control model complexity and benefit from having a large memory window. In this section, we introduce a regularized loss inspired by Bayesian arguments which allows us to use an architecture with a "large enough" past horizon n_B (i.e., larger than the true memory) and automatically select the *relevant past* to avoid overfitting. Such information is exposed to the user through an interpretable parameter λ that directly measures the *relevant time scale* of the signal. Our method exploits the same Bayesian hyper-parameters selection framework we discussed in Section 2.3.3 based on Type II maximum likelihood.

5.3.1 Bayesian Automatic Complexity Selection

To begin with, we consider the same model structures proposed in Section 5.2.2 and Section 5.2.3 [Zancato and Chiuso, 2021, Zancato et al., 2022]. In particular we shall directly refer to Eq. (5.2) (anything can be carried out without modifications for Eq. (5.3)).

Assumption 5.1 (Gaussian Likelihood)**.** *Assume the likelihood function associated to*

Eq. (5.1) *is Gaussian.*

Hence we have:

$$y(t) \mid y_{-}^{t-1} \sim \mathcal{N}\left(f_0(y_{-}^{t-1}), \sigma^2\right) \tag{5.4}$$

where σ is the innovation variance (we assume it does not depend on time Section 6.1) and f_0 is the optimal regression function which, in principle, is allowed to depend on the infinite past of $y(t)$. Note that this assumption does not restrict our framework and is used only to justify the use of the squared loss to learn the regression function.

In practice, we do not know f_0 and we approximate it with our non-linear parametric model $f_{\theta,w}$ Eq. (5.2) where θ represent the *fading parameters* while w represent either the parameters of the non-linear blocks used in Eq. (5.2) or the parameters of a TCN model (if Eq. (5.3) is used). We denote with N the number of data available from the time series y, moreover we shall partition this dataset in two groups of consecutive samples: n_p (representing the "past") and n_f (representing the "future").

We denote the likelihood of the future window of length n_f of y with the model structure parametrized by θ and w as:

$$p\left(y_{t+1}^{t+n_f} | y_{t-n_p+1}^{t}, \theta, w\right) = \prod_{k=1}^{n_f} p\left(y(t+k) | y_{t+k-n_p}^{t+k-1}, \theta, w\right)$$

To make the notation simpler we shall call the vector $y_f := y_{t+1}^{t+n_f} \in \mathbb{R}^{n_f}$ and call the stacked prediction outputs of the non-linear model:

$$\hat{y}_{\theta,w} := (\hat{y}_{\theta,w}(t+1), ..., \hat{y}_{\theta,w}(t+n_f))^T = \left(f_{\theta,w}\left(y_{t+1-n_p}^{t}\right), ..., f_{\theta,w}\left(y_{t+n_f-n_p}^{t+n_f-1}\right)\right)^T$$

Note that $\hat{y}_{\theta,w} \in \mathbb{R}^{n_f}$ is a linear predictor w.r.t. to the output of each block (or TCN) so that it holds: $\hat{y}_{\theta,w} = F_w\theta$, where $(F_w)_{i,j} := f_{W_j}\left(y_{t+i-j-p+1}^{t+i-j}\right) \in \mathbb{R}^{n_f \times n_B}$.

In a Bayesian framework, the optimal set of parameters can be found maximizing the posterior $p(\theta, w \mid y_f)$ over the model parameters. We assume θ and w as independent random variables:

$$p(\theta, w \mid y_f) \propto p(y_f \mid \theta, w)p(\theta)p(w) \tag{5.5}$$

where $p(\theta)$ is the prior associated to the fading coefficients and $p(w)$ is the prior on the remaining parameters. $p(\theta)$ encodes our prior belief that the complexity of the predictor should not be too high and therefore it should depend only on the *most relevant past*.

Assumption 5.2 (Fading prior on θ)**.** *We enforce the fading prior over θ assuming the components of θ have zero mean and exponentially decaying variances: $\mathbb{E}\theta_j^2 = \kappa\lambda^j$*

for $j = 0, ..., n_B - 1$, *where* $\kappa \in \mathbb{R}^+$ *and* $\lambda \in (0, 1)$.

Remark 5.6 (Maximum entropy prior). To specify the prior, we need a density function $p(\theta)$ but up to now we only specified constraints on the first and second order moments. We therefore need to constrain the parametric family of prior distributions we consider. Any choice on the class of prior distributions lead to different optimal estimators. Among all the possible choices of prior families we choose the maximum entropy prior [Cover and Thomas, 1991]. Under constraints on first and second moment, the maximum entropy family of priors is the exponential family [Cover and Thomas, 1991]. In our setting, we write it as:

$$\log(p_{\lambda,\kappa}(\theta)) \propto -\|\theta\|^2_{\Lambda^{-1}} - \log\det\Lambda \tag{5.6}$$

where $\Lambda \in \mathbb{R}^{n_B \times n_B}$ is a diagonal matrix with elements $\Lambda_{j,j} = \kappa\lambda^j$ with $j = 0, ..., n_B - 1$.

The parameter λ represents how fast the output of the predictor "forgets" the past. Therefore, λ regulates the complexity of the predictor: the smaller λ, the lower the complexity.

In practice, we need to estimate model hyper-parameters (corresponding to the prior) from data, so that a Type II maximum likelihood is necessary Section 2.3.3. Unfortunately the task of computing (or even approximating) the marginal likelihood in this setup is prohibitive and we should resort to Monte Carlo sampling techniques which might not scale well with the number of parameters in the non-linear blocks. To overcome such limitation we adopt the following variational strategy which directly exploits the decoupling between the fading parameters and the ones associated to non-linear blocks (or TCN). In particular we exploit the fact that the output $\hat{y}_{\theta,w}$ is linear w.r.t. fading parameters.

To begin with, we observe that jointly estimating $\theta, w, \lambda, \kappa$ (and possibly σ) by minimizing the negative log of the joint posterior leads to degeneracy because the joint negative log posterior goes to $-\infty$ as $\lambda \to 0$. Indeed, typically the parameters describing the prior (such as λ) are estimated by maximizing the marginal likelihood, i.e., the likelihood of the data once the parameters (θ, w) have been integrated out.

Theorem 5.1 (Variational upper bound on the marginal likelihood)**.** *Consider a model on the form:* $\hat{Y}_{\theta,w} = F_w\theta$ *(linear in* θ *and possibly non-linear in* w*) and its posterior in Eq.* (5.5)*. Assume the prior on the parameters* θ *is given by the maximum entropy prior Eq.* (5.6) *and* w *is fixed. Then, the following is an upper bound on the marginal likelihood associated to the posterior in Eq.* (5.5) *with marginalization taken* only *w.r.t.*

θ:

$$\mathcal{U}_{\theta,w,\Lambda} = \frac{1}{\sigma^2}\left\|Y_f - \hat{Y}_{\theta,w}\right\|^2 + \theta^\top \Lambda^{-1}\theta + \log\det(F_w \Lambda {F_w}^\top + \sigma^2 I). \tag{5.7}$$

Note we hide the dependency of prior hyper-parameters λ and κ inside Λ. The complete proof of this is in Appendix C.1.

The upper bound $\mathcal{U}_{\theta,w,\Lambda}$ provides an alternative loss function to the negative log posterior which does not suffer from the degeneracy alluded above while optimizing over θ, w, λ and κ. So that the optimization problem we solve is:

$$\underset{\theta,w,\lambda\in(0,1),\kappa>0}{\arg\min} \frac{1}{\sigma^2}\left\|Y_f - \hat{Y}_{\theta,w}\right\|^2 + \|\theta\|^2_{\Lambda^{-1}} + \log\det(F_w \Lambda F_w^T + \sigma^2 I) + \log p(w) \tag{5.8}$$

Remark 5.7 (Prior on w). $\log p(w)$ defines the regularization applied on the remaining parameters of our architecture. Different choices are possible: regularization based on the squared norm, sparsity inducing regularization or any other type or regularization suited to constrain the complexity of the non-linear blocks (or TCN).

Remark 5.8 (Fading memory in multivariate time series). In the case of multivariate time series, fading regularization can be applied either with a single fading coefficient λ for all the time series or with different fading coefficients for each time series. In all the experiments of the following chapters Chapter 6 and Chapter 7, we chose to keep one single λ for all the time series. In practice, this choice is sub-optimal and might lead to more overfitting than treating each time series separately: the "dominant" (slower) time series will highly influence the optimal λ.

5.3.2 Block Normalization

As mentioned above, the blocks f_{W_i} should be rich enough to model non-linearities of the regression function, yet they should not undo the fading memory regularization we introduced. In particular, we note that non-identifiability occurs due to the product $F_w\theta$ which in turn can reduce the effects of Fading Regularization: if $f_{W_i}\left(y_{t-i-p}^{t-i-1}\right)$ (the features extracted by each block) have different scales across block indices (columns of the matrix F_w) it can happen that features associated with small θ_i have large scale so that the overall contribution of the past does not fade.

We can avoid degeneracy due to non-identifiability by properly normalizing the output of each block; we choose to apply a modern regularization method which is typically applied to regularize DNNs: Batch Normalization (see [Ioffe and Szegedy, 2015] or Section 1.5). In particular we impose that the outputs of different blocks $f_{W_i}\left(y_{t-i-p}^{t-i-1}\right)$ have comparable means and scales across indices $i = 0, ..., n_B - 1$.

The main idea behind batch normalization is to maintain running statistics (means and standard deviations) of the outputs of the hidden nodes of a DNN model during training and apply a normalizing affine transformation to these outputs. In this way the inputs at each layer have zero mean and unit variance. In our case we do not want the output of each block $f_{W_i}\left(y_{t-i-p}^{t-i-1}\right)$ to have zero mean and unit variance, rather we need comparable means and scales across each block output. We therefore use batch normalization to normalize block features. Then we use an affine transformation (with parameters to be optimized) to jointly re-scale all the output blocks before the linear combination with θ.

Denoting with $\bar{f}_{W_i}$ the normalized i-th block output, the output of our regularized fading architecture is: $F_{\theta,W} = \sum_{i=0}^{n_B} \theta_i \bar{f}_{W_i}$. The normalization is performed according to:

$$\bar{f}_{W_i} = \frac{f_{W_i} - \mathbb{E}[f_{W_i}]}{\sqrt{\text{Var}[f_{W_i}] + \epsilon_1}}\gamma + \beta \quad i = 0, ..., n_B - 14 \tag{5.9}$$

where $\mathbb{E}[f_{W_i}]$ and $\text{Var}[f_{W_i}]$ are estimated using running averages along the optimization iterations (as standard practice with batch normalization) and ϵ_1 is a small number used to avoid numerical issues in case the estimated variance becomes too small.

Remark 5.9. γ and β are jointly optimized with other parameters and are shared among the outputs of the blocks such that the relative scale among them is preserved.

5.4 Discussions and conclusions

Deep Learning without inductive bias and regularization is doomed to fail. In this chapter we describe the main design principles that are usually considered while building successful DNNs both in image processing and time series analysis application domains. After this general discussion we specifically focus on time series analysis and propose a novel inductive bias and regularization scheme both for fully-connected DNNs and TCNs. Our inductive bias and regularization are designed to exploit domain knowledge and encode the so called "fading memory" property of time series. "Fading" regularization allows to automatically perform model selection (automatic complexity selection) based solely on available data [Zancato and Chiuso, 2021, Zancato et al., 2022] and therefore can be employed to improve model generalization and reduce the number of hyperparameters needed to be tuned by the user.

6

Fading Memory for Non-Linear System Identification

The main goal of system identification is to build a dynamical model from observed data which is expected to generalize well on unseen data. In the context of non-linear systems, both parametric [Sjöberg et al., 1995, Juditsky et al., 1995, Masti and Bemporad, 2018] and non-parametric models [Pillonetto et al., 2011] are viable alternatives used in practice. Recently, many efforts have been devoted to extend classical results for linear systems to non-linear ones. Instances of parametric and non-parametric model classes are respectively NARX/NARMAX and kernel based methods ([Pillonetto et al., 2011]). Typically, the identification problem can be divided in two steps: first, find the best model class given the available data and then find the best model within that particular model class. None of these two problems can be easily solved in general and often model optimization is a non-convex problem. Finding the proper model complexity (structure) requires a complexity criterion (Section 2.3.6). Beyond classical complexity criteria such as Akaike's Information Criterion and Bayesian Information Criterion [Sjöberg et al., 1995] many other automatic model complexity criteria have been introduced, both in the parametric [Lind and Ljung, 2008] and non-parametric frameworks [Pillonetto et al., 2011].

The aim of this chapter is to extend ideas proposed in [Pillonetto et al., 2011] to the parametric framework. More precisely, we shall build a parametric estimator for non-linear system identification using Neural Networks as building blocks. Due to their high capacity, NNs are prone to overfitting unless constrained by regularization or inductive bias (Chapter 5). As such, our architecture and optimization loss are specifically designed to exploit domain knowledge (fading memory systems Section 5.2) and to

automatically detect and choose the best model complexity from training data (Section 5.3). The inductive bias relies on the assumption that the system to be identified belongs to the class of fading memory systems [Matthews and Moschytz, 1994].

Previously proposed non-parametric methods such as in [Pillonetto et al., 2011] might not scale well with the number of data; on the contrary, our architecture can scale to hundred of thousand datapoints as is typically the case for NNs based models [Sjöberg et al., 1995, Masti and Bemporad, 2018]. Furthermore, the parametric model and the loss function are designed so that standard Deep Learning regularization techniques [Bansal et al., 2018, Srivastava et al., 2014, Ioffe and Szegedy, 2015] and Stochastic Optimization methods [Kingma and Ba, 2014, Welling and Teh, 2011] can be applied.

6.1 Problem formulation

We now proceed by briefly stating the goals of system identification.

Let $\{u(t)\}$ and $\{y(t)\}$, $t \in \mathbb{Z}$ be respectively the input and output of a discrete time, time invariant, nonlinear state-space stochastic system:

$$\begin{aligned} x(t+1) &= f(x(t), u(t), w(t)) \\ y(t) &= h(x(t)) + v(t) \end{aligned} \tag{6.1}$$

where $\{w(t)\}$ and $\{v(t)\}$ are respectively process and measurement noises. A rather standard assumption is that both $\{w(t)\}$ and $\{v(t)\}$ are strictly white and independent. For ease of exposition we shall assume that both y and u are scalar, but extension to the vector case is straightforward. We will denote with $z(t) := (y(t), u(t))^\top$ the joint input-output process and with z_-^t the infinite past $(z(t), z(t-1), ...,)^T$ of $z(t)$ w.r.t. t.

Starting from Eq. (6.1) it is always possible to define the optimal one-step-ahead predictor of $y(t)$ given the joint past z_-^{t-1} as:

$$\hat{y}_{t|t-1} = f_0(z_-^{t-1}) := \mathbb{E}[y(t)|z_-^{t-1}] \tag{6.2}$$

hence the input-output behaviour of the state space model Eq. (6.1) can be written in *innovation* form as:

$$y(t) = f_0(z_-^{t-1}) + e(t) \tag{6.3}$$

where $e(t)$ is, by definition, the one step ahead prediction error (or *innovation sequence*) of $y(t)$ given the joint past $\{z(s), s < t\}$. The innovation $e(t)$ is a martingale difference sequence w.r.t. the sigma algebra generated by past data $\mathcal{P}_t := \sigma\{z(s), s < t\} = \sigma\{y_-^{t-1}, u_-^{t-1}\}$ and, thanks to the time-invariance assumption on Eq. (6.1), it has

constant conditional variance:

$$\text{Var}[e(t)] = \text{Var}[y(t)|z_-^{t-1}] = \eta^2, \quad \forall t \tag{6.4}$$

We shall also assume that $e(t)$ is strictly white.

Our main goal is to find an estimate $\hat{f}$ of the predictor map f_0 in Eq. (6.2). This problem can be framed in the classical regularized Prediction Error Method (PEM) framework, i.e. defining[1]:

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{t=1}^{N} (y(t) - f(z_-^{t-1}))^2 + \lambda P(f)$$

where $\mathcal{F}$ is the model class and $P(f)$ is a penalty function.

This framework includes both classical parametric approaches (i.e. where $\mathcal{F}$ is a parametric model class $\mathcal{F} := \{f_w, w \in \mathbb{R}^D\}$ and the penalty $P(f)$ is expressed as a function of the parameters w) as well as non-parametric ones where f lives in an infinite dimensional space such as a Reproducing Kernel Hilbert space and $P(f)$ is the norm in the space (Section 2.3.1).

In particular, we shall compare state-of-the art nonparametric methods introduced in [Pillonetto et al., 2011] that use RKHS/GPs with the class of fully-connected Neural Networks with Fading Memory inductive bias introduced in Section 5.2.

6.2 Fading Memory Inductive Bias

We now briefly describe the Fading Memory property in the context of system identification (the notion of Fading Memory has already been introduced in Section 5.2 for general time series). We shall now consider the class of non-linear systems also known as Fading Memory systems (see e.g. [Matthews and Moschytz, 1994] and references therein), a property that can be informally described by saying that the effect of past inputs $\{u(s)\}$ with $s < t$ on the output $y(t)$ becomes negligible (tends to zero asymptotically) as $|t - s|$ goes to infinity. This property guarantees that the system behaviour can be uniformly approximated on compact sets. Hence, as already done in Section 5.2, the universal approximation properties of Neural Networks [Cybenko, 1989] suggests that NNs can be seen as natural candidates to tackle the identification problem. Yet, NNs are known to suffer from severe overfitting. To cure this limitation we shall exploit the model structure defined in Section 5.2.2 and introduce an inductive

[1] W.l.o.g we use the square loss.

bias in the architecture and a suitable regularization both fostering the fading memory.

Under the fading memory assumption we shall assume that the optimal predictor model $f_0(y_t^-, u_t^-)$ in Eq. (6.2) depends only upon a finite, yet arbitrarily long window of past data:

$$f_0\left(y_-^t, u_-^t\right) = f_0\left(y_{t-T}^{t-1}, u_{t-T}^{t-1}\right) = f_0\left(z_{t-T}^{t-1}\right) \tag{6.5}$$

The past horizon T is finite but *arbitrarily long* so that no significant bias is introduced.

6.2.1 Network Architecture

We now describe how to apply the block-structured DNN we introduced in Section 5.2.2 for non-linear system identification. As in [Pillonetto et al., 2011] we assume the predictor function f_0 can be written as a linear combination of (in principle) infinitely many elementary building blocks $f_{W_i^*}$, each of them described by a DNN. In particular we assume that each $f_{W_i^*}$ is actually a function of only a small window of past data (of length p), namely:

$$\begin{aligned} f_{W_i^*} :=& f_{W_i^*}(y_{t-i-1}, u_{t-i-1}, ..., y_{t-i-p}, u_{t-i-p}) \\ =& f_{W_i^*}\left(z_{t-i-p}^{t-i-1}\right) \in \mathbb{R} \end{aligned} \tag{6.6}$$

where w.l.o.g. we consider the same horizon p both for the past of y and u. Note that each block $f_{W_i^*}\left(z_{t-i-p}^{t-i-1}\right)$ outputs a scalar feature.

The output predictor is then parametrized in the form:

$$f_{\theta^*, w^*}(z_-^t) = \sum_{i=0}^{\infty} \theta_i^* f_{W_i^*}\left(z_{t-i-p}^{t-i-1}\right)$$

where $w^* := (\operatorname{vec} W_0^{*T}, \operatorname{vec} W_1^{*T}, ...)^T$, $\theta^* := (\theta_0^*, \theta_1^* ...)^T$.

The fading memory assumption guarantees that the contribution to output prediction of blocks $f_{W_i^*}\left(z_{t-i-p}^{t-i-1}\right)$ should fade to zero as the index i increases. Thus, w.l.o.g., we shall consider a finite number of blocks n_B and truncate the model f_{θ^*, w^*} to the form:

$$f_{\theta, w^*}\left(z_{t-n_B-p+1}^{t-1}\right) = \sum_{i=0}^{n_B-1} \theta_i^* f_{W_i^*}\left(z_{t-i-p}^{t-i-1}\right)$$

We can now approximate the optimal one-step-ahead predictor with the parametric

model introduced in Section 5.2.2:

$$f_{\theta,w}\left(z_{t-n_B-p+1}^{t-1}\right) = \sum_{i=0}^{n_B-1} \theta_i f_{W_i}\left(z_{t-i-p}^{t-i-1}\right) \tag{6.7}$$

as illustrated in Fig. 5.2.

Ideally n_B should be large enough to capture the memory of the system, so that the network can approximate arbitrarily well the "true" f_0 and should *not* be chosen to face a bias-variance trade-off. Regularization shall be used to control the model complexity, by automatically assigning fading weights to each block.

Remark 6.1 (How to choose each block?). The choice of f_{W_i} is completely arbitrary, e.g. it could be a single layer, multilayer or basis functions Neural Network; each block has its own set of parameters, so that it can potentially extract different features from different lagged past windows. In the following we shall assume DNNs are used.

6.3 Fading Memory Regularization

In Eq. (6.7) some important hyper-parameters need to be specified: how should one choose the number of blocks n_B and the horizon of each lagged window p? We use Fading Memory regularization we introduced in Section 5.3 to optimally choose the right number of blocks and use cross-validation to choose the best horizon length (Section 2.3.6).

The main idea of Fading Memory regularization [Zancato and Chiuso, 2021] is to let the user choose an architecture with a "large enough" number of blocks n_B (i.e. larger than the actual system memory) and then automatically select the fading decay rate to constrain the complexity of the parametric model in Eq. (6.7) to avoid overfitting. We refer to Section 5.3 for the details on Fading Memory regularization [Zancato and Chiuso, 2021].

6.3.1 Controlling block complexity

Without regularization, each single block could overfit and hence reduce generalization capabilities of our architecture. We now recall some standard regularization techniques that are used to improve trainability and generalization of DNNs: batch normalization [Ioffe and Szegedy, 2015], dropout [Srivastava et al., 2014] and penalty terms on DNNs' weights norm. Despite all these methods can be applied simultaneously, in the following, we shall mainly focus on a type of regularization which can directly be imposed following

the Bayesian argument we used in Eq. (5.8): we shall impose a prior on w (for simplicity we consider each block W_i, $i = 0, ..., n_B - 1$ independently).

Consider a single block f_{W_i}, which is a DNN with L layers, parametrized by W_l for $l = 1, ..., L$ (the bias parameter can be considered within W_l). Inspired by [Bansal et al., 2018] we exploit Soft Orthogonality (SO) to enforce that the Gram matrix of the weights is close to the identity. Therefore we consider the following per-layer regularization term:

$$\log(W_l) = \left\| W_l^\top W_l - I_{d_l - 1} \right\|_F^2 \quad l = 1, ..., L \tag{6.8}$$

where $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$.

Remark 6.2 (Priors independence). We assume the priors are independent both across layers and across blocks.

Soft Orthogonality regularization is known to foster network trainability by stabilizing the distribution of activations over layers [Bansal et al., 2018].

6.4 Optimization

The optimization problem Eq. (5.8) can be solved using off-the-shelf stochastic optimization tools such as Stochastic Gradient Descent and Adam [Welling and Teh, 2011, Kingma and Ba, 2014]. Both these methods rely on gradients to find the best set of parameters, therefore we must require the fading architecture and its blocks to be differentiable w.r.t. their parameters (some extensions are applicable, e.g. with ReLU activations functions). Note the stochasticity introduced by the choice of the mini-batches in SGD has been proven to be highly effective and provide properties which are not shared with Gradient Descent, such as the ability to avoid saddle points and spurious local minima of the loss function (Section 1.4).

Remark 6.3. The stochasticity in the choice of minibatches only affects the computation of the fit (minus log likelihood) and log det terms in Eq. (5.8) since the regularization term does not need any datum to be computed.

6.5 Experiments

Similarly to [Pillonetto et al., 2011] we tested our architecture using Monte Carlo studies on 4 nonlinear systems of increasing complexities, as listed in Table 6.1. For each nonlinear system we generate random trajectories of length N starting from the system initially at rest, we take $u(t) \sim \mathcal{N}(0, 1)$ (whenever possible) and $e(t) \sim \mathcal{N}(0, 1)$. We

Table 6.1: Non-linear systems benchmark from [Pillonetto et al., 2011]. The innovation variance for systems (1) and (2) is 1, for system (3) is 0.22^2 and for system (4) is 0.14^2.

$$
\begin{aligned}
(1) \quad y(t) &= e^{-0.1y(t-1)^2}(2y(t-1) - y(t-2)) + e(t) \\
(2) \quad y(t) &= -2y(t-1)\mathbf{1}(y(t-1) < 0) + 0.4y(t-1)\mathbf{1}(y(t-1) \geq 0) + e(t) \\
(3) \quad y(t) &= 0.5y(t-1) - 0.05y(t-2)^2 + u^2(t-1) + 0.8u(t-2) + 0.22e(t) \\
(4) \quad y(t) &= 0.8y(t-1) + u(t-1) - 0.3u(t-1)^3 + 0.25u(t-1)u(t-2) \\
&\quad -0.3u(t-2) + 0.25u(t-2)^3 - 0.2u(t-2)u(t-3) - 0.4u(t-3) \\
&\quad +0.14e(t)
\end{aligned}
$$

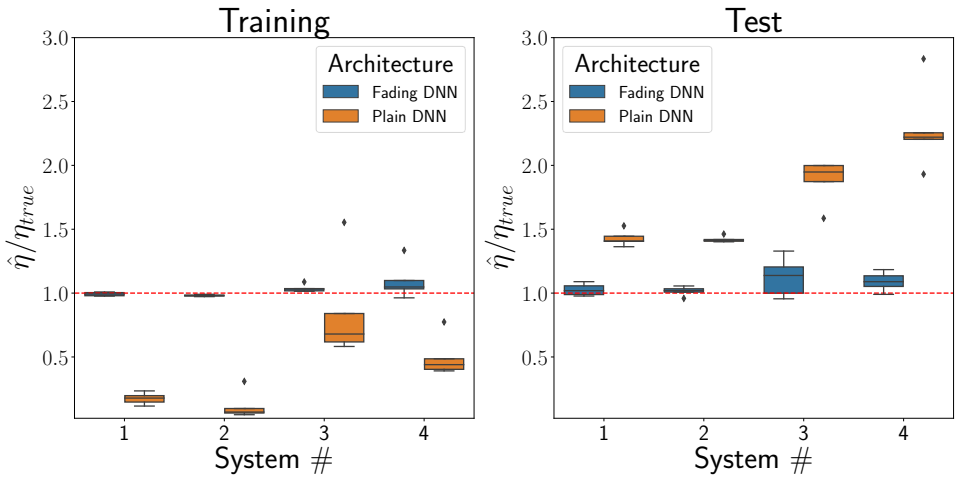


Figure 6.1: **Fading architecture vs plain DNN model.** Monte Carlo results: box plot for train and generalization on systems from Table 6.1 (20 runs, N=10k). Both architectures have the same input horizon (12), activations (Tanh), hidden layers (5) and a similar number of parameters. Note fading architecture avoids overfitting and reduce generalization gap for every benchmark system.

test generalization capabilities of each model on test data generated as the training ones and we measure generalization error comparing η_{true} (Eq. (6.4)) with $\hat{\eta}$ where $\hat{\eta}^2 := \frac{1}{N}\sum_{i=1}^{N}(y(i) - \hat{y}(i))^2$ for each system.

In each experiment we choose over-parametrized DNNs to parametrize each block f_{W_i}: 5 hidden layers, 100 hidden units with Tanh activation function ($\approx 41k$ parameters). In such a scenario we expect that without any regularization severe overfitting occurs. In Fig. 6.1 we show this is indeed the case and compare a plain DNN (without any particular structure) against our fading architecture. Both models take the same number of data as input and have a similar number of parameters.

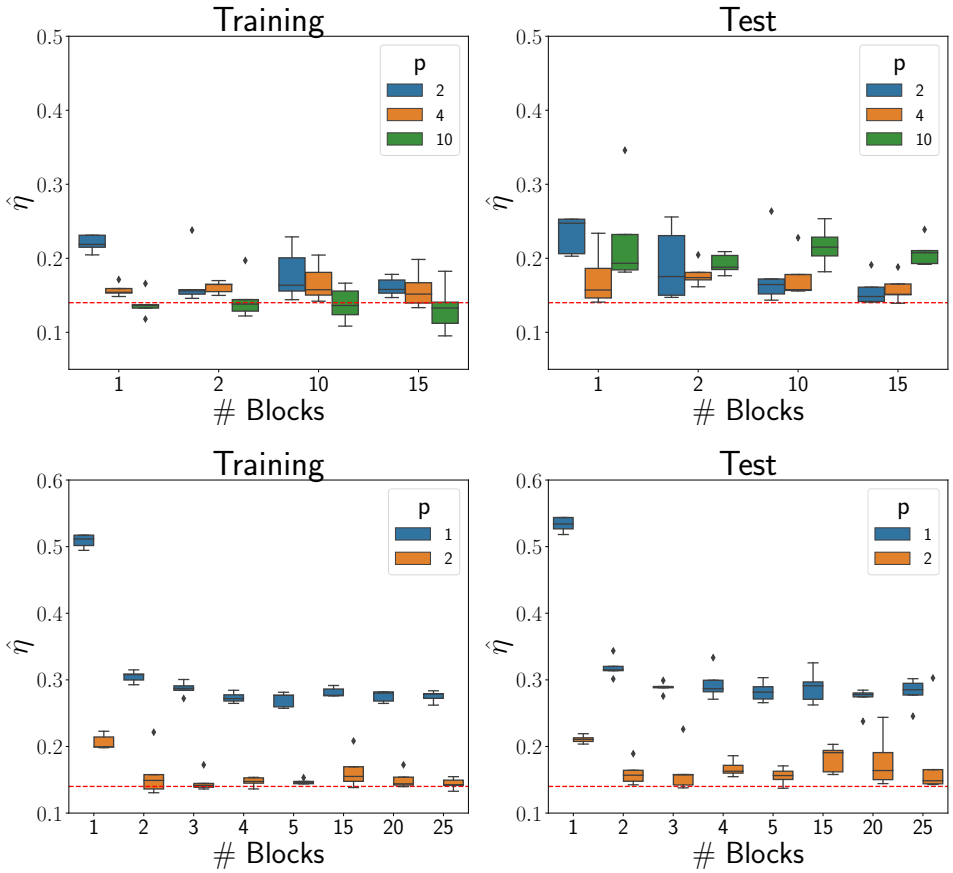


Figure 6.2: **Robustness to the horizon choice.** Monte Carlo results on system 4 (runs=20, N=10k) for different values of n_B and p. **Upper panels**: When p is such that a single block does not overfit, our method prevents overfitting as n_B grows. **Lower panels**: Degenerate choice of p: when p is too small it introduces a bias in the estimation. In this particular case we are not able to model mixed terms such as $u(t-2)u(t-3)$ which are present in System 4.

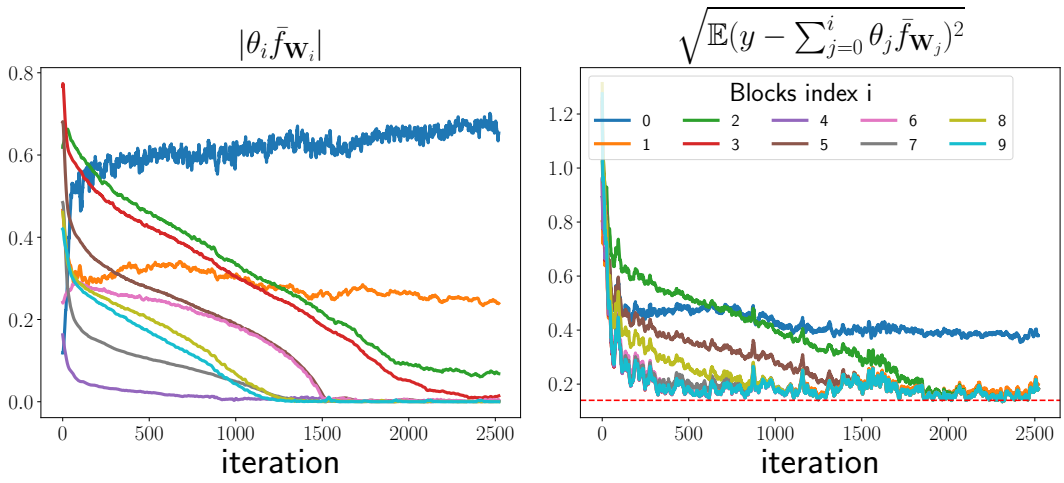


Figure 6.3: **Blocks' relative importance.** Single run on system 4, $n_B = 9$ and $p = 2$. Importance is measured both by $|\theta_i \bar{f}_{W_i}|$ $i = 0, ..., n_B - 1$ (**left**) and by the prediction error standard deviation of the truncated predictor up to the i-th block: $\sqrt{\mathbb{E}(y - \sum_{j=0}^{i} \theta_j \bar{f}_{W_j})^2}$ for $i = 0, ..., n_B - 1$ (**right**).

The proper fading horizon length is not known a priori: we tested automatic complexity selection in Fig. 6.2. We compare different architectures optimized according to Eq. (5.8) using different number of blocks and block horizons p. We show that generalization for fixed p does not worsen as the number of blocks (and therefore representational capability) increases. The robustness on the choice of the number of blocks n_B proves the effectiveness of our regularization scheme. Moreover from the user's perspective it reduces the sensitivity of the identified model w.r.t. a wrong choice of the input horizon and allows the user to safely choose large n_B without incurring overfitting. Regarding the actual value of n_B we have no other prescription than choosing it large enough so that the relevant past is processed by the architecture and automatic complexity selection can select the optimal λ based on available data.

One last question remains open: how to choose the horizon of each block p? Other than trial and error, cross validation could be used to choose the best hyper-parameter p. In Fig. 6.2 we compare the effects of different p: our regularization does not impose fading constraints on the input of each block, we therefore expect that large p (despite SO regularization) might overfit. From the user's perspective the choice of p should be as small as possible without introducing too modeling bias on each block (see Fig. 6.2 for an example of a degenerate choice: $p = 1$).

For the sake of completeness in Fig. 6.3 we show the importance of each block on the prediction $\hat{y}(t)$ during optimization. We use $|\theta_i \bar{f}_{W_i}|$ and the residual error of the truncated (in the number of blocks n_B) predictor. The latter is measured by an empirical estimate of $\sqrt{\mathbb{E}(y - \sum_{j=0}^{i} \theta_j \bar{f}_{W_j})^2}$ for $i = 0, ..., n_B - 1$. In Fig. 6.3 the block processing data closer to the present is indeed the one which mostly affects $\hat{y}(t)$. Note the convergence of each block's relevance to its asymptotic value is not uniform across different blocks: the farther into the past the fastest to converge (and become negligible). We leave to future work the design of optimization schemes which could improve convergence speed (e.g. using adaptive learning rates algorithms other than Adam, and other stochastic optimization methods designed to improve DNN convergence and generalization).

In Table 6.2 we directly compare our architecture with the GP solution proposed in [Pillonetto et al., 2011]. We use system 4 to generate datasets of increasing lengths (up to 100k). Note that in the large data regime GPs cannot be used without approximation schemes. Our architecture shows a larger generalization gap in the low data regime but achieves increasingly better results as the dataset size increases. We believe the lower performance in the low data regime of our method w.r.t. the GP model in [Pillonetto et al., 2011] is due to the fact that the GP model is heavily regularized and the fact that the underlying system to be identified is relatively simple.

Table 6.2: **Data efficiency.** Comparison among: (a) GP model from [Pillonetto et al., 2011], (b) Our architecture w/o regularization, (c) Our complete architecture. $\hat{\eta}$ median value on Monte Carlo study on system 4 ($\eta_{true} = 0.14$).

	N=400		N=1000		N=10k		N=100k	
	Train	Test	Train	Test	Train	Test	Train	Test
GP model from [Pillonetto et al., 2011]	0.14	0.27	0.13	0.19	0.14	0.17	-	-
Our architecture w/o regularization	0.02	0.49	0.03	0.45	0.07	0.23	0.12	0.20
Our complete architecture	0.10	0.32	0.15	0.22	0.16	0.17	0.15	0.15

6.6 Discussions and conclusions

In this chapter we show that overparametrized DNNs without a proper inductive bias and regularization fail to solve non-linear system identification benchmarks. We overcome such a limitation introducing both a new architecture inspired by fading memory systems and a new regularized loss inspired by Bayesian arguments which in turn allows for automatic complexity selection based on the observed data. We showed when DNN based parametric architectures are good alternatives to state of the art non-parametric models for modelling non-linear systems (mid-large data regime). Moreover we proved our method does not suffer from typical non-parametric models limitations on large dataset sizes and favourably scales with the number of samples. We leave to future work the design of optimization schemes which could improve convergence speed (e.g. using adaptive learning rates algorithms other than Adam and other stochastic optimization methods designed to improve DNN convergence and generalization).

7

Interpretable Residual Temporal Convolutional Networks

Time series data is being generated in increasing volume from industrial, medical, commercial and scientific applications. Such growth is fueling demand for anomaly detection algorithms that are general enough to be applicable across domains, yet reliable enough to operate on real-world time series data [Munir et al., 2019, Geiger et al., 2020, Su et al., 2019]. While recent developments have focused on Deep Neural Networks, simple linear models still outperform DNNs in applications that require robustness to dataset-specific tuning [Braei and Wagner, 2020] and interpretable failure modes [Geiger et al., 2020, Su et al., 2019].

To harvest the flexibility and interpretability of engineered modules while enabling end-to-end differentiable training, we introduce STRIC (Time series Reliable and Interpretable Anomaly Detection). STRIC is composed of three modules: A *local predictor* for each component of the time series, tasked with isolating interpretable factors such as trends, quasi-periodicity, and linearly predictable statistics. Its prediction residual is fed to a *global predictor* that takes into account other time series as context to predict each time series. The prediction residual of each time series is then fed to an *anomaly detector* based on a likelihood ratio test.

More specifically, STRIC uses a parametric model implemented by a sequence of residuals blocks with each layer capturing the prediction residual of previous layers. The first layer models trends, the second layer models seasonality, the third layer is a general linear predictor, and the last is a general non-linear model in the form of a Temporal Convolution Network. While the first three layers are *local* to each component of the time series, the last also integrates *global* statistics from additional time series,

as in [Flunkert et al., 2017, Sen et al., 2019]. The model is trained with a predictive loss and an automatic model selection criterion which exploits the upper bound of the marginal likelihood introduced in Chapter 5.

The detection module simply tests the hypothesis that the prediction residual is stationary. This is done via the likelihood ratio between the distributions before and after a given point in time of the candidate anomaly time t_a, as customary, but without requiring knowledge of these distributions. Instead, we introduce a closed-form approximation of the likelihood ratio derived from an upper bound of the f-divergence between the two distributions computed directly from the prediction residuals. The sequence of likelihood ratios is aggregated over time and compared to a data-dependent adaptive threshold using the CUMSUM method.

7.1 Anomaly detection for time series data

A *time series* is an ordered sequence of data points, which can be represented as a map from a set of time indices to a vector space. We focus on discrete and regularly spaced time indices, and thus ignore literature specific to asynchronous time processes. An *anomaly* is, fundamentally, a *violation of continuity*. When analyzing time series for anomaly detection (AD), there is an underlying assumption that the mechanisms that generate the data are stationary, so there are some (unknown, latent) parameters that are constant during normal operation, and change as a result of an anomaly. Since we operate in a discrete time domain, there is no natural notion of *continuity*. Any anomaly detection system thus implicitly or explicitly *defines* a criterion for continuity, which necessarily depends on what (discriminant) function is assumed to be constant, over what window of observation, to within what level of tolerance. Different methods for time series anomaly detection can therefore be taxonomized by their choice of (i) discriminant function, (ii) continuity criterion, and (iii) optimization method to determine the tolerance threshold. It is common to use the prediction error as the discriminant [Braei and Wagner, 2020], and the likelihood ratio between the distribution of the prediction error before and after a given time instant as the continuity criterion [Yashchin, 1993]. More recent methods compute the discriminant using DNNs [Munir et al., 2019, Geiger et al., 2020, Su et al., 2019].

The method we proposed in this chapter follows these standard and well established principles, but introduces novel elements in the ingredients (i) and (ii). Specifically, our **contributions** are:

(i) A deep neural network architecture that explicitly isolates interpretable factors

such as slow trends, quasi-periodicity, and linearly predictable statistics [Oreshkin et al., 2019, Cleveland et al., 1990], and incorporates statistics from other time series as context/side information (Section 7.2). The network is trained with a predictive criterion that requires no supervision, and the prediction residual is the discriminant function on which the hypothesis of continuity is tested. Explicit regularization added to the prediction loss is used for model complexity selection (Section 7.3)

(ii) The decision function for whether a time instant t is an anomaly is based on the cumulation of likelihood ratios between the distributions of prediction residuals in (i) before and after t (Section 7.4). Since we do not know the distributions, we present a closed-form approximation of the likelihood ratio from the given samples (Section 7.4.2). This is derived from an approximation of the f-divergence between the distribution of residuals before and after a candidate anomaly time instant t_a [Nguyen et al., 2010, Liu et al., 2012], which entails a tunable parameter corresponding to the length of observation.

The resulting method, STRIC, has the advantage of separating interpretable components due to trends and seasonality without reducing the representative power of a generic architecture. The parameters of each of the layers are trained after random initialization, but both the initialization and the regularization act to bias the solution towards trends (poles of the corresponding filters close to the origin), periodicity (poles on the unit circle), and linear predictability. At initialization, the network is approximately equivalent to a multi-scale SARIMA model [Adhikari and Agrawal, 2013], which can be reliably applied out-of-the-box on most time series. However, as more data is acquired, any part of the system can be further fine-tuned in an end-to-end fashion.

Differently from previous work [Bai et al., 2018, Munir et al., 2019, Sen et al., 2019], we provide our temporal model with an interpretable structure (first module) which is similar to [Oreshkin et al., 2019]. Compared to previous works on interpretability of DNNs [Tsang et al., 2018] our architecture is the first one to explicitly solve interpretability for time series by means of TCN and regularization without assuming any prior knowledge on the data [Guen et al., 2020]. Moreover, we endow our architecture the capability to extract global statistics from other time series and locally aggregate such information as context to predict each time series (second module) [Sen et al., 2019]. The basic building block of these modules are causal convolutions [Bai et al., 2018]. Since TCNs tend to overfit, we constrain our TCN's representational power by enforcing fading memory [Zancato and Chiuso, 2021].

Our method outperforms both classical statistical methods [Braei and Wagner, 2020] and DNNs [Munir et al., 2019, Geiger et al., 2020, Su et al., 2019] on different anomaly

detection benchmarks [Laptev and Amizadeh, 2020, Lavin and Ahmad, 2015] (Section 7.5). Moreover, we show it can be employed to detect anomalous patterns on complex data such as text embeddings of newspaper articles (Figure 7.9).

Note that an anomaly, as defined above, is neither a property of a datum, nor of the underlying system that generates the data. Rather, an anomaly is a *time instant*: at that time instant, either we receive an isolated observation that is inconsistent with normal operation (outlier measurement), or a discrete change occurs in the mechanism that generates the data (change-point) that persists beyond that time instant [Geiger et al., 2020, Braei and Wagner, 2020, Basseville and Nikiforov, 1993]. A setpoint change can only be determined *post-mortem*, so setpoint change detection involves a "look-ahead" window. Our method treats these two phenomena in a unified manner, without the need to differentiate between outliers and setpoint changes, with specialized detectors for each. Once the predictor is built our method can be used online, detecting anomalies soon after occurrence and without waiting for the entire data stream to be observed.

7.2 Interpretable Residual Temporal Convolutional Architecture

In this chapter we shall be mainly focused on multi-variate time series $\{y(t)\}$ with $t \in \mathbb{Z}$ and $y(t) \in \mathbb{R}^n$, we stack observations from time t to $t+k-1$ and denote the resulting matrix as $Y_t^{t+k-1} := [y(t), y(t+1), ..., y(t+k-1)] \in \mathbb{R}^{n \times k}$. The row index refers to the dimension of the time series while the column index refers to the temporal dimension. At time t, sub-sequences containing the n_p past samples up to time $t-n_p+1$ are given by $Y_{t-n_p+1}^{t}$ (note that we include the present data into the past data), while future samples up to time $t+n_f$ are $Y_{t+1}^{t+n_f}$. We will use past data to predict future ones, where the length of past and future intervals is an hyper-parameter that is up to the user to design.

Our architecture is depicted in Figure 7.1. Its basic building blocks are causal convolutions [Bai et al., 2018], with a fixed-size 1-D kernel with input elements from time t and earlier. Rather than initializing the convolutional filters randomly, as commonly done in deep learning, we initialize the weights so that each layer is biased to attend at different components of the signal, as explained in the following.

Linear module. The first (*linear*) module is interpretable and captures local statistics of a given time series by means of a cascade of learnable linear filters. Its first layer models and removes slow-varying components in the input data using causal Hodrick Prescott (HP) filters [Ravn and Uhlig, 2002]. The second layer models and removes

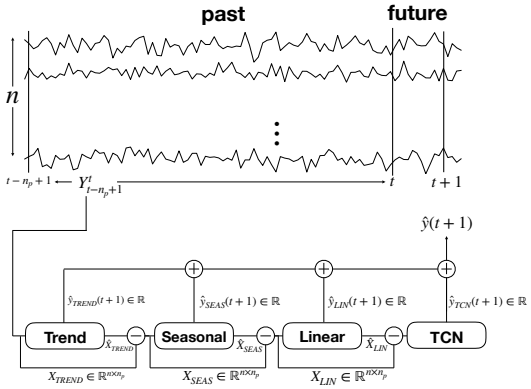


Figure 7.1: STRIC predictor architecture.

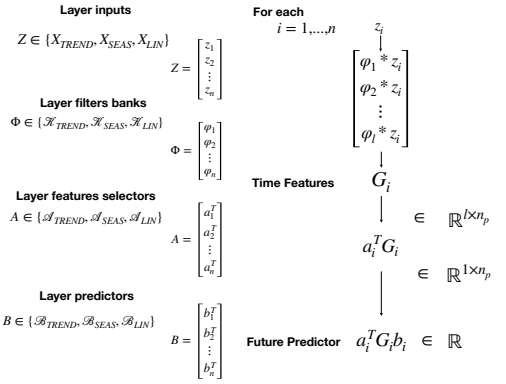


Figure 7.2: **Interpretable blocks structure**. Time features are extracted independently for each time series (see Appendix D.1 for more details).

periodic components using filters with periodic impulse responses. Finally, the third layer implements a linear stationary filter bank.

We treat the impulse responses parameters of the linear filters as trainable parameters. We initialize the trend layer with different HP smoothness degrees while we initialize the periodic and linear-stationary layers with randomly chosen poles [Farahmand et al., 2017] (on the unit circle and within the unit circle).

Remark 7.1. The number of linear filters used in each layer is chosen so that their poles provide a good covering of the unit circle (see [Farahmand et al., 2017] for more details).

Non-linear module. The second (*non-linear*) module aggregates global statistics from different time series (Section 5.1.2). It takes as input the prediction residual of the linear module and outputs a matrix $G(Y_{t-n_p+1}^t) \in \mathbb{R}^{l \times n_p}$ where l is the number of output features extracted by the TCN model. The row $(G(Y_{t-n_p+1}^t)^T)_j$ with $j \in [n_p]$ of the non-linear features is computed using data from $t - n_p + j$ (due to the internal structure of a TCN network, see Section 5.1.2). We build a linear predictor on top of $G(Y_{t-n_p+1}^t)$ for each single time series independently: the predictor for the i-th time series is given by: $(\hat{y}_{\text{TCN}}(t+1))_i := a_i^T G(Y_{t-n_p+1}^t) b_i$ where $a_i \in \mathbb{R}^l$ and $b_i \in \mathbb{R}^{n_p}$ Eq. (5.3). Note a_i combines features (uniformly in time) so that we can interpret it as a feature selector. While b_i aggregates relevant features across time indices to build the one-step ahead predictor (see Appendix D.1 for more details).

Note that the third layer of the linear module is a superset of preceding ones, and

the non-linear module is a superset of the whole linear module. While this makes the model redundant, we show that this design, coupled with proper initialization and regularization, improves the reliability and intepretability of the final model. We improve filters optimization by sharing their kernel parameters among different time series so that global information (e.g., common trend shapes, periodicities, or linear stationary components) can be extracted. In the following we shall exploit the automatic complexity selection mechanism based on the upper bound on the marginal likelihood we introduced in Chapter 5, so that the non-linear model is forced to only look at the *most relevant past*.

7.3 Bayesian Automatic Complexity Selection

For simplicity, we consider for now a scalar time series so that the TCN-based future predictor can be written as: $\hat{y}_{\text{TCN}}(t+1) := a^T G(Y_{t-n_p+1}^t)b = \hat{X}_{\text{TCN}}b$ where $\hat{X}_{\text{TCN}} \in \mathbb{R}^{1\times n_p}$ is the output of the TCN block. The predictor depends on non-linear statistics $\hat{X}_{\text{TCN}}$ w.r.t. the past window $Y_{t-n_p+1}^t$ (the memory of the predictor). Ideally, n_p should be large enough to capture the memory of the system, so that the predictor can approximate arbitrarily well the "optimal" predictor and should *not* be chosen to face a bias-variance trade-off. In practice, too flexible feature extractors (such as TCNs or plain DNNs) are prone to overfitting (Section 5.2 and Section 5.3). Therefore, some regularization is needed to control model complexity and benefit from having a large memory window. We shall employ the regularized loss inspired by Bayesian arguments which we introduced in Section 5.3. Fading regularization allows us to use an architecture with a "large enough" past horizon n_p (i.e., larger than the true system memory) and automatically select the *relevant past* to avoid overfitting. Such information is exposed to the user through an interpretable parameter λ (Section 5.3) that directly measures the *relevant time scale* of the signal.

The model structure we consider is linear in b and we can therefore stack the predictions of each available time index t to get the following linear predictor on the whole future data available: $\hat{Y}_{b,W} = F_W b$ where $F \in \mathbb{R}^{n_f\times n_p}$ is obtained by stacking $\hat{X}_{\text{TCN}}(Y_{i-n_p+1}^i)$ for $i = t, ..., t+n_f-1$. In this way Theorem 5.1 can be directly applied (with $b = \theta$ together with the normalization scheme in Section 5.3.2).

7.4 A novel non-parametric anomaly detector

In this section, we present our anomaly detection method based on a variational approximation of the likelihood ratio between two windows of model residuals. Our temporal

residual architecture model produces the prediction residual after removing trends, periodicity, and stationary (linear) components, as well as considering global covariates. Such a prediction residual is used to test the hypothesis that the time instant t is anomalous by comparing its statistics before t on temporal windows of length n_p and n_f. The detector is based on the likelihood ratios aggregated sequentially using the classical CUMSUM algorithm [Page, 1954, Yashchin, 1993]. CUMSUM, however, requires knowledge of the distributions, which we do not have.

The problem of estimating the densities is hard [Vapnik, 1998] and generally intractable for high-dimensional time series [Liu et al., 2012]. We circumvent this problem by directly estimating the likelihood ratio with a variational characterization of f-divergences [Nguyen et al., 2010] which involves solving a convex risk minimization problem in closed form.
In Section 7.4.1, we summarize the standard material necessary to derive our new estimator and the resulting anomaly test. The overall method is entirely unsupervised, and users can tune the scale parameter (corresponding to the window of observation when computing the likelihood ratios) and the coefficient of CUMSUM, depending on the application and desired operating point in the trade-off between missed detection and false alarms.

7.4.1 Likelihood Ratios and CUMSUM

CUMSUM [Page, 1954] is a classical Sequential Probability Ratio Test [Basseville and Nikiforov, 1993, Liu et al., 2012] of the null hypothesis H_0 that the data after the given time c comes from the same distribution as before, against the alternative hypothesis H_c that the distribution is different. We denote the distribution before c as p_p and the distribution after the anomaly at time c as p_f.

If the density functions p_p and p_f were known (we shall relax this assumption later), the optimal statistic to decide whether a datum $y(i)$ is more likely to come from one or the other is the likelihood ratio $s(y(i))$. According to the Neyman-Pearson lemma: H_0 is accepted if the likelihood ratio $s(y(i))$ is less than a threshold chosen by the operator, otherwise H_c is chosen. In our case, the competing hypotheses are $H_0 =$ no anomaly has happened and $H_c =$ an anomaly happened at time c. We denote with p_{H_0} and p_{H_c} the PDFs under H_0 and H_c so that: $p_{H_0}(Y_1^K) = p_p(Y_1^K)$ and $p_{H_c}(Y_1^{c-1}) = p_p(Y_1^{c-1})$, $p_{H_c}(Y_c^K \mid Y_1^{c-1}) = p_f(Y_c^K \mid Y_1^{c-1})$. Therefore the likelihood ratio is:

$$\Omega_c^t := \frac{p_{H_c}(Y_1^t)}{p_{H_0}(Y_1^t)} = \frac{p_p(Y_1^{c-1})p_f(Y_c^t \mid Y_1^{c-1})}{p_p(Y_1^t)} = \frac{p_f(Y_c^t \mid Y_1^{c-1})}{p_p(Y_c^t \mid Y_1^{c-1})} = \prod_{i=c}^{t} \frac{p_f(y(i) \mid Y_1^{i-1})}{p_p(y(i) \mid Y_1^{i-1})} \quad (7.1)$$

To determine the presence of an anomaly, we can compute the cumulative sum $S_c^t := \log \Omega_c^t$ of the (log) likelihood ratios, which depends on the time c, and estimate c^* using a maximum likelihood criterion, corresponding to the detection function $h_t = \max_{1 \leq c \leq t} S_c^t$. The first instant at which we can confidently assess the presence of a change point (a.k.a. stopping time) is: $c_{\text{stop}} = \min\{t : h_t \geq \tau\}$ where τ is a design parameter that modulates the sensitivity of the detector depending on the application. The final estimate $\hat{c}$ of the true change point c^* after the detection c_{stop} is simply given by the timestamp c at which the maximum of $h_t = \max_{1 \leq c \leq t} S_c^t$ is achieved. In Appendix D.3, we provide an alternative derivation that shows the CUMSUM is a comparison of the test statistic with an adaptive threshold that keeps complete memory of past ratios. The next step is to relax the assumption of known densities, which we bypass in the next section by directly approximating the likelihood ratios to compute the cumulative sum.

Likelihood ratio estimation with Pearson divergence

The goal of this section is to tackle the problem of estimating the likelihood ratio of two general distributions p_p and p_f given samples. To do so, we leverage a variational approximation of f-divergences [Nguyen et al., 2010] whose optimal solution is directly connected to the likelihood ratio. For different choices of divergence function, different estimators of the likelihood ratio can be built. We focus on a particular divergence choice, the Pearson divergence, since it provides a direct estimate of the likelihood ratio and the variational approximation can be solved in closed form (Appendix D.4).

Proposition 7.1 (Variational approximation of likelihood ratios [Nguyen et al., 2010, Liu et al., 2012])**.** *Let $\phi := p_f/p_p$ be the likelihood ratio of the unknown distributions p_f and p_p. Let $\mathcal{F} := \{f_i : f_i \sim p_f, i = 1, ..., n_f\}$ and $\mathcal{H} := \{h_i : h_i \sim p_p, i = 1, ..., n_p\}$ be two sets containing n_f and n_p samples i.i.d. from p_f and p_p respectively. An empirical estimator $\hat{\phi}$ of the likelihood ratio ϕ is given by the solution to the following convex optimization problem:*

$$\hat{\phi} = \arg\min_{\phi} \frac{1}{2n_p} \sum_{i=1}^{n_p} \phi(h_i)^2 - \frac{1}{n_f} \sum_{i=1}^{n_f} \phi(f_i) \tag{7.2}$$

Proposition 7.2 (Optimal regularized likelihood ratio estimator [Liu et al., 2012, Kanamori et al., 2009])**.** *Let ϕ in Equation (7.2) belong to the Reproducing Kernel Hilbert Space Φ induced by the kernel k. Let the kernel sections be centered on the set of data $\mathcal{S}_{tr}$ and let the kernel matrices evaluated on the data from p_f and p_p be*

$K_f := K(\mathcal{F}, \mathcal{S}_{tr})$ and $K_p := K(\mathcal{H}, \mathcal{S}_{tr})$ respectively. The optimal regularized empirical likelihood ratio estimator on a new datum e is given by:

$$\hat{\phi}(e) = \frac{n_p}{n_f} K(e, \mathcal{S}_{tr}) \Big(K_p^T K_p + n_p \gamma I_{n_p+n_f} \Big)^{-1} K_f^T \mathbb{1}. \tag{7.3}$$

Remark 7.2. The estimator in Equation (7.3) is not constrained to be positive. Nonetheless, the positivity constraints can be enforced. In this case, the closed form solution is no longer valid but the problem remains convex. Note the kernel sections centers are $\mathcal{S}_{tr} = \{\mathcal{H}, \mathcal{F}\}$.

7.4.2 Subspace likelihood ratio estimation and CUMSUM

In this section, we present our anomaly detector estimator. We test for an anomaly in the data Y_1^t by looking at the prediction residuals E_1^t Eq. (5.1), which provide a sufficient representation of Y_1^t (Appendix D.5). We therefore assume we are given the prediction errors E_1^t obtained from the time series predictor that is assumed to model the normal behaviour of the time series (Section 7.2). This guarantees that the sequence E_1^t is white in each of its normal subsequences. On the other hand, if the model is applied to a data subsequence which contains the abnormal condition, the residuals are not white.

To apply the CUMSUM we need the likelihood ratio given in Eq. (7.1), we can express the probability density functions of the two competing hypotheses for Y_c^t as:

$$p(Y_c^t) = \prod_{i=c}^{t} p(y(i) \mid Y_c^{i-1}) = \prod_{i=c}^{t} p(e(i)) \qquad \text{normal conditions}$$

$$p(Y_c^t) = \prod_{i=c}^{t} p(y(i) \mid Y_c^{i-1}) = \prod_{i=c}^{t} p(e(i) \mid E_c^{i-1}) \qquad \text{abnormal conditions}$$

These two conditions in turn influence the log likelihood ratio test as follows: under $H_0 \implies \prod_{i=c}^{t} \frac{p_f(e(i))}{p_p(e(i))}$ while under $H_c \implies \prod_{i=c}^{t} \frac{p_f(e(i)|E_c^{i-1})}{p_p(e(i))}$. The main issue here is the numerator under H_c: the distribution of residuals changes at each time-stamp (it is a conditional distribution) and $p_f(e(i) \mid E_c^{i-1})$ is difficult to approximate (it requires the model of the fault). In the following we show that replacing $p_f(e(i) \mid E_c^{i-1})$ with $p_f(e(i))$ allows us to compute a lower bound on the cumulative sum. Such an approximation is necessary to estimate the likelihood ratio in abnormal conditions, the main downside of this approximation is that the detector becomes slower (it needs more time to reach the stopping time threshold).

Applying the independent likelihood test in a correlated setting: In Ap-

pendix D.5 we prove that treating $p_f(e(i) \mid E_c^{i-1})$ as independent random variables $p_f(e(i))$ for $i = 1, ..., t$ allows us to compute a lower bound on the log likelihood $\log \Omega_c^t$ (i.e. the cumulative sum). We denote the cumulative sum of the log likelihood ratio using independent variables as $\log \bar{\Omega}_c^t = \sum_{i=c}^t \log \frac{p_f(e(i))}{p_p(e(i))}$.

Proposition 7.3 (Lower bound on the cumulative sum)**.** *Assume a change happens at time c so that H_c is true and the following log likelihood ratio holds true: $\log \Omega_c^t = \sum_{i=c}^t \log \frac{p_f(e(i)|E_c^{i-1})}{p_p(e(i))}$. Then it holds $\log \Omega_c^t \geq \log \bar{\Omega}_c^t$.*

Estimating the likelihood ratio on the prediction residuals: We estimate the likelihood ratio of p_f and p_p on a datum e_t as $\hat{\phi}_t(e_t)$. $\hat{\phi}_t$ is obtained by applying Equation (7.3) on the past window of size $n_p + n_f$. At each time instant t, we compute the necessary kernel matrices as:

$$K_f(E_{t-n_f+1}^t, E_{t-n_p-n_f+1}^t)$$
$$K_p(E_{t-n_p-n_f+1}^{t-n_f}, E_{t-n_p-n_f+1}^t)$$

Finally, we can compute the detector function by composing the cumulative sum of the estimated likelihood ratios: $\hat{S}_c^t := \sum_{i=c}^t \log \hat{\phi}_i(e_i)$.

Remark 7.3 (Effects of the independence assumption). At time t, the likelihood ratio is estimated assuming i.i.d. data. This assumption holds if no anomaly happened but does not hold in the abnormal situation since residuals are not i.i.d. As we proved in Proposition 7.3 treating correlated variables as uncorrelated provides a lower bound on the actual cumulative sum of likelihood ratios. In practice, for a fixed threshold, this means that the detector cumulates less and therefore it requires more time to reach the threshold value.

How do n_p and n_f affect our detector? The choice of the windows length (n_p and n_f) is fundamental and highly influences the likelihood estimator. Using small windows makes the detector highly sensible to both point and sequential outliers, while larger windows are better suited to estimate only sequential outliers. We now assume $n_p = n_f$ and study how both small and large values affect the behaviour of our detector in simple working conditions.

In Figure 7.3 and Figure 7.4 we compute the cumulative sum of log likelihood ratios estimated from data on equally sized windows. Intuitively any local minimum after a "large" (depending on the threshold τ) increase of the cumulative sum is a candidate abnormal point.

In Figure 7.5 and Figure 7.6 we compare the cumulative sum of estimated likelihood

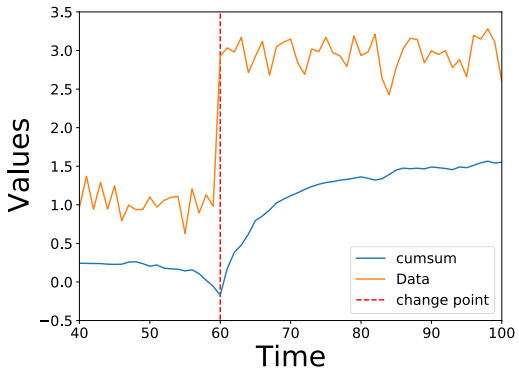


Figure 7.3: **Change point**: Cumulative sum (blue) obtained with our method in a synthetic example. We use the cumulative sum of estimated likelihood ratios on data in which a change point is present at $t = 60$. We use $n_p = n_f = 20$ and kernel length scale=0.2

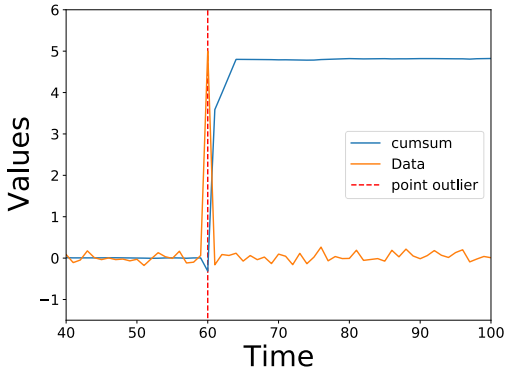


Figure 7.4: **Point anomaly**: Cumulative sum (blue) obtained with our method in a synthetic example. We use the cumulative sum of estimated likelihood ratios on data in which a point outlier is present at $t = 60$. We use $n_p = n_f = 2$ and kernel length scale=2.

ratios on data in which both sequential and point outliers are present. In particular we highlight that large window sizes n_p and n_f are usually not able to capture point anomalies Figure 7.5 while using small window sizes allow to detect both (at the expenses of a more sensitive detector) Figure 7.6.

Remark 7.4. The choice of the windows length (n_p and n_f) is fundamental and highly influences the likelihood estimator. Using small windows makes the detector highly sensible to point outliers, while larger windows are better suited to estimate sequential outliers.

7.5 Experiments

In this section, we show STRIC can be successfully applied to detect anomalous behaviours on different anomaly detection benchmarks. In particular, we test our novel residual temporal structure, the automatic complexity regularization and the anomaly detector on the following datasets: Yahoo [Laptev and Amizadeh, 2020], NAB [Lavin and Ahmad, 2015], CO2 Dataset[1] (Appendix D.6). To show the general applicability and flexibility of our method, we test STRIC on the challenging task of detecting anomalous events in time series generated from embeddings of articles from the New

[1] https://www.kaggle.com/txtrouble/carbon-emissions

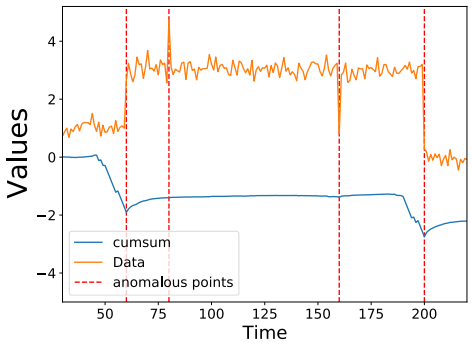


Figure 7.5: **Large n_p and n_f**: Cumulative sum (blue) obtained with our method in a synthetic example. We use the cumulative sum of estimated likelihood ratios on data which contain both change points ($t = 60$ and $t = 200$) and point outliers ($t = 80$ and $t = 160$). We use $n_p = n_f = 20$ and kernel length scale=1.

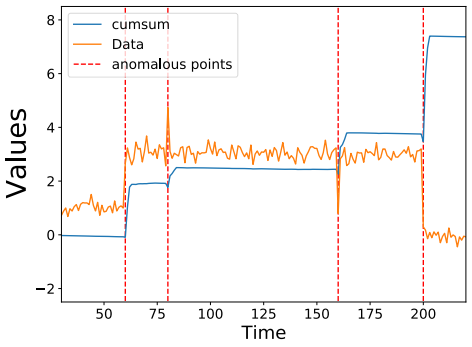


Figure 7.6: **Small n_p and n_f**: Cumulative sum (blue) obtained with our method in a synthetic example. We use the cumulative sum of estimated likelihood ratios on data which contain both change points ($t = 60$ and $t = 200$) and point outliers ($t = 80$ and $t = 160$). We use $n_p = n_f = 3$ and kernel length scale=5.

York Times [Sandhaus, 2008]. See Appendix D.7 for a description of the data normalization and experimental setup.

STRIC interpretable time series decomposition. In Figure 7.7 we show STRIC's interpretable decomposition. We report predicted signals (first row), estimated trends (second row) and seasonalities (third row) for different datasets. For all experiments, we plot both training data (first 40% of each time series) and test data. Note the interpretable components of STRIC generalize outside the training data, thus making STRIC work well on non-stationary time series (e.g. where the trend component is non negligible and typical non linear models overfit, see Section D.7.2).

Remark 7.5 (Additive time series decomposition). Our residual framework assumes that the input time series is the addition of trend, seasonality and residual series. However, it might be possible that the three components are not linearly dependent (e.g. a multiplicative decomposition might be necessary). In these circumstances it would simply be necessary to pre-process the time-series by means of a logarithmic function. This is a standard approach used for time series decomposition techniques and it is indeed applicable to our method. We point out that the current version of our algorithm does not automatically discover which type of decomposition to use and it is the user's responsibility to recognize whether the additive or multiplicative

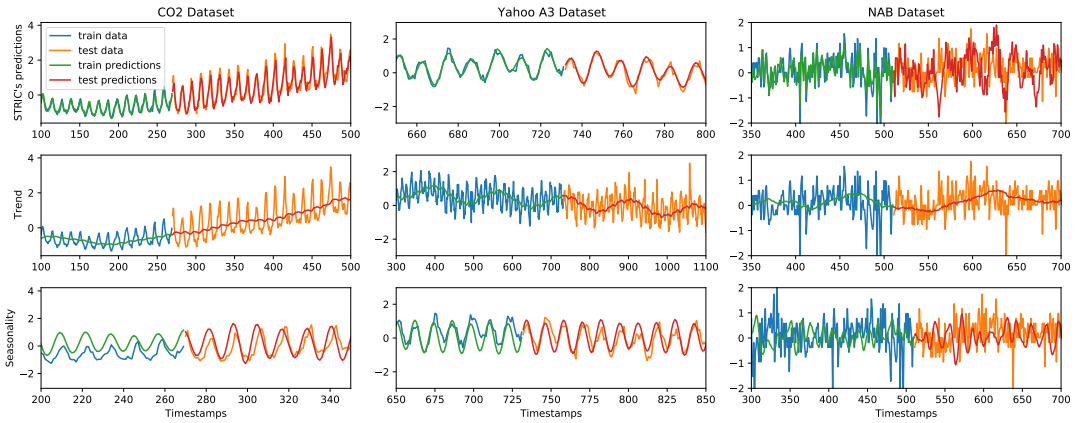


Figure 7.7: We test STRIC time series intepretability on different datasets (columns). In each panel we are displaying both training data and test data a reported (see colors). **First row**: STRIC time series predictor (output of non-linear module). **Second row**: Trend components extracted by the interpretable blocks. **Third row**: Seasonal components extracted by the interpretable blocks.

decomposition is better suited, this is not different to standard practice on STL based methods [Cleveland et al., 1990].

Ablation study. We now compare the prediction performance of a general TCN model with our STRIC method in which we remove the interpretable module and the fading regularization one at the time. In Table 7.1, we report the training and test RMSE prediction errors for different datasets while keeping all the training parameters the same (e.g. training epochs, learning rates etc.) and model parameters (e.g. $n_b = 100$). Overall, the addition of the linear interpretable model before the TCN slightly improves the test error while not changing the training error. We note this effect is more visible on Yahoo A2, A3, A4, mainly due to the non-stationary nature of these datasets and the fact that TCNs do not easily approximate trends [Braei and Wagner, 2020] (we further tested this in Section D.7.1). While STRIC generalization is always better than a standard TCN model and STRIC's ablated components, we note that applying fading memory regularization alone to a standard TCN might not be enough: this highlights that the benefits of combining the linear module and the fading regularization together are not a trivial "sum of the parts". Consider for example Yahoo A1: STRIC achieves 0.62 test error, the best ablated model (TCN + Linear) 0.88 while TCN + Fading does not improve over the baseline TCN. A similar observation holds for the CO2 Dataset. We believe fading regularization might not be beneficial (nor detrimental) for times series containing purely periodic components

Table 7.1: **Ablation study on the RMSE of prediciton errors**: We compare a standard TCN model with our STRIC predictor and some variation of it (using the same train hyper-parameters).

	TCN		**TCN + Linear**		**TCN + Fading**		**STRIC pred**	
Datasets	Train	Test	Train	Test	Train	Test	Train	Test
Yahoo A1	**0.10**	0.92	**0.10**	0.88	0.44	0.92	0.43	**0.62**
Yahoo A2	**0.11**	0.82	0.13	0.35	0.20	0.71	0.14	**0.30**
Yahoo A3	**0.13**	0.43	0.16	**0.22**	0.15	0.40	0.19	**0.22**
Yahoo A4	**0.15**	0.61	0.19	0.35	0.17	0.55	0.23	**0.24**
CO2 Dataset	**0.14**	0.62	0.15	0.45	0.18	0.61	0.33	**0.41**
NAB Traffic	**0.03**	1.06	0.04	1.00	0.62	0.93	0.83	**0.74**
NAB Tweets	**0.18**	1.02	0.20	0.98	0.47	0.83	0.70	**0.77**

which correspond to infinite memory systems (systems with unitary fading coefficient). In such cases the interpretable module is essential in removing the periodicities and providing the regularized non-linear module (TCN + Fading) with an easier to model residual signal. We refer to Figure 7.7 (first column) for a closer look on a typical time series in CO2 dataset, note it contains a periodic component which is captured by the seasonal part of the interpretable model. To conclude, our proposed fading regularization has (on average) a beneficial effect in controlling the complexity of a standard TCN model and reduces its generalization gap ($\approx 40\%$ reduction). Moreover, coupling fading regularization with the interpretable module guarantees the best generalization overall.

Automatic complexity selection. In Figure 7.8, we test the effects of our automatic complexity selection (fading memory regularization) on STRIC. We compare STRIC with a standard TCN model and STRIC without regularization as the memory of the predictor increases. The test error of STRIC is uniformly smaller than a standard TCN (without interpretable blocks nor fading regularization). Adding interpretable blocks to a standard TCN improves generalization for a fixed memory w.r.t. standard TCN, but gets worse (overfitting occurs) as soon as the available past data horizon increase. On the other hand, the generalization gap of STRIC does not deteriorate as the memory of the predictor increases (see Section D.7.1 for a comparison with other metrics).

Anomaly detection. While recent works show deep learning models might not be well suited to solve AD on standard anomaly detection benchmarks [Braei and Wagner, 2020], we prove deep models can be effective, provided they are used with a proper inductive bias and regularization. In Table 7.2, we compare STRIC against statistical

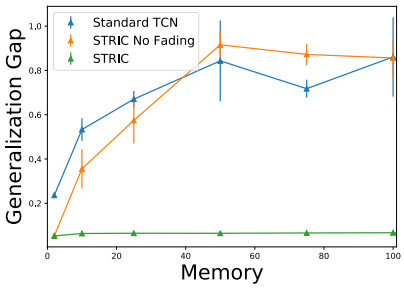


Figure 7.8: **Automatic complexity selection**: Fading memory regularization preserves generalization gap as the memory of the predictor increases on NAB Tweets.

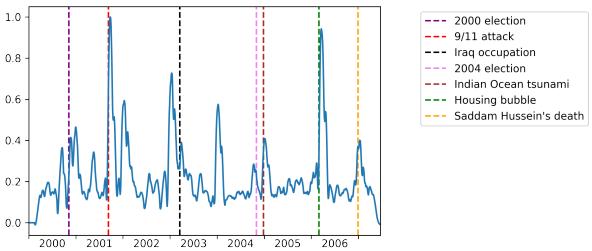


Figure 7.9: **Anomaly score on the New York Times dataset.** Our method finds anomalies in a complex time series consisting of the BERT embedding of articles from the New York Times. Peaks in the anomaly score correspond to historical events that sensibly changed the content of the news cycle.

and deep learning based anomaly detection methods. Our experiments follow the experimental setup and evaluation criteria used in [Braei and Wagner, 2020] and [Munir et al., 2019]. STRIC outperforms most of the statistical based and DNNs based methods. Note no other method performs consistently (across different datasets) as good as STRIC. In particular, STRIC achieves the greatest increase in F1 score on Yahoo A3 compared to other methods based on DNNs. In Appendix D.7, we show this is mainly due to STRIC's predictor. In fact, most of the time series in Yahoo A3 are characterized by trend components and seasonalities which STRIC's interpretable predictor can easily model (see Section D.7.2). In Appendix D.7, we show some ablation studies on the effects of the hyper-parameters of STRIC on its performance. In particular, we find out that STRIC is highly affected by the choice of the length of the windows used to estimate the likelihood ratio, while not being much sensitive to the choice of the memory of the predictor (Section D.7.1 and Remark 7.4). Interestingly, STRIC does not achieve the optimal AUC compared to linear models on Yahoo A4. The ability of linear models to outperform non-linear ones on Yahoo A4 is known in literature (e.g. in [Geiger et al., 2020] any non-linear model is outperformed by AR/MA models of the proper complexity). The main motivation of this is the fact that modern (non-linear) methods tend to overfit on Yahho A4 and therefore generalization is usually low. We highlight that, thanks to fading regularization and model architecture, STRIC does not exhibit severe overfitting despite having larger complexity than a SOTA linear model on Yahoo A4. To conclude, we believe that STRIC merges both the advantages of simple and interpretable linear models and the flexibility of non-linear ones while discounting

the major drawbacks: lack of flexibility of linear models and lack of interpretability and overfitting of non-linear ones (see Appendix D.8 for a more in depth discussion).

Anomaly detection on the New York Times dataset. We test STRIC on a time series consisting of BERT embeddings [Devlin et al., 2019] of New York Times articles [Sandhaus, 2008] from 2000 to 2007. We set $n_p = n_f = 30$ days, to be able to detect change-point anomalies that altered the normal distribution of news articles for a prolonged period of time. Without any human annotation, STRIC is able to detect major historical events such as the 9/11 attack, the 2004 Indian Ocean tsunami, and U.S. elections (Figure 7.9). Additional details and comparison with a baseline model built on PCA are given in Section D.8.1.

7.6 Discussions and conclusions

In this chapter we show how to build an anomaly detector based on an interpretable deep learning architecture. Our anomaly detection scheme is composed by two main building blocks: a time series predictor and an anomaly detector that operates on the prediction residuals. The three novel ingredients that play a key role are (i) the inductive bias on the architecture, (ii) the automatic complexity selection that enforces fading memory of the predictors, and (iii) the unsupervised estimation of the likelihood ratio, used to compute the test statistic.

Our ablation studies proved the effectiveness and impact of the different design choices. In particular, we showed that the interpretable module (which extracts trends, sesonalities, and linear stationary components) helps STRIC generalize correctly on non-stationary time series on which standard deep models (such as TCNs) overfit [Braei and Wagner, 2020]. Moreover, we showed that the fading regularization alone can improve the generalization error up to $\approx 40\%$ over standard TCN models.

The unsupervised estimation of the likelihood ratio test statistic allows us to extend CUMSUM-type algorithms to the realistic situation where the data distribution (nominal and faulty) is unknown. In addition, our framework allows us to handle detection of both outliers and change-points, by tuning suitable parameters.

Thanks to these advances, STRIC outperforms SOTA anomaly detection methods using several evaluation criteria. Interesting open issues remain: how to tune the window lengths of the anomaly detector as a function of the signal's time scale, and how to design statistically optimal rules to calibrate the detector's threshold to trade-off false alarms and missed detections.

Table 7.2: **Comparison with SOTA anomaly detectors:** We compare STRIC with other anomaly detection methods on the experimental setup and the same evaluation metrics proposed in [Braei and Wagner, 2020, Munir et al., 2019]. The baseline models are: MA, ARIMA, LOF [Shen et al., 2020], LSTM [Braei and Wagner, 2020, Munir et al., 2019], Wavenet [Braei and Wagner, 2020] , Yahoo EGADS [Munir et al., 2019] , GOAD [Bergman and Hoshen, 2020], OmniAnomaly [Su et al., 2019], Twitter AD [Munir et al., 2019], TanoGAN [Bashar and Nayak, 2020], TadGAN [Geiger et al., 2020] , DeepAR [Flunkert et al., 2017] and DeepAnT [Munir et al., 2019] . STRIC outperforms most of the other methods based on statistical models and based on DNNs. See Table D.4 for the same table obtained by looking at the relative performance w.r.t. STRIC.

	F1-score	**Yahoo A1**	**Yahoo A2**	**Yahoo A3**	**Yahoo A4**	**NAB Tweets**	**NAB Traffic**
Models	ARIMA	0.35	0.83	0.81	**0.70**	0.57	0.57
	LSTM.	0.44	0.97	0.72	0.59		
	Yahoo EGADS	0.47	0.58	0.48	0.29		
	OmniAnomaly	0.47	0.95	0.80	0.64	0.69	0.70
	Twitter AD	**0.48**	0	0.26	0.31		
	TanoGAN	0.41	0.86	0.59	0.63	0.54	0.51
	TadGAN	0.40	0.87	0.68	0.60	0.61	0.49
	DeepAR	0.27	0.93	0.47	0.45	0.54	0.60
	DeepAnT	0.46	0.94	0.87	0.68		
	STRIC (ours)	**0.48**	**0.98**	**0.89**	0.68	**0.71**	**0.73**

	AUC	**Yahoo A1**	**Yahoo A2**	**Yahoo A3**	**Yahoo A4**	**NAB Tweets**	**NAB Traffic**
Models	MA	0.8681	0.9942	0.9942	**0.9864**		
	ARIMA	0.8730	0.9891	0.990	0.9709		
	LOF	0.9037	0.9011	0.6405	0.6403	0.4906	0.4283
	Wavenet	0.8239	0.7614	0.5796	0.5924		
	LSTM	0.8121	0.7348	0.5781	0.5891		
	GOAD	0.8933	0.9212	0.8879	0.866	0.5724	0.6414
	DeepAnT	0.8976	0.9614	0.9283	0.8597	0.5542	0.6371
	STRIC (ours)	**0.9308**	**0.9999**	**0.9999**	0.9348	**0.6578**	**0.6849**

IV

Conclusions

8 Conclusions

The main questions posed in the thesis are:

1. What is the connection between learning dynamics and generalization of over-parametrized DNNs?

2. How does modern deep learning theory compares with standard practice?

3. Can we exploit domain prior knowledge and explicit regularization to improve DNNs learning and interpretablity?

To answer both the first and the second questions we exploited the theory of the Neural Tangent Kernel, which characterizes the learning dynamics of over-parametrized DNNs. In Chapter 3 and Chapter 4 we empirically show the NTK theory still holds for finite sized modern DNNs architectures so that the linear approximation of the learning dynamics is applicable in the fine-tuning case. By means of the model linearization around initialization we study both Training Time and Generalizability of SOTA DNNs models trained with fine-tuning.

Both these works started to unblock the adoption of a real-world Computer Vision AutoML system, nonetheless our theory has a broader applicability range, e.g. it can be applied to time series without modifications. Typically, in AutoML systems users fine-tune models selected from a large model zoo testing hundreds of combinations of different architectures, pre-training sets and hyper-parameters, but are reluctant to do so without visibility of the expected rough order of magnitude cost of the training. We focus on fine-tuning since it is faster and typically performs better than training from scratch. For these reasons, it is generally the choice in AutoML systems, where users pay by the hour.

Our work is an enabler of large-scale AutoML, which we expect will further foster academic research in the years ahead. The main contributions of this part of thesis are the methods we propose, which enable a cost estimate and allow reducing a large search space to fit a user's budget, a small step toward better accessibility and democratization of ML. Moreover, our results also go towards better understanding of the functioning of deep networks, so in that sense our results are contributing to improve interpretability of deep learning, in a broad sense.

To answer the last question we restrict our scope to time series domain. In particular, we introduce a novel regularization scheme for DNNs temporal models which only assumes the fading memory assumption on the underlying system or time series to be modeled. In Chapter 5, Chapter 6 and Chapter 7 we show how to take advantage of such prior knowledge to design architectures and explicit regularization schemes which foster both generalization and interpretability. The flexibility of our method makes it suitable for different tasks related to time series modeling: we specifically looked at non-linear system identification in Chapter 6 and multivariate time series anomaly detection in Chapter 7. In both cases we proved Bayesian automatic complexity selection criteria can be applied to DNNs models on time series data.

Summary of Contributions

8.0.1 Training Time Prediction

To summarize, the main contributions of Chapter 3 are:

1. We present both a qualitative and quantitative analysis of the fine-tuning Training Time as a function of the Gram-Matrix Θ of the gradients at initialization (empirical NTK matrix).
2. We show how to reduce the cost of estimating the matrix Θ using random projections of the gradients, which makes the method efficient for common architectures and large datasets.
3. We introduce a method to estimate how much longer a network will need to train if we increase the size of the dataset without actually having to see the data (under the hypothesis that new data is sampled from the same distribution).
4. We test the accuracy of our predictions on off-the-shelf state-of-the-art models trained on real datasets. We are able to predict the correct training time within a 20% error with 95% confidence over several different datasets and hyperparameters at only a small fraction of the time it would require to actually run the training (30-45x faster in our experiments).

In particular, we have shown that we can predict with a 13-20% accuracy the time that it will take for a pre-trained network to reach a given loss, in only a small fraction of the time that it would require to actually train the model. We do this by studying the training dynamics of a linearized version of the model – using the SDE in Eq. (3.1) – which, being in the smaller function space compared to parameters space, can be solved numerically. We have also studied the dependency of training time from pre-training and hyper-parameters (Section 3.3.1), and how to make the computation feasible for larger datasets and architectures (Section 3.4). Moreover, we have observed that our method yields predictions that have lower accuracy on some tasks rather than others, for instance it has lower accuracy on texture-based tasks than object classification. However, since we consider datasets as a whole, prediction inaccuracies do not impact any particular cohort or segment of the data.

While we do not necessarily expect a linear approximation around a random initialization to hold during training of a real (non wide) network, we exploit the fact that when using a pre-trained network the weights are more likely to remain close to initialization [Mu et al., 2020], improving the quality of the approximation.

We believe that important avenues for further research can be focused on extending the applicability of our method to other DNN architectures and on improving Training Time predictions by considering feedback from training. Recently, some authors [Yang, 2020] extended the NTK theory to a larger class of architectures (e.g. Transformers, RNNs), we believe the extension of our Training Time prediction algorithm to such models is feasible and could be very impactful on the community. Moreover we note that the procedure described in Chapter 3 only relies on information available at initialization and does not require any feedback from the actual training. Hence we believe that exploiting training feedback (e.g. gradients updates) with the goal of improving the prediction is an interesting research direction.

8.0.2 Model Selection

Fine-tuning using model zoo is a simple method to boost accuracy. In Chapter 4 we show that while a model zoo may have modest gains in the high-data regime, it outperforms Imagenet experts networks in the low-data regime. Moreover we show our model selection saves the cost of brute-force fine-tuning and makes model zoos viable in practice. In particular, in Chapter 4 by performing fine-tuning and model selection on our benchmark, we discover the following:

1. We show (Fig. 4.1 and Fig. 4.2) that fine-tuning models in the model zoo can outperform the standard method of fine-tuning with Imagenet pre-trained ar-

chitectures and HPO. We obtain better fine-tuning than Imagenet expert with, both model zoo of single-domain experts (Fig. 4.3) and multi-domain experts (Fig. 4.4). While in the high-data regime using a model zoo leads to modest gains, it improves accuracy in the low-data regime.

2. For any given target task, we show that only a small subset of the models in the zoo lead to accuracy gain (Fig. 4.3). In such a scenario, brute-force fine-tuning all models to find the few that improve accuracy is wasteful. Fine-tuning with all our single-domain experts in the model zoo is 40× more compute intensive than fine-tuning an Imagenet Resnet-101 expert in Table 4.3.
3. Our LGC model selection, and particularly its approximation LFC, can find the best models from which to fine-tune without requiring an expensive brute-force search (Table 4.3). With only 3 selections, we can select models that show gain over Imagenet expert (Fig. 4.5). Compared to Domain Similarity [Cui et al., 2018], RSA [Dwivedi and Roig, 2019] and Feature Metrics [Ueno and Kondo, 2020], our LFC score can select the best model to fine-tune in fewer selections, and it shows the highest ranking correlation to the fine-tuning test accuracy (Fig. 4.7) among all model selection methods.

Both the empirical results in Chapter 3 and Chapter 4 only consider Computer Vision tasks, nonetheless our framework is general enough to be applied to other domains too (e.g. Time Series) provided feedforward and highly overparametrized DNNs are used. Under these assumptions we believe an interesting, yet incremental future avenue would be to empirically validate our methods on AutoML systems that are specifically designed for time series data. Furthermore, we believe that exploiting novel extensions of the NTK theory [Yang, 2020] to a broader class of DNNs is an interesting research direction which can unlock and reduce the cost of general AutoML systems for a broader class of Machine Learning problems.

8.0.3 Fading Memory for Non-Linear System Identification

In Chapter 6 we show that overparametrized DNNs without a proper inductive bias and regularization fail to solve non-linear system identification benchmarks. We overcome such a limitation introducing both a new architecture inspired by fading memory systems and a new regularized loss inspired by Bayesian arguments which in turn allows for automatic complexity selection based on the observed data. We showed when DNN based parametric architectures are good alternatives to state of the art non-parametric models for modelling non-linear systems (mid-large data regime). Moreover we proved

our method does not suffer from typical non-parametric models limitations on large dataset sizes and favourably scales with the number of samples.

We leave to future work the design of optimization schemes which could improve convergence speed (e.g. using adaptive learning rates algorithms other than Adam and other stochastic optimization methods designed to improve DNN convergence and generalization).

8.0.4 Interpretable Residual Temporal Convolutional Networks

In Chapter 7 we show how to build an anomaly detector based on a deep learning architecture that is both interpretable and reliable. Our anomaly detection scheme is composed by two main building blocks: a time series predictor and an anomaly detector that operates on the prediction residuals. The three novel ingredients that play a key role are (i) the inductive bias on the architecture, (ii) the automatic complexity selection that enforces fading memory of the predictors, and (iii) the unsupervised estimation of the likelihood ratio, used to compute the test statistic.

Our ablation studies proved the effectiveness and impact of the different design choices. In particular, we showed that the interpretable module (which extracts trends, sesonalities, and linear stationary components) helps STRIC generalize correctly on non-stationary time series on which standard deep models (such as TCNs) overfit [Braei and Wagner, 2020]. Moreover, we showed that the fading regularization alone can improve the generalization error up to $\approx 40\%$ over standard TCN models.

The unsupervised estimation of the likelihood ratio test statistic allows us to extend CUMSUM-type algorithms to the realistic situation where the data distribution (nominal and faulty) is unknown. In addition, our framework allows us to handle detection of both outliers and change-points, by tuning suitable parameters.

Thanks to these advances, STRIC outperforms SOTA anomaly detection methods using several evaluation criteria. Interesting open issues remain: how to tune the window lengths of the anomaly detector as a function of the signal's time scale, and how to design statistically optimal rules to calibrate the detector's threshold to trade-off false alarms and missed detections.

Appendices

A

Training Time Prediction

A.1 Target datasets

Dataset	Number of images	Classes	Mean samples per class	Imbalance factor
cifar10 [Krizhevsky, 2009]	50000	10	5000	1
cifar100 [Krizhevsky, 2009]	50000	100	500	1
cub200 [Welinder et al., 2010]	5994	200	29.97	1.03
fgvc-aircrafts [Maji et al., 2013]	6667	100	66.67	1.02
mit67 [Quattoni and Torralba, 2009]	5360	67	80	1.08
opensurfacesminc2500 [Bell et al., 2015]	48875	23	2125	1.03
stanfordcars [Krause et al., 2013]	8144	196	41.6	2.83

Table A.1: Training Time Target datasets.

A.2 Additional Experiments

A.2.1 Effective learning rate

In Section 3.3.1 we note that as long as the effective learning rate $\tilde{\eta} = \eta/(1-m)$ remains constant, runs with different learning rate η and momentum m will have similar learning curve. We show a formal derivation in Appendix A.4. In Figure A.1 we show additional experiments, similar to Figure 3.2, on several other datasets to further confirm this point.

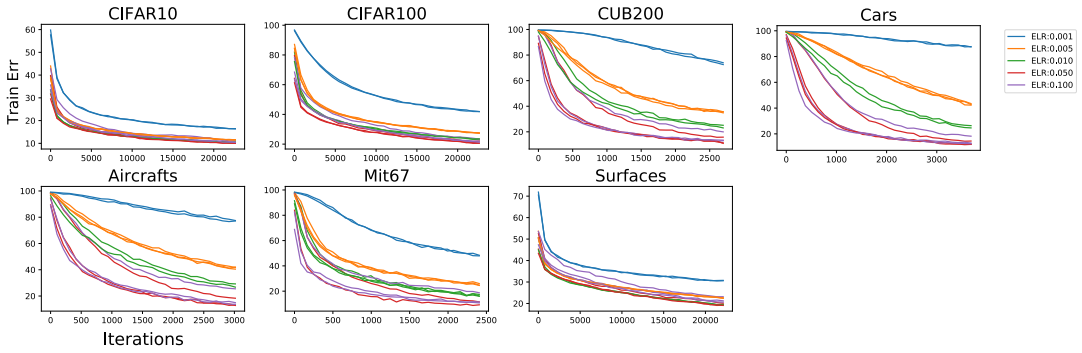


Figure A.1: **Additional experiments on the effective learning rate.** We show additional plots showing the error curves obtained on different datasets using different values of the effective learning rate $\tilde{\eta} = \eta/(1-m)$, where η is the learning rate and m is the momentum. Each line is the observed error curve of a model trained with a different learning rate η and momentum m. Lines with the same color have the same ELR $\tilde{\eta}$, but each has a different η and m. As we note in Section 3.3.1, as long as $\tilde{\eta}$ remains the same, training dynamics with different hyper-parameters will have similar error curves.

A.2.2 Different training time definitions

Our method to predict training time can be extended to other training time definitions. We started defining training time as the first time the (smoothed) loss is below a given threshold (which we then normalized w.r.t. the total computational budget allowed, see Section 3.6). Similarly one can define training time as the first time training error decreases by less than ϵ over the last T epochs. With this definition we achieve 13% avg. relative prediction error and 0.94 Pearson's correlation between predicted and ground-truth time in the same experimental setting we used in Section 3.6.

A.3 Training Time prediction on larger datasets

In Section 3.4 we suggest that, in the case of MSE loss, it is possible to predict the training time on a large dataset using a subset of the samples. To do so we leverage the fact that the eigenvalues of Θ follows a power-law [Fan and Wang, 2020] which is independent on the size of the dataset for large enough sizes (see Figure 3.4, right). More precisely, from Proposition 3.2, we know that given the eigenvalues λ_k of Θ and

the projections $p_k = (\delta y \cdot v_k)^2$ it is possible to predict the loss curve using

$$L_t = \sum_k p_k e^{-2\eta\lambda_k^2 t}$$

Let Θ_0 be the Gram-matrix of the gradients computed on the small subset of N_0 samples, and let Θ be the Gram-matrix of the whole dataset of size N. Using the fact that, as we increase the number of samples, the eigenvalues (once normalized by the dataset size) converge to a fixed limit (Figure 3.4, right), we estimate the eigenvalues λ_k of Θ as follow: we fit the coefficients s and c of a power law $\lambda_k = ck^{-s}$ to the eigenvalues of Θ_0, and use the same coefficients to predict the eigenvalues of Θ. However, we notice that the coefficient s (slope of the power law) estimated using a small subset of the data is often smaller than the slope observed on larger datase (note in Figure 3.4 (right) that the curves for smaller datasets are more flat). We found that using the following corrected power law increases the precision of the prediction:

$$\hat{\lambda}_k = ck^{-s+\alpha\left(\frac{N_0}{N}-1\right)}.$$

Empirically, we determined $\alpha \in [0.1, 0.2]$ to give a good fit over different combinations of N and N_0. In Figure A.2 (center) we compare the predicted eigenspectrum of Θ with the actual eigenspectrum of Θ.

The projections p_k follow a similar power-law, albeit more noisy (see Figure A.2, right), so directly fitting the data may give an incorrect result. However, notice that in this case we can exploit an additional constraint, namely that $\sum_k p_k = \|\delta y\|^2$ ($\|\delta y\|^2$ is a known quantity: labels and initial model predictions on the large dataset). Let $p_k = (\delta y \cdot v_k)^2$ and let $p'_k = (\delta y \cdot v'_k)^2$ where v_k and v'_k are the eigenvectors of Θ and Θ_0 respectively. Fix a small (w.r.t. number of data) k_0 (in our experiments, $k_0 = 100$). By convergence laws [Shawe-Taylor et al., 2005], we have that $p'_k \simeq p_k$ when $k < k_0$. The remaining tail of p_k for $k > k_0$ must now follow a power-law and also be such that $\sum_k p_k = \|\delta y\|^2$. This uniquely identify the coefficients of a power law. Hence, we use the following prediction rule for p_k:

$$\hat{p}_k = \begin{cases} p'_k & \text{if } k < k_0 \\ ak^{-b} & \text{if } k \geq k_0 \end{cases}$$

where a and b are such that $\hat{p}_{k_0} = p'_{k_0}$ and $\sum_k \hat{p}_k = \|\delta y\|^2$.

In Figure A.2 (left), we use the approximated $\hat{\lambda}_k$ and $\hat{p}_k$ to predict the loss curve on a dataset of $N = 1000$ samples using a smaller subset of $N_0 = 100$ samples. Notice that we

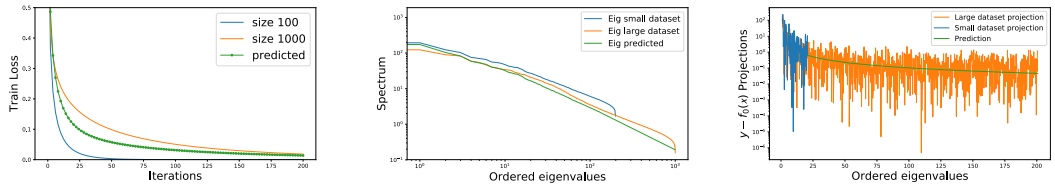


Figure A.2: **Training time prediction using a subset of the data. (Left)** We predict the loss curve on a large dataset of $N = 1000$ samples using a subset of $N_0 = 100$ samples on CIFAR10 (similar results hold for other datasets presented so far). **(Center)** Eigenspectrum of Θ computed using $N_0 = 100$ samples (orange), $N = 1000$ samples (green) and predicted spectrum using our method (blue). **(Right)** Value of the projections p_k of δy on the eigenvectors of Θ, computed at $N_0 = 100$ (orange) and $N = 1000$ (blue). Note that while they approximatively follow a power-law on average, it is much more "noisy" than that of the eigenvalues. In green we show the predicted trend using our method.

correctly predict that the convergence is slower on the larger dataset. Moreover, while training on the smaller dataset quickly reaches zero, we correctly estimate the much slower asymptotic phase on the larger dataset. Increasing both N and N_0 increases the accuracy of the estimate, since the eigenspectrum of Θ is closer to convergence: In Figure A.3 we show the same experiment as Figure A.2 with $N_0 = 1000$ and $N = 4000$. Note the increase in accuracy on the predicted curve.

A.3.1 Prediction of training time using a subset of samples

In Section 3.4 we suggest that in the case of MSE loss, it is possible to predict the training time on a large dataset using a smaller subset of samples (we discuss the details in Appendix A.3). In Figure A.3 we show the result of predicting the loss curve on a dataset of $N = 4000$ samples using a subset of $N = 1000$ samples. Similarly, in Figure A.2 (top row) we show the more difficult example of predicting the loss curve on $N = 1000$ samples using a very small subset of $N_0 = 100$ samples. In both cases we correctly predict that training on a larger dataset is slower, in particular we correctly predict the asymptotic convergence phase. Note in the case $N_0 = 100$ the prediction is less accurate, this is in part due to the eigenspectrum of Θ being still far from its limiting behaviour achieved for large number of data (see Appendix A.3).

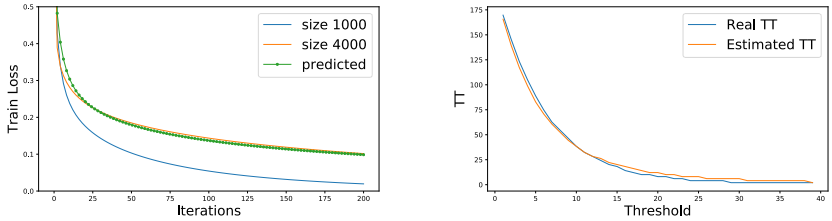


Figure A.3: **Training-time prediction using a subset of the data. (Left)** Using the method described in Appendix A.3, we predict (green) the loss curve on a large dataset of $N = 4000$ samples (orange) using a subset of $N_0 = 1000$ samples (blue). In Figure A.2 we show a similar result using a much smaller subset of $N_0 = 100$ samples. **(Right)** Corresponding estimated training time on the larger dataset at different thresholds ϵ compared to the real training time on the larger dataset.

A.4 Effective learning rate

We now show that having a momentum term has the effect of increasing the effective learning rate in the deterministic part of Eq. (3.1). A similar treatment of the momentum term is also in [Smith and Le, 2018, Appendix D]. Consider the update rule of SGD with momentum:

$$a_{t+1} = m\, a_t + g_{t+1},$$
$$w_{t+1} = w_t - \eta\, a_{t+1},$$

If η is small, the weights w_t will change slowly and we can consider g_t to be approximately constant on short time periods, that is $g_{t+1} = g$. Under these assumptions, the gradient accumulator a_t satisfies the following recursive equation:

$$a_{t+1} = m\, a_t + g,$$

which is solved by (assuming $a_0 = 0$ as common in most implementations):

$$a_t = (1 - m^t)\frac{g}{1-m}.$$

In particular, a_t converges exponentially fast to the asymptotic value $a^* = g/(1-m)$. Replacing this asymptotic value in the weight update equation above gives:

$$w_{t+1} = w_t - \eta a^* = w_t - \frac{\eta}{1-m} g = w_t - \tilde{\eta}\, g,$$

that is, once a_t reaches its asymptotic value, the weights are updated with an higher effective learning rate $\tilde{\eta} = \frac{\eta}{1-m}$. Note that this approximation remains true as long as the gradient g_t does not change much for the time that it takes a_t to reach its asymptotic value. This happens whenever the momentum m is small (since a_t will converge faster), or when η is small (g_t will change more slowly). For larger momentum and learning rate, the effective learning rate may not properly capture the effect of momentum.

A.5 Proof of propositions

A.5.1 SDE in function space for linearized networks trained with SGD

We now prove Proposition 3.1 and show how we can approximate the SGD evolution in function space rather than in parameters space. We follow the standard method used in [Hayou et al., 2019] to derive a general SDE for a DNN, then we specialize it to the case of linearized deep networks. Our notation follows [Lee et al., 2019], we define $f_{\theta_t}(\mathcal{X}) = \text{vec}(f_t(\mathcal{X})) \in \mathbb{R}^{CN}$ the stacked vector of model output logits for all examples, where C is the number of classes and N the number of samples in the training set.

Proof. To describe SGD dynamics in function space we start from deriving the SDE in parameter space. In order to derive the SDE required to model SGD we will start describing the discrete update of SGD as done in [Hayou et al., 2019].

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}^B(\theta_t) \tag{A.1}$$

where $\mathcal{L}^B(\theta_t) = \mathcal{L}(f_{\theta_t}(\mathcal{X}^B), \mathcal{Y}^B)$ is the average loss on a mini-batch B (for simplicity, we assume that B is a set of indexes sampled with replacement).

The mini-batch gradient $\nabla_\theta \mathcal{L}^B(\theta_t)$ is an unbiased estimator of the full gradient, in particular the following holds:

$$\mathbb{E}[\nabla_\theta \mathcal{L}^B(\theta_t)] = 0 \qquad \text{Cov}[\nabla_\theta \mathcal{L}^B(\theta_t)] = \frac{\Sigma(\theta_t)}{|B|} \tag{A.2}$$

Where we defined the covariance of the gradients as:

$$\Sigma(\theta_t) := \mathbb{E}\big[(g_i \nabla_{f_t(x_i)} \mathcal{L}) \otimes (g_i \nabla_{f_t(x_i)} \mathcal{L})\big] - \mathbb{E}\big[g_i \nabla_{f_t(x_i)} \mathcal{L}\big] \otimes \mathbb{E}\big[g_i \nabla_{f_t(x_i)} \mathcal{L}\big]$$

and $g_i := \nabla_w f_0(x_i)$. The first term in the covariance is the second order moment matrix while the second term is the outer product of the average gradient.

Following standard approximation arguments (see [Chaudhari and Soatto, 2017] and

references there in) in the limit of small learning rate η we can approximate the discrete stochastic equation Eq. (A.1) with the SDE:

$$d\theta_t = -\eta \nabla_\theta \mathcal{L}(\theta_t) dt + \frac{\eta}{\sqrt{|B|}} \Sigma(\theta_t)^{\frac{1}{2}} dn \tag{A.3}$$

where $n(t)$ is a Brownian motion.

Given this result, we are going now to describe how to derive the SDE for the output $f_t(\mathcal{X})$ of the network on the train set $\mathcal{X}$. Using Ito's lemma (see [Hayou et al., 2019] and references there in), given a random variable θ that evolves according to an SDE, we can obtain a corresponding SDE that describes the evolution of a function of θ. Applying the lemma to $f_\theta(\mathcal{X})$ we obtain:

$$df_t(\mathcal{X}) = [-\eta \Theta_t \nabla_{f_t} \mathcal{L}(f_t(\mathcal{X}), \mathcal{Y}) + \frac{1}{2} \text{vec}(A)] dt + \frac{\eta}{\sqrt{|B|}} \nabla_\theta f(\mathcal{X}) \Sigma(\theta_t)^{\frac{1}{2}} dn \tag{A.4}$$

where $\nabla_\theta f(\mathcal{X}) \in \mathbb{R}^{CN \times D}$ is the jacobian matrix and D is the number of parameters. Note A is a $N \times C$ matrix which, denoting by $f_\theta^{(j)}(x)$ the j-th output of the model on a sample x, is given by:

$$A_{ij} = \text{Tr}[\Sigma(\theta_t) \nabla_\theta^2 f_\theta^{(j)}(x_i)].$$

Using the fact that in our case the model is linearized, so $f_\theta(x)$ is a linear function of θ, we have that $\nabla_\theta^2 f^{(j)}(x) = 0$ and hence $A = 0$. This leaves us with the SDE:

$$df_t(\mathcal{X}) = -\eta \Theta_t \nabla_{f_t} \mathcal{L} dt + \frac{\eta}{\sqrt{|B|}} \nabla_\theta f(\mathcal{X}) \Sigma(\theta_t)^{\frac{1}{2}} dn \tag{A.5}$$

as we wanted. ■

A.5.2 Loss decomposition

We now prove Proposition 3.2.

Proof. Let $\nabla_w f_w(\mathcal{X}) = V \Lambda U$ be the singular value decomposition of $\nabla_w f_w(\mathcal{X})$ where Λ is a rectangular matrix (of the same size of $\nabla_w f_w(\mathcal{X})$) containing the singular values $\{\lambda_1, \dots, \lambda_N\}$ on the diagonal. Both U and V are orthogonal matrices. Note that we have

$$S = \nabla_w f_w(\mathcal{X})^T \nabla_w f_w(\mathcal{X}) = U^T \Lambda^T \Lambda U,$$
$$\Theta = \nabla_w f_w(\mathcal{X}) \nabla_w f_w(\mathcal{X})^T = V \Lambda \Lambda^T V^T.$$

We now use the singular value decomposition to derive an expression for $\mathcal{L}_t$ in case of gradient descent and MSE loss (which we call L_t). In this case, the differential equation Eq. (3.1) reduces to:

$$\dot{f}_t^{\text{lin}}(\mathcal{X}) = -\eta\Theta(\mathcal{Y} - f_t^{\text{lin}}(\mathcal{X})),$$

which is a linear ordinary differential equation that can be solved in closed form. In particular, we have:

$$f_t^{\text{lin}}(\mathcal{X}) = (I - e^{-\eta\Theta t})\mathcal{Y} + e^{-\eta\Theta t} f_0(\mathcal{X}).$$

Replacing this in the expression for the MSE loss at time t we have:

$$\begin{aligned} L_t &= \sum_i (y_i - f_t^{\text{lin}}(x_i))^2 \\ &= (\mathcal{Y} - f_t^{\text{lin}}(\mathcal{X}))^T (\mathcal{Y} - f_t^{\text{lin}}(\mathcal{X})) \\ &= (\mathcal{Y} - f_0(\mathcal{X}))^T e^{-2\eta\Theta t} (\mathcal{Y} - f_0(\mathcal{X})). \end{aligned}$$

Now recall that, by the properties of the matrix exponential, we have:

$$e^{-2\eta\Theta t} = e^{-2\eta V\Lambda\Lambda^T V^T t} = V e^{-2\eta\Lambda\Lambda^T t} V^T,$$

where $e^{-2\Lambda\Lambda^T t} = \text{diag}(e^{-2\eta\lambda_1^2 t}, e^{-2\eta\lambda_2^2 t}, \dots)$. Then, defining $\delta y = \mathcal{Y} - f_0(\mathcal{X})$ and denoting with v_k the k-th column of V we have:

$$\begin{aligned} L_t &= \delta y^T V e^{-2\eta\Lambda\Lambda^T t} V^T \delta y \\ &= \sum_{k=1}^{n} e^{-2\eta\lambda_k^2 t} (\delta y \cdot v_k). \end{aligned}$$

where $n \leq \min(N, D)$ is the number of non-null singular values of $\nabla_w f_w(\mathcal{X})$.

Now let u_k denote the k-th column of U^T and $\nabla_w f_w(x_i)$ is the gradient of the i-th sample (that is the i-th column of $\nabla_w f_w(\mathcal{X})^T$). To conclude the proof we only need to show that $\lambda_k v_k = (\nabla_w f_w(x_i) \cdot u_k)_{i=1}^N$. But this follows directly from the SVD decompostion $\nabla_w f_w(\mathcal{X}) = V\Lambda U$, since then $V\Lambda = \nabla_w f_w(\mathcal{X}) U^T$. ■

B

Model Selection

B.1 Approximating NTK matrix with feature similarity

Proposition B.1 (Gradients back-propagation formula [Arora et al., 2019b]). *The partial derivative of the output of a DNN with respect to the parameters of its l-th hidden layer can be expressed as:*

$$\frac{\partial f_w(x)}{\partial W^{(h)}} = b^{(h)}(x) \otimes g^{h-1}(x), \qquad h \in [L+1]$$

where

$$b^{(h)}(x) := \begin{cases} 1 \in \mathbb{R}, & h = L+1 \\ \sqrt{\frac{c_\sigma}{d_h}} D^h(x)(W^{h+1})^T b^{(h+1)}(x) \in \mathbb{R}^{d_h \times d_h}, & h \in [L] \end{cases}$$
$$D^h(x) := \operatorname{diag}(\dot{\sigma}(f^{(h)})(x)) \in \mathbb{R}^{d_h \times d_h}, \qquad h \in [L]$$

Proposition B.2 (Approximating NTK matrix with activations similarity). *Let $\hat{\mathbb{E}}$ be the empirical measure associated to the dataset $\mathcal{D}$. Let $f_w(x)$ be a DNN and consider the explict formula of gradients given in Proposition B.1. Assuming the random variables (w.r.t. x) $yg^{h-1}(x)$ and $b^{(h)}(x)$ are uncorrelated, it holds:*

$$y^T \Theta y = N^2 \sum_{h=1}^{L} \left\| \mathbb{E}[b^{(h)}(x)] \right\|_2^2 \left\| \mathbb{E}[yg^{h-1}(x)] \right\|_2^2 \tag{B.1}$$

Proof.

$$
\begin{aligned}
y^T \Theta y &= \sum_{i,j}^{N} y_i y_j (\Theta)_{i,j} = \sum_{i,j}^{N} y_i y_j \nabla_w f_w(x_i)^T \nabla_w f_w(x_j) \\
&= \left(\sum_{i=1}^{N} y_i \nabla_w f_w(x_i) \right)^T \left(\sum_{j=1}^{N} y_j \nabla_w f_w(x_j) \right) \\
&= \sum_{h=1}^{L} \left(\sum_{i=1}^{N} y_i b^{(h)}(x_i) \otimes g^{h-1}(x_i) \right)^T \left(\sum_{j=1}^{N} y_j b^{(h)}(x_j) \otimes g^{h-1}(x_j) \right) \\
&= N^2 \sum_{h=1}^{L} \hat{\mathbb{E}}[y b^{(h)}(x) \otimes g^{h-1}(x)]^T \hat{\mathbb{E}}[y b^{(h)}(x) \otimes g^{h-1}(x)] \\
&= N^2 \sum_{h=1}^{L} \left\| \mathbb{E}[y b^{(h)}(x) \otimes g^{h-1}(x)] \right\|_2^2 \\
&= N^2 \sum_{h=1}^{L} \left\| \mathbb{E}[b^{(h)}(x)] \otimes \mathbb{E}[y g^{h-1}(x)] \right\|_2^2 \qquad \text{(B.2)} \\
&= N^2 \sum_{h=1}^{L} \left\| \mathbb{E}[b^{(h)}(x)] \right\|_2^2 \left\| \mathbb{E}[y g^{h-1}(x)] \right\|_2^2
\end{aligned}
$$

where in Eq. (B.2) we used the assumption $y g^{h-1}(x)$ and $b^{(h)}(x)$ are uncorrelated. In the last equation we applied standard results on the euclidean norm of outer products. ■

Remark B.1 (Uncorrelatedness assumption). This assumption does not hold in general, nonetheless in practice it can be show to be approximately true, see Section 3.1 in [Martens and Grosse, 2015] for theoretical and empirical justifications.

Remark B.2 (Approximating NTK matrix with feature similarity). Note the term $\hat{\mathbb{E}}[y g^{h-1}(x)]$ measures the correlation between each individual activation and the label. If activations are correlated with labels, then $y^T \Theta y$ is larger. Note that in general it should be necessary to consider all the hidden layers of the DNN. However the contribution of earlier layers is discounted by a factor $\left\| \mathbb{E}[b^{(h)}(x)] \right\|_2^2$. So that, as we progress further down the network (closer to the input), this term may become smaller (since the mobility of earlier weights may be smaller see Proposition B.1) and hence the contribution of activation-label correlations for these layers is small.

Dataset	Training Images	Testing Images	# Classes	URL
NWPU-RESISC45 [Cheng et al., 2017]	25,200	6300	45	https://www.tensorflow.org/datasets/catalog/resisc45
Food-101 [Bossard et al., 2014]	75,750	25,250	101	https://www.tensorflow.org/datasets/catalog/food101
Logo 2k [Wang et al., 2020]	134,907	32,233	2341	https://github.com/msn199959/Logo-2k-plus-Dataset
Goog. Landmark [Noh et al., 2017]	200,000	15,601	256	https://github.com/cvdfoundation/google-landmark
iNaturalist [Horn et al., 2017]	265,213	3030	1010	https://github.com/visipedia/inat_comp
iMaterialist [MalongTech, 2019]	965,782	9639	2019	https://github.com/malongtech/imaterialist-product-2019
Imagenet [Deng et al., 2009]	1,281,167	50,000	1000	http://image-net.org/download
Places-365 [Zhou et al., 2017]	1,803,460	36,500	365	http://places2.csail.mit.edu/download.html
Magnetic Tile Defects [Huang et al., 2018]	1008	336	6	https://github.com/abin24/Magnetic-tile-defect-datasets
UC Merced Land Use [Yang and Newsam, 2010]	1575	525	21	http://weegee.vision.ucmerced.edu/datasets/landuse.html
Oxford Flowers 102 [Nilsback and Zisserman, 2008]	2040	6149	102	https://www.robots.ox.ac.uk/~vgg/data/flowers/102/
Cucumber [dataset, 2016]	2326	597	30	https://github.com/workpiles/CUCUMBER-9
European Flood Depth [Barz et al., 2019]	3153	557	2	https://github.com/cvjena/eu-flood-dataset
Oxford-IIIT Pets [Parkhi et al., 2012]	3680	3669	37	https://www.robots.ox.ac.uk/~vgg/data/pets/
Describable Textures [Cimpoi et al., 2014]	4230	1410	47	https://www.robots.ox.ac.uk/~vgg/data/dtd/
iCassava [Mwebaze et al., 2019]	5367	280	5	https://sites.google.com/view/fgvc6/competitions/icassava-2019
CUB-200 [Welinder et al., 2010]	5994	5793	200	http://www.vision.caltech.edu/visipedia/CUB-200-2011.html
Belga Logos [Joly and Buisson, 2009]	7500	2500	27	http://www-sop.inria.fr/members/Alexis.Joly/BelgaLogos/BelgaLogos.html
Stanford Cars [Krause et al., 2013]	8144	8041	196	https://ai.stanford.edu/~jkrause/cars/car_dataset.html

Table B.1: **Datasets description.** The number of training images, testing images and classes as well as the URL to download the dataset are listed above. The top part contains our source datasets used to train the model zoo and the bottom part lists our target tasks used for fine-tuning and model selection with our model zoo.

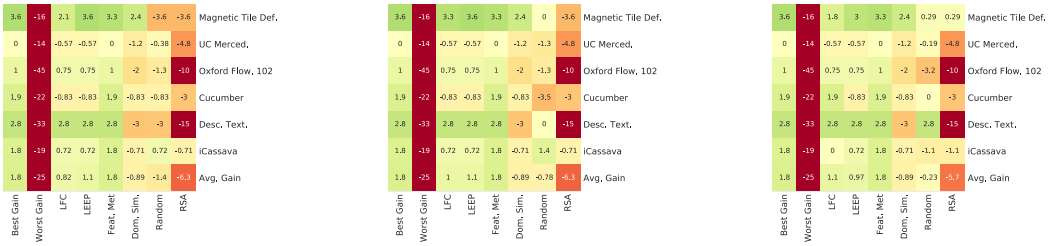
B.2 Datasets

We choose our source and target datasets such that they cover different domains, and are publicly available for download. Detailed data statistics are in the respective citations for the datasets, and we include a few statistics e.g. training images, testing images, number of classes in Table B.1. For all the datasets, if available we use the standard train and test split of the dataset, else we split the dataset randomly into 80% train and 20% test images. If images are indexed by URLs in the dataset, we download all accessible URLs with a python script.

B.3 Details of model selection methods

Domain Similarity [Cui et al., 2018]. As per [Cui et al., 2018], we extract avg. features for every class for source and target datasets using pre-trained model. We compute an earth movers distance between these average class vectors and convert them to domain similarity score. We use the code provided by the authors at https://github.com/richardaecn/cvpr18-inaturalist-transfer. We exclude classes with less than 5 training images for Earth-Movers Distance computation.

RSA [Dwivedi and Roig, 2019]. Following the procedure outlined in [Dwivedi and Roig, 2019], we extract features before the classification layer (e.g. 2048 dim features of Resnet-101 after average pool) for images in the target dataset. We denote this set of features as $f(x)$, $\forall(x, y) \in \mathcal{D}$. We build a representation dissimilarity matrix (RDM)



(a) 25 samples per class (b) 50 samples per class (c) Full target task

Figure B.1: **Ablation study of dataset size for model selection.** Above we use 25,50-samples per class and full target task to perform model selection with different methods. We plot accuracy gain vs. Imagenet expert for top-3 selected models for every method (similar to Fig. 4 of the paper). The accuracy gain increases for LFC, LEEP and RSA with more samples of the target task. However, we see that even as small as 25 samples suffice to obtain good accuracy gain with low computational cost.

as follows:

$$\text{rdm}_f(i,j) = 1 - correlation(f(x_i), f(x_j)) \tag{B.3}$$

We train a small neural network f_{small} on target dataset. Note, this is much cheaper to train than fine-tuning the model zoo. Features are extracted from f_{small} and we build another rdm:

$$\text{rdm}_{f_{\text{small}}}(i,j) = 1 - correlation(f_{\text{small}}(x_i), f_{\text{small}}(x_j)) \tag{B.4}$$

If rdm's of trained small network f_{small} and our pre-trained model f are similar, then the pre-trained model is a good candidate for fine-tuning with target dataset. The final RSA model selection score is:

$$S_{\text{RSA}}(f, \mathcal{D}) = \text{spearmanr}(\text{rdm}_f, \text{rdm}_{f_{\text{small}}}) \tag{B.5}$$

Since the method requires training a small neural network on target task, we train a Resnet-18 as the small neural network with the same fine-tuning configuration used in Section 4.1 of the paper with initial learning rate .005.

Feature Metrics [Ueno and Kondo, 2020]. Features are extracted for all images of target dataset from pre-trained model, i.e. $f(x), \forall x \in \mathcal{D}$. We use same features as RSA, our LFC/LGC and compute variance, sparsity metrics of [Ueno and Kondo, 2020]. We use the sparsity metrics as model selection score, $S_{\text{Feat. Metrics}}(f, \mathcal{D}) =$

sparsity$(f(x), \forall x \in \mathcal{D})$. Note, we use the optimal linear combination of the two sparsity metrics proposed in the paper. For feature metrics, the hypothesis is that if the pre-trained model generates more sparse representations, they are can generalize with fine-tuning to the target task.

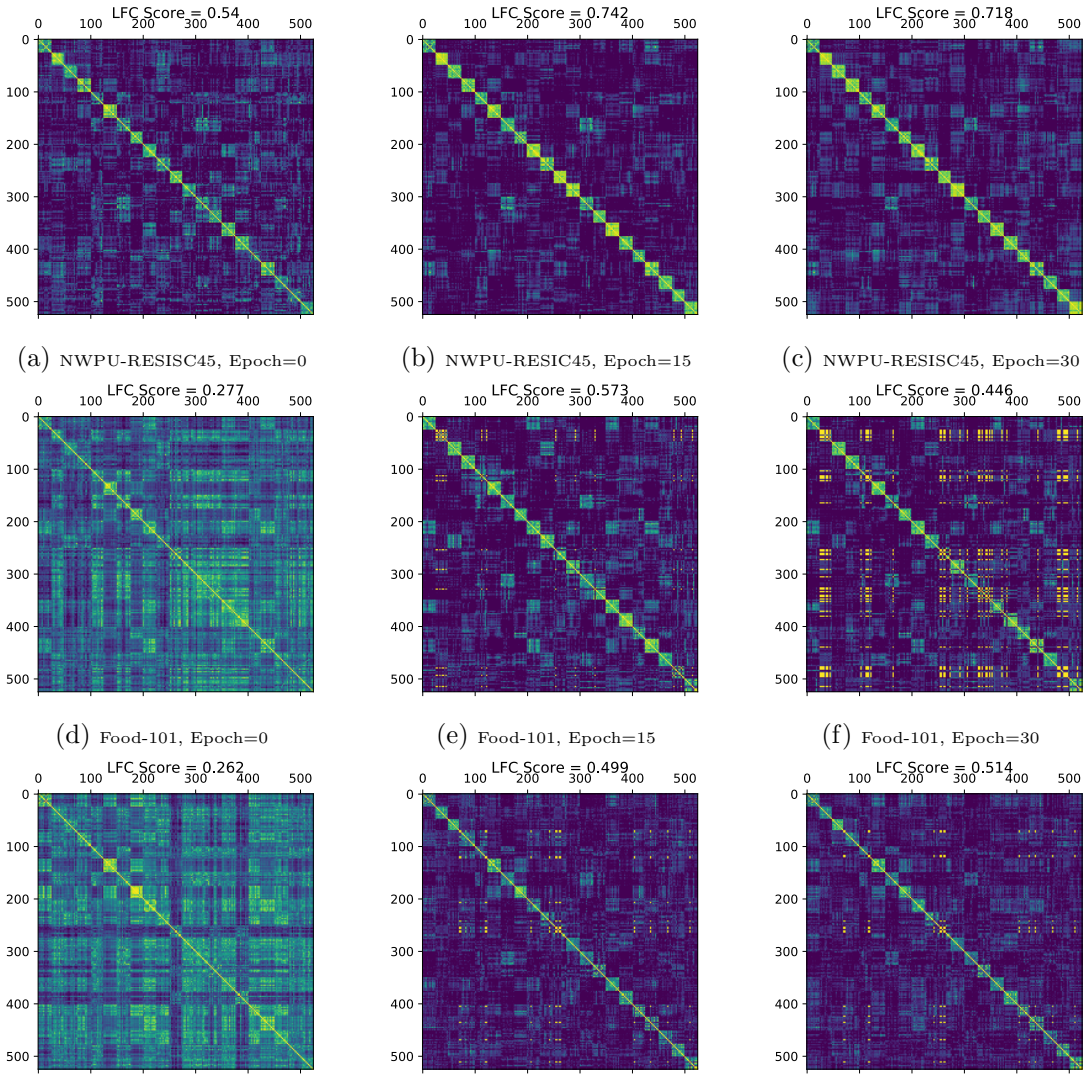
LEEP [Nguyen et al., 2020]. LEEP builds an empirical classifier from source dataset label space to target dataset label space using base model f. The likelihood of target dataset $\mathcal{D}$ under this empirical classifier is the model selection score for the pre-trained model and target dataset. See [Nguyen et al., 2020] for a detailed explanation.

B.4 Different dataset sizes for model selection

In Fig. B.1, we perform an ablation study on different sampling sizes of the target task used for model selection. We find that, our choice of 25 samples per class for model selection, suffices to select good models to fine-tune in top-3 selections at low-computational cost.

B.5 Visualization of feature correlation with fine-tuning

In Fig. B.2, we plot the feature correlation matrix for different pre-trained models across different epochs of fine-tuning (i.e. 0^{th}, 15^{th}, 30^{th} epoch) for the UC Merced Land Use [Yang and Newsam, 2010] target task. We see that the pre-trained model on NWPU-RESISC45 [Cheng et al., 2017], exhibits the ideal correlation wherein features of the images with the same class are correlated and features of images with different classes are uncorrelated. This NWPU-RESISC45 [Cheng et al., 2017] also has the highest LFC score.



(a) NWPU-RESISC45, Epoch=0 (b) NWPU-RESIC45, Epoch=15 (c) NWPU-RESISC45, Epoch=30

(d) Food-101, Epoch=0 (e) Food-101, Epoch=15 (f) Food-101, Epoch=30

(g) Logo 2k, Epoch=0 (h) Logo 2k, Epoch=15 (i) Logo 2k, Epoch=30

Figure B.2: **Feature correlation matrix visualization.** We plot the *feature correlation matrix*, Θ_F, for different pre-trainings (row) and different epochs (columns) of fine-tuning. Above, we fine-tune on the UC Merced Land Use [Yang and Newsam, 2010] dataset comprising of aerial images. Images with same class label are grouped along the vertical/horizontal axis. Since, features of the same class should be correlated and features of different classes should be uncorrelated, the matrix is expected to have higher values along block diagonal and zero elsewhere. We observe that the matrix exhibits this ideal behaviour for pre-training on semantically related domain (aerial images) of NWPU-RESISC45 [Cheng et al., 2017] (top row) and has highest LFC score for this pre-training.

C

Inductive Bias and Regularization

C.1 Variational upper bound proof

To begin with we prove a simple lemma which we shall use in the proof of Theorem 5.1.

Lemma C.1 (Optimal loss value on Regularized Least Squares)**.** *Consider the regularized optimization problem given by:*

$$\min_{\theta} \frac{1}{\sigma^2} \|Y_f - F_w\theta\|^2 + \theta^T \Lambda^{-1} \theta \tag{C.1}$$

The optimal minimum value is given by:

$$\min_{\theta} \frac{1}{\sigma^2} \|Y_f - F_w\theta\|^2 + \theta^T \Lambda^{-1} \theta = Y_f^\top \Sigma^{-1} Y_f$$

with $\Sigma := F\Lambda F^\top + \sigma^2 I$.

Proof. To begin with, let μ be the optimal θ for Eq. (C.1):

$$\mu := \left(F_w^T F_w + \sigma^2 \Lambda^{-1}\right)^{-1} F_w^T Y_f = A^{-1} F_w^T Y_f$$

where $A := F_w^T F_w + \sigma^2 \Lambda^{-1}$. We now apply the matrix inversion lemma to obtain the following:

$$\left(F\Lambda F^\top + \sigma^2 I\right)^{-1} = \sigma^{-2} I - \sigma^{-2} F_w^T A^{-1} F_w \tag{C.2}$$

So that we get:

$$\begin{aligned} Y_f^\top \Sigma^{-1} Y_f =& \sigma^{-2} Y_f^\top \left(Y_f - F_w \mu\right) \\ =& \sigma^{-2} \left(Y_f^\top Y_f - 2 Y_f^\top F_w \mu + \mu^T F_w^T F_w \mu \right) + \sigma^{-2} Y_f^\top F_w \mu - \sigma^{-2} \mu^T F_w^T F_w \mu \\ =& \sigma^{-2} \left\| Y_f - F_w \mu \right\|_2^2 + \sigma^{-2} \mu^T \left(A - F_w^T F_w \right) \mu \\ =& \sigma^{-2} \left\| Y_f - F_w \mu \right\|_2^2 + \mu^T \Lambda^{-1} \mu \end{aligned}$$

which is the Regularized Least Squares loss evaluated at the optimum. ■

We now prove the upper bound to the marginal likelihood associated to the posterior given by Eq. (5.5) with marginalization taken only w.r.t. θ Theorem 5.1.

Proof. Consider the regularized linear least squares problem in Eq. (C.1) with feature matrix $F_w \theta$ and parameters θ (where w is assumed fixed). Then, Lemma C.1 guarantees that:

$$\frac{1}{\sigma^2} \left\| Y_f - F_w \theta \right\|^2 + \theta^T \Lambda^{-1} \theta + \log\det \Sigma \geq Y_f^\top \Sigma^{-1} Y_f + \log\det \Sigma$$

where the right hand side is proportional to the negative log marginal likelihood with marginalization taken *only* w.r.t. θ. Therefore, for fixed w,

$$\frac{1}{\sigma^2} \| Y_f - F_w \theta \|^2 + \theta^\top \Lambda^{-1} \theta + \log\det(F_w \Lambda F_w^\top + \sigma^2 I)$$

is an upper bound of the marginal likelihood with marginalization over θ. ■

D

Interpretable Residual Temporal Convolutional Networks

D.1 Implementation

D.1.1 Architecture

Let $Y_{t-n_p+1}^{t} \in \mathbb{R}^{n \times n_p}$ be the input data to our architecture at time step t (a window of n_p past time instants). The main blocks of the architecture are defined to encode trend, seasonality, stationary linear and non-linear part. In the following we shall denote each quantity related to a specific layer using either the subscripts {TREND, SEAS, LIN, TCN} or $\{0, 1, 2, 3\}$.

We shall denote the input of each block as $X_k \in \mathbb{R}^{n \times n_p}$ and the output as $\hat{X}_k \in \mathbb{R}^{n \times n_p}$ for $k = 0, 1, 2, 3$. The residual architecture we propose is defined by the following: $X_0 = Y_{t-n_p+1}^{t}$ and $X_k = X_{k-1} - \hat{X}_{k-1}$ for $k = 1, 2, 3$. At each layer we extract l_k temporal features from the input X_k. We denote the temporal features extracted from the input of the k-th block as: $G_k := G_k(X_k) \in \mathbb{R}^{l_k \times n_p}$. The i-th column of the feature matrix G_k is a feature vector (of size l_k) extracted from the input X_k up to time $t - n_p - i$. To do so, we use causal convolutions of the input signal X_k with a set of filter banks [Bai et al., 2018].

D.1.2 Interpretable Residual TCN on scalar time series

Modeling Interpretable blocks: In this section, we shall describe the main design criteria of the linear module. For each interpretable layer (TREND, SEAS, LIN), we convolve the input signal with a filter bank designed to extract specific components of the input.

For example, consider the trend layer, denoting its scalar input time series by x and

its output by g_{TREND}. Then g_{TREND} is defined as a multidimensional time series (of dimension $l_{\text{TREND}} := l_0$) obtained by stacking l_0 time series given by the convolution of x with l_0 causal linear filters: $\varphi_{\text{TREND}i} * x$ for $i = 0, ..., l_0 - 1$. In other words, $g_{\text{TREND}} := [\varphi_{\text{TREND}1} * x, \varphi_{\text{TREND}2} * x, ..., \varphi_{\text{TREND}l_1} * x]^T$. We denote the set of linear filters $\varphi_{\text{TREND}i}$ for $i = 0, ..., l_0 - 1$ as $\mathcal{K}_{\text{TREND}}$ and parametrize each filter in $\mathcal{K}_{\text{TREND}}$ with its truncated impulse response (i.e. kernel) of length $k_0 := k_{\text{TREND}}$.

We interpret each time series in g_{TREND} as an approximation of the trend component of x computed with the i-th filter. We design each $\varphi_{\text{TREND}i}$ so that each filter extracts the trend of the input signal on different time scales [Ravn and Uhlig, 2002] (i.e., each filter outputs a signal with a different smoothness degree). We estimate the trend of the input signal by recombining the extracted trend components in g_{TREND} with the linear map a_{TREND}. Moreover, we predict the future trend of the input signal (on the next time-stamp) with the linear map b_{TREND}.

We construct the blocks that extract seasonality and linear part in a similar way.

Implementing Interpretable blocks: The input of each layer is given by a window of measurements of length n_p. We zero-pad the input signal so that the convolution of the input signal with the i-th filter is a signal of length n_p (note this introduces a spurious transient whose length is the length of the filter kernel). We therefore have the following temporal feature matrices: $G_0 = G_{\text{TREND}} \in \mathbb{R}^{l_0 \times n_p}$, $G_1 = G_{\text{SEAS}} \in \mathbb{R}^{l_1 \times n_p}$ and $G_2 = G_{\text{LIN}} \in \mathbb{R}^{l_2 \times n_p}$.

The output of each layer $\hat{X}_k$ is an estimate of the trend, seasonal or stationary linear component of the input signal on the past interval of length n_p, so that we have $\hat{X}_k \in \mathbb{R}^{1 \times n_p}$ (same dimension as the input X_k). On the other hand, the linear predictor $\hat{y}_k$ computed at each layer is a scalar. Intuitively, $\hat{X}_k$ and $\hat{y}_k$ should be considered as the best linear approximation of the trend, seasonality or linear part given block's filter bank in the past and future. Our architecture performs the following computations: $\hat{X}_k := a_k^T G_k$ and $\hat{y}_k := \hat{X}_k b_k$ for $k = 0, 1, 2$ where $a_i \in \mathbb{R}^{l_k}$ and $b_k \in \mathbb{R}^{n_p}$. Note a_k combines features (uniformly in time) so that we can interpret it as a feature selector while b_k aggregates relevant features across different time indices to build the one-step ahead predictor.

Non-linear module The non-linear module is based on a standard TCN network. Its input is defined as $X_3 = Y_{t-n_p+1}^t - \hat{X}_0 - \hat{X}_1 - \hat{X}_2$, which is to be considered as a signal whose linearly predictable component has been removed. The TCN extracts a set of l_3 non-linear features $G_3(X_3) \in \mathbb{R}^{l_3 \times n_p}$ which we combine with linear maps as done for the previous layers. The j-th column of the non-linear features G_3 is computed using data up to time $t - n_p + j$ (due to the internal structure of a TCN network [Bai

et al., 2018]). The linear predictor on top of G_3 is $\hat{y}_{\text{TCN}} := a_3^T G_3 b_3$, where $a_3 \in \mathbb{R}^{l_3}$ and $b_3 \in \mathbb{R}^{n_p}$.

Finally, the output of our time model is given by:

$$\hat{y}(t+1) = \sum_{k=0}^{3} \hat{y}_k = \sum_{k=0}^{3} \hat{X}_k b_k = \sum_{k=0}^{3} a_k^T G_k(X_k) b_k.$$

D.1.3 Interpretable Residual TCN on multi-dimensional time series

We extend our architecture to multi-dimensional time series according to the following principles: preserve interpretability (first module) and exploit global information to make local predictions (second module).

In this section, the input data to our model is $Y_{t-n_p+1}^t \in \mathbb{R}^{n \times n_p}$ (a window of length n_p from an n-dimensional time series).

Interpretable module: Each time series undergoes the sequence of 3 interpretable blocks independently from other time series: the filter banks are applied to each time series independently. Therefore, each time series is processed by the same filter banks: $\mathcal{K}_{\text{TREND}}$, $\mathcal{K}_{\text{SEAS}}$ and $\mathcal{K}_{\text{LIN}}$. For ease of notation we shall now focus only on the trend layer. Any other layer is obtained by substituting "TREND" with the proper subscript ("SEAS" or "LIN").

We denote by $G_{\text{TREND}}, i \in \mathbb{R}^{l_0 \times n_p}$ the set of time features extracted by the trend filter bank $\mathcal{K}_{\text{TREND}}$ from the i-th time series. Each feature matrix is then combined as done in the scalar setting using linear maps, which we now index by the time series index i: $a_{\text{TREND}i}$ and $b_{\text{TREND}i}$. The rationale behind this choice is that each time series can exploit differently the extracted features. For instance, slow time series might need a different filter than faster ones (chosen using $a_{\text{TREND}i}$) or might need to look at values further in the past (retrieved using $b_{\text{TREND}i}$). We stack the combination vectors $a_{\text{TREND}i}$ and $b_{\text{TREND}i}$ into the following matrices:

$$\mathcal{A}_{\text{TREND}} = [a_{\text{TREND}1}, a_{\text{TREND}2}, ..., a_{\text{TREND}n}]^T \in \mathbb{R}^{n \times l_0}$$
$$\mathcal{B}_{\text{TREND}} = [b_{\text{TREND}1}, b_{\text{TREND}2}, ..., b_{\text{TREND}n}]^T \in \mathbb{R}^{n \times n_p}.$$

Non-linear module: The second (*non-linear*) module aggregates global statistics from different time series [Sen et al., 2019] using a TCN model. It takes as input the prediction residual of the linear module and outputs a matrix $G_{\text{TREND}}(Y_{t-n_p+1}^t) \in \mathbb{R}^{l_3 \times n_p}$ where l_3 is the number of output features that are extracted by the TCN model (which is a design parameter). The j-th column of the non-linear features $G_{\text{TREND}}(Y_{t-n_p+1}^t)$

is computed using data up to time $t - n_p + j$. This is due to the internal structure of a TCN network [Bai et al., 2018] which relies on causal convolutions. As done for the time features extracted by the interpretable blocks, we build a linear predictor on top of $G_{\text{TREND}}(Y^t_{t-n_p+1})$ for each single time series independently: the predictor for the i-th time series is given by: $\hat{y}_{\text{TCN}}(t+1)_i := a_i^T G_{\text{TREND}}(Y^t_{t-n_p+1}) b_i$ where $a_i \in \mathbb{R}^{l_3}$ and $b_i \in \mathbb{R}^{n_p}$. We stack the combination vectors $a_{\text{TCN}i}$ and $b_{\text{TCN}i}$ into the following matrices: $\mathcal{A}_{\text{TCN}} = [a_{\text{TCN}1}, a_{\text{TCN}2}, ..., a_{\text{TCN}n}]^T \in \mathbb{R}^{n \times l_3}$ and $\mathcal{B}_{\text{TCN}} = [b_{\text{TCN}1}, b_{\text{TCN}2}, ..., b_{\text{TCN}n}]^T \in \mathbb{R}^{n \times n_p}$.

Finally, the outputs of the predictor on the i-th time series are given by:

$$\hat{y}(t+1)_i = \sum_{k \in \{\text{TREND,SEAS,LIN}\}} a_{ki}^T G_{ki} b_{ki} + a_{\text{TCN}i}^T G_{\text{TCN}} b_{\text{TCN}i}.$$

D.1.4 Block structure and initialization

In this section, we shall describe the internal structure and the initialization of each block.

Structure: Each filter is implemented by means of depth-wise causal 1-D convolutions [Bai et al., 2018]. We call the tensor containing the k-th block's kernel parameters $\mathcal{K}_k \in \mathbb{R}^{l_k \times N_k}$, where l_k and N_k are the block's number of filters and block's kernel size, respectively (without loss of generality, we assume all filters have the same dimension). Each filter (causal 1D-convolution) is parametrized by the values of its impulse response parameters (kernel parameters). When we learn a filter bank, we mean that we optimize over the kernel values for each filter jointly. For multidimensional time series, we apply the filter banks to each time series independently (depth-wise convolution) and improve filter learning by sharing kernel parameters across different time series.

Initialization: The *first block* (trend) is initialized using l_0 causal Hodrick Prescott (HP) filters [Ravn and Uhlig, 2002] of kernel size N_0. HP filters are widely used to extract trend components of signals [Ravn and Uhlig, 2002]. In general a HP filter is used to obtain from a time series a smoothed curve which is not sensitive to short-term fluctuations and more sensitive to long-term ones [Ravn and Uhlig, 2002]. In general, a HP filter is parametrized by a hyper-parameter λ_{HP} which defines the regularity of the filtered signal (the higher λ_{HP}, the smoother the output signal). We initialize each filter with λ_{HP} chosen uniformly in log-scale between 10^3 and 10^9. Note the impulse response of these filters decays to zero (i.e., the latest samples from the input time series are the most influential ones). When we learn the optimal set of trend filter banks, we do not consider them parametrized by λ_{HP} and search for the optimal λ_{HP}.

Instead, we optimize over the impulse response parameters of the kernel which we do not assume live in any manifold (e.g., the manifold of HP filters). Since this might lead to optimal filters which are not in the class of HP filters, we impose a regularization which penalizes the distance of the optimal impulse response parameters from their initialization.

The *second block* (seasonal part) is initialized using l_1 periodic kernels which are obtained as linear filters whose poles (i.e., frequencies) are randomly chosen on the unit circle (this guarantees to span a range of different frequencies). Note the impulse responses of these filters do not go to zero (their memory does not fade away). Similarly to the HP filter bank, we do no optimize the filters over frequencies, but rather we optimize them over their impulse response (kernel parameters). This optimization does not preserve the strict periodicity of filters. Therefore, in order to keep the optimal impulse response close to initialization values (purely periodic), we exploit weight regularization by penalizing the distance of the optimal set of kernel values from initialization values.

The *third block* (stationary linear part) is initialized using l_2 randomly chosen linear filters whose poles lie inside the unit circle, as done in [Farahmand et al., 2017]. As the number of filters l_2 increases, this random filter bank is guaranteed to be a universal approximator of any (stationary) linear system (see [Farahmand et al., 2017] for details).

Remark D.1. This block could approximate any trend and periodic component. However, we assume to have factored out both trend and periodicities in the previous blocks.

The last module (*non-linear part*) is composed by a randomly initialized TCN model. We employ a TCN model due to its flexibility and capability to model both long-term and short-term non-linear dependencies. As is standard practice, we exploit dilated convolutions to increase the receptive field and make the predictions of the TCN (on the future horizon) depend on the most relevant past [Bai et al., 2018].

Remark D.2. Our architecture provides an interpretable justification of the initialization scheme proposed for TCN in [Sen et al., 2019]. In particular our convolutional architecture allows us to handle high-dimensional time series data without a-priori standardization (e.g., trend or seasonality removal).

D.2 Automatic Complexity Determination for STRIC

In this section, we briefly recall the *fading regularization* introduced in Section 5.3.1.

The output of the TCN model is $G(Y_{t-n_p+1}^t) \in \mathbb{R}^{l_3 \times n_p}$ where l_3 is the number of output features extracted by the TCN model. The predictor built from TCN features is given by: $a_{\text{TCN}i}^T G_{\text{TCN}}(Y_{t-n_p+1}^t) b_{\text{TCN}i}$, where the predictor $b_{\text{TCN}i} \in \mathbb{R}^{n_p}$ takes as

input a linear combination of the TCN features (weighted by $a_{\text{TCN}i}$). The j-th column of the non-linear features $G(Y_{t-n_p+1}^t)$ is computed using data up to time $t-n_p+j$ (due to causal convolutions used in the internal structure of the TCN network [Bai et al., 2018]). One expects that the influence on the TCN predictor as j increases should increase too (in case $j = n_p$, the statistic is the one computed on the closest window of time w.r.t. present time stamp). The exact *relevance* on the output is not known a priori and needs to be estimated. In other words, the predictor should be less sensitive to statistics (features) computed on a far past, a property which is commonly known as *fading memory*. Currently, this property is not built in the predictor $b_{\text{TCN}i}$, which treats each time instant equally and might overfit while trying to explain the future by looking into far and possibly non-relevant past. We impose our fading memory property on STRIC's predictor to constrain it complexity and reduce overfitting.

Remark D.3 (Prior on the parameters). The prior $\log p(W)$ in Eq. (5.8) defines the regularization applied on the remaining parameters of our architecture. In particular, we induce sparsity by applying L^1 regularization on $\mathcal{A}_{\text{TREND}}$, $\mathcal{A}_{\text{SEAS}}$, $\mathcal{A}_{\text{LIN}}$ and $\mathcal{A}_{\text{TCN}}$. Also, we constrain filters parameters to stay close to initialization by applying L^2 regularization on $\mathcal{K}_{\text{TREND}}$, $\mathcal{K}_{\text{SEAS}}$ and $\mathcal{K}_{\text{LIN}}$.

D.3 Alternative CUMSUM Derivation and Interpretation

In this section, we describe an equivalent formulation of the CUMSUM algorithm we derived in Section 7.4.1. Before a change point, by construction we are under the distribution of the past. Therefore, $\log \frac{p_f(y)}{p_p(y)} \leq 0 \ \forall y$, which in turn means that the cumulative sum S_1^t will decrease as t increases (negative drift). After the change, the situation is opposite and the cumulative sum starts to show a positive drift, since we are sampling $y(i)$ from the future distribution p_f. This intuitive behaviour shows that the relevant information to detect a change point can be obtained directly from the cumulative sum (along timestamps). In particular, all we need to know is the difference between the value of the cumulative sum of log-likelihood ratios and its minimum value.

The CUMSUM algorithm can be expressed using the following equations: $v_t := S_1^t - m_t$, where $m_t := \min_{j,1\leq j\leq t} S_j^t$. The stopping time is defined as: $t_{\text{stop}} = \min\{t : v_t \geq \tau\} = \min\{t : S_1^t \geq m_t + \tau\}$. With the last equation, it becomes clear that the CUMSUM detection equation is simply a comparison of the cumulative sum of the log likelihood ratios along time with an adaptive threshold $m_t + \tau$. Note that the adaptive threshold keeps complete memory of the past ratios. The two formulations are equivalent because $S_1^t - m_t = h_t$.

D.4 Variational Approximation of the Likelihood Ratio

In this section, we present some well known facts on f-divergences and their variational characterization. Most of the material and the notation is from [Nguyen et al., 2010]. Given a probability distribution $\mathbb{P}$ and a random variable f measurable w.r.t. $\mathbb{P}$, we use $\int f d\mathbb{P}$ to denote the expectation of f under $\mathbb{P}$. Given samples $x(1), ..., x(n)$ from $\mathbb{P}$, the empirical distribution $\mathbb{P}_n$ is given by $\mathbb{P}_n = \frac{1}{n}\sum_{i=1}^{n} \delta_{x(i)}$. We use $\int f d\mathbb{P}_n$ as a convenient shorthand for the empirical expectation $\frac{1}{n}\sum_{i=1}^{n} f(x(i))$.

Consider two probability distributions $\mathbb{P}$ and $\mathbb{Q}$, with $\mathbb{P}$ absolutely continuous w.r.t. $\mathbb{Q}$. Assume moreover that both distributions are absolutely continuous with respect to the Lebesgue measure μ, with densities p_0 and q_0, respectively, on some compact domain $\mathcal{X} \subset \mathbb{R}^d$.

Variational approximation of the f-divergence: The f-divergence between $\mathbb{P}$ and $\mathbb{Q}$ is defined as [Nguyen et al., 2010]

$$D_f(\mathbb{P}, \mathbb{Q}) := \int p_0 f\Big(\frac{q_0}{p_0}\Big) d\mu \tag{D.1}$$

where $f : \mathbb{R} \to \mathbb{R}$ is a convex and lower semi-continuous function. Different choices of f result in a variety of divergences that play important roles in various fields [Nguyen et al., 2010]. Eq. (D.1) is usually replaced by the variational lower bound:

$$D_f(\mathbb{P}, \mathbb{Q}) \geq \sup_{\phi \in \Phi} \int [\phi d\mathbb{Q} - f^*(\phi) d\mathbb{P}] \tag{D.2}$$

and equality holds iff the subdifferential $\partial f(\frac{q_0}{p_0})$ contains an element of Φ. Here f^* is defined as the convex dual function of f.

In the following, we are interested in divergences whose conjugate dual function is smooth (which in turn defines commonly used divergence measures such as Kullback Leibler and Pearson divergence), so that we shall assume that f is convex and differentiable. Under this assumption, the notion of subdifferential is not required and the previous statement reads as: *equality holds iff* $\partial f(\frac{q_0}{p_0}) = \phi$ *for some* $\phi \in \Phi$.

Remark D.4. The infinite-dimensional optimization problem in Eq. (D.2) can be written as $D_f(\mathbb{P}, \mathbb{Q}) = \sup_{\phi \in \Phi} \mathbb{E}_{\mathbb{Q}} \phi - \mathbb{E}_{\mathbb{P}} f^*(\phi)$.

In practice, one can have an estimator of any f-divergence restricted to a functional class Φ by solving Eq. (D.2) [Nguyen et al., 2010]. Moreover, when $\mathbb{P}$ and $\mathbb{Q}$ are not known one can approximate them using their empirical counterparts: $\mathbb{P}_n$ and $\mathbb{Q}_n$. Then an empirical estimate of the f-divergence is: $\hat{D}_f(\mathbb{P}, \mathbb{Q}) = \sup_{\phi \in \Phi} \mathbb{E}_{\mathbb{Q}_n} \phi - \mathbb{E}_{\mathbb{P}_n} f^*(\phi)$.

Approximation of the likelihood ratio: An estimate of the likelihood ratio can be directly obtained from the variational approximation of f-divergences. The key observation is the following: *equality on Eq. (D.2) is achieved iff* $\phi = \partial f(\frac{q_0}{p_0})$. This tells us that the optimal solution to the variational approximation provides us with an estimator of the composite function $\partial f(\frac{q_0}{p_0})$ of the likelihood ratio $\frac{q_0}{p_0}$. As long as we can invert ∂f, we can uniquely determine the likelihood ratio.

In the following, we shall get an empirical estimator of the likelihood ratio in two separate steps. We first solve the following:

$$\hat{\phi}_n := \arg\max_{\phi \in \Phi} \mathbb{E}_{\mathbb{Q}_n} \phi - \mathbb{E}_{\mathbb{P}_n} f^*(\phi) \tag{D.3}$$

which returns an estimator of $\partial f(\frac{q_0}{p_0})$, not the ratio itself. And then we apply the inverse of ∂f to $\hat{\phi}_n$. We therefore have a family of estimation methods for the likelihood function by simply ranging over choices of f.

Remark D.5. If f is not differentiable, then we cannot invert ∂f but we can obtain estimators of other functions of the likelihood ratio. For instance, we can obtain an estimate of the thresholded likelihood ratio by using a convex function whose subgradient is the sign function centered at 1.

Likelihood ratio estimation with Pearson divergence

In this section, we show how to estimate the likelihood ratio when the Pearson divergence is used. With this choice, many computations simplify and we can write the estimator of the likelihood ratio in closed form. Other choices (such as the Kullback-Leibler divergence) are possible and legitimate, but usually do not lead to closed form expressions (see [Nguyen et al., 2010]).

The Pearson, or χ^2, divergence is defined by the following choice: $f(t) := \frac{(t-1)^2}{2}$. The associated convex dual function is :

$$f^*(v) = \sup_{u \in \mathbb{R}} \left\{ uv - \frac{(u-1)^2}{2} \right\} = \frac{v^2}{2} + v.$$

Therefore the Pearson divergence is characterized by the following:

$$PE(\mathbb{P}||\mathbb{Q}) := \int p_0 \Big(\frac{q_0}{p_0} - 1 \Big)^2 d\mu \geq \sup_{\phi \in \Phi} \mathbb{E}_{\mathbb{Q}} \phi - \frac{1}{2} \mathbb{E}_{\mathbb{P}} \phi^2 - \mathbb{E}_{\mathbb{P}} \phi. \tag{D.4}$$

Solving the lower bound for the optimal ϕ provides us an estimator of $\partial f(\frac{q_0}{p_0}) = \frac{q_0}{p_0} - 1$. For the special case of the Pearson divergence, we can apply a change of variables which

preserves convexity of the variational optimization problem Eq. (D.4) and provides a more straightforward interpretation. Let the new variable be $z := \phi + 1$ with $z \in \mathcal{Z}$, which in this case is nothing but the inverse function of ∂f. We get

$$\sup_{\phi \in \Phi} \mathbb{E}_{\mathbb{Q}} \phi - \frac{1}{2}\mathbb{E}_{\mathbb{P}} \phi^2 - \mathbb{E}_{\mathbb{P}} \phi = \sup_{z \in \mathcal{Z}} \mathbb{E}_{\mathbb{Q}} z - \frac{1}{2}\mathbb{E}_{\mathbb{P}} z^2 - \frac{1}{2} \tag{D.5}$$

It is now trivial to see that z is a "direct" approximator of the likelihood ratio (i.e., it does not estimate a composite map of the likelihood ratio). Therefore for simplicity, we shall employ

$$\arg\min_{\phi \in \Phi} \frac{1}{2}\mathbb{E}_{\mathbb{P}} \phi^2 - \mathbb{E}_{\mathbb{Q}} \phi \tag{D.6}$$

to build our "direct" estimator of the likelihood ratio.

Let the samples from $\mathbb{P}$ and $\mathbb{Q}$ be, respectively, $x_p(i)$ with $i = 1, ..., n_p$ and $x_q(i)$ with $i = 1, ..., n_q$. We define the empirical estimator of the likelihood ratio $\hat{\phi}_n$:

$$\hat{\phi}_n = \arg\min_{\phi \in \Phi} \frac{1}{2n_p} \sum_{i=1}^{n_p} \phi(x_p(i))^2 - \frac{1}{n_q} \sum_{i=1}^{n_q} \phi(x_f(i)). \tag{D.7}$$

A closed form solution: Up to now we have not defined in which class of functions our approximator ϕ lives. As done in [Nguyen et al., 2010, Liu et al., 2012], we choose $\phi \in \Phi$ where Φ is a RKHS induced by the kernel k.

We exploit the representer theorem to write a general function within Φ as:

$$\phi(x) = \sum_{i=1}^{n_{tr}} k(x, x_{tr}(i))\alpha_i,$$

where we use n_{tr} data which are the centers of the kernel sections used to approximate the unknown likelihood ratio. We shall use both data from $\mathbb{P}_n$ and from $\mathbb{Q}_n$ as centers.

Let us define the following kernel matrices: $K_p := K(X_p, X_{tr}) \in \mathbb{R}^{n_p \times n_{tr}}$, $K_q := K(X_q, X_{tr}) \in \mathbb{R}^{n_q \times n_{tr}}$, where $X_p := \{x_p(i)\}$, $X_q := \{x_q(i)\}$ and $X_{tr} := \{x_{tr}(i)\}$.

We therefore have:

$$\begin{aligned}\hat{\phi}_n &= \arg\min_{\phi\in\Phi} \frac{1}{2n_p}\sum_{i=1}^{n_p}\phi(x_p(i))^2 - \frac{1}{n_q}\sum_{i=1}^{n_q}\phi(x_f(i)) \\ &= \arg\min_{\alpha,\alpha\geq 0} \frac{1}{2n_p}\sum_{i=1}^{n_p}(\sum_{j=1}^{n_{tr}}k(x_p(i),x_{tr}(j))\alpha_j)^2 - \frac{1}{n_f}\sum_{i=1}^{n_f}\sum_{j=1}^{n_{tr}}k(x_f(i),x_{tr}(j))\alpha_j \\ &= \arg\min_{\alpha,\alpha\geq 0} \frac{1}{2n_p}\alpha^T K_p^T K_p\alpha - \frac{1}{n_f}\mathbb{1}^T K_f\alpha\end{aligned}$$

Remark D.6. We impose the recombination coefficients α to be non negative since the likelihood ratio is a non negative quantity. The resulting optimization problem is a standard convex optimization problem with linear constraints which can be efficiently solved with Newton methods, nonetheless in general it does not admit any closed form solution.

We now relax the positivity constraints so that the optimal solution can be obtained in closed form. Moreover we add a quadratic regularization term as done in [Nguyen et al., 2010] which lead us to the following regularized optimization problem:

$$\arg\min_{\alpha} \frac{1}{2n_p}\alpha^T K_p^T K_p\alpha - \frac{1}{n_f}\mathbb{1}^T K_f\alpha + \frac{\gamma}{2}\|\alpha\|_\Phi^2$$

whose solution is trivially given by:

$$\hat{\alpha} = \frac{n_p}{n_f}\Big(K_p^T K_p + n_p\gamma I_{n_{tr}}\Big)^{-1}K_f^T\mathbb{1} := \frac{n_p}{n_f}H^{-1}K_f^T\mathbb{1} \tag{D.8}$$

The estimator of the likelihood ratio for an arbitrary location x is given by the following:

$$\frac{p_q(x)}{p_p(x)} \approx \hat{\phi}_n(x) = K(x,X_{tr})\hat{\alpha} = \frac{n_p}{n_f}K(x,X_{tr})\Big(K_p^T K_p + n_p\gamma I_{n_{tr}}\Big)^{-1}K_f^T\mathbb{1} \tag{D.9}$$

Remark D.7. In the following we shall exploit RBF kernels which are defined by the length scales σ.

D.5 Subspace likelihood ratio estimation and CUMSUM

In this section we describe our subspace likelihood ratio estimator and its relation to the CUMSUM algorithm. The CUMSUM algorithm requires to compute the likelihood ratio $\frac{p_f(y(t)|Y_c^{t-1})}{p_p(y(t)|Y_c^{t-1})}$ for each time t. We denote p_p as the normal density and p_f as the

abnormal one (after the anomaly has occurred).

We shall proceed to express the conditional probability $p(y(t) \mid Y_1^{t-1})$ using our predictor. In particular it is always possible to express the optimal (unknown) one-step ahead predictor as Eq. (5.1):

$$\hat{y}_{t|t-1} = f_0(Y_{t-K+1}^t) := \mathbb{E}[y(t) \mid Y_{t-K+1}^t]$$

which is a deterministic function given the past of the time series (whose length is K). So that the data density distribution can be written in innovation form (based on the optimal prediction error) as:

$$y(t) = f_0(Y_{t-K+1}^t) + e(t)$$

where $e(t) := y(t) - f_0(Y_{t-K+1}^t)$ is, by definition, the one step ahead prediction error (or *innovation sequence*) of $y(t)$ given its past. We therefore have: $p(y(t) \mid Y_{t-K+1}^t) = p(e(t) \mid Y_{t-K+1}^t)$. Where $e(t)$ is the optimal prediction error for each time t and is therefore indipendent on each time t.

Remark D.8. In practice we do not know f_0 and we use our predictor learnt from normal data as a proxy. This implies the prediction residuals are approximately independent on normal data (the predictor can explain data well), while the prediction residuals are, in general, correlated on abnormal data.

Applying the independent likelihood test in a correlated setting: We now prove Proposition 7.3.

Proof. By simple algebra we can write:

$$\frac{p_f(e(i) \mid E_c^{i-1})}{p_p(e(i))} = \frac{p_f(e(i) \mid E_c^{i-1})}{p_f(e(i))} \frac{p_f(e(i))}{p_p(e(i))} \qquad \forall i$$

Now recall the cumulative sum of the log-likelihood ratios taken under the current data generating mechanism $p_f(E_1^t)$ provides an estimate of the expected value of the log-likelihood ratio. Due to the correlated nature of data E_1^t the samples are drawn from a multidimensional distribution of dimension t (a sample from this distribution is an entire trajectory from c to t).

We now take the expectation of previous formula w.r.t. the "true" distribution

$p_f(E_1^t)$:

$$\begin{aligned}
\mathbb{E}_{p_f(E_c^t)}\Omega_c^t &= \mathbb{E}_{p_f(E_c^t)} \log \prod_{i=c}^{t} \frac{p_f(e(i) \mid E_c^{i-1})}{p_p(e(i))} \\
&= \mathbb{E}_{p_f(E_c^t)} \log \prod_{i=c}^{t} \frac{p_f(e(i) \mid E_c^{i-1})}{p_f(e(i))} + \mathbb{E}_{p_f(E_c^t)} \log \prod_{i=c}^{t} \frac{p_f(e(i))}{p_p(e(i))} \\
&= MI\Big(p_f(E_c^t); \prod_{i=c}^{t} p_f(e(i))\Big) + KL\Big(\prod_{i=c}^{t} p_f(e(i)) \Big|\Big| \prod_{i=c}^{t} p_p(e(i))\Big) \\
&\geq KL\Big(\prod_{i=c}^{t} p_f(e(i)) \Big|\Big| \prod_{i=c}^{t} p_p(e(i))\Big)
\end{aligned}$$

where KL is the Kullback-Leibler divergence and we use the fact the mutual information (MI) is always non negative. ■

D.6 Datasets

D.6.1 Yahoo Dataset

Yahoo Webscope dataset [Laptev and Amizadeh, 2020] is a publicly available dataset containing 367 real and synthetic time series with point anomalies, contextual anomalies and change points. Each time series contains 1420-1680 time stamps. This dataset is further divided into 4 sub-benchmarks: A1 Benchmark, A2 Benchmark, A3 Benchmark and A4 Benchmark. A1Benchmark is based on the real production traffic to some of the Yahoo! properties. The other 3 benchmarks are based on synthetic time series. A2 and A3 Benchmarks include point outliers, while the A4Benchmark includes change-point anomalies. All benchmarks have labelled anomalies. We use such information only during evaluation phase (since our method is completely unsupervised).

D.6.2 NAB Dataset

NAB (Numenta Anomaly Benchmark) [Lavin and Ahmad, 2015] is a publicly available anomaly detection benchmark. It consists of 58 data streams, each with 1,000 - 22000 instances. This dataset contains streaming data from different domains including read traffic, network utilization, on-line advertisement, and internet traffic. As done in [Geiger et al., 2020] we choose a subset of NAB benchmark, in particular we focus on the NAB Traffic and NAB Tweets benchmarks.

Table D.1: **Anomaly detection datasets summaries**. We report some properties of the datasets used (see [Geiger et al., 2020] for mode details).

		Yahoo				NAB		Kaggle
		A1	A2	A3	A4	Traffic	Tweets	CO2
Properties	# signals	67	100	100	100	7	10	9
	# anomalies	178	200	939	835	14	33	
	point	68	33	935	833	0	0	
	sequential	110	167	4	2	14	33	
	# anomalous points	1669	466	943	837	1560	15651	
	(% tot)	1.8%	0.32%	0.56%	0.5%	9.96%	9.87%	
	# data points	94866	142100	168000	168000	15662	158511	4323

D.6.3 CO2 Dataset

We test the prediction and interpretability capabilities of our model on the CO2 dataset from kaggle[1]. The main goal here is to predict both trend and periodicity of CO2 emission rates on different years. This is not an Anomaly detection task.

D.6.4 NYT Dataset

The New York Times Annotated Corpus[2] [Sandhaus, 2008] contains over 1.8 million articles written and published by the New York Times between January 1, 1987 and June 19, 2007. We pre-processed the lead paragraph of each article with a pre-trained BERT model [Devlin et al., 2019] from the HuggingFace Transformers library [Wolf et al., 2020] and extracted the 768-dimensional hidden state of the [CLS] token (which serves as an article-level embedding). For each day between January 1, 2000 and June 19, 2007, we took the mean of the embeddings of all articles from that day. Finally, we computed a PCA and kept the first 200 principal components (which explain approximately 95% of the variance), thus obtaining a 200-dimensional time series spanning 2727 consecutive days. Note that we did not use any of the dataset's annotations, contrary to prior work such as [Rayana and Akoglu, 2015].

D.7 Experimental setup

In this section, we shall describe the experimental setup we used to test STRIC.

Data preprocessing: Before learning the predictor we standardize each dataset to

[1] https://www.kaggle.com/txtrouble/carbon-emissions
[2] https://catalog.ldc.upenn.edu/LDC2008T19

have zero mean and standard deviation equals to one. As done in [Braei and Wagner, 2020] we note standardization is not equal to normalization, where data are forced to belong to the interval $(0, 1)$. Normalization is more sensitive to outliers, thus it would be inappropriate to normalize our datasets, which contain outliers.

We do not apply any deseasonalizing or detrending pre-processing.

Data splitting: We split each dataset into training and test sets preserving time ordering, so that the first data are used as train set and the following ones are used as test set. The data used to validate the model during optimization are last 10% of the training dataset. Depending on the experiment, we choose a different percentage in splitting train and test. When comparing with [Braei and Wagner, 2020] we used 30% as training data, while when comparing to [Munir et al., 2019] we use 40%. Such a choice is dictated by the particular (non uniform) experimental setup reported in [Braei and Wagner, 2020, Munir et al., 2019] and has been chosen to produce comparable results with state of the art methods present in literature.

Evaluation metrics: We compare different predictors by means of the RMSE (root mean squared error) on the one-step ahead prediction errors. Given a sequence of data Y_1^N and the one-step ahead predictions $\hat{Y}_1^N$ the RMSE is defined as:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\|y(i)-\hat{y}(i)\|^2}$$

As done in [Braei and Wagner, 2020] we compare different anomaly detection methods taking into account several metrics. We use F-Score which is defined as the armonic mean of Precision and Recall (see [Braei and Wagner, 2020, Munir et al., 2019]) and another metric that is often used is *receiver operating characteristic curve*, ROC-Curve, and its associated metric *area under the curve* (AUC). The AUC is defined as the area under the ROC-Curve. This metric is particularly useful in our anomaly detection setting since it describes with an unique number *true positive rate* and *false positive rate* on different threshold values. We now follow [Braei and Wagner, 2020] to describe how AUC is computed. Let the *true positive rate* and *false positive rate* be defined, respectively, as: $TPR = \frac{TP}{P}$ and $FPR = \frac{FP}{N}$, where TP stands for *true positive*, P for *positive*, FP for *false positive* and N for *negative*. To copute the ROC-Curve we use different thresholds on our anomaly detection method. We therefore have different pairs of TPR and FPR for each threshold. These values can be plotted on a plot whose x and y axes are, respectively: FPR and TPR. The resulting curve starts at the origin and ends in the point (1,1). The AUC is the area under this curve. In anomaly detec-

tion, the AUC expresses the probability that the measured algorithm assigns a random anomalous point in the time series a higher anomaly score than a random normal point.

Hardware: We conduct our experiments on the following hardware setup:

- Processor: Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz
- RAM: 128 Gb
- GPU: Nvidia TITAN V 12Gb and Nvidia TITAN RTX 24Gb

Hyper-parameters: All the experiments we carried out are uniform on the optimization hyper-parameters. In particular we fixed the maximum number of epochs to 300, the learning rate to 0.001 and batch size to 100. We optimize each model using Adam and early stopping.

We fix STRIC's first module hyper-parameters as follows:

- number of filter per block: $l_0 = 10$, $l_1 = 100$, $l_2 = 200$
- linear filters kernel lengths (N_0, N_1, N_2): half predictor's memory

In all experiments we either use a TCN composed of 3 hidden layers with 300 nodes per layer or a TCN with 8 layers and 32 nodes per layer. Moreover we chose $N_3 = 5$ (TCN kernels' lengths) and ReLU activation functions [Bai et al., 2018].

Comparison with SOTA methods: We tested our model against other SOTA methods (Table 7.2) in a comparable experimental setup. In particular, we chose comparable window lengths and architecture sizes (same order of magnitude of the number of parameters) to make the comparison as fair as possible. For the hyper-parameters details of any SOTA method we used we refer the relative cited references. We point out that while the window length is a critical hyper-parameter for the accuracy of many methods, our architecture is robust w.r.t. choice of window length: thanks to our fading regularization, the user is required only to choose a window length larger than the optimal one and then our automatic complexity selection is guaranteed to find the optimal model complexity given the available data Section 5.3.1.

Anomaly scores: When computing the F-score we use the predictions of the CUMSUM detector which we collect as a binary vector whose length is the same as the number of available data. Ones are associated to the presence of an anomalous time instants while zeros are associated to normality.

When computing the AUC we need to consider a continuous anomaly score, therefore the zero-one encoded vector from the CUMSUM is not usable. We compute the anomaly scores for each time instant as the estimated likelihood ratios. Since we write the

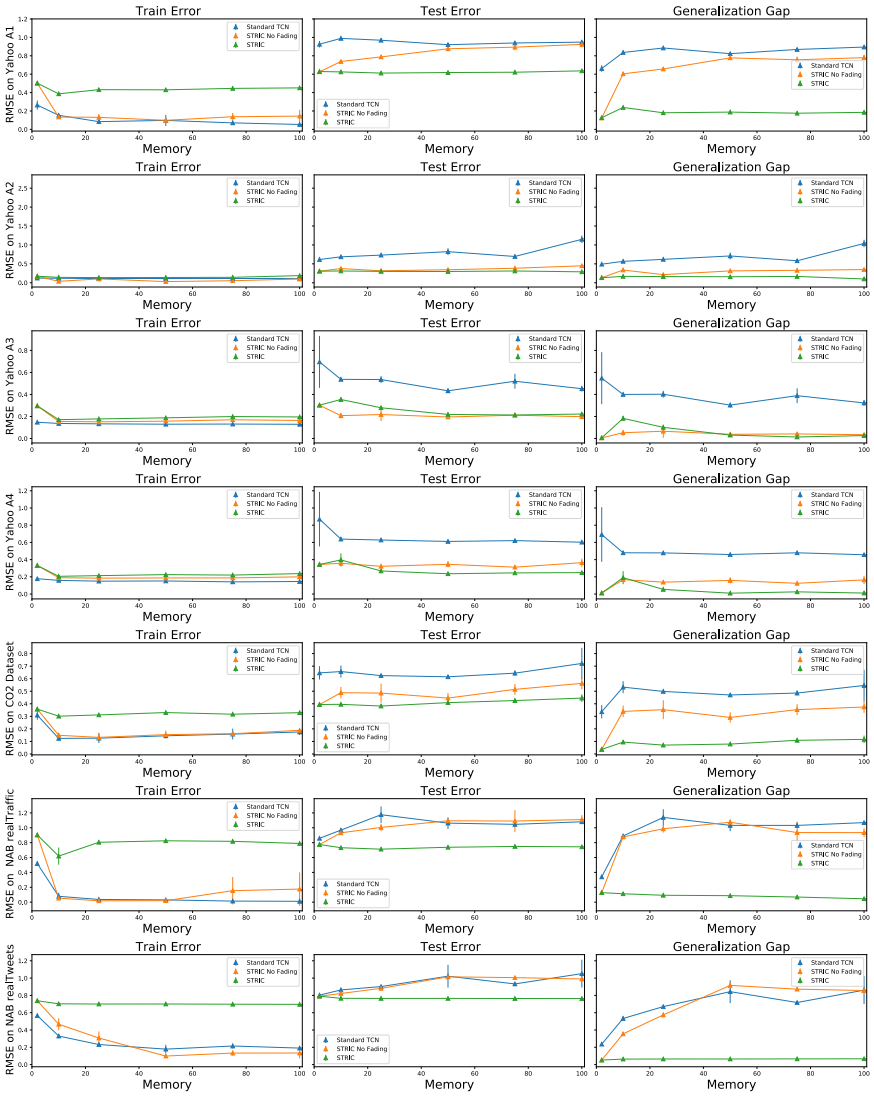


Figure D.1: **STRIC ablation studies on different datasets**: Effects of interpretable blocks and fading regularization on model's forecasting as the available window of past data increases (memory). **Left Panel**: Train error. **Center Panel**: Test error. **Right Panel**: Generalization Gap. The test error of STRIC is uniformiy smaller than a standard TCN (without interpretable blocks nor fading regularization). Adding interpretable blocks to a standard TCN improves generalization for fixed memory w.r.t. Standard TCN but get worse (overfitting occurs) as soon as the available past data horizon increase. Fading regularization is effective: STRIC generalization GAP is almost constant w.r.t. past horizon.

likelihood ratio as $\frac{p_f}{p_p}$, it is large when data does not come from p_p (which we consider the reference distribution).

D.7.1 Ablation study

In Fig. D.1 we show different metrics based on the predictor's RMSE (training, test and generalization gap) as a function of the memory of the predictor. We test our fading regularization on a variety of different datasets. In all situations fading regularization helps improving test generalization and preserving the generalization gap (by keeping it constant) as the model complexity increases. All plots show confidence intervals around mean values evaluated on 10 different random seeds.

In Table D.2 we extend the results we show in the main paper by adding uncertainties (measured by standard deviations on 10 different random seeds) to the values of train and test RMSE on different ablations of STRIC. Despite the high variability across different datasets STRIC achieves the most consistent results (smaller standard deviations both on training and testing).

Finally, in Table D.3 we show the effects on different choices of the predictor's memory n_{pred} and length of the anomaly detectors windows $n_{\text{det}} = n_f = n_p$ on the detection performance of STRIC. Note both F-score and AUC are highly sensible to the choice of n_{det}: the best results are achieve for small windows. On the other hand when n_{det} is large the performance drops. This is due to the type of anomalies present in the Yahoo benchmark: most of the them can be considered to be point anomalies. In fact, as we showed in Section 7.4.2, our detector is less sensible to point anomalies when a large window n_{det} is chosen.

In Table D.3 we also report the reconstruction error of the optimal predictor given it's memory n_{pred}. Note small memory in the predictor introduce modelling bias (higher training error) while a large memory does not (thanks to fading regularization). As we observed in Appendix D.5 better predictive models provide the detection module with more discriminative residuals: the downstream detection module achieves better F-scores and AUC.

D.7.2 Comparison TCN vs STRIC

In this section we show standard non-linear TCN without regularization and proper inductive bias might not generalize on non-stationary time series (e.g. time series with non zero trend component) and TCN architecture. In Fig. D.2 we compare the prediciton errors of a standard TCN model against our STRIC on the A3 Yahoo dataset.

Table D.2: **Ablation study on the RMSE of prediciton errors with standard deviation on 10 different seeds**: We compare a standard TCN model with our STRIC predictor and some variation of it (using the same train hyper-parameters).

		TCN		**TCN + Linear**		**TCN + Fading**		**STRIC pred**	
		Train	Test	Train	Test	Train	Test	Train	Test
Datasets	Yahoo A1	**0.10** ± 0.06	0.92 ± 0.06	**0.10** ± 0.03	0.88 ± 0.03	0.44 ± 0.03	0.92 ± 0.03	0.43 ± 0.02	**0.62** ± 0.02
	Yahoo A2	**0.11** ± 0.02	0.82 ± 0.02	0.13 ± 0.01	0.35 ± 0.02	0.20 ± 0.01	0.71 ± 0.01	0.14 ± 0.01	**0.30** ± 0.01
	Yahoo A3	**0.13** ± 0.01	0.43 ± 0.01	0.16 ± 0.01	**0.22** ± 0.01	0.15 ± 0.01	0.40 ± 0.01	0.19 ± 0.01	**0.22** ± 0.01
	Yahoo A4	**0.15** ± 0.01	0.61 ± 0.01	0.19 ± 0.01	0.35 ± 0.01	0.17 ± 0.01	0.55 ± 0.01	0.23 ± 0.01	**0.24** ± 0.01
	CO2 Dataset	**0.14** ± 0.02	0.62 ± 0.02	0.15 ± 0.02	0.45 ± 0.02	0.18 ± 0.03	0.61 ± 0.03	0.33 ± 0.01	**0.41** ± 0.01
	NAB Traffic	**0.03** ± 0.01	1.06 ± 0.02	0.04 ± 0.01	1.00 ± 0.02	0.62 ± 0.01	0.93 ± 0.01	0.83 ± 0.02	**0.74** ± 0.02
	NAB Tweets	**0.18** ± 0.05	1.02 ± 0.05	0.20 ± 0.05	0.98 ± 0.05	0.47 ± 0.02	0.83 ± 0.02	0.70 ± 0.01	**0.77** ± 0.01

Table D.3: **Sensitivity of STRIC to hyper-parameters**: We compare STRIC on different anomaly detection benchmarks datasets using different hyper-parameters: memory of the predictor n_{pred} and length of anomaly detector windows $n_p = n_f = n_{\text{det}}$.

		Yahoo A1		**Yahoo A2**		**Yahoo A3**		**Yahoo A4**	
		F1	AUC	F1	AUC	F1	AUC	F1	AUC
Models	$n_{\text{pred}} = 10$, $n_{\text{det}} = 2$	0.45	0.89	0.63	0.99	0.87	0.99	0.64	0.89
	$n_{\text{pred}} = 100$, $n_{\text{det}} = 2$	**0.48**	**0.9308**	**0.98**	**0.9999**	**0.99**	**0.9999**	**0.68**	**0.9348**
	$n_{\text{pred}} = 10$, $n_{\text{det}} = 20$	0.10	0.58	0.63	0.99	0.47	0.83	0.37	0.72
	$n_{\text{pred}} = 100$, $n_{\text{det}} = 20$	0.10	0.55	**0.98**	**0.9999**	0.49	0.86	0.35	0.76
		Yahoo A1		**Yahoo A2**		**Yahoo A3**		**Yahoo A4**	
		Train	Test	Train	Test	Train	Test	Train	Test
Models	$n_{\text{pred}} = 10$	0.44	0.62	0.16	0.31	0.22	0.23	0.25	0.26
	$n_{\text{pred}} = 100$	0.42	0.61	0.14	0.30	0.19	0.22	0.23	0.24

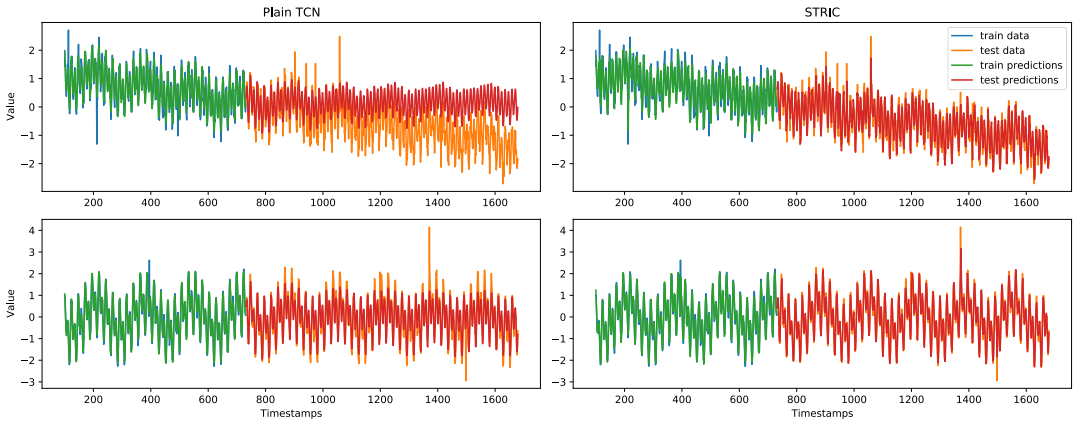


Figure D.2: **STRIC vs off-the-shelf TCN**: We compare an off-the-shelf TCN against STRIC (time series predictor) on the Yahoo dataset A3 Benchmark. Note the standard TCN overfits compared to STRIC: the standard TCN does not handle correctly the trend component of the signal (**First row**). If we consider a time series without trend, the standard TCN model performs better but overfitting is still present. In particular the generalization gap (measured using squared reconstruction error) for the two models is: Standard TCN 0.3735 and STRIC 0.0135.

We train both models using the same optimization hyper-parameters (as described in previous section). Note a plain TCN does not necessarily capture the trend component in the test set.

D.8 STRIC vs SOTA Anomaly Detectors

In this section we further expand the discussion on the main differences between STRIC and other SOTA anomaly detectors by commenting results obtained in Table 7.2 and Table D.4. Table D.4 highlights the relative performance of STRIC when the pefor-mance are nearly saturated (e.g. Yahoo A2 and A3). For each comparing SOTA method we report the following: $\frac{\text{AUC}_{\text{method}} - \text{AUC}_{\text{STRIC}}}{1 - \text{AUC}_{\text{STRIC}}} \cdot 100$ (similarly for F1).

To begin with, STRIC outperforms "traditional" methods (LOF and One-class SVM) which are considered as baselines models for comparing time series anomaly detectors.

Comparison with other Deep Learning based methods: STRIC outperforms most of the SOTA Deep Learning based methods reported in Table 7.2: TadGAN, TAnoGAN, DeepAnT and DeepAR (the last one is a SOTA time series predictor). Note the relative improvement of STRIC is higher on the Yahoo dataset where statistical models outperforms deep learning based ones. We believe this is due to both fading regularization and the seasonal-trend decomposition performed by STRIC.

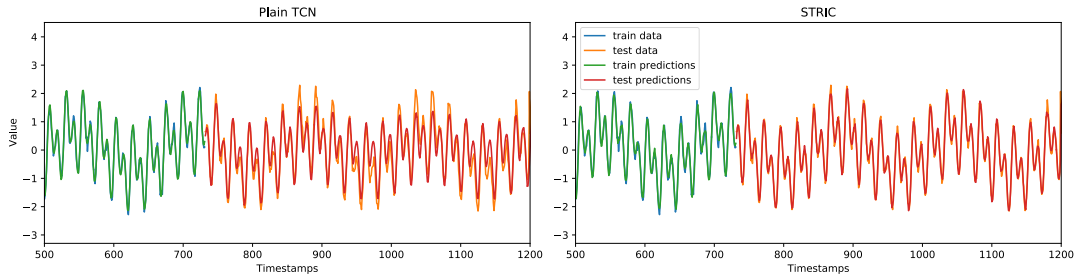


Figure D.3: Zoom on the second row of panels in Fig. D.2. We show the interface between train and test data both on a plain TCN and on our STRIC predictor. A plain TCN overfits w.r.t. STRIC also when not trend is present.

Despite the general applicability of GOAD [Bergman and Hoshen, 2020] this method has not been designed to handle time series, but images and tabular data. "Geometric" transformations which have been considered in GOAD and actually have inspired it (rotations, reflections, translations) might not be straightforwardly applied to time series. Nevertheless, while we have not been able to find in the literature any direct and principled extension of this work to the time series domain, we have implemented and compared against [Bergman and Hoshen, 2020] by extending the main design ideas of GOAD to time-series. So that we applied their method on lagged windows extracted from time series (exploiting the same architectures proposed for tabular data case with some minor modifications). We report the results we obtained by running the GOAD's official code on all our benchmark datasets. Overall, STRIC performs (on average) 70% better than GOAD on the Yahoo dataset and 15% better on the NAB dataset.

D.8.1 Details on the NYT experiment

Fig. 7.9 shows the normalized anomaly score computed by STRIC on the NYT dataset, following the setup described in Section D.6.4. Some additional insights can be gained by zooming in around some of the detected change-points. In Fig. D.4 (left), we see that the anomaly score (blue line) rapidly increases immediately after the 9/11 attack and reaches its peak some days later. Such delay is inherently tied to our choice of time scale, that privileges the detection of prolonged anomalies as opposed to single-day anomalies (which are not meaningful due to the high variability of the news content). The change-point which occurs the day after the 9/11 attack is reflected by a sudden increase of the relative frequency of article descriptors such as "Terrorism" (orange line). Article descriptors are annotated in the NYT dataset, but they are not given as input to STRIC so that we do not rely on any human annotations. However, they can

Table D.4: **Comparison with SOTA anomaly detectors:** We compare STRIC with other anomaly detection methods on the experimental setup and the same evaluation metrics proposed in [Braei and Wagner, 2020, Munir et al., 2019]. The baseline models are: MA, ARIMA, LOF [Shen et al., 2020], LSTM [Braei and Wagner, 2020, Munir et al., 2019], Wavenet [Braei and Wagner, 2020] , Yahoo EGADS [Munir et al., 2019] , GOAD [Bergman and Hoshen, 2020], OmniAnomaly [Su et al., 2019], Twitter AD [Munir et al., 2019], TanoGAN [Bashar and Nayak, 2020], TadGAN [Geiger et al., 2020] , DeepAR [Flunkert et al., 2017] and DeepAnT [Munir et al., 2019] . STRIC outperforms most of the other methods based on statistical models and based on DNNs. Same as Table 7.2, here we report the relative improvements w.r.t. STRIC (the higher the better).

	Relative F1-score improvement over STRIC in %	**Yahoo A1**	**Yahoo A2**	**Yahoo A3**	**Yahoo A4**	**NAB Tweets**	**NAB Traffic**
Models	ARIMA	-20	-88	-42	**6**	-33	-37
	LSTM	-7	-33	-60	- 21		
	Yahoo EGADS	-1	-95	-78	-54		
	OmniAnomaly	-1	-60	-45	-11	-6	-10
	Twitter AD	**0**	-98	-85	-53		
	TanoGAN	-11	-85	-73	-13	-36	-44
	TadGAN	-13	-85	-65	-20	-25	-47
	DeepAR	-29	-72	-79	-41	-37	-32
	DeepAnT	-4	-67	-15	0		
	STRIC (ours)	**0**	**0**	**0**	0	**0**	**0**

	Relative AUC improvement over STRIC in %	**Yahoo A1**	**Yahoo A2**	**Yahoo A3**	**Yahoo A4**	**NAB Tweets**	**NAB Traffic**
Models	MA	-47	-98	-98	**379**		
	ARIMA	-45	-99	-99	124		
	LOF	-28	-99	-99	-81	-32	-44
	Wavenet	-60	-99	-99	-84		
	LSTM	-63	-99	-99	-84		
	GOAD	-37	-99	-99	-51	-19	-12
	DeepAnT	-32	-99	-99	-53	-23	-13
	STRIC (ours)	**0**	**0**	**0**	0	**0**	**0**

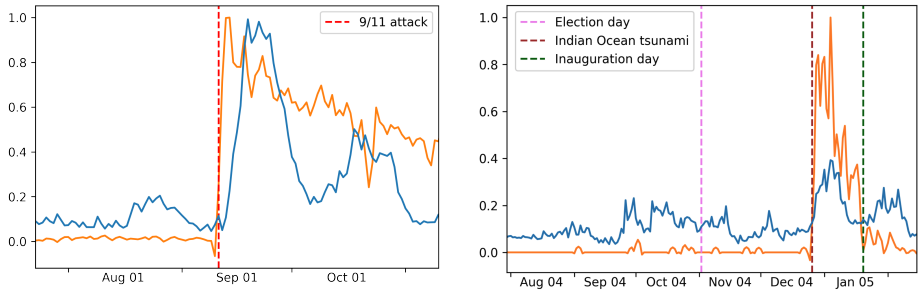


Figure D.4: A closer look at some of the change-points detected by STRIC. **Left**: Normalized anomaly score (blue line) and normalized frequency of the "Terrorism" descriptor (orange line) around the 9/11 attack. **Right**: Normalized anomaly score (blue line) and normalized frequency of the "Earthquakes" descriptor (orange line) in the second half of 2004 and beginning of 2005. The 2004 U.S. election causes an increase in the anomaly score, but the most significant change-point occurs after the Indian Ocean tsunami.

help interpreting the change-points found by STRIC.

In Fig. D.4 (right), we can observe that the anomaly score (blue line) is higher in the months around the 2004 U.S. election and immediately after the inauguration day. However, the highest values for the anomaly score occur around the end of 2004, shortly after the Indian Ocean tsunami. Indeed, this is reflected by an abrupt increase of the frequency of descriptors like "Earthquakes" (orange line) and "Tsunami".

We note this experiment is qualitative and unfortunately we are not aware of any ground truth or metrics we can compare against (e.g. in [Rayana and Akoglu, 2015] a similar qualitative result has been reported on the NYT dataset). We therefore tested STRIC against a simple baseline which uses PCA on BERT features and a threshold to detect anomalies. Despite being a simple baseline thie method prooved to be highly applied in practice due to its simplicity [Blázquez-García et al., 2020]. The PCA + threshold baseline is able to pick up some events (2000 election, 9/11 attack, housing bubble) but is otherwise more noisy than STRIC's anomaly score. This is likely due to the lack of a modeling of seasonal/periodic components. For instance, the anomaly score of the simple baseline contains many false alarms which are related to normal weekly periodicity that is not easily modeled by the baseline. This does not affect STRIC's predictions since normal weekly periodicity is directly modeled and identified as normal behaviour.

References

[Achille et al., 2021] Achille, A., Golatkar, A., Ravichandran, A., Polito, M., and Soatto, S. (2021). Lqf: Linear quadratic fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15729–15739. (Cited on page 49.)

[Achille et al., 2019] Achille, A., Lam, M., Tewari, R., Ravichandran, A., Maji, S., Fowlkes, C. C., Soatto, S., and Perona, P. (2019). Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6430–6439. (Cited on pages 96 and 97.)

[Achille et al., 2019] Achille, A., Paolini, G., Mbeng, G., and Soatto, S. (2019). The Information Complexity of Learning Tasks, their Structure and their Distance. *arXiv e-prints*, page arXiv:1904.03292. (Cited on pages 1, 96, and 97.)

[Achille et al., 2019] Achille, A., Rovere, M., and Soatto, S. (2019). Critical learning periods in deep networks. In *International Conference on Learning Representations*. (Cited on pages 1 and 21.)

[Achille and Soatto, 2017] Achille, A. and Soatto, S. (2017). On the emergence of invariance and disentangling in deep representations. *CoRR*, abs/1706.01350. (Cited on pages 1, 2, and 102.)

[Achlioptas, 2003] Achlioptas, D. (2003). Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687. (Cited on page 68.)

[Adhikari and Agrawal, 2013] Adhikari, R. and Agrawal, R. K. (2013). An introductory study on time series modeling and forecasting. *CoRR*, abs/1302.6613. (Cited on page 131.)

[Allen-Zhu et al., 2019a] Allen-Zhu, Z., Li, Y., and Liang, Y. (2019a). Learning and generalization in overparameterized neural networks, going beyond two layers. 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. (Cited on pages 15, 23, 24, 45, and 50.)

[Allen-Zhu et al., 2018] Allen-Zhu, Z., Li, Y., and Song, Z. (2018). On the convergence rate of training recurrent neural networks. *arXiv preprint arXiv:1810.12065*. (Cited on pages 48, 50, and 104.)

[Allen-Zhu et al., 2019b] Allen-Zhu, Z., Li, Y., and Song, Z. (2019b). A convergence theory for deep learning via over-parameterization. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 242–252. PMLR. (Cited on pages 45, 48, 57, 60, and 62.)

[Aronszajn, 1950] Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404. (Cited on page 29.)

[Arora et al., 2019a] Arora, S., Du, S., Hu, W., Li, Z., and Wang, R. (2019a). Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332. (Cited on pages 21, 22, 24, 45, 50, 51, 52, 60, 63, and 67.)

[Arora et al., 2019b] Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R. R., and Wang, R. (2019b). On exact computation with an infinitely wide neural net. 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. (Cited on pages 3, 22, 23, 24, 25, 26, 28, 38, 45, and 165.)

[Bai et al., 2018] Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*. (Cited on pages 2, 101, 102, 103, 105, 106, 107, 110, 131, 132, 173, 175, 176, 177, 178, and 187.)

[Bansal et al., 2018] Bansal, N., Chen, X., and Wang, Z. (2018). Can we gain more from orthogonality regularizations in training deep networks? *NeurIPS*, 31:4261–4271. (Cited on pages 2, 102, 118, and 122.)

[Barz et al., 2019] Barz, B., Schröter, K., Münch, M., Yang, B., Unger, A., Dransch, D., and Denzler, J. (2019). Enhancing flood impact analysis using interactive retrieval of social media images. *ArXiv*, abs/1908.03361. (Cited on pages 91 and 167.)

[Bashar and Nayak, 2020] Bashar, M. A. and Nayak, R. (2020). Tanogan: Time series anomaly detection with generative adversarial networks. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1778–1785. IEEE. (Cited on pages 145 and 193.)

[Basseville and Nikiforov, 1993] Basseville, M. and Nikiforov, I. V. (1993). *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Inc., USA. (Cited on pages 132 and 135.)

[Bell et al., 2015] Bell, S., Upchurch, P., Snavely, N., and Bala, K. (2015). Material recognition in the wild with the materials in context database. *Computer Vision and Pattern Recognition (CVPR)*. (Cited on pages 71 and 157.)

[Bergman and Hoshen, 2020] Bergman, L. and Hoshen, Y. (2020). Classification-based anomaly detection for general data. In *International Conference on Learning Representations*. (Cited on pages 145, 192, and 193.)

[Bingham and Mannila, 2001] Bingham, E. and Mannila, H. (2001). Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250. (Cited on page 68.)

[Blázquez-García et al., 2020] Blázquez-García, A., Conde, A., Mori, U., and Lozano, J. A. (2020). A review on outlier/anomaly detection in time series data. *arXiv preprint arXiv:2002.04236*. (Cited on page 194.)

[Bossard et al., 2014] Bossard, L., Guillaumin, M., and Van Gool, L. (2014). Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*. (Cited on pages 88, 89, and 167.)

[Bottou, 1998] Bottou, L. (1998). On-line learning and stochastic approximations. In *In On-line Learning in Neural Networks*, pages 9–42. Cambridge University Press. (Cited on page 17.)

[Bottou et al., 2018] Bottou, L., Curtis, F., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311. (Cited on page 102.)

[Braei and Wagner, 2020] Braei, M. and Wagner, S. (2020). Anomaly detection in univariate time-series: A survey on the state-of-the-art. *CoRR*, abs/2004.00433. (Cited on pages 129, 130, 131, 132, 141, 142, 143, 144, 145, 153, 186, and 193.)

[Brutzkus et al., 2017] Brutzkus, A., Globerson, A., Malach, E., and Shalev-Shwartz, S. (2017). SGD learns over-parameterized networks that provably generalize on linearly separable data. *CoRR*, abs/1710.10174. (Cited on page 59.)

[Chaudhari and Soatto, 2017] Chaudhari, P. and Soatto, S. (2017). Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. *CoRR*, abs/1710.11029. (Cited on pages 64, 65, and 162.)

[Chaudhari and Soatto, 2018] Chaudhari, P. and Soatto, S. (2018). Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. *2018 Information Theory and Applications Workshop (ITA)*, pages 1–10. (Cited on pages 1, 2, 69, and 102.)

[Chen et al., 2020] Chen, Z., Cao, Y., Gu, Q., and Zhang, T. (2020). A generalized neural tangent kernel analysis for two-layer neural networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13363–13373. Curran Associates, Inc. (Cited on pages 15, 23, and 53.)

[Cheng et al., 2017] Cheng, G., Han, J., and Lu, X. (2017). Remote sensing image scene classification: Benchmark and state of the art. (Cited on pages 88, 89, 167, 169, and 170.)

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*. (Cited on pages 1 and 105.)

[Cimpoi et al., 2014] Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., , and Vedaldi, A. (2014). Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. (Cited on pages 91, 93, and 167.)

[Cleveland et al., 1990] Cleveland, R. B., Cleveland, W. S., McRae, J. E., and Terpenning, I. (1990). Stl: A seasonal-trend decomposition procedure based on loess (with discussion). *Journal of Official Statistics*, 6:3–73. (Cited on pages 131 and 141.)

[Cortes et al., 2012] Cortes, C., Mohri, M., and Rostamizadeh, A. (2012). Algorithms for learning kernels based on centered alignment. *J. Mach. Learn. Res.*, 13(1):795–828. (Cited on pages 39, 42, 43, 44, 83, 84, and 85.)

[Cover and Thomas, 1991] Cover, T. and Thomas, J. (1991). *Elements of Information Theory*. Series in Telecommunications and Signal Processing. Wiley. (Cited on page 113.)

[Cristianini et al., 2002] Cristianini, N., Shawe-Taylor, J., Elisseeff, A., and Kandola, J. (2002). On kernel-target alignment. In Dietterich, T., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press. (Cited on pages 39, 40, 41, 44, 83, and 84.)

[Cui et al., 2018] Cui, Y., Song, Y., Sun, C., Howard, A., and Belongie, S. (2018). Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4109–4118. (Cited on pages 79, 80, 81, 88, 92, 95, 96, 97, 152, and 167.)

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314. (Cited on pages 1, 13, 101, 107, and 119.)

[dataset, 2016] dataset, C.-. (2016). https://github.com/workpiles/cucumber-9. (Cited on pages 91 and 167.)

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*. (Cited on pages 79, 88, 89, 96, and 167.)

[Deshpande et al., 2021] Deshpande, A., Achille, A., Ravichandran, A., Li, H., Zancato, L., Fowlkes, C. C., Bhotika, R., Soatto, S., and Perona, P. (2021). A linearized framework and a new benchmark for model selection for fine-tuning. *CoRR*, abs/2102.00084. (Cited on pages 4, 21, 44, 45, 49, 50, 52, and 53.)

[Devlin et al., 2019] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics. (Cited on pages 1, 101, 144, and 185.)

[Dosovitskiy et al., 2021] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*. (Cited on pages 2, 101, and 102.)

[Du et al., 2019a] Du, S., Lee, J., Li, H., Wang, L., and Zhai, X. (2019a). Gradient descent finds global minima of deep neural networks. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1675–1685, Long Beach, California, USA. PMLR. (Cited on pages 57, 60, and 62.)

[Du et al., 2018a] Du, S. S., Lee, J. D., Li, H., Wang, L., and Zhai, X. (2018a). Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*. (Cited on pages 15, 23, 45, 48, 50, and 59.)

[Du et al., 2018b] Du, S. S., Zhai, X., Poczos, B., and Singh, A. (2018b). Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*. (Cited on pages 45 and 60.)

[Du et al., 2019b] Du, S. S., Zhai, X., Poczos, B., and Singh, A. (2019b). Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*. (Cited on pages 15, 23, 48, and 50.)

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159. (Cited on page 17.)

[Dvornik et al., 2020] Dvornik, N., Schmid, C., and Mairal, J. (2020). Selecting relevant features from a universal representation for few-shot classification. *arXiv preprint arXiv:2003.09338*. (Cited on page 97.)

[Dwivedi and Roig, 2019] Dwivedi, K. and Roig, G. (2019). Representation similarity analysis for efficient task taxonomy & transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 12387–12396. Computer Vision Foundation / IEEE. (Cited on pages 80, 81, 87, 88, 92, 95, 97, 152, and 167.)

[Fan and Wang, 2020] Fan, Z. and Wang, Z. (2020). Spectra of the conjugate kernel and neural tangent kernel for linear-width neural networks. *arXiv preprint arXiv:2005.11879*. (Cited on page 158.)

[Farahmand et al., 2017] Farahmand, A.-m., Pourazarm, S., and Nikovski, D. (2017). Random projection filter bank for time series data. 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. (Cited on pages 133 and 177.)

[Flunkert et al., 2017] Flunkert, V., Salinas, D., and Gasthaus, J. (2017). Deepar: Probabilistic forecasting with autoregressive recurrent networks. *CoRR*, abs/1704.04110. (Cited on pages 130, 145, and 193.)

[Friedman et al., 2001] Friedman, J., Hastie, T., Tibshirani, R., et al. (2001). *The elements of statistical learning*, volume 1. Springer series in statistics New York. (Cited on pages 21, 32, 39, 83, and 84.)

[Garriga-Alonso et al., 2018] Garriga-Alonso, A., Rasmussen, C. E., and Aitchison, L. (2018). Deep convolutional networks as shallow gaussian processes. *arXiv preprint arXiv:1808.05587*. (Cited on pages 24, 28, and 45.)

[Geiger et al., 2020] Geiger, A., Liu, D., Alnegheimish, S., Cuesta-Infante, A., and Veeramachaneni, K. (2020). Tadgan: Time series anomaly detection using generative adversarial networks. *arXiv preprint arXiv:2009.07769*. (Cited on pages 129, 130, 131, 132, 143, 145, 184, 185, and 193.)

[Golatkar et al., 2019] Golatkar, A. S., Achille, A., and Soatto, S. (2019). Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. 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. (Cited on pages 2, 21, and 102.)

[Goldblum et al., 2019] Goldblum, M., Geiping, J., Schwarzschild, A., Moeller, M., and Goldstein, T. (2019). Truth or backpropaganda? an empirical investigation of deep learning theory. (Cited on pages 45, 58, 60, and 70.)

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org. (Cited on pages 1, 2, 13, 18, 19, 103, 104, and 105.)

[Goyal et al., 2019] Goyal, P., Mahajan, D., Gupta, A., and Misra, I. (2019). Scaling and benchmarking self-supervised visual representation learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6391–6400. (Cited on page 97.)

[Guen et al., 2020] Guen, V. L., Yin, Y., Dona, J., Ayed, I., de Bézenac, E., Thome, N., and Gallinari, P. (2020). Augmenting physical models with deep networks for complex dynamics forecasting. *arXiv preprint arXiv:2010.04456*. (Cited on page 131.)

[Hardt et al., 2015] Hardt, M., Recht, B., and Singer, Y. (2015). Train faster, generalize better: Stability of stochastic gradient descent. *CoRR*, abs/1509.01240. (Cited on page 49.)

[Hayou et al., 2019] Hayou, S., Doucet, A., and Rousseau, J. (2019). Mean-field behaviour of neural tangent kernel for deep neural networks. *arXiv preprint arXiv:1905.13654*. (Cited on pages 58, 60, 63, 162, and 163.)

[He et al., 2019] He, K., Girshick, R., and Dollár, P. (2019). Rethinking imagenet pre-training. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4918–4927. (Cited on page 96.)

[He et al., 2016] 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*, pages 770–778. (Cited on pages 1, 2, 101, 102, 103, and 107.)

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. (Cited on pages 82, 87, and 89.)

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780. (Cited on pages 1 and 105.)

[Horn et al., 2017] Horn, G. V., Mac Aodha, O., Song, Y., Shepard, A., Adam, H., Perona, P., and Belongie, S. J. (2017). The inaturalist challenge 2017 dataset. *CoRR*, abs/1707.06642. (Cited on pages 88, 89, 96, and 167.)

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366. (Cited on pages 1, 13, and 101.)

[Huang et al., 2016] Huang, G., Liu, Z., and Weinberger, K. Q. (2016). Densely connected convolutional networks. *CoRR*, abs/1608.06993. (Cited on pages 87 and 89.)

[Huang et al., 2018] Huang, Y., Qiu, C., Guo, Y., Wang, X., and Yuan, K. (2018). Surface defect saliency of magnetic tile. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 612–617. (Cited on pages 91, 93, and 167.)

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167. (Cited on pages 19, 107, 114, 118, and 121.)

[Jacot et al., 2018] Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural*

information processing systems, pages 8571–8580. (Cited on pages 2, 3, 21, 22, 24, 25, 26, 45, 58, 59, 60, 63, and 79.)

[Jaeger, 2003] Jaeger, H. (2003). Adaptive nonlinear system identification with echo state networks. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems*, volume 15. MIT Press. (Cited on page 105.)

[Joly and Buisson, 2009] Joly, A. and Buisson, O. (2009). Logo retrieval with a contrario visual query expansion. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, pages 581–584. (Cited on pages 93 and 167.)

[Juditsky et al., 1995] Juditsky, A., Hjalmarsson, H., Benveniste, A., Delyon, B., Ljung, L., SjÖberg, J., and Zhang, Q. (1995). Nonlinear black-box models in system identification: Mathematical foundations. *Automatica*, 31(12):1725 – 1750. (Cited on page 117.)

[Justus et al., 2018] Justus, D., Brennan, J., Bonner, S., and McGough, A. S. (2018). Predicting the computational cost of deep learning models. *CoRR*, abs/1811.11880. (Cited on page 59.)

[Kanamori et al., 2009] Kanamori, T., Hido, S., and Sugiyama, M. (2009). A least-squares approach to direct importance estimation. *Journal of Machine Learning Research*, 10(48):1391–1445. (Cited on page 136.)

[Kawaguchi, 2016] Kawaguchi, K. (2016). Deep learning without poor local minima. *arXiv preprint arXiv:1605.07110*. (Cited on page 19.)

[Keskar et al., 2016a] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016a). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*. (Cited on page 18.)

[Keskar et al., 2016b] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016b). On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836. (Cited on page 59.)

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. (Cited on pages 2, 17, 118, and 122.)

[Klein et al., 2017] Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F. (2017). Learning curve prediction with bayesian neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. (Cited on page 59.)

[Kolesnikov et al., 2020] Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., and Houlsby, N. (2020). Big transfer (bit): General visual representation learning. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J.-M., editors, *Computer Vision – ECCV 2020*, pages 491–507, Cham. Springer International Publishing. (Cited on page 79.)

[Krause et al., 2013] Krause, J., Stark, M., Deng, J., and Fei-Fei, L. (2013). 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia. (Cited on pages 71, 93, 157, and 167.)

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Master's thesis, Computer Science Department, University of Toronto. (Cited on pages 71 and 157.)

[Krizhevsky, 2012] Krizhevsky, A. (2012). Learning multiple layers of features from tiny images. *University of Toronto*. (Cited on page 80.)

[Krizhevsky et al., 2012] 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. (Cited on pages 1, 2, 101, 102, and 103.)

[Laptev and Amizadeh, 2020] Laptev, N. and Amizadeh, S. (2020). Yahoo! webscope dataset ydata-labeled-time-series-anomalies-v1_0. *CoRR*. (Cited on pages 132, 139, and 184.)

[Lavin and Ahmad, 2015] Lavin, A. and Ahmad, S. (2015). Evaluating real-time anomaly detection algorithms - the numenta anomaly benchmark. *CoRR*, abs/1510.03336. (Cited on pages 132, 139, and 184.)

[Lee et al., 2017] Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. (2017). Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*. (Cited on pages 25, 45, and 59.)

[Lee et al., 2019] Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. (2019). Wide neural networks of any depth evolve as linear models under gradient descent. (Cited on pages 3, 22, 26, 27, 45, 47, 48, 52, 53, 58, 62, 63, 65, 67, 75, 79, and 162.)

[Li et al., 2020] Li, H., Chaudhari, P., Yang, H., Lam, M., Ravichandran, A., Bhotika, R., and Soatto, S. (2020). Rethinking the hyperparameters for fine-tuning. In *ICLR*. (Cited on pages 1, 60, 65, 79, and 96.)

[Li et al., 2018] Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the loss landscape of neural nets. 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. (Cited on pages 1, 2, 18, 21, and 102.)

[Li and Yuan, 2017] Li, Y. and Yuan, Y. (2017). Convergence analysis of two-layer neural networks with relu activation. *CoRR*, abs/1705.09886. (Cited on page 60.)

[Lind and Ljung, 2008] Lind, I. and Ljung, L. (2008). Regressor and structure selection in narx models using a structured anova approach. *Automatica*, 44(2):383 – 395. (Cited on page 117.)

[Liu et al., 2012] Liu, S., Yamada, M., Collier, N., and Sugiyama, M. (2012). Change-point detection in time-series data by relative density-ratio estimation. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 363–372, Berlin, Heidelberg. Springer Berlin Heidelberg. (Cited on pages 131, 135, 136, and 181.)

[Maji et al., 2013] Maji, S., Rahtu, E., Kannala, J., Blaschko, M. B., and Vedaldi, A. (2013). Fine-grained visual classification of aircraft. *CoRR*, abs/1306.5151. (Cited on pages 71 and 157.)

[Mallya and Lazebnik, 2018] Mallya, A. and Lazebnik, S. (2018). Packnet: Adding multiple tasks to a single network by iterative pruning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7765–7773. (Cited on page 94.)

[MalongTech, 2019] MalongTech (2019). Imaterialist dataset, https://github.com/malongtech/imaterialist-product-2019. (Cited on pages 88 and 167.)

[Mandt et al., 2017] Mandt, S., Hoffman, M. D., and Blei, D. M. (2017). Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907. (Cited on page 61.)

[Martens and Grosse, 2015] Martens, J. and Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. (Cited on page 166.)

[Masti and Bemporad, 2018] Masti, D. and Bemporad, A. (2018). Learning nonlinear state-space models using deep autoencoders. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 3862–3867. (Cited on pages 117 and 118.)

[Matthews and Moschytz, 1994] Matthews, M. B. and Moschytz, G. S. (1994). The identification of nonlinear discrete-time fading-memory systems using neural network models. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 41(11):740–751. (Cited on pages 107, 118, and 119.)

[Montufar et al., 2014] Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc. (Cited on pages 14 and 101.)

[Mu et al., 2020] Mu, F., Liang, Y., and Li, Y. (2020). Gradients as features for deep representation learning. In *International Conference on Learning Representations*. (Cited on pages 45, 57, 58, 60, 62, 74, 79, 83, 87, and 151.)

[Munir et al., 2019] Munir, M., Siddiqui, S. A., Dengel, A., and Ahmed, S. (2019). Deepant: A deep learning approach for unsupervised anomaly detection in time series. *IEEE Access*, 7:1991–2005. (Cited on pages 105, 129, 130, 131, 143, 145, 186, and 193.)

[Mwebaze et al., 2019] Mwebaze, E., Gebru, T., Frome, A., Nsumba, S., and Tusubira, J. (2019). icassava 2019 fine-grained visual categorization challenge. (Cited on pages 91 and 167.)

[Nguyen et al., 2020] Nguyen, C. V., Hassner, T., Archambeau, C., and Seeger, M. (2020). Leep: A new measure to evaluate transferability of learned representations. *arXiv preprint arXiv:2002.12462*. (Cited on pages 80, 88, 92, 95, 96, 97, and 169.)

[Nguyen et al., 2010] Nguyen, X., Wainwright, M. J., and Jordan, M. I. (2010). Estimating divergence functionals and the likelihood ratio by convex risk minimization. *IEEE Transactions on Information Theory*, 56(11):5847–5861. (Cited on pages 131, 135, 136, 179, 180, 181, and 182.)

[Nilsback and Zisserman, 2008] Nilsback, M. and Zisserman, A. (2008). Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*, pages 722–729. (Cited on pages 91 and 167.)

[Nitanda and Suzuki, 2020] Nitanda, A. and Suzuki, T. (2020). Optimal rates for averaged stochastic gradient descent under neural tangent kernel regime. *arXiv preprint arXiv:2006.12297*. (Cited on pages 15, 22, 23, 24, 48, 53, and 54.)

[Noh et al., 2017] Noh, H., Araujo, A., Sim, J., Weyand, T., and Han, B. (2017). Large-scale image retrieval with attentive deep local features. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3476–3485. (Cited on pages 88, 89, and 167.)

[Oreshkin et al., 2019] Oreshkin, B. N., Carpov, D., Chapados, N., and Bengio, Y. (2019). N-BEATS: neural basis expansion analysis for interpretable time series forecasting. *CoRR*, abs/1905.10437. (Cited on pages 2, 102, and 131.)

[Page, 1954] Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, 41(1/2):100–115. (Cited on page 135.)

[Parkhi et al., 2012] Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. V. (2012). Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*. (Cited on pages 91 and 167.)

[Paszke et al., 2019] 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 Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc. (Cited on page 89.)

[Pillonetto et al., 2011] Pillonetto, G., Quang, M. H., and Chiuso, A. (2011). A new kernel-based approach for nonlinearsystem identification. *IEEE Transactions on Automatic Control*, 56(12):2825–2840. (Cited on pages 107, 117, 118, 119, 120, 122, 123, and 126.)

[Quattoni and Torralba, 2009] Quattoni, A. and Torralba, A. (2009). Recognizing indoor scenes. In *CVPR*, pages 413–420. IEEE Computer Society. (Cited on pages 71 and 157.)

[Ramírez-Chavarría and Schoukens, 2021] Ramírez-Chavarría, R. G. and Schoukens, M. (2021). Nonlinear finite impulse response estimation using regularized neural networks. *IFAC-PapersOnLine*, 54(7):174–179. 19th IFAC Symposium on System Identification SYSID 2021. (Cited on pages 108 and 109.)

[Rasmussen and Williams, 2006] Rasmussen, C. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA. (Cited on pages 31, 32, 34, 35, 37, 39, 40, and 83.)

[Ravn and Uhlig, 2002] Ravn, M. and Uhlig, H. (2002). On adjusting the hodrick-prescott filter for the frequency of observations. *The Review of Economics and Statistics*, 84:371–375. (Cited on pages 132, 174, and 176.)

[Rayana and Akoglu, 2015] Rayana, S. and Akoglu, L. (2015). Less is more: Building selective anomaly ensembles with application to event detection in temporal graphs. In Venkatasubramanian, S. and Ye, J., editors, *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, BC, Canada, April 30 - May 2, 2015*, pages 622–630. SIAM. (Cited on pages 185 and 194.)

[Rebuffi et al., 2017] Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2017). Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*. (Cited on page 94.)

[Rebuffi et al., 2018] Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2018). Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8119–8127. (Cited on pages 89, 90, and 94.)

[Sandhaus, 2008] Sandhaus, E. (2008). The new york times annotated corpus ldc2008t19. web download. *Linguistic Data Consortium, Philadelphia*, 6(12):e26752. (Cited on pages 140, 144, and 185.)

[Saxe et al., 2013] Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*. (Cited on page 59.)

[Scholkopf and Smola, 2001] Scholkopf, B. and Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA. (Cited on pages 29, 31, and 36.)

[Sen et al., 2019] Sen, R., Yu, H.-F., and Dhillon, I. S. (2019). Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. 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. (Cited on pages 130, 131, 175, and 177.)

[Shawe-Taylor et al., 2005] Shawe-Taylor, J., Williams, C. K. I., Cristianini, N., and Kandola, J. (2005). On the eigenspectrum of the gram matrix and the generalization error of kernel-pca. *IEEE Transactions on Information Theory*, 51(7):2510–2522. (Cited on pages 63, 69, and 159.)

[Shen et al., 2020] Shen, L., Li, Z., and Kwok, J. (2020). Timeseries anomaly detection using temporal hierarchical one-class network. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13016–13026. Curran Associates, Inc. (Cited on pages 145 and 193.)

[Sjöberg et al., 1995] Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P.-Y., Hjalmarsson, H., and Juditsky, A. (1995). Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12):1691 – 1724. Trends in System Identification. (Cited on pages 117 and 118.)

[Smith and Le, 2018] Smith, S. L. and Le, Q. V. (2018). A bayesian perspective on generalization and stochastic gradient descent. *ArXiv*, abs/1710.06451. (Cited on pages 59, 65, and 161.)

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958. (Cited on pages 107, 118, and 121.)

[Su and Yang, 2019] Su, L. and Yang, P. (2019). On learning over-parameterized neural networks: A functional approximation perspective. *arXiv preprint arXiv:1905.10826*. (Cited on page 48.)

[Su et al., 2019] Su, Y., Zhao, Y., Niu, C., Liu, R., Sun, W., and Pei, D. (2019). Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2828–2837, New York, NY, USA. Association for Computing Machinery. (Cited on pages 104, 129, 130, 131, 145, and 193.)

[Tieleman et al., 2012] Tieleman, T., Hinton, G., et al. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31. (Cited on page 17.)

[Tikhonov and Arsenin, 1977] Tikhonov, A. N. and Arsenin, V. Y. (1977). *Solutions of Ill-posed problems*. W.H. Winston. (Cited on page 29.)

[Tran et al., 2019] Tran, A. T., Nguyen, C. V., and Hassner, T. (2019). Transferability and hardness of supervised classification tasks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1395–1405. (Cited on page 97.)

[Triantafillou et al., 2020] Triantafillou, E., Zhu, T., Dumoulin, V., Lamblin, P., Evci, U., Xu, K., Goroshin, R., Gelada, C., Swersky, K. J., Manzagol, P.-A., and Larochelle, H. (2020). Meta-dataset: A dataset of datasets for learning to learn from few examples. In *International Conference on Learning Representations (submission)*. (Cited on pages 79 and 97.)

[Tsang et al., 2018] Tsang, M., Liu, H., Purushotham, S., Murali, P., and Liu, Y. (2018). Neural interaction transparency (nit): Disentangling learned interactions for improved interpretability. 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. (Cited on page 131.)

[Ueno and Kondo, 2020] Ueno, Y. and Kondo, M. (2020). A base model selection methodology for efficient fine-tuning. (Cited on pages 80, 81, 88, 92, 95, 96, 152, and 168.)

[van der Maaten and Hinton, 2008] van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605. (Cited on page 66.)

[Van der Vaart, 2000] Van der Vaart, A. W. (2000). *Asymptotic statistics*, volume 3. Cambridge university press. (Cited on page 11.)

[Vapnik, 1998] Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley-Interscience. (Cited on pages 1, 21, and 135.)

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. 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. (Cited on pages 1, 2, 101, and 102.)

[Wahba, 1990] Wahba, G. (1990). *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics, Philadelphia. (Cited on pages 29, 31, 39, and 84.)

[Wang et al., 2020] Wang, J., Min, W., Hou, S., Ma, S., Zheng, Y., Wang, H., and Jiang, S. (2020). Logo-2k+: A large-scale logo dataset for scalable logo classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6194–6201. (Cited on pages 88 and 167.)

[Wang et al., 2012] Wang, T., Zhao, D., and Tian, S. (2012). An overview of kernel alignment and its applications. *Artificial Intelligence Review*, 43:179–192. (Cited on pages 39, 40, 41, 44, 83, 84, and 85.)

[Welinder et al., 2010] Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., and Perona, P. (2010). Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology. (Cited on pages 71, 93, 157, and 167.)

[Welling and Teh, 2011] Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. (Cited on pages 118 and 122.)

[Wolf et al., 2020] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics. (Cited on page 185.)

[Yang, 2020] Yang, G. (2020). Tensor programs ii: Neural tangent kernel for any architecture. *arXiv preprint arXiv:2006.14548*. (Cited on pages 151 and 152.)

[Yang and Newsam, 2010] Yang, Y. and Newsam, S. (2010). Bag-of-visual-words and spatial extensions for land-use classification. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, page 270–279, New York, NY, USA. Association for Computing Machinery. (Cited on pages 93, 167, 169, and 170.)

[Yashchin, 1993] Yashchin, E. (1993). Performance of cusum control schemes for serially correlated observations. *Technometrics*, 35(1):37–52. (Cited on pages 130 and 135.)

[Zamir et al., 2018] Zamir, A. R., Sax, A., Shen, W., Guibas, L. J., Malik, J., and Savarese, S. (2018). Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722. (Cited on pages 80, 96, and 97.)

[Zancato et al., 2022] Zancato, L., Achille, A., Giovanni, P., Alessandro, C., and Soatto, S. (2022). STRIC: Stacked residuals of interpretable components for time series anomaly detection. In *Submitted to The Tenth International Conference on Learning Representations*. under review. (Cited on pages 5, 101, 102, 105, 107, 110, 111, and 115.)

[Zancato et al., 2020] Zancato, L., Achille, A., Ravichandran, A., Bhotika, R., and Soatto, S. (2020). Predicting training time without training. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6136–6146. Curran Associates, Inc. (Cited on pages 2, 3, 21, 45, 49, 50, 52, 57, and 102.)

[Zancato and Chiuso, 2021] Zancato, L. and Chiuso, A. (2021). A novel deep neural network architecture for non-linear system identification. *IFAC-PapersOnLine*, 54(7):186–191. 19th IFAC Symposium on System Identification SYSID 2021. (Cited on pages 4, 101, 102, 107, 111, 115, 121, and 131.)

[Zhang et al., 2017] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. (Cited on pages 1, 2, 21, 50, and 67.)

[Zhou et al., 2017] Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., and Torralba, A. (2017). Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. (Cited on pages 88, 89, and 167.)

[Zoph et al., 2020] Zoph, B., Ghiasi, G., Lin, T.-Y., Cui, Y., Liu, H., Cubuk, E. D., and Le, Q. V. (2020). Rethinking pre-training and self-training. (Cited on page 96.)

[Zou et al., 2018] Zou, D., Cao, Y., Zhou, D., and Gu, Q. (2018). Stochastic gradient descent optimizes over-parameterized deep relu networks. *CoRR*, abs/1811.08888. (Cited on pages 57, 60, and 62.)