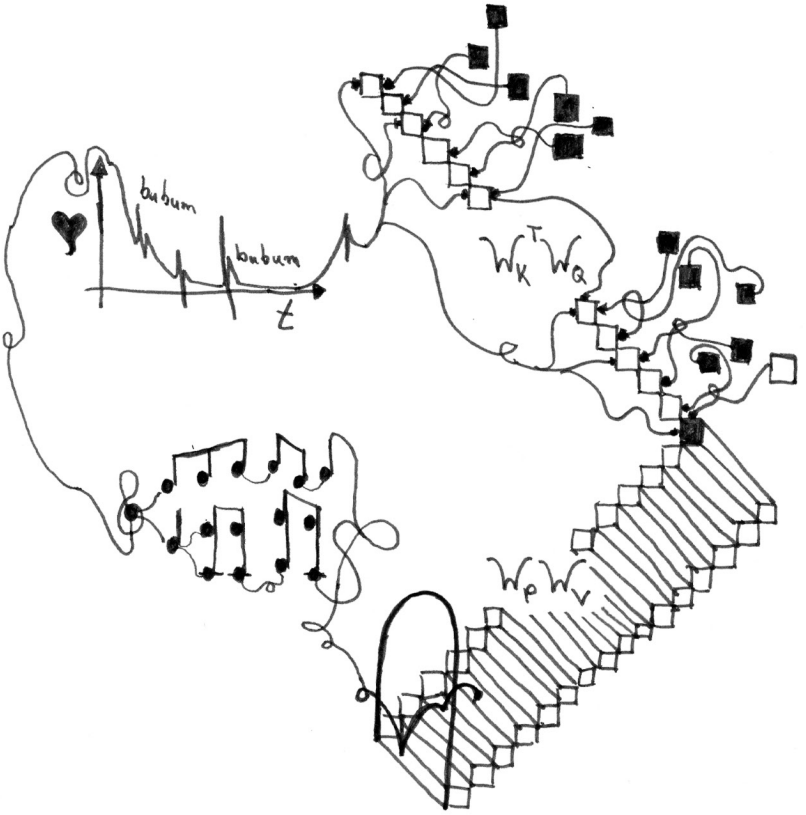
Johannes von Oswald

Interpretability of Learning Algorithms Encoded in Deep Neural Networks



Diss. ETH No. 29862

JOHANNES VON OSWALD

INTERPRETABILITY OF LEARNING ALGORITHMS ENCODED IN DEEP NEURAL NETWORKS

DISS. ETH NO. 29862

INTERPRETABILITY OF LEARNING ALGORITHMS ENCODED IN DEEP NEURAL NETWORKS

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

JOHANNES VON OSWALD
MSc., Technische Universität München

born on 28 September 1990

accepted on the recommendation of

Prof. Dr. Angelika Steger, examiner
Dr. João Sacramento, co-examiner
Dr. Razvan Pascanu, co-examiner
Prof. Dr. Guillaume Lajoie, co-examiner

2024

DOI: 10.3929/ethz-b-000660213

This thesis is dedicated to my parents
Anne & Moritz

ABSTRACT

We are currently experiencing a revolution in artificial intelligence (AI). Considerable improvements in generative AI and particular large language models (LLMs) are driving this revolution. Yet powerful, these large-scale AI systems are more difficult to understand. Once trained, their inner workings remain a fascinating and potentially frightening mystery. The question remains of how we, the creators of these systems, can understand and control them and what drives their behavior.

In this thesis, I will present an attempt to understand certain characteristics of deep neural networks with the tools of *mechanistic interpretability* (MI). These tools are reminiscent of those used by a neuroscientist: 1) analyzing the connectivity of brain cells (connectomics) 2) measuring and analyzing neuronal activity and 3) measuring effects of active interventions in ongoing computations. Although a rigorous understanding of large deep learning models is out of reach, in this thesis, I will provide evidence of a possible path towards this goal by *iterative interpretability*: an iterative process of designing, training, and analyzing AI systems in which insights obtained by MI lead to more capable and interpretable models.

First, I provide evidence that it is possible to gain an understanding of the intriguing in-context learning characteristics of Transformers, the artificial neural network architecture used in LLMs when studied in isolation. We analyze, as a first step, the weights of small Transformer models trained on few-shot regression data. By using the tools of MI, we can reverse engineer the trained Transformers, equipped with linear self-attention layers, and show that they implicitly learn internal models in their forward dynamics based on gradient descent and the data given in context.

Second, I will address one important shortcoming of this simplistic setting and move closer to LLMs by training autoregressive Transformers. Here, we train models to predict the next element in a sequence of elements obtained by linear dynamics instead of few-shot data. Again, with the tools of a neuroscientist, we can reverse engineer these autoregressive models and identify the model internally i) constructing optimization problems and ii) solving them by gradient descent-based algorithms. This algorithm hidden inside the weights of the model allows us to repurpose the model as an in-context learner post-training. Based on these insights, we then close the interpretability cycle and propose a novel self-attention layer that can

solve the identified optimization problem within a single layer by design. While offering better interpretability, we show improved performance in our simplistic experiments as well as in language modeling.

Third, I will present another example of iterative interpretability in the context of meta-learning and continual learning where we improve the performance and interpretability of the prominent *model-agnostic meta learning* (MAML) [1]. The goal of MAML is to learn a network initialization from which the network can quickly adapt to new tasks. Based on prior insights obtained through mechanistic interpretability, we propose sparse-MAML, a MAML variant that additionally decides to actively stop learning particular weights: it learns where to learn. Despite performance improvements in common few-shot classification and continual learning benchmarks, sparse-MAML provides another example of a successful interpretability cycle as the learned solution allows for better interpretability by design.

ZUSAMMENFASSUNG

Wir erleben derzeit eine Revolution der künstlichen Intelligenz (KI). Erhebliche Verbesserungen bei der generativen KI und insbesondere bei großen Sprachmodellen (LLMs) treiben diese Revolution voran. Diese leistungsstarken, groß angelegten KI-Systeme sind jedoch schwierig zu verstehen. Einmal trainiert, bleibt ihr Innenleben ein faszinierendes und potenziell beängstigendes Geheimnis. Es bleibt die Frage, wie wir, die Schöpfer dieser Systeme, sie verstehen und kontrollieren können und was ihr Verhalten steuert.

In dieser Arbeit werde ich versuchen, bestimmte Eigenschaften von tiefen neuronalen Netzen mit den Werkzeugen der mechanistischen Interpretierbarkeit (MI) zu verstehen. Diese Werkzeuge erinnern an diejenigen, die ein Neurowissenschaftler verwendet: 1) Analyse der Konnektivität von Gehirnzellen (Connectomics), 2) Messung und Analyse der neuronalen Aktivität und 3) Messung der Auswirkungen aktiver Eingriffe in laufende Berechnungen. Obwohl ein rigoroses Verständnis großer Deep-Learning-Modelle noch in weiter Ferne liegt, werde ich in dieser Arbeit einen möglichen Weg zu diesem Ziel aufzeigen: einen iterativen Prozess der Entwicklung, des Trainings und der Analyse von KI-Systemen, bei dem die durch MI gewonnenen Erkenntnisse zu leistungsfähigeren und besser interpretierbaren Modellen führen.

Zunächst zeige ich, dass es möglich ist, ein Verständnis für die faszinierenden kontextbezogenen Lerneigenschaften von Transformers zu erlangen, der künstlichen neuronalen Netzwerkarchitektur, die in LLMs verwendet wird, wenn sie isoliert untersucht wird. In einem ersten Schritt analysieren wir die Gewichte kleiner Transformer-Modelle, die auf Regressionsdaten mit wenigen Schüssen trainiert wurden. Mit Hilfe der MI-Tools können wir die trainierten Transformers, die mit linearen self-attention-Schichten ausgestattet sind, zurückentwickeln und zeigen, dass sie implizit interne Modelle in ihrer Vorwärtsdynamik auf der Grundlage von Gradientenabstieg und den im Kontext gegebenen Daten lernen.

Zweitens werde ich ein wichtiges Defizit dieser vereinfachten Einstellung beheben und mich LLMs annähern, indem ich autoregressive Transformer trainiere. Hier trainieren wir Modelle zur Vorhersage des nächsten Elements in einer Sequenz von Elementen, die durch ein lineares dynamisches System gewonnen werden. Auch hier können wir mit den Werkzeugen eines Neu-

rowissenschaftlers diese autoregressiven Modelle zurückentwickeln und interne Modelle identifizieren, die i) Optimierungsprobleme konstruieren und ii) diese durch Algorithmen auf der Grundlage des Gradientenabstiegs lösen. Dieser Algorithmus, der in den Gewichten des Modells versteckt ist, ermöglicht es uns, das Modell nach dem Training als kontextabhängigen Lerner neu zu verwenden. Auf der Grundlage dieser Erkenntnisse schließen wir dann den Interpretierbarkeitszyklus und schlagen eine neuartige self-attention-Schicht vor, die das identifizierte Optimierungsproblem innerhalb einer einzigen Schicht lösen kann. Während wir eine bessere Interpretierbarkeit bieten, zeigen wir auch verbesserte Leistung in unseren vereinfachten Experimenten sowie in der Sprachmodellierung.

Drittens werde ich ein weiteres Beispiel für iterative Interpretierbarkeit im Kontext von Meta-Lernen und kontinuierlichem Lernen vorstellen, bei dem wir die Leistung und Interpretierbarkeit des prominenten *model-agnostic meta learning* (MAML) [1] verbessern. Das Ziel von MAML ist es, eine Netzwerkinitialisierung zu erlernen, von der aus sich das Netzwerk schnell an neue Aufgaben anpassen kann. Basierend auf früheren Erkenntnissen, die durch mechanistische Interpretierbarkeit gewonnen wurden, schlagen wir sparse-MAML vor, eine MAML-Variante, die zusätzlich entscheidet, aktiv mit dem Lernen bestimmter Gewichte aufzuhören: Sie lernt, wo sie lernen soll. Trotz der Leistungsverbesserungen in den üblichen Benchmarks zur Klassifizierung mit wenigen Schüssen und zum kontinuierlichen Lernen bietet sparse-MAML ein weiteres Beispiel für einen erfolgreichen Interpretierbarkeitszyklus, da die gelernte Lösung eine bessere Interpretierbarkeit ermöglicht.

ACKNOWLEDGEMENTS

This thesis is a collection of articles that I co-authored with amazing colleagues and friends. I am honored to have worked with and alongside you over the course of my PhD. I am forever grateful and proud of the dedication and love we all put into the scientific idea and the articles that make up this thesis. None of this would have been possible without you and I feel blessed to have witnessed your incredible excitement for science and research of this thing called intelligence with me every day!

Out of this large group of companions there are some I need to single out:

Christian - You are an incredible role model for me and remind me what it means to be kind, compassionate and a true friend.

Seijin - Thank you for everything, my friend! I am especially humbled and blessed to have been able to work with you throughout my PhD.

Marc - Thank you for being this extraordinary person and friend that forces me to always think, reflect and consider more.

João - Your excitement about science and life is breathtaking. I thank you for everything and much more. This is just the beginning!

Angelika - You hopefully know what you are to us: A true leader and role model, an admirable scientist full of integrity, belief and conviction to do the right thing. The day you stood up for us is marked in my calendar forever. I thank you from the bottom of my heart.

I also want to thank Eyvind Niklasson and Zeke Turner for valuable feedback on this manuscript.

Finally, I want to share my deepest gratitude with regards to my parents, my sisters, my family, all my friends and of course my Géraldine. This thesis is a dedication to you all - the one love you are in my heart.

CONTENTS

1

INTRODUCTION

In this thesis I want to advocate for the use of the tools and insights of an emerging research field termed *mechanistic interpretability* (MI). With them I will study artificial neural networks and in particular their learning mechanisms. To contextualize this thesis, in this introduction, I will attempt to describe what mechanistic interpretability is and highlight its parallels and major advantages concerning neuroscience. Based on these, I will build on a framework I term *iterative interpretability*, a procedure that turns insights derived from interpretability work into designing models with improved interpretability by design. This design methodology could lead to AI down the road which we can understand and control.

SETTING THE STAGE The overarching goal of mechanistic interpretability and the research presented in this thesis can be described as the dream of *understanding* highly complex and capable artificial neural networks — which are now undoubtedly changing our world as we know it. But what does understanding in this context even mean? To try to formulate an answer for this question it is important to realize both the inherent difficulty but also our familiarity with the inquiry of understanding. Indeed, one could argue that the scientific field as a whole is in pursuit of understanding. In particular, most disciplines of quantitative sciences follow a path of *modeling* with which, given some observations, one can predict the unknown. Therefore it can be argued that the development of interpretable scientific models, and the insights that come with them, increase our understanding of the subject of study.

Nevertheless, even if successful, this form of understanding, while constantly improving due to better data and tools as well as more clever (but potentially less comprehensible) mathematics, might be fundamentally limited. Given the limited compute of our brains and their highly specialized intelligence, we simply might not be able to formulate the right questions let alone answer them. Even if allowed access to external compute which we control and use in pursuit of these goals. This line of thinking awakens visualizations of a trace left behind by a hopeless rabbit tottering into a dark unpleasant hole — a path I choose not to follow.

I believe it is nevertheless of importance to note that even in the simplistic problem settings studied in this thesis, my co-authors and I were quickly confronted with the difficult question: What is it that we can study to increase our *understanding* of the system at hand? It often seemed difficult to even formulate a hypothesis that could lead, if verified, to an improved understanding of the behavior of the complex neural networks at hand — which are soon approaching trillions of parameters.

Despite this immense difficulty, one of the sources of hope that drove me to do this research is finding my place within a large and rich research community. In it, a small group of fascinating scientists laid the groundwork and built tools with that we can hope to understand artificial neural networks and AI. This kind of research has been termed *mechanistic interpretability* which for their scientists can be described as the study of artificial neural networks as, for example, neuroscience is the study of the biological nervous system and the brain. My thesis is positioned within this young research field and will build upon its parallels and, crucially as we will see below, its advantages compared to neuroscience.

To end these introductory comments, I want to provide an in-complete, optimistic, and hope-filled list of insights, very much inspired from [2], motivating my pursuit of this field, and the field as a whole:

- Given the very scarce current understanding of AI as well as the immaturity of the definitions, tools, and results in the field of MI, it seems reasonable to assume that many low-hanging fruits await researchers entering this field.

- Even if ultimately unsuccessful in the quest for the currently unreachable goal of fully understanding a highly complex system like large neural networks, MI might offer explanations of smaller, less complex phenomena along the way. These might have of immense impact on their own and lead to, for example, improved trust towards an AI system.

- Equipped with these results of improved understanding, it seems possible to steer current deep learning research into building more capable models with crucially baked-in interpretability. Therefore, even if current systems might be impenetrable black boxes, future systems might not be. This motivates iterative interpretability as one path towards understanding neural networks proposed in this thesis.

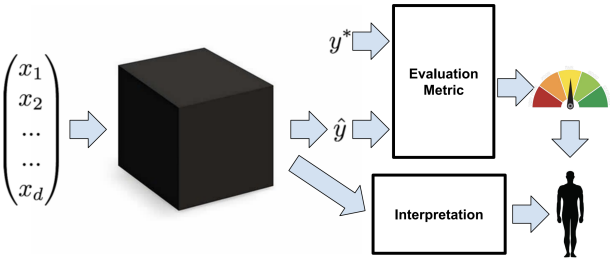


FIGURE 1.1: Despite the main desire of good performance of a black-box machine learning model, mechanistic interpretability aims to provide additional useful information to the user post training. Figure taken from [3].

The research that makes up this thesis presents examples covering all these three points and beyond. Therefore, I am filled with excitement to pursue this research direction further in the coming years fueled by the eagerness to discover and fill the immense gaps in our understanding of AI.

MECHANISTIC INTERPRETABILITY VS NEUROSCIENCE At the core of mechanistic interpretability is the desire to understand machine learning models. As machine learning consists of a huge variety of subfields with their respective tools and goals which again all use vastly different kinds of models, including different types of neural network architectures, it might not come as a surprise that the term *mechanistic interpretability* means different things to different people. It lacks a clear definition [3]. The field can be better described as a loose term that entails a collection of many different ideas and goals and different techniques at their disposal. Nevertheless, generally, MI refers to the task of obtaining additional useful information concerning a black-box machine learning model, see Illustration taken from [3] in Figure 1.1.

Deep neural networks are black-box machine learning models usually obtained by optimization on huge amounts of data. Once optimized the neural network model and its parameters describe a function that compresses the input and output relationship of the data. Therefore interpretability work encompasses all details of the training pipeline but is mainly influenced by the training data. Nevertheless, MI usually only starts after the model is assembled i.e. in a top-down or post hoc fashion. This poses huge difficulty

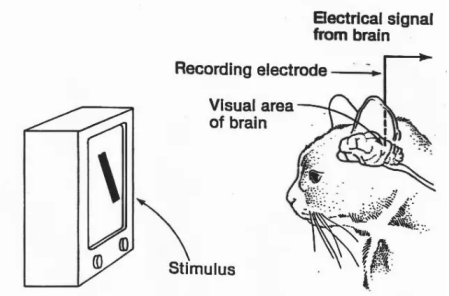


FIGURE 1.2: Illustration of the experimental setup of Hubel & Wiesel in 1954 leading to the discovery of single neurons responsive of distinct visual features. Figure taken from [5].

to the interpretability work as the final complex system, the subject of study, is a consequence of again itself a highly complex process. Therefore, causal relationships between the final optimized model with e.g. training data, the network architecture and other important factors such as the training algorithm a hard very to infer [4].

In contrast, I stress here the difference to a hand-designed model for which interpretability, arguably, is obtained bottom-up. As human design and analyses, in the form of mathematical statements and proofs, were necessary to design the model in the first place it required understanding beforehand. Importantly, the design usually is not at all, only sparsely or implicitly influenced by data. Therefore the model is not influenced by the complicated, mathematically difficult to describe dependency of noisy real-world data.

In this thesis, I will nevertheless show some first examples in which MI was the major tool to obtain these rigorous mathematical statements about models in a top-down fashion i.e. about models optimized on data. But what should these mathematical statements address? I list here a few high-level desiderata, summarized from [3], which motivate MI and research, in general, that aims at understanding AI models.

- *Trust*: A user, despite obtaining a (possibly correct) prediction from a model, should be able to *trust* it. This is related to the difficult problem of calibration of the predictions as well as the possibility of a model doing out-of-distribution detection [6]. Nevertheless, even if these problems were solved it feels hard that users of an AI would be able to trust a machine learning model to make e.g. life-critical decisions

without providing additional information. Although unclear, a possibly useful information spectrum could range from understanding the inner workings of a model to less mechanistic information such as additional self-generated explanations related to the basis of predictions. This again is an imprecise objective towards understanding but I believe close to the heart of everyone who uses or will use AI in the future.

- *Causality*: Related to understanding how a system forms its predictions is the urge to understand how for example the training data as well as features of the input causally relates to the model prediction. This quest seems particularly difficult as it is widely known how difficult the inference of causal relationships from observational data alone is [7].

- *Fair and Ethical Decision-Making*: As machine learning models are already used as a new interface between computers and humans through chatbots, used in critical decision-making processes and becoming more and more autonomous, there has been a huge interest in the *controllability* of these models. This is deeply connected, especially in the context of large language models (LLMs), to the desire of their *alignment* with certain values and understand biases which models might posses [8]. Interpretability could be one tool to understand and control this important and desirable AI feature [9].

Having listed some meaningful desiderata, I want to give a short overview of the applicable MI tools useful for this thesis. These all have direct or indirect relations to techniques that neuroscientists especially computational neuroscientists use to study biological neural networks (BNNs) since decades [10, 11]. However, we will see that there are major advantages when it comes to studying artificial neural networks [12]. These advantages are fundamental and at the heart of *iterative interpretability* which I will introduce below. Note that the techniques and tools of neuroscience might not be sufficient to understand BNNs [13, 14].

Neural recordings: One of the most common techniques to study the living brain of animals and humans is to measure and record neural activity, often in response to given stimuli. Driven by the desire to understand how these stimuli are processed and e.g. form memories or drive behavior, neuroscientists have developed numerous techniques allowing for neural recordings from single or large groups of neurons throughout the brain and

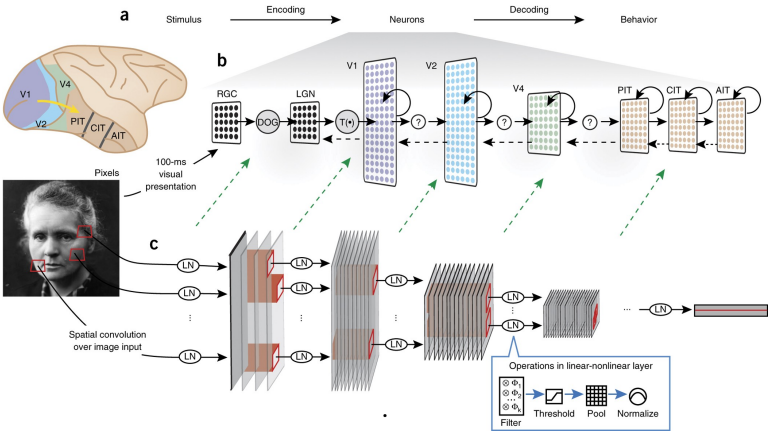


FIGURE 1.3: Striking similarities of the activation strength of neurons within optimized convolutional neural networks and the spiking probability of neurons within the visual pathway. This is indicative of the computational similarity (or optimality) of these two vastly different systems when processing visual information. Figure taken from [15]

nervous system. In a landmark result, Hubel & Wiesel in 1954 were able to show that neurons in the cat's visual cortex robustly responded when presented with simple moving black bars of specific angles on a screen [16], see Figure 1.2. These studies of correlating stimuli with neural activity have been vastly improved, see for example [17] for an intriguing study of the navigation system of the Drosophila, and is still a leading paradigm to study the brain and how it drives behavior. These studies can therefore shed light, for example, on where in the brain certain stimuli activate neurons and furthermore how activity changes during learning.

I want to highlight the importance of these findings to machine learning. Numerous studies in neuroscience were able to show that neurons along the visual pathway, ranging from the retina deep into our brain, can be divided into various brain areas with firing rates that correlate with different, more complex features along its information stream. For example, giving rise to the famous (debated) existence of celebrity neurons [18]. In a highly celebrated article by Yamins & DiCarlo [15] were able to show that striking similarities between these features in brains and optimized artificial neural networks can be made, visualized in Figure 1.3. The analogy of artificial neural networks with their biological ancestors seems therefore not only inspirational but also functional as their activity patterns show similarities.

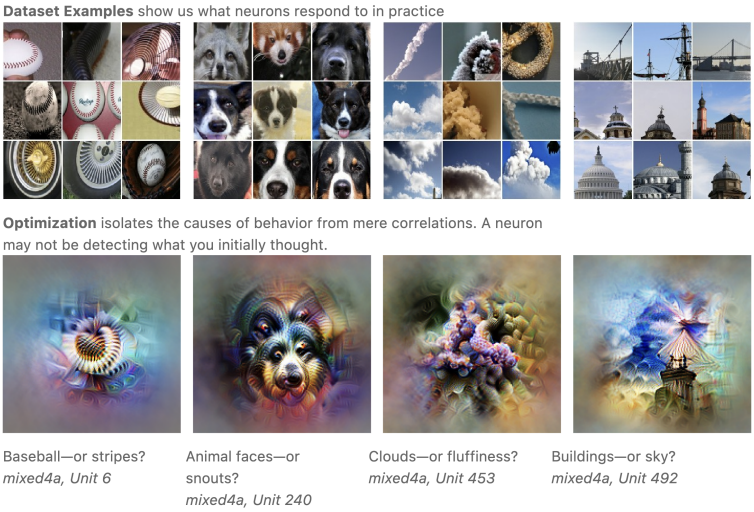


FIGURE 1.4: Feature visualization of neurons inside trained convolutional neural networks, Figure taken from [19]. *Top row*: Real word images that lead to strong activations of certain neurons inside the network. *Bottom row*: Images obtained by optimization that again lead to strong firing of the same neurons.

When moving to mechanistic interpretability, correlating the input stimuli with neural activity is again a common tool for understanding the information processing in artificial neural networks. A prominent example of this is the studies led by Christopher Olah [19–21], a notable figure in interpretability research. Their work was able to identify families of neuron activations, as in biological networks, strongly correlated with certain stimuli. Furthermore, in a somewhat reversed fashion, the authors then constructed images by optimization that target the same neurons to fire particularly strongly. This led to images, by visual inspection, that can be interpreted once more as detectors of these highly complex features related to the ones found in the corresponding real-world images, see Figure 1.4.

Connectomics & Circuits: The study of how neurons are connected, and form distinct brain areas and circuits is the study of connectomics. This neuroscience field has made incredible advances both technologically and on the data science side, also fuelled by the advances of machine learning. Around the world, large-scale efforts towards obtaining precise brain atlases of animals giving exact information of where neuron bodies and their

dendrites are positioned. It is even possible to image their connections and their strength with other neurons by synapses. Although these connectomes are obtained after the death of the animals and can therefore only give limited answers about their role functionally [22], brain connectomics can still provide insights into how information is processed inside the animal's highly complex neural systems. See an example of one of these fascinating maps showing precisely the wiring of a Drosophila brain in Figure 1.5.

Although the inner workings of single cells and even single synapses are highly complex and of large interest to study in neuroscience, usually neurons in artificial networks are simplistic, and their mechanistic study in isolation is of less importance. I focus therefore on the more relevant study of their connections i.e. through the wiring defined by the strength of the corresponding optimized neural network weights. Here, two interconnected directions of study are of importance:

First, it can be shown, within both biological and artificial networks, that features of early processing layers, are combined in later layers leading to more complex feature detectors. This, for example, can be the evolution of features that correspond first to simple Gabor filters in early layers to neurons that activate in response to highly complex features such as *pose-invariant dog heads* in later layers [19, 23]. These feature combinations form circuits of high complexity, including the branching into subnetworks as well as the cancellation and competition between features influencing the activation in later layers and finally the network's prediction [24, 25]. Note that the existence and combination of features and their circuits in artificial neural networks is solely based on the optimization of the network for a given task, for example, image classification leading to features visualized in Figure 1.4, which changes the weights of the network by gradient descent into a state of low loss.

Second, it is exactly the study of these weights, the program of the network, in isolation that can provide interpretability. Here, the structure and strength of the weights govern e.g. the combination of features, branching, and the formation of larger-scale circuits. The study of weights in isolation can lead to a surprising and insightful understanding of the network as well [26]. We will see below that this will be of particular interest to this thesis and one of our most useful tools for interpretability.

Interventions: Another interesting neuroscience technique I want to motivate for MI are interventions into ongoing computation. The techniques developed here, in mechanistic interpretability as well as in neuroscience,

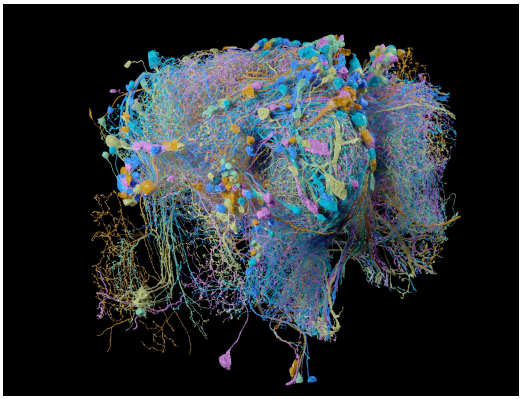


FIGURE 1.5: Part of a Drosophilia brain's connectome, Image from [27].

all build on targeted disturbance of computation in artificial networks as well as biological brains. In the latter case, neuroscientists have developed highly creative and technically elaborate ways to influence the computation in the brain by lesions or by electrical, chemical, and optical interventions. This can lead to controlling the subject's neural wiring as well as activity and therefore influence the behavior of living animals [28, 29]. The major advantage of this paradigm is to establish a causal link between behavior and neural activity instead of relying on a correlation between these. In mechanistic interpretability, these ideas find usability by again establishing a causal relationship of computation between inputs, intermediate neurons, and outputs. This can also include the study of causally linking between the activations and prediction with the network weights.

To end this introduction to MI, I summarise notes of Christopher Olah [12] and provide a list of major advantages of mechanistic interpretability when compared to neuroscience with which I then will motivate *iterative interpretability* below.

- *Accessibility & Causality*: Compared to the immense efforts one has to put into attaining neural activity or carrying out intervention in biological neural networks, the activities, as well as all the weights of neural networks, are, if open sourced, accessible and available to analyze, correlate and perturb causally without major difficulty. On a negative note, this can be, as already mentioned, a burden due to the immense amount of data available.

- *Uniqueness & Reproducibility*: The study of biological neural networks suffers from reproducibility issues with numerous causes [30]. One apparent advantage when studying artificial neural networks is that the entire system can be transferred without problems offering the study of a single network through many different independent scientists. In principle, the reproducibility of any results in MI and deep learning should be more simplistic and reliable compared to other lab work.

- *Active interventions*: Research of biological neural networks assumes that the subject of study e.g. an animal is given, although it can be altered with tremendous efforts, see above. Artificial neural networks, on the other hand, are crucially objects of our own creation and we enjoy full freedom of their design and training objectives that have seen dramatic changes since the time they were invented and studied. Therefore, we are given full flexibility to design the subject of study as we please, provided it is useful. It might therefore be that although current models are not interpretable, we are able to design capable models that allow for better interpretability bottom-up.

ITERATIVE INTERPRETABILITY IN THE CONTEXT OF LEARNING ALGORITHMS In this thesis, I strive to understand deep neural networks and their algorithms within a common framework described in [31] that I term iterative interpretability. It is an iterative process cycling over the following three steps with the overarching goal of answering a specific hypotheses H:

1. Training of a deep neural network: Given the choices about the specific neural network model, the dataset, the training algorithm, and an optimization objective among other things, the result of this step is an optimized neural network model.

2. Mechanistic interpretability to study H: Based on the previous decisions, obtain evidence for or against H with the tools of MI through studying for example correlations, causation, or the connectomics of the neural network. These findings can be of considerable value in isolation.

3. Design novel or alter parts of the deep learning pipeline: Given insights in support of H obtained previously, this step aims to alter parts of the training pipeline to obtain 1) a similar or more capable model while 2) allowing for better interpretability bottom-up supporting H.

Various aspects of this pipeline are of course steps of scientific inquiry generally. Nevertheless, I think it is of value to restate them here in the context of AI research and specifically mechanistic interpretability. Other aspects of this pipeline, especially the third step are somewhat unusual for the empirical sciences and more closely related to engineering. This highlights an interesting tension of iterative interpretability between the science of understanding a complex system like deep neural networks through mechanistic interpretability and, on the other hand, the engineering of the subject of study in the first place: While the primary goal of the latter is the development of more capable and therefore arguably more complex models, the former aims towards understanding exactly these models. I, therefore, stress that successful iterative interpretability should and often can fulfill both seemingly orthogonal goals [31, 32]. Here, two arguments come to mind:

First, the current AI and economic climate aims purely towards improved capabilities that can be measured on benchmarks. Hence, a path toward improved interpretability might be pursued more readily if it does not entail sacrificing model performance. Furthermore, it might be useful for a benchmark-driven community to accept interpretability as a valuable benchmark that we can work towards. I thus advocate for the establishment of benchmarks for interpretability that are currently missing.

Second, I believe that it is an explicit empirical verification of understanding if active interventions in the model design when based on MI, lead to performance improvements. I draw here parallels to designing models, entirely by hand, that describe data better. Nevertheless, I stress again the pressure to condition interpretability work and iterative interpretability on model performance. This is necessary and prohibits fallacies since the analyzed data and its complexity obtained by MI is directly dependent on the training pipeline and hence under the control of the scientist as well. These arguments are certainly up for debate.

In this thesis, I will showcase examples of successful iterative interpretability focusing on specific hypotheses H related to the ability of neural networks to quickly adapt to novel and small amounts of data. Deep neural networks that learn-to-learn on different time scales are the subject of study in AI and neural networks research at least since the late 1980s [33, 34] and have seen substantial interest in the last deep learning decade [35]. Nevertheless, fast learning has now regained major interest as it is been celebrated as arguably the most powerful characteristic of large language

models [36]. Although diverse use cases exist such as zero- or few-shot learning and chain-of-thought prompting [37], all can be described as a way to implicitly reprogram an LLM, and therefore alter its behavior, without changing or access its weights. Intriguingly, this reprogramming is achieved only with instructions by text and happens post-training, indicating that LLMs generalize to these behaviors. These capabilities are currently among if not the most celebrated features of LLMs and AI today.

This feature of LLMs, best known as in-context learning (ICL), is the central subject of study in the next chapter. In it, inspired by [38], in-context learning abilities of the Transformer architecture, the most common neural network architecture currently used for natural language processing, is studied in isolation. A Transformer model $t_\theta(x_1, y_1, \ldots, x_N, y_N, x_{\text{test}})$, provided a supervised dataset and an additional test point as input, is trained to predict given the corresponding target y_{test}. For every sequence, we sample a new teacher W with which we build the dataset $y_i = Wx_i$ presented as the sequence to the model. Simply studying the weights of the Transformer, trained on this linear regression data, shows surprising sparsity and offers precise interpretability, at least in some restricted cases. In particular, when studying a single trained layer of linear self-attention, it can be shown that by construction and in practice, the layer implements a single step of gradient descent with an optimal learning rate on the squared error regression loss. Key to these findings is actually a slightly different *tokenization* of the data, see Chapter 2.

These results have been the subject of considerable interest in the research community, reproduced numerous times, and studied in mathematical rigor in multiple independent follow-up works [39–42]. These studies show among other things that the solution found by optimization, i.e. a single step of gradient descent, corresponds to the global optimum of the underlying optimization problem explaining our findings.

When moving to deep Transformers, again by studying its weights, we can identify that a single layer of self-attention implements a step of GD while simultaneously iteratively pre-conditioning the input data. This simple, possibly up-until-that-point unknown, gradient descent-based algorithm that we term GD++ outperforms plain gradient descent considerably. Although we can not perfectly relate the trained Transformer to GD++ by studying its weights, we fall back to studying the correlations of neuron activations. Here, we analyze differences between the trained Transformer and a Transformer implementing GD++ inside its weights show strong similari-

ties. We also provide evidence that our findings translate to Transformers trained to learn non-linear regression in their forward dynamics.

Based on these results and related concurrent work, numerous follow-up works have extended our findings and, for example, studied in similar vein chain-of-thought prompting [43], multi-task in-context learning [44] or even the implicit training of small Transformers in large Transformers [45].

Although all of these studies shed light on in-context learning abilities and their relations to gradient descent from bottom-up i.e. by studying toy problems in rather small models, there are major gaps towards these findings and the original question of how in-context learning comes about and it is implemented in LLMs. One of these shortcomings we address in Chapter 3 where we move from training Transformer on in-context few-shot data to an autoregressive training setting. Now, the Transformers $t_\theta(s_1, \ldots, s_t)$ are trained to predict the next token s_{t+1} given a sequence of elements obtained from a linear dynamical system $s_{t+1} = Ws_t$. We find that a similar 2-step algorithm as in the in-context learning setting is identifiable, by means of neural activity correlation, weight, and circuit inspection as well as causal interventions in the autoregressive models. This enables us to understand, in this very simplistic model, post-training in-context learning in autoregressive models — as the (approximate) least-squares solver that is implemented inside the model to predict the next token can easily be *repurposed* for ICL.

I want to stress that in these two chapters, we are able to, in certain simplistic settings, reverse engineer identifiable *algorithms*, parametrized with less than 99% parameters of the original Transformer model. These findings show that Transformers, with linear self-attention, trained on clean algorithmic data implement algorithms that we can fully interpret. Once extracted from the trained models, the algorithms can be analyzed and characterized with mathematical rigor. Although this very desirable mechanistic interpretability precision can not be expected when training models on more noisy data [13, 46], our work is an example of MI that offers a particularly precise mathematical understanding of the inner workings of the trained neural networks.

Given these insights, we furthermore propose in Chapter 3 a new attention layer that we term *mesa-layer* that solves the underlying recursive least-squares error optimization problem within a single layer. Equipped with the mesa-layer, Transformers show improved performance in our simplistic problems as well as language modeling. We therefore show a

compelling example of iterative interpretability where mechanistic interpretability results led to insights that allowed designing more powerful and interpretable neural network models from the bottom-up.

In Chapter 4, we provide another example of iterative interpretability. This work diverges from our studies of Transformers and the study of optimization algorithms implemented inside these models. Here we build on prior mechanistic interpretability work of the well-known few-shot meta-learning algorithm termed *model-agnostic meta-learning* (MAML) [1]. MAML aims towards learning a deep neural network weight initialization, with *backpropagation through training*, which can be quickly fine-tuned towards a new task without overfitting. Prior work [47] showed that MAML tends to find initializations of the network weights that, except the last layer, can be kept fixed during fine-tuning without significant performance loss. Inspired by this prior interpretability work, we, therefore, propose a MAML variant, which we term sparse-MAML, where not only the initial weights but also a binary 0-1 mask applied to the gradients used for fine-tuning is meta-learned. Therefore, we allow MAML to explicitly decide which parameters to adapt in the fine-tuning phase and allow the algorithm to choose actively where to stop learning parameters [47]. Despite performance gains in few-shot as well as few-shot continual learning tasks, we see that sparse-MAML learns to drastically sparsify gradients as indicated by the prior interpretability work. We therefore propose again a more performant and crucially interpretable deep learning algorithm based on the tools of mechanistic interpretability.

2

TRANSFORMERS LEARN IN-CONTEXT BY GRADIENT DESCENT

This chapter's content was published and awarded an oral presentation at the Proceedings of the 40th International Conference on Machine Learning and can be found online in an extended form including all experimental details which we omit here for clarity. The original publication is authored by Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, Max Vladymyrov [48].

At present, the mechanisms of in-context learning in Transformers are not well understood and remain mostly an intuition. In this chapter, we suggest that training Transformers on auto-regressive objectives is closely related to gradient-based meta-learning formulations. We start by providing a simple weight construction that shows the equivalence of data transformations induced by 1) a single linear self-attention layer and by 2) gradient-descent (GD) on a regression loss. Motivated by that construction, we show empirically that when training self-attention-only Transformers on simple regression tasks either the models learned by GD and Transformers show great similarity or, remarkably, the weights found by optimization match the construction. Thus we show how trained Transformers become mesa-optimizers i.e. learn models by gradient descent in their forward pass. This allows us, at least in the domain of regression problems, to mechanistically understand the inner workings of in-context learning in optimized Transformers. Building on this insight, we furthermore identify how Transformers surpass the performance of plain gradient descent by learning an iterative curvature correction and learn linear models on deep data representations to solve non-linear regression tasks. Finally, we discuss intriguing parallels to a mechanism identified to be crucial for in-context learning termed *induction-head* [49] and show how it could be understood as a specific case of in-context learning by gradient descent learning within Transformers.

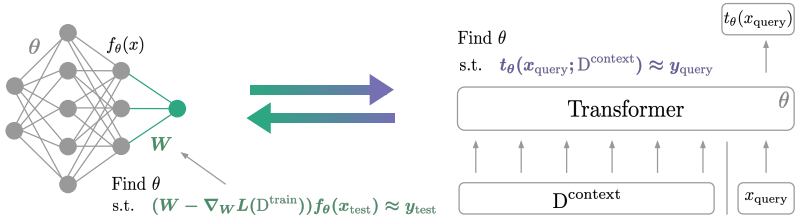


FIGURE 2.1: **Illustration of our hypothesis: gradient-based optimization and attention-based in-context learning are equivalent.** *Left*: Learning a neural network output layer by gradient descent on a dataset $\mathrm{D}^{\text{train}}$. The task-shared meta-parameters θ are obtained by meta-learning with the goal that after adjusting the neural network output layer, the model generalizes well on unseen data. *Right*: Illustration of a Transformer that adjusts its query prediction on the data given in-context i.e. $t_\theta(x_{\text{query}}; \mathrm{D}^{\text{context}})$. The weights of the Transformer are optimized to predict the next token y_{query}.

2.1 INTRODUCTION

In recent years Transformers [TFs; 50] have demonstrated their superiority in numerous benchmarks and various fields of modern machine learning, and have emerged as the *de-facto* neural network architecture used for modern AI [51–54]. It has been hypothesised that their success is due in part to a phenomenon called *in-context learning* [36, 55]: an ability to flexibly adjust their prediction based on additional data given *in context* (i.e. in the input sequence itself). In-context learning offers a seemingly different approach to few-shot and meta-learning [36], but as of today the exact mechanisms of how it works are not fully understood. It is thus of great interest to understand what makes Transformers pay attention to their context, what the mechanisms are, and under which circumstances, they come into play [49, 56].

In this chapter, we aim to bridge the gap between in-context and meta-learning, and show that in-context learning in Transformers can be an emergent property approximating gradient-based few-shot learning within its forward pass, see Figure 2.1. For this to be realized, we show how Transformers **(1)** construct a loss function dependent on the data given in sequence and **(2)** learn based on gradients of that loss. We will first focus on the latter, the more elaborate learning task, in sections 2.2 and 2.3, after which we provide evidence for the former in section 2.4.

We summarize our contributions as follows[1]:

- We construct explicit weights for a linear self-attention layer that induces an update identical to a single step of gradient descent (GD) on a mean squared error loss. Additionally, we show how several self-attention layers can iteratively perform curvature correction improving on plain gradient descent.
- When optimized on linear regression datasets, we demonstrate that linear self-attention-only Transformers either converge to our weight construction and therefore implement gradient descent, or generate linear models that closely align with models trained by GD, both in in- and out-of-distribution validation tasks.
- We resolve the dependency on the specific token construction by providing evidence that learned Transformers first encode incoming tokens into a format amenable to the in-context gradient descent learning that occurs in the later layers of the Transformer.

These findings allow us to connect *learning* Transformer weights and the concept of *meta-learning* a learning algorithm [34, 57–65]. In this extensive research field, meta-learning is typically regarded as learning that takes place on various time scales namely fast and slow. The slowly changing parameters control and prepare for fast adaptation reacting to sudden changes in the incoming data by e.g. a context switch. Notably, we build heavily on the concept of fast weights [34] which has shown to be equivalent to linear self-attention [66] and show how optimized Transformers implement interpretable learning algorithms within their weights.

Another related meta-learning concept, termed MAML [67], aims to meta-learn a deep neural network initialization which allows for fast adaptation on novel tasks. It has been shown that in many circumstances, the solution found can be approximated well when only adapting the output layer i.e. learning a linear model on a meta-learned deep data representations [47, 67–72]. In section 2.3, we show the equivalence of this framework to in-context learning implemented in a common Transformer block i.e. when combining self-attention layers with a multi-layer-perceptron.

In the light of meta-learning we show how optimizing Transformer weights can be regarded as learning on two time scales. More concretely,

1 Main experiments can be reproduced with notebooks provided under the following link: `https://github.com/google-research/self-organising-systems/tree/master/transformers_learn_icl_by_gd`

we find that solely through the pressure to predict correctly Transformers discover learning algorithms inside their forward computations, effectively meta-learning a learning algorithm. Recently, this concept of an emergent optimizer within a learned neural network, such as a Transformer, has been termed "mesa-optimization" [73]. We find and describe one possible realization of this concept and hypothesize that the in-context learning capabilities of language models emerge through mechanisms similar to the ones we discuss here.

Transformers come in different "shapes and sizes", operate on vastly different domains, and exhibit varying forms of phase transitions of in-context learning [74, 75], suggesting variance and significant complexity of the underlying learning mechanisms. As a result, we expect our findings on linear self-attention-only Transformers to only explain a limited part of a complex process, and it may be one of many possible methods giving rise to in-context learning. Nevertheless, our approach provides an intriguing perspective on, and novel evidence for, an in-context learning mechanism that significantly differs from existing mechanisms based on associative memory [76], or by the copying mechanism termed *induction heads* identified by [49]. We, therefore, state the following

Hypothesis 1 (Transformers learn in-context by gradient descent) *When training Transformers on auto-regressive tasks, in-context learning in the Transformer forward pass is implemented by gradient-based optimization of an implicit auto-regressive inner loss constructed from its in-context data.*

We acknowledge work done in parallel, investigating the same hypothesis. Akyürek *et al.* [77] puts forward a weight construction based on a chain of Transformer layers (including MLPs) that together implement a single step of gradient descent with weight decay. Similar to work done by Garg *et al.* [78], they then show that trained Transformers match the performance of models obtained by gradient descent. Nevertheless, it is not clear that optimization finds Transformer weights that coincide with their construction.

Here, we present a much simpler construction that builds on Schlag, Irie & Schmidhuber [66] and *only requires a single linear self-attention layer* to implement a step of gradient descent. This allows us to (1) show that optimizing self-attention-only Transformers finds weights that match our weight construction (Proposition 1), demonstrating its practical relevance, and (2) explain in-context learning in shallow two layer Transformers intensively studied by Olsson *et al.* [49]. Therefore, although related work

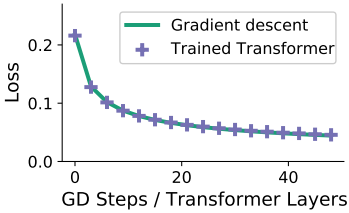


FIGURE 2.2: **Evidence for our hypothesis: gradient-based optimization and attention-based in-context learning are equivalent.** Our results, particular circumstances, confirm the hypothesis that learning with K steps of gradient descent matches trained Transformers with K linear self-attention layers when given $\mathrm{D}^{\mathrm{train}}$ as in-context data $\mathrm{D}^{\mathrm{context}}$.

provides comprehensive empirical evidence that Transformers indeed seem to implement gradient descent based learning on the data given in-context, we will in the following present mechanistic verification of this hypothesis and provide compelling evidence that our construction, which implements GD in a Transformer forward pass, is found in practice.

2.2 LINEAR SELF-ATTENTION *can* EMULATE GRADIENT DESCENT ON A LINEAR REGRESSION TASKS

We start by reviewing a standard multi-head self-attention (SA) layer with parameters θ. A SA layer updates each element e_j of a set of tokens $\{e_1, \ldots, e_N\}$ according to

$$\begin{aligned} e_j &\leftarrow e_j + \mathrm{SA}_\theta(j, \{e_1, \ldots, e_N\}) \\ &= e_j + \sum_h P_h V_h \mathrm{softmax}(K_h^T q_{h,j}) \end{aligned} \tag{2.1}$$

with P_h, V_h, K_h the projection, value and key matrices, respectively, and $q_{h,i}$ the query, all for the h-th head. To simplify the presentation, we omit bias terms here and throughout. The columns of the value $V_h = [v_{h,1}, \ldots, v_{h,N}]$ and key $K_h = [k_{h,1}, \ldots, k_{h,N}]$ matrices consist of vectors $v_{h,i} = W_{h,V} e_i$ and $k_{h,i} = W_{h,K} e_i$; likewise, the query is produced by linearly projecting the tokens, $q_{h,j} = W_{h,Q} e_j$. The parameters $\theta = \{P_h, W_{h,V}, W_{h,K}, W_{h,Q}\}_h$ of a SA layer consist of all the projection matrices, of all heads.

The self-attention layer described above corresponds to the one used in the standard Transformer model. Following Schlag, Irie & Schmidhuber [66],

we now introduce our first (and only) departure from the standard model, and omit the softmax operation in equation 2.1, leading to the *linear* self-attention (LSA) layer $e_j \leftarrow e_j + \mathrm{LSA}_\theta(j, \{e_1, \ldots, e_N\}) = e_j + \sum_h P_h V_h K_h^T q_{h,j}$ We next show that with some simple manipulations we can relate the update performed by an LSA layer to one step of gradient descent on a linear regression loss.

Data transformations induced by gradient descent

We now introduce a reference linear model $y(x) = Wx$ parameterized by the weight matrix $W \in \mathbb{R}^{N_y \times N_x}$, and a training dataset $D = \{(x_i, y_i)\}_{i=1}^N$ comprising of input samples $x_i \in \mathbb{R}^{N_x}$ and respective labels $y_i \in \mathbb{R}^{N_y}$. The goal of learning is to minimize the squared-error loss:

$$L(W) = \frac{1}{2N} \sum_{i=1}^{N} \|Wx_i - y_i\|^2. \tag{2.2}$$

One step of gradient descent on L with learning rate η yields the weight change

$$\Delta W = -\eta \nabla_W L(W) = -\frac{\eta}{N} \sum_{i=1}^{N} (Wx_i - y_i) x_i^T. \tag{2.3}$$

Considering the loss after changing the weights, we obtain

$$\begin{aligned} L(W + \Delta W) &= \frac{1}{2N} \sum_{i=1}^{N} \|(W + \Delta W)x_i - y_i\|^2 \\ &= \frac{1}{2N} \sum_{i=1}^{N} \|Wx_i - (y_i - \Delta y_i)\|^2 \end{aligned} \tag{2.4}$$

where we introduced the transformed targets $y_i - \Delta y_i$ with $\Delta y_i = \Delta W x_i$. Thus, we can view the outcome of a gradient descent step as an update to our regression loss (equation 2.2), where data, and not weights, are updated. Note that this formulation is closely linked to predicting based on nonparametric kernel smoothing, see Appendix 2.6.1 for a discussion.

Returning to self-attention mechanisms and Transformers, we consider an in-context learning problem where we are given N context tokens together with an extra query token, indexed by $N + 1$. In terms of our linear regression problem, the N context tokens $e_j = (x_j, y_j) \in \mathbb{R}^{N_x + N_y}$ correspond to the N training points in D, and the N+1-th token $e_{N+1} = (x_{N+1}, y_{N+1}) =$

$(x_{\text{test}}, \hat{y}_{\text{test}}) = e_{\text{test}}$ to the test input x_{test} and the corresponding prediction $\hat{y}_{\text{test}}$. We use the terms training and in-context data interchangeably, as well as query and test token/data, as we establish their equivalence now.

Transformations induced by gradient descent and a linear self-attention layer can be equivalent

We have re-cast the task of learning a linear model as directly modifying the data, instead of explicitly computing and returning the weights of the model (equation 2.4). We proceed to establish a connection between self-attention and gradient descent. We provide a construction where learning takes place simultaneously by directly updating all tokens, including the test token, through a linear self-attention layer. In other words, the token produced in response to a query (test) token is transformed from its initial value $W_0 x_{\text{test}}$, where W_0 is the initial value of W, to the post-learning prediction $\hat{y} = (W_0 + \Delta W) x_{\text{test}}$ obtained after one gradient descent step.

Proposition 1 *Given a 1-head linear attention layer and the tokens $e_j = (x_j, y_j)$, for $j = 1, \ldots, N$, one can construct key, query and value matrices W_K, W_Q, W_V as well as the projection matrix P such that a Transformer step on every token e_j is identical to the gradient-induced dynamics $e_j \leftarrow (x_j, y_j) + (0, -\Delta W x_j) = (x_j, y_j) + PVK^T q_j$ such that $e_j = (x_j, y_j - \Delta y_j)$. For the test data token (x_{N+1}, y_{N+1}) the dynamics are identical.*

We provide the weight matrices in block form: $W_K = W_Q = \begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix}$ with I_x and I_y the identity matrices of size N_x and N_y respectively. Furthermore, we set $W_V = \begin{pmatrix} 0 & 0 \\ W_0 & -I_y \end{pmatrix}$ with the weight matrix $W_0 \in \mathbb{R}^{N_y \times N_x}$ of

the linear model we wish to train and $P = \frac{\eta}{N} I$ with identity matrix of size $N_x + N_y$. With this simple construction we obtain the following dynamics

$$\begin{aligned}
\begin{pmatrix} x_j \\ y_j \end{pmatrix} &\leftarrow \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \frac{\eta}{N} \sum_{i=1}^{N} W_V \begin{pmatrix} x_i \\ y_i \end{pmatrix} \otimes \left(W_Q \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right) W_K \begin{pmatrix} x_j \\ y_j \end{pmatrix} \\
&= \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \frac{\eta}{N} I \sum_{i=1}^{N} \begin{pmatrix} 0 \\ W_0 x_i - y_i \end{pmatrix} \otimes \begin{pmatrix} x_i \\ 0 \end{pmatrix} \begin{pmatrix} x_j \\ 0 \end{pmatrix} \qquad (2.5) \\
&= \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \frac{\eta}{N} I \sum_{i=1}^{N} \begin{pmatrix} 0 & 0 \\ (W_0 x_i - y_i) x_i^T & 0 \end{pmatrix} \begin{pmatrix} x_j \\ 0 \end{pmatrix} \qquad (2.6) \\
&= \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} 0 \\ -\Delta W x_j \end{pmatrix}. \qquad (2.7)
\end{aligned}$$

for every token $e_j = (x_j, y_j)$ including the query token $e_{N+1} = e_{\text{test}} = (x_{\text{test}}, -W_0 x_{\text{test}})$ which will give us the desired result. We denote the corresponding self-attention weights by θ_{GD}.

Below, we provide some additional insights on what is needed to implement the provided LSA-layer weight construction, and further details on what it can achieve:

- **Full self-attention.** Our dynamics model training is based on in-context tokens only, i.e., only $e_1, \ldots, e_N$ are used for computing key and value matrices; the query token e_{N+1} (containing test data) is excluded. This leads to a linear function in x_{test} as well as to the correct ΔW, induced by gradient descent on a loss consisting only of the training data. This is a minor deviation from full self-attention. In practice, this modification can be dropped, which corresponds to assuming that the underlying initial weight matrix is zero, $W_0 \approx 0$, which makes ΔW in equation 2.5 independent of the test token even if incorporating it in the key and value matrices. In our experiments, we see that these assumptions are met when initializing the attention weights θ to small values.

- **Reading out predictions.** When initializing the y-entry of the test-data token with $-W_0 x_{N+1}$, i.e. $e_{\text{test}} = (x_{\text{test}}, -W_0 x_{\text{test}})$, the test-data prediction $\hat{y}$ can be easily read out by simply multiplying again by -1 the updated token, since $-y_{N+1} + \Delta y_{N+1} = -(y_{N+1} - \Delta y_{N+1}) = y_{N+1} + \Delta W x_{N+1}$. This can easily be done by a final projection matrix, which incidentally is usually found in Transformer architectures. Importantly, we see that a single head of self-attention is sufficient to

transform our training targets as well as the test prediction simultaneously.

- **Uniqueness.** We note that the construction is not unique; in particular, it is only required that the products PW_V as well as $W_K W_Q$ match the construction. Furthermore, since no nonlinearity is present, any rescaling s of the matrix products, i.e., $PW_V s$ and $W_K W_Q / s$, leads to an equivalent result. If we correct for these equivalent formulations, we can experimentally verify that weights of our learned Transformers indeed match the presented construction.

- **Meta-learned task-shared learning rates.** When training self-attention parameters θ across a family of in-context learning tasks τ, where the data $(x_{\tau,i}, y_{\tau,i})$ follows a certain distribution, the learning rate can be implicitly (meta-)learned such that an optimal loss reduction (averaged over tasks) is achieved given a fixed number of update steps. In our experiments, we find this to be the case. This kind of meta-learning to improve upon plain gradient descent has been leveraged in numerous previous approaches for deep neural networks [79–83].

- **Task-specific data transformations.** A self-attention layer is in principle further capable of exploiting statistics in the current training data samples, beyond modeling task-shared curvature information in θ. More concretely, a LSA layer updates an input sample according to a data transformation $x_j \leftarrow x_j + \Delta x_j = (I + P(X)V(X)K(X)^T W_Q)x_j = H_\theta(X)x_j$, with X the $N_x \times N$ input training data matrix, when neglecting influences by target data y_i. Through $H_\theta(X)$, a LSA layer can encode in θ an algorithm for carrying out data transformations which depend on the actual input training samples in X. In our experiments, we see that trained self-attention learners employ a simple form of $H(X)$ and that this leads to substantial speed ups in for GD and TF learning.

2.3 TRAINED TRANSFORMERS *do* MIMIC GRADIENT DESCENT ON LINEAR REGRESSION TASKS

We now experimentally investigate whether trained attention-based models implement gradient-based in-context learning in their forward passes. We gradually build up from single linear self-attention layers to multi-layer nonlinear models, approaching full Transformers. In this section, we follow

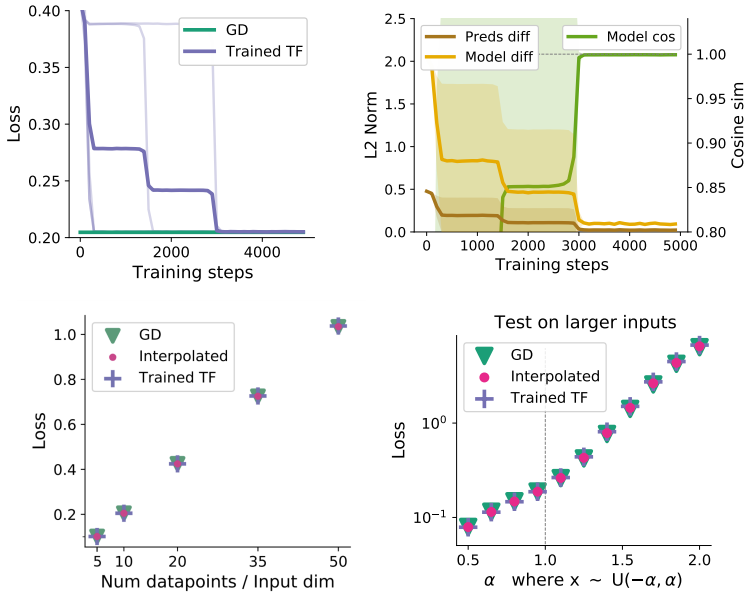


FIGURE 2.3: **Comparing one step of GD with a trained single linear self-attention layer.** *Upper left:* Trained single LSA layer performance is identical to the one of gradient descent. *Upper right*: Almost perfect alignment of GD and the model generated by the SA layer after training, measured by cosine similarity and the L2 distance between models as well as their predictions. *Lower right:* Identical loss of GD, the LSA layer model as well as the model obtained by interpolating between the construction and the optimized LSA layer weights for different $N = N_x$. *Lower right:* The trained LSA layer, gradient descent and their interpolation show identically loss (in log-scale) when provided input data different than during training i.e. with scale of 1. We display the mean/std. or the single runs of 5 seeds.

the assumption of Proposition 1 tightly and construct our tokens by concatenating input and target data, $e_j = (x_j, y_j)$ for $1 \leq j \leq N$, and our query token by concatenating the test input and a zero vector, $e_{N+1} = (x_{\text{test}}, 0)$. We show how to lift this assumption in the last section of the chapter. The prediction $\hat{y}_\theta(\{e_{\tau,1}, \ldots, e_{\tau,N}\}, e_{\tau,N+1})$ of the attention-based model, which depends on all tokens and on the parameters θ, is read-out from the y-entry of the updated $N+1$-th token as explained in the previous section.

The objective of training, visualized in Figure 2.1, is to minimize the expected squared prediction error, averaged over tasks

$$\min_\theta \mathbb{E}_\tau[||\hat{y}_\theta(\{e_{\tau,1}, \ldots, e_{\tau,N}\}, e_{\tau,N+1}) - y_{\tau,\text{test}}||^2]. \tag{2.8}$$

. We achieve this by minibatch online minimization (by Adam [84]): At every optimization step, we construct a batch of novel training tasks and take a step of stochastic gradient descent on the loss function:

$$\mathcal{L}(\theta) = \frac{1}{B} \sum_{\tau=1}^{B} ||\hat{y}_\theta(\{e_{\tau,i}\}_{i=1}^N, e_{\tau,N+1}) - y_{\tau,\text{test}}||^2 \tag{2.9}$$

where each task (context) τ consists of in-context training data $D_\tau = \{(x_{\tau,i}, y_{\tau,i})\}_{i=1}^N$ and test point $(x_{\tau,N+1}, y_{\tau,N+1})$, which we use to construct our tokens $\{e_{\tau,i}\}_{i=1}^{N+1}$ as described above. We denote the optimal parameters found by this optimization process by θ^*. In our setup, finding θ^* may be thought of as meta-learning, while learning a particular task τ corresponds to simply evaluating the model $\hat{y}_\theta(\{e_{\tau,1}, \ldots, e_{\tau,N}\}, e_{\tau,N+1})$. Note that we therefore never see the exact same training task twice during training.

We focus on solvable tasks and similarly to Garg *et al.* [78] generate data for each task using a teacher model with parameters $W_\tau \sim \mathcal{N}(0, I)$. We then sample $x_{\tau,i} \sim U(-1,1)^{n_I}$ and construct targets using the task-specific teacher model, $y_{\tau,i} = W_\tau x_{\tau,i}$. In the majority of our experiments we set the dimensions to $N = n_I = 10$ and $n_O = 1$. Since we use a noiseless teacher for simplicity, we can expect our regression tasks to be well-posed and analytically solvable as we only compute a loss on the Transformers last token, which stands in contrast to usual autoregressive training and the training setup of Garg *et al.* [78].

One-step of gradient descent vs. a single trained self-attention layer

Our first goal is to investigate whether a trained single, linear self-attention layer can be explained by the provided weight construction that implements

GD. To that end, we compare the predictions made by a LSA layer with trained weights θ^* (which minimize equation 2.9) and with constructed weights θ_{GD} (which satisfy Proposition 1).

Recall that a LSA layer yields the prediction

$$\hat{y}_\theta(x_{\mathrm{test}}) = e_{N+1} + \mathrm{LSA}_\theta(\{e_1, \ldots, e_N\}, e_{N+1}) = \Delta W_{\theta,D} x_{\mathrm{test}} \tag{2.10}$$

, which is linear in x_{test}. We denote by $\Delta W_{\theta,D}$ the matrix generated by the LSA layer following the construction provided in Proposition 1, with query token e_{N+1} set such that the initial prediction is set to zero, $\hat{y}_{\mathrm{test}} = 0$. We compare $\hat{y}_\theta(x_{\mathrm{test}})$ to the prediction of the control LSA $\hat{y}_{\theta_{\mathrm{GD}}}(x_{\mathrm{test}})$, which under our token construction corresponds to a linear model trained by one step of gradient descent starting from $W_0 = 0$. For this control model, we determine the optimal learning rate η by minimizing $\mathcal{L}(\eta)$ over a training set of 10^4 tasks through line search, with $\mathcal{L}(\eta)$ defined analogously to equation 2.9.

More concretely, to compare trained and constructed LSA layers, we sample $T_{\mathrm{val}} = 10^4$ validation tasks and record the following quantities, averaged over validation tasks: (1) the difference in predictions measured with the L2 norm, $\|\hat{y}_\theta(x_{\tau,\mathrm{test}}) - \hat{y}_{\theta_{\mathrm{GD}}}(x_{\tau,\mathrm{test}})\|$, (2) the cosine similarity between the sensitivities $\frac{\partial \hat{y}_{\theta_{\mathrm{GD}}}(x_{\tau,\mathrm{test}})}{\partial x_{\mathrm{test}}}$ and $\frac{\partial \hat{y}_\theta(x_{\tau,\mathrm{test}})}{\partial x_{\mathrm{test}}}$ as well as (3) their difference $\| \frac{\partial \hat{y}_{\theta_{\mathrm{GD}}}(x_{\tau,\mathrm{test}})}{\partial x_{\mathrm{test}}} - \frac{\partial \hat{y}_\theta(x_{\tau,\mathrm{test}})}{\partial x_{\mathrm{test}}} \|$ again according to the L2 norm, which in both cases yields the explicit models computed by the algorithm. We show the results of these comparisons in Figure 2.3. We find an excellent agreement between the two models over a wide range of hyperparameters. We note that as we do not have direct access to the initialization of W in the attention-based learners (it is hidden in θ), we cannot expect the models to agree exactly.

Although the above metrics are important to show similarities between the resulting learned models (in-context vs. gradient-based), the underlying algorithms could still be different. We therefore carry out an extended set of analyses:

1. **Interpolation.** We take inspiration on recent work [85, 86] that showed approximate equivalence of models found by SGD after permuting weights within the trained neural networks. Since our models are deep linear networks with respect to x_{test} we only correct for scaling mismatches between the two models – in this case the construction that implements GD and the trained weights. As shown in Figure 2.3, we observe (and can actually inspect by eye, see Appendix Figure

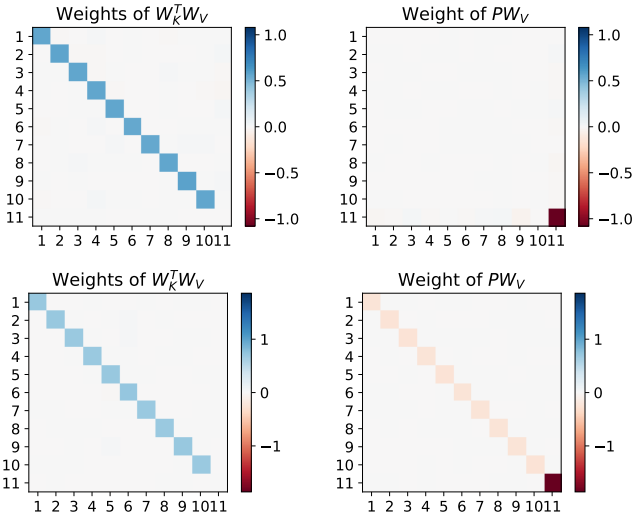


FIGURE 2.4: **Visualizing the weight matrices of trained Transformers**. *Upper Left & upper right:* Weight matrix products of a trained single linear self-attention layer. We see (after scalar correction) a perfect resemblance of our construction. *Lower right & outer left:* Weight matrix products of a trained 3-layer recurrent linear self-attention Transformer. Again, we see (after scalar correction) a perfect resemblance of our construction and an additional curvature correction i.e. diagonal values in PW_V of the same magnitude except the last entry that functions as the learning rate.

2.4) that a simple scaling correction on the trained weights is enough to recover the weight construction implementing GD. This leads to an identical loss of GD, the trained Transformer and the linearly interpolated weights $\theta_{\mathrm{I}} = (\theta + \theta_{\mathrm{GD}})/2$. See details in Appendix 2.6.3 on how our weight correction and interpolation is obtained.

2. **Out-of-distribution validation tasks.** To test if our in-context learner has found a generalizable update rule, we investigate how GD, the trained LSA layer and its interpolation behave when providing in-context data in regimes different to the ones used during training. We therefore visualize the loss increase when (1) sampling the input data from $U(-\alpha, \alpha)^{N_x}$ or (2) scaling the teacher weights by α as αW when sampling validation tasks. For both cases, we set $\alpha = 1$ during training. We again observe that when training a single linear self-attention Transformer, for both interventions, the Transformer performs equally to gradient descent outside of this training setups, see Figure 2.3. Note that the loss obtained through gradient descent also starts degrading quickly outside the training regime. Since we tune the learning rate for the input range $[-1, 1]$ and one gradient step, tasks with larger input range will have higher curvature and the optimal learning rate for smaller ranges will lead to divergence and a drastic increase in loss also for GD.

3. **Repeating the LSA update.** Since we claim that a single trained LSA layer implements a GD-like learning rule, we further test its behavior when applying it repeatedly, not only once as in training. After we correct the learning rate of both algorithms, i.e. for GD and the trained Transformer with a dampening parameter $\lambda = 0.75$ (details in Appendix 2.6.5), we see an identical loss decrease of both GD and the Transformer, see Figure 2.2.

To conclude, we present evidence that optimizing a single LSA layer to solve linear regression tasks finds weights that (approximately) coincide with the LSA-layer weight construction of Proposition 1, hence implementing a step of gradient descent, leading to the same learning capabilities on in- and out-of-distribution tasks.

Multiple steps of gradient descent vs. multiple layers of self-attention

We now turn to deep linear self-attention-only Transformers. The construction we put forth in Proposition 1, can be immediately stacked up over K lay-

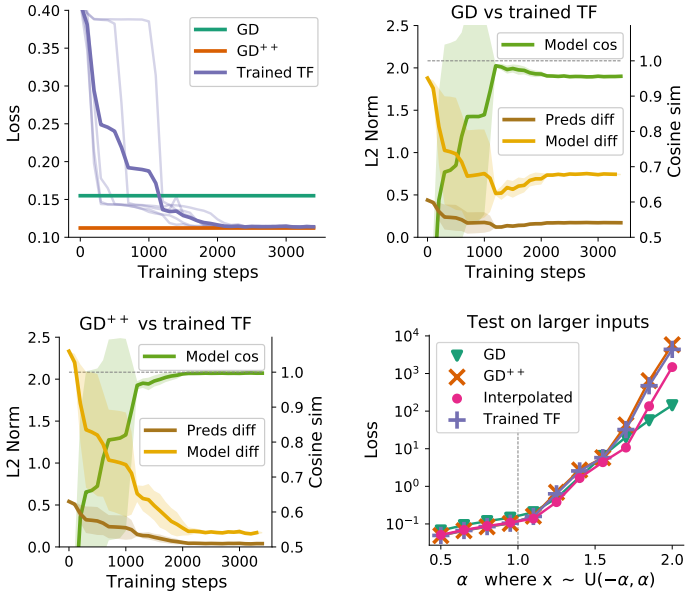


FIGURE 2.5: **Comparing two steps of gradient descent with trained *recurrent* two-layer Transformers.** *Top left:* The trained TF performance surpasses standard GD but matches GD^{++}, our GD variant with simple iterative data transformation. On both cases, we tuned the gradient descent learning rates as well as the scalar γ which governs the data transformation $H(X)$. *Top right & lower left*: We measure the alignment between the GD as well as the GD^{++} models and the trained TF. In both cases the TF aligns well with GD in the beginning of training but aligns much better with GD^{++} after training. *Lower right:* TF performance (in log-scale) mimics the one of GD^{++} well when testing on OOD tasks ($\alpha \neq 1$).

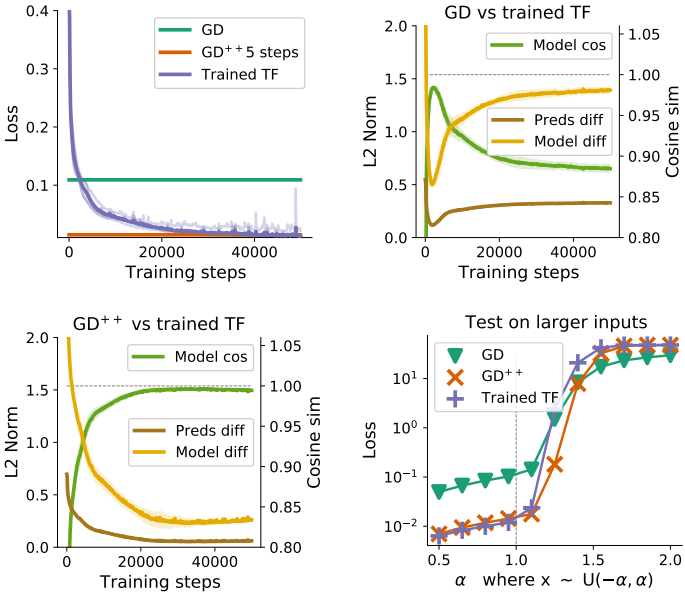


FIGURE 2.6: **Comparing five steps of gradient descent with trained five-layer Transformers.** *Top left:* The trained TF performance surpasses standard GD but matches GD^{++}, our GD variant with simple iterative data transformation. On both cases, we tuned the gradient descent learning rates as well as the scalar γ which governs the data transformation $H(X)$. *Top right & lower left*: We measure the alignment between the GD as well as the GD^{++} models and the trained TF. In both cases the TF aligns well with GD in the beginning of training but aligns much better with GD^{++} after training. *Lower right:* TF performance (in log-scale) mimics the one of GD^{++} well when testing on OOD tasks ($\alpha \neq 1$).

ers; in this case, the final prediction can be read out from the last layer as before by negating the y-entry of the last test token: $-y_{N+1} + \sum_{k=1}^{K} \Delta y_{k,N+1} = -(y_{N+1} - \sum_{k=1}^{K} \Delta y_{k,N+1}) = y_{N+1} + \sum_{k=1}^{K} \Delta W_k x_{N+1}$, where $y_{k,N+1}$ are the test token values at layer k, and $\Delta y_{k,N+1}$ the change in the y-entry of the test token after applying the k-th step of self-attention, and ΔW_k the k-th implicit change in the underlying linear model parameters W. When optimizing such Transformers with K layers, we observe that these models generally outperform K steps of plain gradient descent, see Figure 2.5. Their behavior is however well described by a variant of gradient descent, for which we tune a single parameter γ defined through the transformation function $H(X)$ which transforms the input data according to $x_j \leftarrow H(X)x_j$, with $H(X) = (I - \gamma XX^T)$. We term this gradient descent variant GD^{++} which we analyze it in the next section.

To analyze the effect of adding more layers to the architecture, we first turn to the arguably simplest extension of a single SA layer and analyze a *recurrent* or *looped* 2-layer LSA model. Here, we simply repeatably apply the same layer (with the same weights) multiple times i.e. drawing the analogy to learning an iterative algorithm that applies the same logic multiple times.

Somewhat surprisingly, we find that the trained model surpasses plain gradient descent, which also results in decreasing alignment between the two models (see center left column), and the recurrent Transformer realigns perfectly with GD^{++} while matching its performance on in- and out-of distribution tasks. Again, we can interpolate between the Transformer weights found by optimization and the LSA-weight construction with learned η, γ, see Figure 2.5.

We next consider deeper, non-recurrent 5-layer LSA-only Transformers, with different parameters per layer (i.e. no weight tying). We see that a different GD learning rate as well as γ per step (layer) need to be tuned to match the Transformer performance. This slight modification leads again to almost perfect alignment between the trained TF and GD^{++} with in this case 10 additional parameters and loss close to 0, see Figure 2.5. Nevertheless, we see that the naive correction necessary for model interpolation used in the aforementioned experiments is not enough to interpolate without a loss increase. We leave a search for better weight corrections to future work. We further study Transformers with different depths for recurrent as well as non-recurrent architectures with multiple heads and equipped with MLPs, and find qualitatively equivalent results.

Additionally, we provide results obtained when using softmax-SA layers as well as LayerNorm, thus essentially retrieving the standard Transformer

architecture. We again observe and are able to explain (after slight architectural modifications) good learning performance and as well as alignment with the construction of Proposition 1, though worse than when using linear self-attention. These findings suggest that the in-context learning abilities of the standard Transformer with these common architecture choices can be explained by the gradient-based learning hypothesis explored here. Our findings also question the ubiquitous use of softmax attention, and suggest further investigation is warranted into the performance of linear vs. softmax SA layers in real-world learning tasks, as initiated by Schlag, Irie & Schmidhuber [66].

DETAILS OF GD^{++}

We give here a precise construction showing how to implement in a single head, a step of GD and the discussed data transformation, resulting in GD^{++}. Recall again the linear self-attention operation with a single head

$$e_j \leftarrow e_j + PW_V \sum_i e_i \otimes e_i W_K^T. \tag{2.11}$$

We provide again the weight matrices in block form of the construction of Prop. 1 but now enabling additionally our described data transformation: $W_K = W_Q = \begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix}$ with I_x the identity matrix of size N_x, I_y od size N_y resp. Furthermore, we set $W_V = \begin{pmatrix} I_x & 0 \\ W & -I_y \end{pmatrix}$ with the weight matrix $W \in \mathbb{R}^{N_y \times N_x}$ of the linear model we wish to train and $P = \begin{pmatrix} -\gamma I_x & 0 \\ 0 & \frac{\eta}{N} \end{pmatrix}$. This leads to the following update

$$\begin{aligned} \begin{pmatrix} x_j \\ y_j \end{pmatrix} &\leftarrow \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} -\gamma I_x & 0 \\ 0 & \frac{\eta}{N} \end{pmatrix} \sum_{i=1}^N \left(\begin{pmatrix} I_x & 0 \\ W & -I_y \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right) \otimes \begin{pmatrix} x_i \\ 0 \end{pmatrix} \begin{pmatrix} x_j \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} -\gamma I_x & 0 \\ 0 & \frac{\eta}{N} \end{pmatrix} \sum_{i=1}^N \begin{pmatrix} x_i \\ Wx_i - y_i \end{pmatrix} \otimes \begin{pmatrix} x_i \\ 0 \end{pmatrix} \begin{pmatrix} x_j \\ 0 \end{pmatrix} \qquad (2.12) \\ &= \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} -\gamma X X^T x_j \\ -\Delta W x_j \end{pmatrix}. \qquad (2.13) \end{aligned}$$

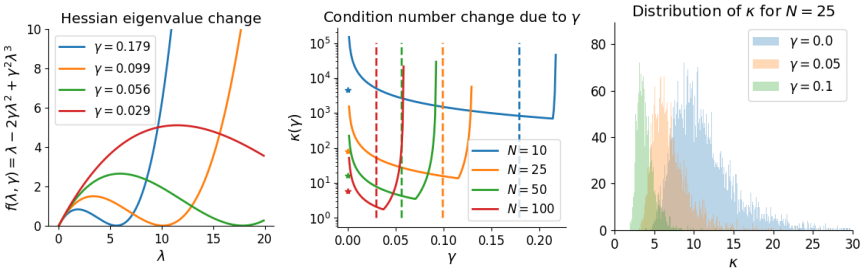


FIGURE 2.7: **GD^{++} analyses**. *Left:* We visualize the change of the eigenspectrum induced by the input data transformation of GD^{++} for different γ observed in practice. *Center:* Given we know the maximum and minimum of eigenvalues λ_1, λ_n of the loss Hessian XX^T with $X = (x_0, \ldots, x_N)$ for different N, we compare the original condition number (depicted by *'s at $\gamma = 0$) and the condition number (in log scale) of the GD^{++} altered loss Hessian when varying γ. We plot in dotted lines the γ values that we observe in practice which are close the optimal ones i.e. the local minimum derived through our analysis. *Right:* For$N = 25$, we plot for different γ values the distribution of condition numbers $\kappa = \lambda_1/\lambda_n$ for 10000 tasks and observe favorable κ values close to 1 when approaching the $\gamma = 0.099$ value was found in practice. The κ values quickly explode for $\gamma > 0.1$.

for every token $e_j = (x_j, y_j)$ including the query token $e_{N+1} = e_{\text{test}} = (x_{\text{test}}, 0)$ which will give us the desired result.

Why does GD^{++} perform better? We give here one possible explanation of the superior performance of GD^{++} compared to GD. Note that there is a close resemblance of the GD transformation and a heavily truncated Neuman series approximation of the inverse XX^T. However, we provide here a more heuristic explanation for the observed acceleration.

Given $\gamma \in \mathbb{R}$, GD^{++} transforms every input according to

$$x_i \leftarrow x_i - \gamma XX^T x_i = (I - \gamma XX^T)x_i. \tag{2.14}$$

We can therefore look at the change of squared regression loss $L(W) = \frac{1}{2}\sum_{i=0}^{N}(Wx_i - y_i)^2$ induced by this transformation i.e.

$$L^{++}(W) = \frac{1}{2}\sum_{i=0}^{N}(W(I-\gamma XX^T)x_i - y_i)^2 = \frac{1}{2}(W(I-\gamma XX^T)X - Y)^2 \tag{2.15}$$

which in turn leads to a change of the loss Hessian from $\nabla^2 L = XX^T$ to

$$\nabla^2 L^{++} = (I - \gamma XX^T)X((I - \gamma XX^T)X)^T \tag{2.16}$$

.

Given the original Hessian and it's diagonalization $H = XX^T = U\Sigma U^T$ with it's set of sorted eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$ and $\lambda_i \geq 0$ on the diagonal matrix Σ we can express the new Hessian through U, Σ i.e.

$$H^{++} = (I - \gamma XX^T)X((I - \gamma XX^T)X)^T \tag{2.17}$$

$$= (I - \gamma U\Sigma U^T)U\Sigma U^T(I - \gamma U\Sigma U^T)^T \tag{2.18}$$

.

We can simplify H^{++} further as

$$H^{++} = (I - \gamma U\Sigma U^T)U\Sigma U^T(I - \gamma U\Sigma U^T)^T \tag{2.19}$$

$$= U(\Sigma - \gamma\Sigma^2)U^T U(I - \gamma\Sigma)U^T \tag{2.20}$$

$$= U(\Sigma - 2\gamma\Sigma^2 + \gamma^2\Sigma^3)U^T \tag{2.21}$$

Given the eigenspectrum $\{\lambda_1, \ldots, \lambda_n\}$ of H, we obtain an (unsorted) eigenspecturm for H^{++} with $\{\lambda_1 - 2\gamma\lambda_1^2 + \gamma^2\lambda_1^3, \ldots, \lambda_n - 2\gamma\lambda_n^2 + \gamma^2\lambda_n^3\}$ which we visualize in Figure 2.7 for different γ observed in practice. We hypotheses that the Transformer chooses γ in a way that on average, across the distribution of tasks, the data transformation (iteratively) decreases the condition number λ_1/λ_n leading to accelerated learning. This could be achieved, for example, by keeping the smallest eigenvalue $\lambda_n \approx \lambda_n^{++}$ fixed and choosing γ such that the largest eigenvalue of the transformed data λ_1^{++} is reduced, while the original λ_1 stays within $[\lambda_1^{++}, \lambda_n^{++}]$.

To support our hypotheses empirically, we computed the minimum and maximum eigenvalues of XX^T across 10000 tasks while changing the number of datapoints $N \in [10, 25, 50, 100]$ i.e. $X = (x_0, \ldots, x_N)$ leading to better conditioned loss Hessians i.e. $[1e{-}10, 0.097, 0.666, 2.870]$ and $[4.6, 7.712, 10.845, 17.196]$ as the minimum and maximum eigenvalues of XX^T across all tasks where we cut the smallest eigenvalue for $N = 10$ at $1e{-}10$. Furthermore, we extract the γ values from the weights of optimized recurrent 2-layer Transformers trained on different task distributions and obtain γ values of $[0.179, 0.099, 0.056, 0.029]$, see again Figure 2.7. Note that the observed eigenvalues stay within $[0, 1/\gamma]$ i.e. the two roots of $f(\lambda, \gamma) = \lambda - 2\gamma\lambda^2 + \gamma^2\lambda^3$.

Given the derived function of eigenvalue change $f(\lambda, \gamma)$, we compute the condition number of H^{++} by dividing the novel maximum eigenvalues

$\lambda_1^{++} = f(1/(3\gamma), \gamma)$ where $\lambda = 1/(3\gamma)$ as the local maximum of $f(\lambda, \gamma)$, for fixed γ, and the novel minimum eigenvalue $\lambda_n^{++} = \min(f(\lambda_1, \gamma), f(\lambda_n, \gamma))$. Note that with too small γ, we move the original λ_n closer to the root of $f(\lambda, \gamma)$ i.e. $\lambda = 1/\gamma$ and therefore can change the smallest eigenvalue.

Given the task distribution and its corresponding eigenvalue distribution, we see that choosing γ reduces the new condition number $\kappa^{++} = \lambda_1^{++}/\lambda_n^{++}$ which leads to better conditioned learning, see center plot of Figure 2.7. Note that the optimal γ based on our derivation above is based on the maximum and minimum eigenvalue across all tasks and does not take the change of the whole eigenvalue distribution into account. We argue therefore that the simplicity of the arguments above does not capture the task statistics and distribution shifts entirely. Therefore we obtain a slightly larger γ as the value observed in practice. We furthermore visualize the condition number change for $N = 25$ and 10000 tasks in the right plot of Figure 2.7 and observe the distribution moving to desirable κ values close to 1. For γ values larger than 0.1 the distribution quickly exhibits exploding condition numbers.

Transformers solve nonlinear regression tasks by gradient descent on deep data representations

It is unreasonable to assume that the astonishing in-context learning flexibility observed in large Transformers is explained by gradient descent on linear models. We now show that this limitation can be resolved by incorporating one additional element of fully-fledged Transformers: preceding self-attention layers by MLPs enables learning linear models by gradient descent on deep representations which motivates our illustration in Figure 2.1. Empirically, we demonstrate this by solving non-linear sine-wave regression tasks, see Figure 2.8. We state

Proposition 2 *Given a Transformer block i.e. a MLP $m(e)$ which transforms the tokens $e_j = (x_j, y_j)$ followed by an attention layer, we can construct weights that lead to gradient descent dynamics descending $\frac{1}{2N}\sum_{i=1}^{N} ||Wm(x_i) - y_i||^2$. Iteratively applying Transformer blocks therefore can solve kernelized least-squares regression problems with kernel function $k(x, y) = m(x)^\top m(y)$ induced by the MLP $m(\cdot)$.*

A detailed discussion on this form of kernel regression as well as kernel smoothing w/wo softmax nonlinearity through gradient descent on the data can be found in Appendix section 2.6.1. The way MLPs transform data

in Transformers diverges from the standard meta-learning approach, where a task-shared *input* embedding network is optimized by backpropagation-through-training to improve the learning performance of a task-specific readout [e.g., 47, 70, 87]. On the other hand, given our token construction in Proposition 1, MLPs in Transformers intriguingly process both inputs *and* targets. The output of this transformation is then processed by a single linear self-attention layer, which, according to our theory, is capable of implementing gradient descent learning. We compare the performance of this Transformer model, where all weights are learned, to a control Transformer where the final LSA weights are set to the construction θ_{GD} which is therefore identical to training an MLP by backpropagation through a GD updated output layer.

Intriguingly, both obtained functions show again surprising similarity on (1) the initial (meta-learned) prediction, read out after the MLP, and (2) the final prediction, after altering the output of the MLP through GD or the self-attention layer. This is again reflected in our alignment measures that now, since the obtained models are nonlinear w.r.t. x_{test}, only represent the two first parts of the Taylor approximation of the obtained functions. Our results serve as a first demonstration of how MLPs and self-attention layers can interplay to support nonlinear in-context learning, allowing to fine-tune deep data representations by gradient descent. Investigating the interplay between MLPs and SA-layer in deep TFs is left for future work.

2.4 DO SELF-ATTENTION LAYERS BUILD REGRESSION TASKS?

The construction provided in Proposition 1 and the previous experimental section relied on a token structure where both input and output data are concatenated into a single token. This design is different from the way tokens are typically built in most of the related work dealing with simple few-shot learning problems as well as in e.g. language modeling. We therefore ask: Can we overcome the assumption required in Proposition 1 and allow a Transformer to build the required token construction on its own? This motivates

Proposition 3 *Given a 1-head linear or softmax attention layer and the token construction* $e_{2j} = (x_j), e_{2j+1} = (0, y_j)$ *with a zero vector* 0 *of dim* $N_x - N_y$ *and concatenated positional encodings, one can construct key, query and value matrix* W_K, W_Q, W_V *as well as the projection matrix* P *such that all tokens* e_j *are transformed into tokens equivalent to the ones required in Proposition 1.*

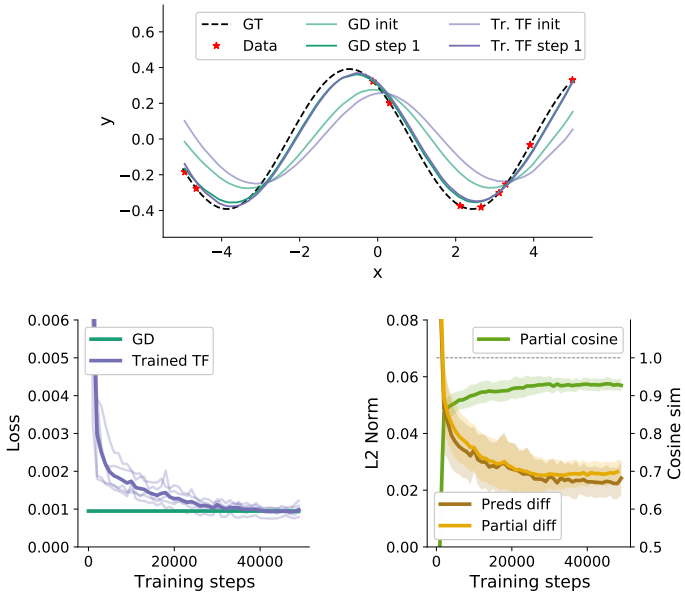


FIGURE 2.8: **Sine wave regression: comparing trained Transformers with meta-learned MLPs for which we adjust the output layer with one step of gradient descent.** *Left:* Plots of the learned initial functions as well as the adjusted functions through either a layer of self-attention or a step of GD. We observe similar initial functions as well as solutions for the trained TF compared fine-tuning a meta-learned MLP. *Center*: The performance of the trained Transformer is matched by meta-learned MLPs. *Left*: We observe strong alignment when comparing the prediction as well as the partial derivatives of the the meta-learned MLP and the trained Transformer.

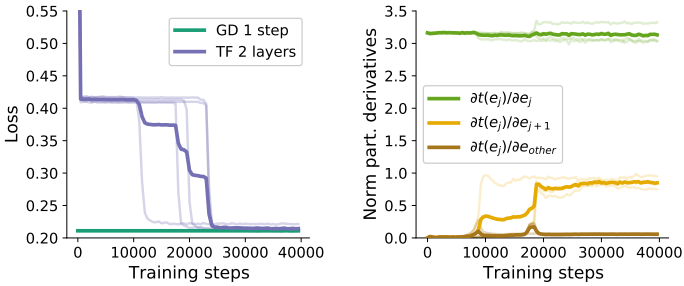


FIGURE 2.9: **Training a two layer SA-only Transformer using the standard token construction.** *Left:* The loss of trained TFs matches one step of GD, not two, and takes an order of magnitude longer to train. *Right*: Norm of the partial derivatives of the output of the first self-attention layer w.r.t. input tokens. Before the Transformer performance jumps to the one of GD, the first layer becomes highly sensitive to the next token.

The construction and its discussion can be found in Appendix section 2.6.2. To provide evidence that copying is performed in trained Transformers, we optimize a two-layer self-attention circuit on in-context data where alternating tokens include input or output data i.e. $e_{2j} = (x_j)$ and $e_{2j+1} = (0, y_j)$. We again measure the loss as well as the mean of the norm of the partial derivative of the first layer's output w.r.t. the input tokens during training, see Figure 2.9. First, the training speeds are highly variant given different training seeds, also reported in Garg *et al.* [78]. Nevertheless, the Transformer is able to match the performance of a *single* (not two) step gradient descent. Interestingly, before the Transformer performance jumps to the one of GD, token e_j transformed by the first self-attention layer becomes notably dependant on the neighboring token e_{j+1} while staying independent on the others which we denote as e_{other} in Figure 2.9.

We interpret this as evidence for a copying mechanism of the Transformer's first layer to merge input and output data into single tokens as required by Proposition 1. Then, in the second layer the Transformer performs a single step of GD. Notably, we were not able to train the Transformer with linear self-attention layers, but had to incorporate the softmax operation in the first layer. These preliminary findings support the study of Olsson *et al.* [49] showing that softmax self-attention layers easily learn to copy; we confirm this claim, and further show that such copying allows the

Transformer to proceed by emulating gradient-based learning in the second or deeper attention layers.

We conclude that copying through (softmax) attention layers is the second crucial mechanism for in-context learning in Transformers. This operation enables Transformers to merge data from different tokens and then to compute dot products of input and target data downstream, allowing for in-context learning by gradient descent to emerge.

2.5 DISCUSSION

Transformers show remarkable in-context learning behavior. Mechanisms based on attention, associative memory and copying by induction heads are currently the leading explanations for this remarkable feature of learning within the Transformer forward pass. In this chapter, we put forward the hypothesis, similar to Garg *et al.* [78] and Akyürek *et al.* [77], that Transformer's in-context learning is driven by gradient descent, in short – *Transformers learn to learn by gradient descent based on their context*. Viewed through the lens of meta-learning, learning Transformer weights corresponds to the outer-loop which then enables the forward pass to transform tokens by gradient-based optimization.

To provide evidence for this hypothesis, we build on Schlag, Irie & Schmidhuber [66] that already provide a linear self-attention layer variant with (fast-)inner loop learning by the error-correcting delta rule [88]. We diverge from their setting and focus on (in-context) learning where we specifically construct a dataset by considering neighboring elements in the input sequence as input- and target training pairs, see assumptions of Proposition 1. This construction could be realized, for example, due to the model learning to implement a copying layer, see section 2.4 and proposition 3, and allows us to provide a simple and different construction to Schlag, Irie & Schmidhuber [66] that solely is built on the standard linear, and approximately softmax, self-attention layer but still implements gradient descent based learning dynamics. We, therefore, are able to explain gradient descent based learning in these standard architectures. Furthermore, we extend this construction based on a single self-attention layer and provide an explanation of how deeper K-layer Transformer models implement principled K-step gradient descent learning, which deviates again from Schlag et al. and allows us to identify that deep Transformers implement GD++, an accelerated version of gradient descent.

We highlight that our construction of gradient descent and GD++ is not suggestive but when training multi-layer self-attention-only Transformers on simple regression tasks, we provide strong evidence that the construction is actually found. This allows us, at least in our restricted problems settings, to explain mechanistically in-context learning in trained Transformers and its close resemblance to GD observed by related work. Further work is needed to incorporate regression problems with noisy data and weight regularization into our hypothesis. We speculate aspects of learning in these settings are meta-learned – e.g., the weight magnitudes to be encoded in the self-attention weights. Additionally, we did not analyze logistic regression for which one possible weight construction is already presented in Zhmoginov, Sandler & Vladymyrov [89].

Our refined understanding of in-context learning based on gradient descent motives us to investigate how to improve it. We are excited about several avenues of future research. First, to exceed upon a single step of gradient descent in every self-attention layer it could be advantageous to incorporate so called *declarative* nodes [90–93] into Transformer architectures. This way, we would treat a single self-attention layer as the solution of a fully optimized regression loss leading to possibly more efficient architectures. Second, our findings are restricted to small Transformers and simple regression problems. We are excited to delve deeper into research trying to understand how further mechanistic understanding of Transformers and in-context learning in larger models is possible and to what extend. Third, we are excited about targeted modifications to Transformer architectures, or their training protocols, leading to improved gradient descent based learning algorithms or allow for alternative in-context learners to be implemented within Transformer weights, augmenting their functionality, as e.g. in Dai *et al.* [94]. Finally, it would be interesting to analyze in-context learning in HyperTransformers [89] that produce weights for target networks and already offer a different perspective on merging Transformers and meta-learning. There, Transformers transform weights instead of data and could potentially allow for gradient computations of weights deep inside the target network lifting the limitation of GD on linear models analyzed here.

2.6 APPENDIX

We present here some additional results to complement the main results discussed in the previous sections.

2.6.1 *Proposition 2 and connections between gradient descent, kernelized regression and kernel smoothing*

Let's consider the data transformation induced by an MLP $\tilde{m}(x)$ and a residual connection commonly used in Transformer blocks i.e. $e_j \leftarrow e_j + \tilde{m}(e_j) = (x_j, y_j) + (\tilde{m}(x_j), 0) = (m(x_j), y_j)$ with $m(x_j) = x_j + \tilde{m}(x_j)$ and $\tilde{m}$ not changing the targets y. When simply applying Proposition 1, it is easy to see that given this new token construction, a linear self-attention layer can induce the token dynamics $e_j \leftarrow (m(x_j), y_j) + (0, -\Delta W m(x_j))$ with $\Delta W = -\eta \nabla L(W)$ given the loss function $L(W) = \frac{1}{2N} \sum_{i=1}^{N} ||Wm(x_i) - y_i||^2$.

Interestingly, for the test token $e_{\text{test}} = (x_{\text{test}}, 0)$ this induces, after a multiplication with -1, an initial prediction after a single Transformer block given by

$$\hat{y} = \Delta W m(x_{\text{test}}) = -\eta \nabla_W L(0) m(x_{\text{test}}) \tag{2.22}$$

$$= \sum_{i=1}^{N} y_i m(x_i)^T m(x_{\text{test}}) = \sum_{i=1}^{N} y_i k(x_i, x_{\text{test}}) \tag{2.23}$$

with $m(x_i)^T m(x_{\text{test}}) = k(x_i, x_{\text{test}}) \in \mathbb{R}$ interpreted as a kernel function. Concluding, we see that the combination of MLPs and a *single* self-attention layer can lead to dynamics induced when descending a kernelized regression (squared error) loss with a *single* step of gradient-descent.

Interestingly, when choosing $W_0 = 0$, we furthermore see that a single self-attention layer or Transformer block can be regarded as doing non-parametric kernel smoothing $\hat{y} = \sum_{i=1}^{N} y_i k(x_i, x_{\text{test}})$ based on the data given in-context [95, 96]. Note that we made a particular choice of kernel function here and that this view still holds when $m(x_j) = \mathbb{I}$ i.e. consider Transformers without MLPs or leverage the well-known view of softmax self-attention layer as a kernel function used to measure similarity between tokens [e.g. 97, 98].

Thus, implementing one step of gradient descent through a self-attention layer (w/wo softmax nonlinearity) is equivalent to performing kernel smoothing estimation. We however argue that this nonparametric kernel smoothing view of in-context learning is limited, and arises from looking

only at a *single* self-attention layer. When considering deeper Transformer architectures, we see that multiple Transformer blocks can iteratively transform the targets based on multiple steps of gradient descent leading to minimization of a kernelized squared error loss $L(W)$. One way to obtain a suitable construction is by neglecting MLPs everywhere except in the first Transformer block.

2.6.2 *Proof and discussion of Proposition 3*

We state here again Proposition 3, provide the necessary construction and a short discussion.

Proposition 3 *Given a 1-head linear- or softmax attention layer and the token construction $e_{2j} = (x_j), e_{2j+1} = (0, y_j)$ with a zero vector 0 of dim $N_x - N_y$ and concatenated positional encodings, one can construct key, query and value matrix W_K, W_Q, W_V as well as the projection matrix P such that all tokens e_j are transformed into tokens equivalent to the ones required in proposition 1.*

To get a simple and clean construction, we choose wlog $x_j \in \mathbb{R}^{2N+1}$ and $(0, y_j) \in \mathbb{R}^{2N+1}$ as well as model the positional encodings as unit vectors $p_j \in \mathbb{R}^{2N+1}$ and concatenate them to the tokens i.e. $e_j = (x_{j/2}, p_j)$. We wish for a construction that realizes

$$e_j \leftarrow \begin{pmatrix} x_{j/2} \\ p_j \end{pmatrix} + PVK^T W_Q \begin{pmatrix} x_{j/2} \\ p_j \end{pmatrix} \tag{2.24}$$

$$= \begin{pmatrix} x_{j/2} \\ p_j \end{pmatrix} + \begin{pmatrix} 0 \\ y_{j/2+1} - p_j \end{pmatrix}. \tag{2.25}$$

This means that a token replaces its own positional encoding by coping the target data of the next token to itself leading to $e_j = (x_{j/2}, 0, y_{j/2+1})$, with slight abusive of notation. This can simply be realized by (for example) setting $P = I$, $W_V = \begin{pmatrix} 0 & 0 \\ I_x & -I_{x,off} \end{pmatrix}$, $W_K = \begin{pmatrix} 0 & 0 \\ 0 & I_x \end{pmatrix}$ and $W_Q = \begin{pmatrix} 0 & 0 \\ 0 & I_{x,off}^T \end{pmatrix}$ with $I_{x,off}$ the lower diagonal identity matrix fo size N_x. Note that then simply $K^T W_Q e_j = p_{j+1}$ i.e. it chooses the $j+1$ element of V which stays p_{j+1} if we apply the softmax operation on $K^T q_j$. Since the $j+1$ entry of V is $(0, y_{j/2+1} - p_j)$ we obtain the desired result.

For the (toy-)regression problems considered in this chapter, the provided result would give $N/2$ tokens for which we also copy (parts) of

x_j underneath y_j. This is desired for modalities such as language where every two tokens could be considered an in-and output pair for the implicit autoregressive inner-loop loss. These tokens do not have be necessarily next to each other, see for this behavior experimental findings presented in [49]. For the experiments conducted here, one solution is to zero out these tokens which could be constructed by a two-head self-attention layer that given uneven j simply subtracts itself resulting in a zero token. For all even tokens, we use the construction from above which effectively coincides with the token construction required in Proposition 1.

2.6.3 *Linear mode connectivity between the weight construction of Prop 1 and trained Transformers*

In order to interpolate between the construction θ_{GD} and the trained weights of the Transformer θ, we need to correct for some scaling ambiguity. For clarification, we restate here the linear self-attention operation for a single head

$$e_j \leftarrow e_j + PW_V \sum_i e_i \otimes e_i W_K^T W_Q e_j \tag{2.26}$$

$$= e_j + W_{PV} \sum_i e_i \otimes e_i W_{KQ} e_j \tag{2.27}$$

Now, to match the weight construction of Prop. 1 we have the aim for the matrix product W_{KQ} to match an identify matrix (except for the last diagonal entry) after re-scaling. Therefore we compute the mean of the diagonal of the matrix product of the trained Transformer weights W_{KQ} which we denote by β. After resealing both operations i.e. $W_{KQ} \leftarrow W_{KQ}/\beta$ and $W_{PV} \leftarrow W_{PV}\beta$ we interpolate linearly between the matrix products of GD as well as these rescaled trained matrix products i.e. $W_{I,KQ} = (W_{GD,KQ} + W_{TF,KQ})/2$ as well as $W_{I,PV} = (W_{GD,PV} + W_{TF,PV})/2$. We use these parameters to obtain results throughout the chapter denote with *Interpolated*. We do so for GD as well as GD^{++} when comparing to recurrent Transformers. Note that for non-recurrent Transformers, we face more ambiguity that we have to correct for since e.g. scalings influence each other across layer. We also see this in practice and are not able (only for some seeds) to interpolate between weights with our simple correction from above. We leave the search for more elaborate corrections for future work.

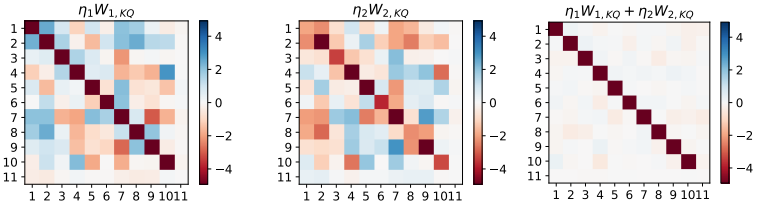


FIGURE 2.10: **Visualizing the correction to the softmax operation when training Transformers on regression tasks.** The left and center plot show the matrix product $W_{KQ} = W_K^T W_Q$ including its scaling by η induced through PW_V of the two heads of the trained softmax self-attention layer. We observe that both of the matrices are approximate diagonal almost perfect sign reversed values on the off-diagonal terms. After adding the matrices (right plot), we observe a diagonal matrix and therefore to much improved approximation of our construction and therefore gradient descent dynamics.

2.6.4 *Linear vs. softmax self-attention as well LayerNorm Transformers*

Although linear Transformers and their variants have been shown to be competitive with their softmax counterpart [99], the removal of this nonlinearity is still a major departure from classic Transformers and more importantly from the Transformers used in related studies analyzing in-context learning. In this section we investigate whether and when gradient-based learning emerges in trained softmax self-attention layers, and we provide an analytical argument to back our findings.

First, we show, see Figure 2.11, that a single layer of softmax self-attention is not able to match GD performance. We tuned the learning rate as well as the weight initialization but found no significant difference over the hyperparameters we used througout this study. In general, we hypothesize that GD is an optimal update given the limited capacity of a single layer of

(single-head) self-attention. We therefore argue that the softmax induces (at best) a linear offset of the matrix product of training data and query vector

$$\text{softmax}(K^T q_j) = (e^{k_1^T q_j}, \dots, e^{k_N^T q_j})^T / (\sum_i e^{k_i^T q_j}) \tag{2.28}$$

$$= (e^{x_1^T W_{KQ} x_j}, \dots, e^{x_N^T W_{KQ} x_j})^T / (\sum_i e^{x_i^T W_{KQ} x_j}) \tag{2.29}$$

$$\approx (1 + x_1^T W_{KQ} x_j, \dots, 1 + x_N^T W_{KQ} x_j)^T / (\sum_i 1 + x_i^T W_{KQ} x_j) \tag{2.30}$$

$$\propto K^T q_j + \epsilon \tag{2.31}$$

proportional to a factor dependent on all $\{x_{\tau,i}\}_{i=1}^{N+1}$. We speculate that the dependency on the specific task τ, for large N_x vanishes or that the x-dependent value matrix could introduce a correcting effect. In this case the softmax operation introduces an additive error w.r.t. to the optimal GD update. To overcome this disadvantageous offset, the Transformer can (approximately) introduce a correction with a second self-attention head by a simple subtraction i.e.

$$P_1 V_1 \text{softmax}(K_1^T W_Q x_j) + P_2 V_2 \text{softmax}(K_2^T W_Q x_j) \tag{2.32}$$

$$\approx PV((1 + x_1^T W_{1,KQ} x_j, \dots, 1 + x_N^T W_{1,KQ} x_j) \tag{2.33}$$

$$- (1 + x_1^T W_{2,KQ} x_j, \dots, 1 + x_N^T W_{2,KQ} x_j)) \tag{2.34}$$

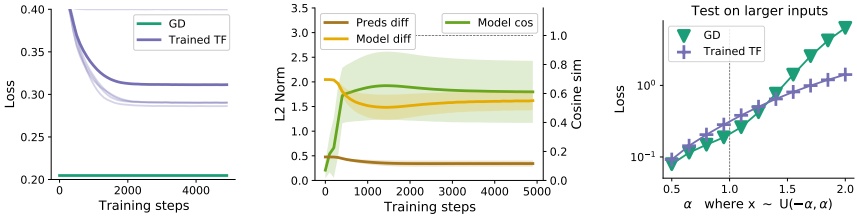
$$= PV(x_1^T (W_{1,KQ} - W_{2,KQ}) x_j, \dots, x_N^T (W_{1,KQ} - W_{2,KQ}) x_j) \tag{2.35}$$

$$\propto PVK^T q_j. \tag{2.36}$$

Here we assume that PV 1) subsumes the dividing factor of the softmax and that 2) is the same (up to scaling) for each head. Note that if $(W_{1,KQ} - W_{2,KQ})$ is diagonal, and W_P and W_V chosen as in the Proposition 1, we recover our gradient descent construction.

We base this derivation on empirical findings, see Figure 2.11, that, first of all, show the softmax self-attention performance increases drastically when using two heads instead of one. Nevertheless, the self-attention layer has difficulties to match the loss values of a model trained with GD. Furthermore, this architecture change leads to a very much improved alignment of the trained model and GD. Second, we can observe that when training a two-headed softmax self-attention layer on regression tasks the correction proposed above is actually observed in weight space, see Figure 2.10. Here, we visualize the matrix product within the softmax operation

(a) *Softmax one*-headed self-attention layer.



(b) *Softmax two*-headed self-attention layer.

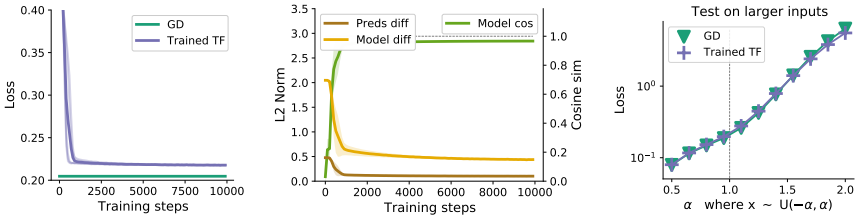


FIGURE 2.11: **Comparing trained two-headed and one-headed single-layer *softmax* self-attention with 1 step of gradient descent on linear regression tasks.** *Left column*: Softmax self-attention is not able to match gradient descent performance with hand-tuned learning rate, but adding a second attention head significantly reduces the gap, as expected by our analytical argument. *Center column*: The alignment suffers significantly for single-head softmax SA. We observe good but not as precise alignment when compared to linear Transformers for the two-headed softmax SA layer. *Right column*: The two-headed self-attention compared to the single-head layer shows similar robust out-of-distribution behavior compared to gradient descent.

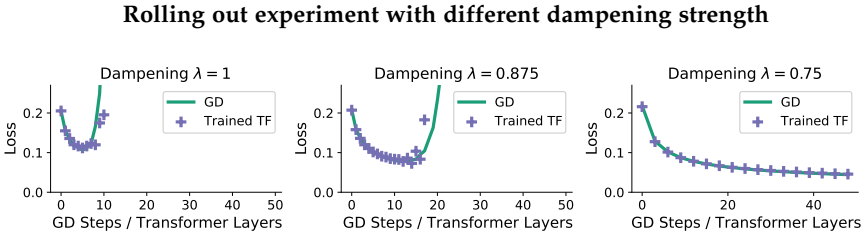


FIGURE 2.12: **Roll-out experiments: applying a trained single linear self-attention layer multiple times.** We observe that different dampening strengths affect the generalization of both methods with slightly better robustness for GD which matching performance for 50 steps when $\lambda = 0.75$.

$W_{h,KQ}$ per head which we scale with the last diagonal entry of $P_h W_{h,V}$ which we denote by $\eta_h = P_h W_{h,V}(-1,-1)$. Intriguingly, this results in an almost perfect cancellation (right plot) of the off-diagonal terms and therefore in sum to an improved approximation of our construction, see the derivation above.

We would like to reiterate that the stronger inductive bias for copying data of the softmax layer remains, and is not invalidated by the analysis above. Therefore, even for our shallow and simple constructions they indeed fulfill an important role in support for our hypotheses: The ability to merge or copy input and target data into single tokens allowing for their dot product computation necessary for the construction in Proposition 1, see Section 2.4 in the main text.

We conclude that common architecture choices like softmax (and LayerNorm, see full paper online) seem sup-optimal for the constructed in-context learning settings when comparing to GD or linear self-attention. Nevertheless, we speculate that the potentially small performance drops of in-context learning are negligible when turning to deep and wide Transformers for which these architecture choices have empirically proven to be superior.

2.6.5 *Dampening the self-attention layer*

As an additional out-of-distribution experiment, we test the behavior when repeating a single LSA-layer trained to lower our objective, see equation 2.9, with the aim to repeat the learned learning/update rule. Note that GD as

well as the self-attention layer were optimized to be optimal for one step. For GD we line search the optimal learning rate η on 10.000 task. Interestingly, for both methods we observe quick divergence when applied multiple times, see left plot of Figure 2.12. Nevertheless, both of our update functions are described by a linear self-attention layer for which we can control the norm, post training, by a simple scale which we denote as λ. This results in the new update $y_{\text{test}} + \lambda \Delta W x_{\text{test}}$ for GD and $y_{\text{test}} + \lambda P V K^T W_Q x_{\text{test}}$ for the trained self-attention layer which effectively re-tunes the learning rate for GD and the trained self-attention layer. Intriguingly, both methods do generalize similarly well (or poorly) on this out-of-distribution experiment when changing λ, see again Figure 2.12. We show in Figure 2.1 the behavior for $\lambda = 0.75$ for which we see both methods steadily decreasing the loss within 50 steps.

3

UNCOVERING MESA-OPTIMIZATION ALGORITHMS IN TRANSFORMERS

This chapter's content can be found online in an extended form including all experimental details which we omit here for clarity. The original publication is authored by Johannes von Oswald, Eyvind Niklasson*, Maximilian Schlegel*, Seijin Kobayashi, Nicolas Zucchet, Nino Scherrer, Nolan Miller, Mark Sandler, Blaise Agüera y Arcas, Max Vladymyrov, Razvan Pascanu, João Sacramento [100].*

** These authors contributed equally.*

Transformers have become the dominant model in deep learning, but the reason for their superior performance is poorly understood. Here, we hypothesize that the strong performance of Transformers stems from an architectural bias towards mesa-optimization, a learned process running within the forward pass of a model consisting of the following two steps: (*i*) the construction of an internal learning objective, and (*ii*) its corresponding solution found through optimization. To test this hypothesis, we reverse-engineer a series of autoregressive Transformers trained on simple sequence modeling tasks, uncovering underlying gradient-based mesa-optimization algorithms driving the generation of predictions. Moreover, we show that the learned forward-pass optimization algorithm can be immediately repurposed to solve supervised few-shot tasks, suggesting that mesa-optimization might underlie the in-context learning capabilities of large language models. Finally, we propose a novel self-attention layer, the mesa-layer, that explicitly and efficiently solves optimization problems specified in context. We find that this layer can lead to improved performance in synthetic and preliminary language modeling experiments, adding weight to our hypothesis that mesa-optimization is an important operation hidden within the weights of trained Transformers.

3.1 INTRODUCTION

Transformers [101] and especially large language models (LLMs) are known to strongly adjust their predictions and learn based on data given in-context [102]. Recently, a number of works have studied this phenomenon in detail by meta-learning Transformers to solve few-shot tasks, providing labeled

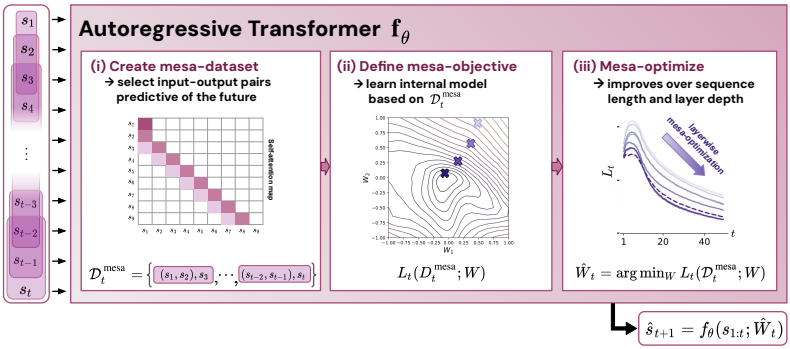


FIGURE 3.1: **Illustration of our hypothesis:** Optimizing the weights θ of an autoregressive Transformer f_θ gives rise to mesa-optimization algorithms implemented in the forward pass of the model. As a sequence of inputs $s_1, \ldots, s_t$ is processed up to timestep t, the Transformer (*i*) creates an internal training set consisting of pairs of input-target associations, (*ii*) defines an internal objective function through the resulting dataset, used to measure the performance of an internal model with weights W, (*iii*) optimizes this objective and uses the learned model to generate a prediction $\hat{s}_{t+1}$ of the future.

training sets in context. These studies discovered that Transformers implement learning algorithms that either closely resemble or exactly correspond to gradient-based optimizers [38–42, 48, 103, 104].

However, it remains unclear how well these findings on meta-trained Transformers translate to models that are *autoregressively-trained* on sequential data, the prevalent LLM training setup. Here, we address this question by building on the theoretical construction of von Oswald *et al.* [48], and show how Transformers trained on sequence modeling tasks predict using gradient-descent learning based on in-context data. Thus, we demonstrate that minimizing a generic autoregressive loss gives rise to a subsidiary gradient-based optimization algorithm running inside the forward pass of a Transformer. This phenomenon has been recently termed mesa-optimization [105]. Moreover, we find that the resulting mesa-optimization algorithms exhibit in-context few-shot learning capabilities, independently of model scale. Our results therefore complement previous reports characterizing the emergence of few-shot learning in large-scale LLMs [102, 106].

Our contributions are as follows:

- We generalize the construction introduced in the previous chapter [48] and show how, in theory, Transformers can autoregressively predict the next element of a sequence by optimizing internally-constructed objectives with gradient-based methods.
- Experimentally, we reverse-engineer Transformers trained on simple sequence modeling tasks, and find strong evidence that their forward pass implements two-step algorithms: (*i*) early self-attention layers construct internal training datasets by grouping and copying tokens, and therefore implicitly define internal objective functions, (*ii*) deeper layers optimize these objectives to generate predictions.
- Similarly to LLMs, we show that these simple autoregressively-trained models become in-context learners, and that prompt-tuning, crucial to improve in-context learning in LLMs, also improves performance in our setting.
- Motivated by our findings that attention layers are attempting to implicitly optimize internal objective functions, we introduce the *mesa-layer*, a novel attention layer that efficiently solves a least-squares optimization problem, instead of taking just a single gradient step towards an optimum. We show that a single mesa-layer outperforms deep linear and softmax self-attention Transformers on simple sequential tasks while offering more interpretability.
- We carry out preliminary language modeling experiments replacing standard self-attention layers with the mesa-layer, and obtain promising results demonstrating strong in-context learning capabilities enabled by the layer.

3.2 PRELIMINARIES

SELF-ATTENTION. We study causally-masked, autoregressive Transformers [101] where self-attention [107] is the elementary building block. Given a sequence of t input tokens $E_t = (e_{t'})_{t'=1}^{t}$, representing the first t time steps, a self-attention layer with H heads and parameters θ updates the current token $e_t \in \mathbb{R}^{D_e}$ as follows:

$$\Delta e_t^{\text{softmax}}(E_t, \theta) = \sum_{h=1}^{H} P_h V_{h,t} \,\text{softmax}(K_{h,t}^{\top} q_{h,t}), \tag{3.1}$$

where $q_{h,t} = W_{h,q} e_t \in \mathbb{R}^{D_a}$ is referred to as a query, each column $k_{h,t'} = W_{h,k} e_{t'} \in \mathbb{R}^{D_a}$ of matrix $K_{h,t} \in \mathbb{R}^{D_a \times t}$ as a key, and each column $v_{h,t'} = W_{h,v} e_{t'} \in \mathbb{R}^{D_v}$ of matrix $V_{h,t} \in \mathbb{R}^{D_v \times t}$ as a value. The nonlinear function $\text{softmax}(a)$ applied to vector $a \in \mathbb{R}^t$ returns an attention vector with entries $[\text{softmax}(a)]_i = \frac{\exp(a_i)}{\sum_{t'=1}^{t} \exp(a_{t'})}$. We absorb bias terms and assume here for conciseness that all heads are equally sized. The parameters θ of this layer are the projection matrices $\{(P_h, W_{h,q}, W_{h,k}, W_{h,v})\}_{h=1}^{H}$ for all heads. Transformers include other layers that we do not review here, notably multi-layer perceptrons (MLPs) and layer normalization (LayerNorm) units.

We also consider linear attention models [e.g., 108–111], which simply omit the softmax nonlinearity:

$$\Delta e_t^{\text{linear}}(E_t, \theta) = \sum_{h=1}^{H} P_h V_{h,t} K_{h,t}^{\top} q_{h,t} = \sum_{h=1}^{H} P_h \hat{W}_{h,t}^{\text{linear}} q_{h,t}. \tag{3.2}$$

Above, we rewrite this equation using a weight matrix $\hat{W}_{h,t}^{\text{linear}} = \sum_{t'=1}^{t} v_{h,t'} k_{h,t'}^{\top}$. The size of this weight matrix does not scale with time, but it encodes information from all past tokens $(e_{t'})_{t'=1}^{t}$, allowing inference at constant memory cost. For this reason, there is at present considerable interest in linear attention [112, 113].

LINEAR SELF-ATTENTION CAN IMPLEMENT ONE STEP OF GRADIENT DESCENT. Our starting point is the main result of von Oswald *et al.* [48], who showed that one such attention layer can implement one step of gradient descent (GD) on a quadratic cost function evaluated on in-context data. Therefore, multi-layer Transformers can, in theory, minimize the loss down to an arbitrary desired level through multiple steps of GD. In this chapter, we extend this result to the autoregressive setting. First, we review the original model and task setting.

In the setup of the previous chapter i.e. as in von Oswald *et al.* [48], the goal is to meta-learn the parameters θ of a linear self-attention layer such that it learns to solve supervised learning tasks, similarly to related work [38–42, 103, 104]. Each task τ is specified in-context by a training set $\mathcal{D}_\tau = \{(x_{\tau,i}, y_{\tau,i})\}_{i=1}^{N}$ and a test input $x_{\tau,\text{test}}$. The goal of meta-learning is then $\min_\theta \mathbb{E}_\tau \left[\| y_{\tau,\text{test}} - f(x_{\tau,\text{test}}, \mathcal{D}_\tau, \theta) \|^2 \right]$, where $y_{\tau,\text{test}}$ is the correct output revealed during meta-learning,
$f(x_{\tau,\text{test}}, \mathcal{D}_\tau, \theta)$ denotes the actual output of the linear self-attention layer, and the expectation is taken over a distribution of linear regression tasks.

A standard approach for solving a linear regression task is to resort to a linear model $f_W(x) = Wx$ with parameters $W \in \mathbb{R}^{D_y \times D_x}$ learned by gradient descent on the squared error loss

$$L(W, \mathcal{D}_\tau) = \sum_{i=1}^{N} \frac{1}{2} \|y_{\tau,i} - f_W(x_{\tau,i})\|^2 \tag{3.3}$$

. Starting from an initial parameter W_0, a gradient-descent learner updates it by taking a step ΔW_0 of size η along the negative of the gradient, $\nabla L = \sum_{i=1}^{N} (y_{\tau,i} - W_0 x_{\tau,i}) x_{\tau,i}^\top$. The main result of von Oswald *et al.* [48] is a theoretical construction showing that a linear self-attention layer can implement exactly one such gradient descent step. We briefly sketch this result now.

First, we construct a set of tokens E_T, with $T = N$, such that $e_t = (y_{\tau,i}, x_{\tau,i})$, with $y_{\tau,i}$ and $x_{\tau,i}$ concatenated. Additionally, we create a query token $e_{T+1} = (-W_0 x_{\tau,\text{test}}, x_{\tau,\text{test}})$ not contained within the set $\mathcal{D}_\tau$, where we place the test input for which a prediction should be made. Under this token construction and using the symbol I_x to denote the identity matrix of size $\dim(x)$, if all bias terms are zero and $W_k^\top W_q = \begin{pmatrix} 0 & 0 \\ 0 & I_x \end{pmatrix}$, and $PW_v = \begin{pmatrix} -\eta I_y & \eta W_0 \\ 0 & 0 \end{pmatrix}$, the query token e_{T+1}, after one such layer, becomes $(-(W_0 + \Delta W_0) x_{\tau,\text{test}}, x_{\tau,\text{test}})$. The y-component of this token contains the (negative) of the prediction obtained by a linear model that underwent one step (ΔW_0) of gradient descent. Therefore, this self-attention layer implicitly constructs a least-squares optimization problem and takes one step of *mesa-gradient descent* towards solving it. This layer can be directly stacked to implement multiple steps of GD, cf. Appendix 3.7.3.2. The term *mesa* reinforces that this optimization occurs within the forward attention dynamics, without any actual change to the parameters of the attention layer itself [105]. We stress the necessary assumption of having $x_{\tau,i}$ and $y_{\tau,i}$ concatenated within a single token.

3.3 SEQUENTIAL PREDICTION BY LEAST-SQUARES MESA-OPTIMIZATION

The construction reviewed above is designed to solve few-shot supervised learning problems. As we see next, moving to a general autoregressive modeling setting requires minimal change. However, the spirit of what follows is markedly different: we no longer ask whether an attention layer

can solve few-shot supervised learning problems that are presented in-context. Instead, we ask whether Transformers can rely on mesa-gradient descent to predict future inputs.

We therefore move to the case where a self-attention layer has to learn sequentially as some inputs $s_{1:T}$ are gradually unveiled. The goal at time t is now to minimize the autoregressive loss:

$$L_t(W) = \sum_{t'=1}^{t-1} \frac{1}{2} \|s_{t'+1} - W s_{t'}\|^2, \tag{3.4}$$

where $s_{t'+1}$ serves as the label for $s_{t'}$. As in the previous section, we assume that the model always starts from the same initial weights W_0, and that learning corresponds to taking only a single gradient step; this appears sub-optimal. We address this concern in the next section.

As is usually done in autoregressive modeling we apply causal masking, and at time t we update token e_t using the in-context data available in E_t. To adapt to the autoregressive setting, we adapt the token construction to a three-channel code, $e_t = (-W_0 s_t, s_t, s_{t-1})$, to include an additional separate first channel to be filled with the prediction $\hat{s}_{t+1}$ of future inputs at every time step t, alongside channels for the previous and current sequence element, with the latter playing the role of target in the construction of von Oswald *et al.* [48]. Note that by providing neighboring elements s_t, s_{t-1} within one token e_t, self-attention is able to compute dot products of *targets* and *inputs* of the loss $L_t(W)$ necessary to compute ∇L_t, see Eq. 3.4. Then, to update the first channel of such a token with the prediction of a linear model learned with one step of gradient descent, it suffices to set

$$PW_v = \begin{pmatrix} 0 & -\eta I_s & \eta W_0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \text{and} \quad W_k^\top W_q = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & I_s & 0 \end{pmatrix}. \tag{3.5}$$

We refer to this result (Eq. 3.5) as the one-step mesa-gradient descent construction.

MULTI-LAYER MESA-OPTIMIZERS. We next move to the case of deep networks comprising stacked linear self-attention layers. While it is natural to hypothesize that K layers simply implement K steps of mesa-gradient descent, as in the few-shot learning (non-autoregressive) case reviewed above, this picture might be too simple to explain actual trained autoregressive Transformers. A first hint towards this view being too narrow lies

in the fact that stacking the one-step mesa-gradient descent construction (Eq. 3.5) over multiple layers does not yield vanilla gradient descent, as explained in Appendix 3.7.3.2. Instead, we obtain an unconventional online gradient-based optimizer, that is expected to behave worse than vanilla gradient descent. This observation, together with a mathematical analysis of the resulting optimization algorithm, can be found in a study arguing for the disadvantages of causally-masked attention for few-shot in-context learning [114]. One may thus wonder if Transformers can implement more efficient mesa-optimizers.

Here, we provide an alternative mesa-optimizer that is also based on causally-masked self-attention layers. The novel optimizer operates in two stages. In a first stage, comprising one or more self-attention layers, the algorithm implements an iterative preconditioning procedure. The result of this stage is a regularized mesa-objective $\bar{L}_t(W) = \sum_{t'=1}^{t-1} \frac{1}{2}\|s_{t'+1} - WH_t s_{t'}\|^2 + \frac{1}{2\lambda}||W||_{\mathrm{F}}^2$, with improved condition number compared to $L_t(W)$. Above, H_t is a preconditioning matrix and the scalar $\lambda^{-1} \geq 0$ controls the regularization strength. This preconditioning procedure has the property that in the many-layer limit and under some mild conditions, H_t converges to $H_t^* = (S_{t-1}S_{t-1}^\top + 1/\lambda I)^{-1}$, with S_t the data matrix whose columns are $(s_{t'})_{t'=1}^t$. In a second stage, a final self-attention layer takes a single gradient descent step on the preconditioned mesa-objective $\bar{L}_t(W)$.

The two-stage algorithm described here is theoretically justified: when $H_t = H_t^*$, the regression problem is solved in a single step, starting from a zero-weight initialization $W_0 = 0$. In Appendix 3.7.3.2, we provide a simple weight and input token construction to implement this algorithm. Our novel construction leverages the truncated Neumann series to iteratively approximate the required inverse-matrix-vector products $H_{t-1}^* s_t$ in parallel for all $t = 2, \ldots, T$, and compactly, without ever explicitly representing any of the H_t matrices.

In Section 3.5 we show empirically that training a Transformer on autoregressive tasks can lead to the solutions presented above. But first, in the next section, we assume that mesa-optimization is a desirable feature for a model to have, and we discuss an architectural modification that makes this behavior built-in by default within a Transformer.

3.4 AN ATTENTION LAYER FOR OPTIMAL LEAST-SQUARES LEARNING

Here we introduce the *mesa-layer*: a novel self-attention layer that fully solves a layer-specific optimization problem, such as the minimization of

Eq. 3.4, instead of only descending a loss function with a single gradient step. The layer we propose is closely related to the Delta-Net model of Schlag, Irie & Schmidhuber [110], which is hardwired to do one gradient descent step per time point. We focus on causally-masked autoregressive problems, while noting that the insights remain the same for other strategies such as BERT-style masking [115].

Given again a sequence of tokens E_t, we design a layer that changes the tokens following the update

$$\Delta e_t^{\text{mesa}}(E_t, \theta) = \sum_{h=1}^{H} P_h \hat{W}_{h,t}^{\text{mesa}} q_{h,t}, \tag{3.6}$$

$$\text{with} \qquad \hat{W}_{h,t}^{\text{mesa}} = \arg\min_{W} \left\{ \frac{1}{2} \sum_{t'=1}^{t} ||v_{h,t'} - W k_{h,t'}||^2 + \frac{1}{2\lambda_h} ||W||_{\text{F}}^2 \right\}. \tag{3.7}$$

Above, the scalar $\lambda_h^{-1} > 0$ controls the strength of a regularizer added to improve generalization, and key, value and query vectors are the usual learned head-specific affine transformations of the tokens, as before. However, through Eq. 3.7 these vectors are now assigned a precise, interpretable role: value vectors specify targets to which an internal model with parameters W should map training and test inputs, represented by keys and queries, respectively. The minimizer of a regularized version of Eq. 3.4 can be immediately mapped to Eq. 3.7 under the token construction discussed in Section 3.3 by appropriately setting the projection matrices $W_{h,v}$, $W_{h,k}$ and $W_{h,q}$.

At any given time step $t = 1, \ldots, T$ computing Δe_t^{mesa} requires solving a regularized least squares problem per head. To efficiently solve this sequence of T optimization problems, we will leverage the recursive dependency of the solutions of these consecutive problems which can be expressed in closed-form as

$$\hat{W}_{h,t}^{\text{mesa}} = V_{h,t} K_{h,t}^{\top} R_{h,t} = \sum_{t'=1}^{t} v_{h,t'} k_{h,t'}^{\top} \left(\sum_{t'=1}^{t} k_{h,t'} k_{h,t'}^{\top} + 1/\lambda_h \, I \right)^{-1}. \tag{3.8}$$

Note that if we drop the inverted matrix $R_{h,t}$, we recover a standard linear self-attention layer, cf. Eq. 3.2. A recent study has also shown that the solution of a least-squares problem can be expressed as a generalized attention layer [116].

We now use the Sherman & Morrison [117] formula to obtain the inverse at time t from the inverse at the previous time step $t-1$. This iterative

update is possible because we only change the inverse by a rank-one update. This solution scheme is known as recursive least squares [118]. We obtain through Sherman-Morrison the recursion

$$R_{h,t} = R_{h,t-1} - \frac{R_{h,t-1} k_{h,t} k_{h,t}^\top R_{h,t-1}}{1 + k_{h,t}^\top R_{h,t-1} k_{h,t}} \tag{3.9}$$

with $R_{h,0} = \lambda_h I$. With this, we can (causally in time) compute

$$\Delta e_t^{\text{mesa}}(E_t, \theta) = \sum_{h=1}^{H} P_h V_{h,t} K_{h,t}^\top R_{h,t} q_{h,t} \tag{3.10}$$

which requires 2 additional vector-matrix and 2 vector-vector multiplications per step compared to the standard self-attention operation. Note that since our intermediates consist of matrices of dimension $D_a \times D_a$ across the timesteps, naive backward gradient computation requires storing them in memory. Fortunately, this memory overhead can be avoided using the Sherman-Morrison formula in reverse during the backward pass, cf. Appendix 3.7.2.1, enabling memory-efficient gradient computation of the output of the mesa-layer w.r.t. its inputs. We further note that while the implementation described here has a desirable $\mathcal{O}(1)$ inference memory cost like standard linear self-attention, it is not parallelizable across time during training. This is a disadvantage for training on contemporary hardware shared with recurrent neural networks, but not with standard softmax or linear self-attention. As discussed in Appendix 3.7.2.1, in practice this significantly slows down our experiments.

We demonstrate the expressivity and performance of the mesa-layer in reverse-engineerable sequence learning tasks as well as in language modeling in the next sections.

3.5 EMPIRICAL ANALYSIS

3.5.1 *Prediction of linear dynamics by in-context learning*

We now attempt to reverse-engineer Transformers trained on simple synthetic autoregressive tasks. We have two main goals. First, we want to understand whether autoregressively-trained Transformers use mesa-optimization algorithms to predict future inputs. We use the constructions presented in Section 3.3 to guide our reverse-engineering analyses. Our second goal is to determine if introducing the mesa-layer improves the performance

of standard Transformers, by subsuming multiple attention layers that are otherwise needed to go beyond one mesa-gradient descent step.

GENERATIVE MODEL. We focus on fully-observed linear dynamical systems. For all experiments described in this section, we use the following generative model. To create a sequence $s_{1:T}$ we first draw a random groundtruth $D_s \times D_s$ weight matrix W^* as well as a random initial state $s_1 \sim \mathcal{N}(0, I_s)$; subsequent states for $t = 2, \ldots, T$ are then generated according to the rule $s_{t+1} = W^* s_t + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, \sigma_s^2 I_s)$ introduces Gaussian noise. We take W^* to be a random orthogonal matrix[1]. The generation of W^* anew for each sequence avoids the memorization solution that stores W^* in θ, and corresponds to a highly simplified toy model meant to capture the diversity present in real-world data. A similar in spirit design choice may be found in the hierarchical generative model of Xie *et al.* [119]. Under such an assumed groundtruth dynamics, the standard way of predicting future states from a given past sequence $s_{1:T}$ is to use a linear model, $s_{t+1} = W s_t$, where the weights W are learned by minimizing $L_t(W)$, Eq. 3.4, possibly with an added regularizer.

TRAINING AND IN-CONTEXT LEARNING OBJECTIVES. Here, we analyze various configurations of Transformers trained through stochastic online minimization of the autoregressive loss

$$\mathcal{L}(\theta) = \mathbb{E}_s \left[\sum_{t=1}^{T-1} \mathcal{L}_t(s_{1:t}, \theta) \right] = \mathbb{E}_s \left[\frac{1}{2} \sum_{t=1}^{T-1} \| s_{t+1} - f_t(s_{1:t}, \theta) \|^2 \right], \qquad (3.11)$$

where the expectation is taken under the sequence distribution described above, $f_t(s_{1:t}, \theta)$ denotes the output of the Transformer model using s_t as query and $s_{1:t}$ as context, and θ are the Transformer parameters, which vary depending on the exact architecture being trained. To avoid confusion with mesa-optimization, we refer to the minimization of $\mathcal{L}(\theta)$ as the base-optimization process.

Here and throughout, to measure in-context learning performance we take the per-timestep loss $\mathcal{L}_t(s_{1:t}, \theta)$ and monitor its evolution as a function of context size t. Thus, we simply measure how future-input predictions improve as more context is provided to the model. This corresponds to the operational definition of in-context learning proposed by Kaplan *et al.* [106].

1 This detail turns out to be important; we found that converging linear dynamics led to different inference algorithms.

HYPOTHESIS STATEMENT. The hypothesis we pursue is that base-optimization of $\mathcal{L}(\theta)$ gives rise to a mesa-optimization process in charge of generating predictions $f_t(s_{1:t}, \theta)$, as illustrated in Figure 3.2A. More concretely, for our linear generative model, we hypothesize that learning yields Transformers that predict future inputs by implicitly, and entirely within their forward dynamics: (*i*) representing a linear model with mesa-parameters W, (*ii*) constructing the least-squares mesa-objective $L_t(W)$, cf. Eq. 3.4, using in-context data $s_{1:t}$, (*iii*) learning W by minimizing the mesa-objective, and (*iv*) applying W to predict the next token s_{t+1}. We note that, according to our hypothesis, the mesa-objective $L_t(W)$ governing the forward pass of our Transformer coincides with the base-objective $\mathcal{L}(\theta)$, but now defined w.r.t. an implicit linear autoregressive model with mesa-parameters W.

SINGLE SELF-ATTENTION LAYER. We begin by verifying our hypothesis on single-layer, linear-attention-only Transformers, using the token construction of Section 3.3, $e_t = (0, s_t, s_{t-1})$. We hypothesize that feeding the Transformer with input-target pairs provides an inductive bias towards mesa-gradient descent. Using this token construction, we then train by online mini-batch gradient descent on $\mathcal{L}(\theta)$, generating new sequences at each base optimization step according to the process described above.

We are able to perfectly identify the algorithm (RevAlg-1) that this single-layer Transformer uses to generate predictions. Visual inspection of the projection matrices is revealing, cf. Figure 3.10: we see that the dominant pattern coincides with our one-step mesa-gradient descent construction, Eq. 3.5, plus some identification noise. We verify quantitatively that the layer is indeed implementing a step of mesa-gradient descent by (*i*) comparing the loss reached by the trained layer with a linear autoregressive model learned through one step of gradient descent, and by (*ii*) studying an interpolated model, obtained by averaging directly in parameter space learned and constructed weights. We find that we can perfectly fit our trained layer when using all degrees of freedom in our construction, including not only a learned learning rate η, but also a learned set of initial weights W_0, reminiscent of the model-agnostic meta-learning method of Finn, Abbeel & Levine [67].

Importantly, as shown in Figure 3.2, the resulting learned one-step algorithm is still vastly outperformed by a single mesa-layer. We note that under a simple setting of its weights, easily discovered by base-optimization, this

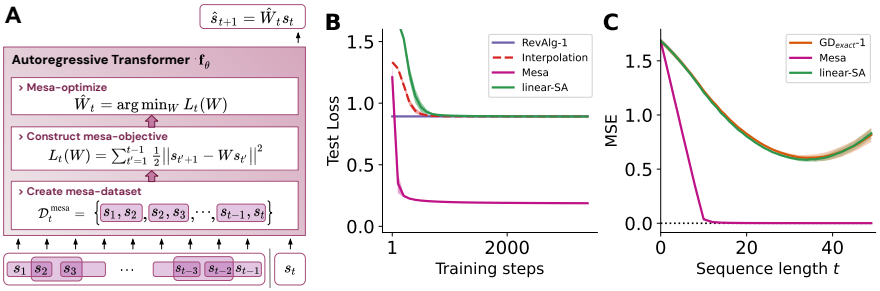


FIGURE 3.2: **Reverse-engineering a trained linear self-attention layer.** (**A**) Transformers mesa-optimize an internal linear model and use it to predict the future state of a linear dynamical system. (**B**) A trained 2-head linear self-attention layer (linear-SA) is perfectly described by a reverse-engineered mesa-gradient descent algorithm (RevAlg-1; see Eq. 3.56). We show also the performance achieved by an interpolation model, obtained by averaging the parameters θ of the trained model and those expected from our reverse-engineered construction. (**C**) In-context learning loss after training: next-input s_{t+1} mean squared prediction error (MSE) as a function of sequence length. The trained linear-SA layer is very well described by a linear model learned by one step of gradient descent with a tuned learning rate (GD_{exact}-1). Linear-SA is greatly outperformed by a single mesa-layer, which optimally solves the autoregressive learning problem at every time point t, reaching minimal mean-squared prediction error after observing enough examples. By contrast, one-step GD runs into capacity issues, exhibiting non-monotonic MSE as a function of sequence length. Averages over 5 different seeds; shaded area represents standard deviation.

layer can optimally solve the task studied here. This result demonstrates the advantage of hardcoded inductive biases in favor of mesa-optimization.

MULTIPLE SELF-ATTENTION LAYERS. Armed with our theoretical insights for the multi-layer case, cf. Section 3.3, we now analyze deep linear and softmax attention-only Transformers. We format our inputs according to a 4-channel construction, $e_t = (0, s_t, s_t, s_{t-1})$, which corresponds to choosing $W_0 = 0$. This makes it possible to implement both multi-step mesa-optimization and our iterative preconditioning algorithm, as well as hybrid variants mixing both, as discussed in Appendix section 3.7.3.2.

Like with single-layer models, we see clean structure in the weights of the trained models, see Figures 3.12 and 3.11. As a first reverse-engineering analysis, we exploit this structure and construct an algorithm (RevAlg-d, where d denotes layer number) comprising 16 parameters (instead of 3200) per layer head. We find that this compressed, albeit convoluted, expression can describe a trained model. In particular, it allows interpolating between actual Transformer and RevAlg-d weights in an almost lossless fashion, cf. Figure 3.3A.

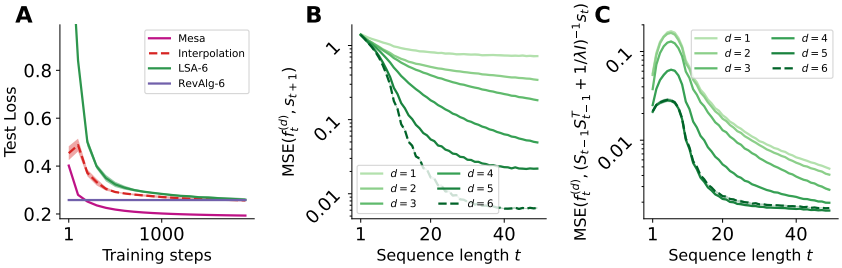


FIGURE 3.3: **Reverse-engineering multi-layer Transformers trained on constructed token inputs.** We report results for a 6-layer linear-self-attention-only Transformer. (**A**) As training proceeds, this multi-layer linear model (LSA-6) is again perfectly described by a reverse-engineered algorithm (RevAlg-6), described in Appendix 3.7.3. Note that the model is still outperformed by a single trained mesa-layer. (**B** & **C**) We linearly regress the activations of each layer against (**B**) final targets (target probing) over the d layers as well as (**C**) the preconditioned inputs $(S_{t-1}S_{t-1}^\top + 1/\lambda I)^{-1}s_t$ predicted by our theory (inverse probing), observing an improvement in linear decoding performance across layers. Averages computed over 5 different seeds; shaded area represents standard deviation.

While the RevAlg-d expression explains a trained multi-layer Transformer with a small number of free parameters, it is difficult to interpret it as a mesa-optimization algorithm. We, therefore, resort to a linear regression probing analysis [103, 120] to look for signatures of our hypothesized mesa-optimization algorithms. In particular, we seek evidence both for the stacked multi-layer gradient descent construction, which should bring the outputs of intermediate layers closer to the desired targets; and for our novel iterative preconditioning algorithm, which should bring layer outputs closer to $H_t^* s_t$. We therefore carry out our probing analysis taking as targets for regression (i) the future state to be predicted s_{t+1} used as the

target to train the Transformer, which we term the *target probe*; and (*ii*) the preconditioned current input, $(S_{t-1}S_{t-1}^\top + 1/\lambda I)^{-1}s_t$, which we term the *inverse probe*, and that would allow for solving the least-squares problem in a single gradient descent step as discussed above.

As shown in Figure 3.3 for deep self-attention Transformers we see that *both* probes can be linearly decoded, with decoding performance increasing with sequence length and network depth. Base-optimization has therefore discovered a hybrid algorithm that descends over layers the original mesa-objective $L_t(W)$ while simultaneously improving the condition number of the mesa-optimization problem. This leads to a fast descent of the mesa-objective $L_t(W)$, Eq. 3.4. Moreover, we find that performance strongly improves with depth, cf. Figure 3.3, with a 6-layer model coming close to but still not matching a single mesa-layer.

Our probing analysis results therefore support our hypothesis that a fast descent on the autoregressive mesa-objective $L_t(W)$ is achieved through mesa-optimization on progressively (across layers) better preconditioned data. We point to Figures 3.13 for an additional confirmation of this effect, showing that when taking regressed inverse probes as inputs to a linear model (instead of raw inputs s_t), the performance of single-step learning significantly improves.

FULL-FLEDGED TRANSFORMERS. To finish our synthetic data experiments, we relax all previous architectural simplifications and turn to training standard Transformers that use positional encodings, input and output projections, and which need to process raw tokens $e_t = s_t$. We hypothesize that after autoregressive training these models operate in two stages. In a first stage, they use positional information to re-create our token construction in the first softmax self-attention layer through a copying mechanism, essentially identical to first stage of the induction heads discovered by Olsson *et al.* [121]. This effectively corresponds to an internal *specification* of a mesa-optimization problem. Since the states are Markovian, i.e. only depend (linearly) on the immediate previous state, a simple next-token copying mechanism suffices in our toy model. The second part of our hypothesis is that subsequent layers implement a mesa-optimizer that solves the self-constructed least-squares problem. For this second part, we again use our two candidate constructions – mesa-gradient descent steps and iterative preconditioning – to guide our analyses.

Following this hypothesis, we compare three model families, namely, softmax-only Transformers, and hybrid models that have a first softmax

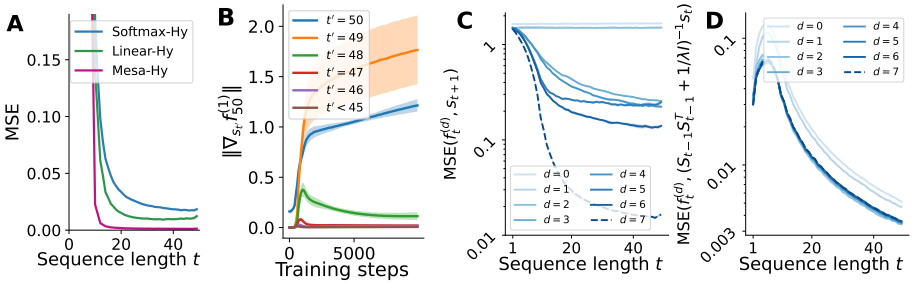


FIGURE 3.4: **Reverse engineering full-fledged trained Transformers.** We study 2-layer hybrid-mesa, 7-layer hybrid-linear, and 7-layer softmax-only Transformers. (**A**) After training, the hybrid-mesa Transformer slightly outperforms the deep hybrid-linear and softmax-only models in terms of autoregressive loss. In (**B** & **C** & **D**), we show results for a softmax-only model. (**B**) The first softmax layer groups together neighboring tokens. This can be seen in the high sensitivity to the current and previous tokens of the outputs of the first layer of a softmax-only Transformer (with even more clean next-token copying behavior for hybrid-linear and hybrid-mesa Transformers; see also complementary attention map visualizations in Appendix 3.7.4). (**B** & **C**) We linearly regress the activations of each layer against final targets (**C**) as well as $(S_{t-1}S_{t-1}^\top + 1/\lambda I)^{-1}s_t$, the preconditioned inputs (**D**) predicted by our theory. Compared to our more constructed models of Figure 3.3, here we observe a rather harsh transition in the last layer when measuring target probing (**C**) while observing a gradual performance increase for early layers when probing for curvature-corrected inputs (**D**). These results are well aligned with our hypothesized two-stage mesa-optimizer. Averages computed over 5 different seeds; shaded area represents standard deviation.

layer followed by either linear or mesa layers. First, we verify that Transformers of all three types learn copy layers when trained on linear dynamics by (*i*) computing the sensitivity norm $\|\nabla_{s_{t'}} f_t^{(1)}(s_{1:t}, \theta)\|$ of the output of the first layer for all $t' \leq t$, and by (*ii*) inspecting attention maps. We use $f_t^{(d)}(s_{1:t}, \theta)$ to denote the intermediate output of the d-th layer of a Transformer, including the residual (skip connection) value. Both experiments provide evidence that after the first layer, every token mostly depends on itself and on the preceding token, as shown in Figure 3.4B.

We now turn to the post-copying behavior of the models. Although some interpretable identity structure can be observed in the weight matrix products $W_K^\top W_Q, PW_V$ of the Transformers, cf. Figures 3.12, we speculate that the initial embedding layer introduces too much ambiguity on how the input data is represented and processed by the subsequent attention layers, complicating reverse-engineering a clean algorithm. We therefore build on insights extracted from our previous analyses and probe hidden layer activations using the same simple linear regression analysis. Even for this more complex model, we find that again hidden activations gradually (over depth) become more predictive for both the target as well as the inverse probes. Interestingly, we observe a hard-transition-like behavior at the last layer in terms of target decoder performance, in line with our constructed two-stage mesa-optimizer, which first preconditions, and then takes an optimization step in the last layer, see Figure 3.4C&D and remarkably clear in Figure 3.8 for softmax resp. linear self-attention Transformers.

Taken together, these findings provide evidence that realistic deep Transformers trained autoregressively on simple linear dynamics implement prediction algorithms based on mesa-optimization principles. These iterative algorithms allow a standard Transformer to harness depth to almost match the performance of a learned mesa-layer, which achieves optimality for the task considered here.

3.5.2 *Simple autoregressive models become few-shot learners*

In the previous section, we established a close connection between autoregressively-trained Transformers to gradient-based mesa-optimization. It is therefore natural to ask whether these models can be repurposed to learn in-context when presented with few-shot regression data. Here, we pursue this question experimentally by changing the generation of the sequences *after* training, from a linear dynamical system to a linear regression task. We illustrate our findings in Figure 3.5A.

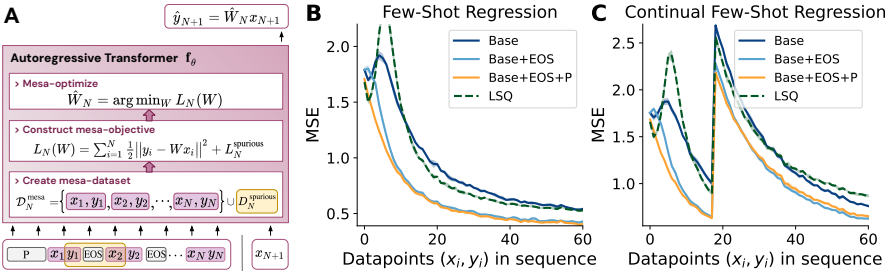


FIGURE 3.5: **Autoregressively-trained Transformers solve supervised few-shot regression problems.** (**A**) In-context learning by autoregressive mesa-optimization. (**B**) The mesa-optimization algorithm acquired by training on autoregressive linear dynamics tasks allows softmax Transformers to learn supervised tasks in-context, i.e., the mean-squared error $\langle (f(x_i;\theta) - y_i)^2 \rangle$ decreases gradually and significantly with the number of labeled examples. When prompted with a special `EOS` token after each pair (x_i, y_i) or a prefix-prompt `P` at the beginning of an input sequence, which we fine-tune for this regression task on a held-out training set, the performance improves considerably, highlighting the usefulness of prompt-tuning already in this very simple setting. (**C**) Autoregressive Transformers already display some continual in-context learning capabilities, being able to learn two tasks consecutively. Here, we show the results for the full-fledged softmax-only transformer. Averages computed over 5 different seeds; shaded area represents standard deviation.

FEW-SHOT TASK GENERATIVE MODEL. To generate our few-shot tasks we still sample a groundtruth W^* as a random orthogonal matrix as done during training, but now use this groundtruth model to generate a labeled training set $\{x_i, y_i\}_{i=1}^N$, with inputs $x_i \sim \mathcal{N}(0, I_x)$ and targets $y_i = W^* x_i$. We then present this dataset to our autoregressively-trained Transformers as a sequence of tokens, $e^{\text{few-shot}} = [x_1, y_1, \ldots, x_N, y_N]$ of length $T = 2N$, cf. Figure 3.5. As the sequence unfolds, and more training data is presented, we measure in-context learning performance through the mean squared error between the Transformer output $f_\theta(e_{2i-1}; e^{\text{few-shot}}_{1:2i-1})$ and the corresponding target $y_i = e_{2i}$. We emphasize that both the sequence generative model and loss function differ from the ones used during training; compare the task performance metric $L^{\text{few-shot}} = \frac{1}{2}\sum_{i=1}^N \|e_{2i} - f_\theta(e_{2i-1}; e^{\text{few-shot}}_{1:2i-1})\|^2$ used to evaluate in-context learning performance in this section with the actual loss used to train the Transformer, Eq. 3.11.

AUTOREGRESSIVE TRANSFORMERS ARE CAPABLE OF FEW-SHOT LEARNING. Although never trained on this setting, we observe that the loss of the Transformer decreases with sequence length, see Figure 3.5B for results obtained when taking the exact same 7-layer softmax Transformer model analyzed in Figure 3.4, repurposing it for in-context linear regression. The model can thus learn in-context, making use of additional in-context training data to improve its predictions. As a control, we further report the performance reached by the least-squares solution (LSQ) obtained on the dataset $D_N^{\text{mesa}} = \{(x_i, y_i)\}_{i=1}^N \cup \{(y_i, x_{i+1})\}_{i=1}^{N-1}$, and observe a similar decrease in loss. This dataset, where half of the associations consist of wrong input-output pairs $D_N^{\text{spurious}} = \{(y_i, x_{i+1})\}_{i=1}^{N-1}$ as illustrated in Figure 3.5A, corresponds to the training set an autoregressive Transformer imbued with the mesa-optimizers uncovered in the previous section learns from. In this sense, our models achieve a few-shot learning performance that is not far from optimal. Thus, our results show that training Transformers on simple autoregressive tasks can give rise to in-context few-shot learning, complementing previous evidence for this phenomenon in large-scale models [102].

PROMPT TUNING IMPROVES IN-CONTEXT LEARNING PERFORMANCE. To mitigate the influence of wrongly-constructed inputs (y_i, x_{i+1}) in a sequence, we fine-tune a single token, which we refer to as the `EOS` token, to improve the in-context-learned predictions. Prompt (or prefix) tuning has been shown to lead to significant performance improvements when applied

to large language models [122, 123]; here we investigate the effectiveness of this technique on our mechanistically-understood models. When presenting data sequentially as $[x_1, y_1, \texttt{EOS}, x_2, y_2, \ldots, \texttt{EOS}, x_N, y_N]$ we observe a considerable performance improvement after prompt-tuning, see Figure 3.5B. Furthermore, to 'guide' the model for few-shot tasks, we learn a single prefix-prompt `P` which we append at the beginning of a sequence with `EOS` tokens. This appears to further improve the few-shot performance for early data-pairs.

CONTINUAL IN-CONTEXT LEARNING. Lastly, we demonstrate the capability of our trained Transformers to learn multiple tasks in a row. We study the minimal setup where the model has to learn two tasks, generated from two distinct groundtruth linear models with parameters $W^{*,1}, W^{*,2}$ sampled as described above, resulting in a sequence of data of the form $[x_1^1, y_1^1, \ldots, x_N^1, y_N^1, x_1^2, y_1^2, \ldots, x_N^2, y_N^2]$. We plot the performance when using `EOS` tokens (constructed as before) and prefix prompts `P`, as well. In Figure 3.5C we see that the trained Transformer has the capability to overwrite the first and learn a second task in-context, even though it was never explicitly trained to solve such sequential learning problems.

A TOY MODEL FOR IN-CONTEXT LEARNING. We conclude that Transformers trained to predict the next element in a sequence can be naturally repurposed as in-context learners due to the similarity of the algorithms implemented within their forward pass. This allows studying in a controlled setting interesting properties of in-context learning, such as the advantages of prompt tuning and the ability to learn continually. Our toy models could serve as a test bed for future work investigating the shortcomings and various particularities of in-context learning observed in LLMs [e.g., 124–126].

3.5.3 *Language models equipped with least-squares solvers*

We now move beyond synthetic tasks and provide results on autoregressive language modeling, a problem domain Transformers have revolutionized in recent years. Because reverse-engineering the ensuing models to the degree of our previous analyses is difficult, we base our claims on performance comparisons between standard Transformers, and new variants based on the mesa-layer. Our hypothesis is that the mesa-layer will improve the in-context learning and working memory capabilities of a Transformer, in particular

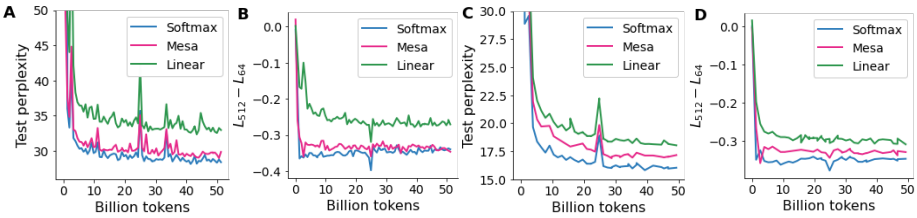


FIGURE 3.6: **Language modeling experiments on the Pile.** We observe improved perplexity and in-context learning scores across all our language modeling experiments when switching from standard linear self-attention to the mesa-layer. When comparing loss values for longer time horizons, we still observe a performance gap between softmax and mesa, possibly pointing towards memory issues over long sequences. As hypothesized, we confirm that in all models various copying heads can be found in the first softmax layer, see Appendix 3.7.4 for visualizations of the attention heads. (**A&B**) 2-layer Transformers without MLPs and first layers softmax self-attention and second layer either softmax, mesa or linear. (**C&D**) 4-layer Transformers with MLPs and first layers softmax self-attention and rest of the layers either all softmax, mesa or linear.

of the linear kind. We further hypothesize that this in turn translates to language modeling improvements, based on the high correlation between in-context learning and actual autoregressive loss reported by Kaplan *et al.* [106]. We therefore quantify performance along two axes: the next-token prediction loss, the actual objective of base-optimization; and the ability to learn in-context, measured as the difference in loss calculated over two timepoints within a sequence, as defined by Kaplan *et al.* [106] and Olsson *et al.* [121].

We train Transformers with various architectural configurations on the Pile [127], a large compilation of various English text datasets including parts of Wikipedia, arXiv, and code. We always model the first layer using softmax self-attention in all experiments. This decision is based on insights from our previous experiments, where base-optimization consistently attributed a mesa-objective creation role to this layer. We then compare pure softmax-only Transformers to two types of hybrid models, where the subsequent layers are either linear or mesa. We vary the depth of our models, from 2-layer attention-only to deeper 4-attention-layer models endowed with tokenwise MLPs which are present by default in standard Transformers. By transforming the data nonlinearly, MLP layers allow solving

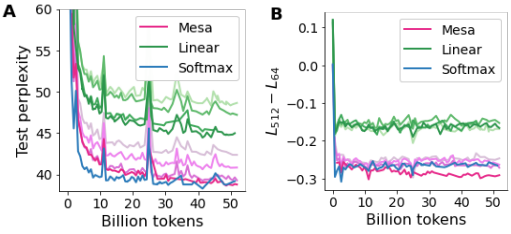


FIGURE 3.7: **Single-layer Transformers trained on the Pile with key-shifts.** We observe improved (**A**) perplexity and (**B**) in-context learning scores when comparing one linear to one mesa layer with different DPFP sizes $\nu \in \{0, 1, 2, 3\}$, corresponding inversely to color fade. Mesa layers consistently outperform linear layers, catching up with softmax.

nonlinear regression problems by mesa-gradient descent. Following this reasoning, we further adopt in our hybrid-linear and hybrid-mesa Transformers the deterministic parameter-free projection (DPFP, size denoted by ν) due to Schlag, Irie & Schmidhuber [110], a non-learned and simple to compute nonlinear transformation of keys and queries. We found that this significantly improved the performance of non-softmax attention layers. Finally, to represent discrete input symbols as real-valued vectors, we learn a vocabulary of real-valued vectors using the standard GPT-2 tokenizer. We note that all models have an (almost) identical number of parameters.

In line with our synthetic experiments, we observe stable learning across all model types of copying layers, indicated by the constant attention to tokens in direct or close proximity, as shown in Figure 3.9. We therefore reproduce the findings of Olsson *et al.* [121], extending them to models that include other forms of attention. This phenomenon is predicted by the mesa-optimization theory presented here, where copy layers serve the purpose of constructing internal mesa-objective functions. We note that, in contrast to our previous synthetic linear prediction tasks, the Pile is no longer Markovian of order 1. This is reflected in the more complicated attention maps, indicating more involved copying behavior. Additionally, we run an ablation where we compare to a single-layer control model whose first softmax layer is removed and replaced by a hardcoded one-step key-shift operator. Interestingly, such an operator can be found in previous work [121, 128]. Again, we verify the findings of [121] and observe strong in-context learning scores, within a single layer, with the mesa-layer performing on-par with softmax, see Figure 3.7. As in [110], DPFP features

substantially improve performance; we fix $\nu = 3$ for the linear as well as the mesa layer for all other language modeling experiments.

We find that the hybrid-mesa Transformers dominate their hybrid-linear counterparts in terms of performance, across all configurations, essentially matching (for 2-layer models) or coming closer (for 4-layer models with MLPs) to pure-softmax Transformers, cf. Figure 3.6. We leave for future work studying the mesa-layer equipped with forgetting factors, see Appendix 3.7.2.1, which could further improve upon our results here. This is reflected both in terms of perplexity and in-context learning scores. Strictly speaking, these results are not sufficient to make claims on whether mesa-optimization is occurring within standard Transformers. However, the high performance achieved by the hybrid-mesa models, which operate on mesa-optimization principles by design, suggests that mesa-optimization might be happening within conventional Transformers. More reverse-engineering work is needed to add weight to this conjecture.

3.6 DISCUSSION

We presented evidence that Transformer models are capable of developing gradient-based inference algorithms when trained on sequence prediction tasks under a standard autoregressive objective. We therefore confirmed that recent results obtained under a multi-task, meta-learning setup translate to the conventional self-supervised LLM training setup. Moreover, we have seen that the resulting autoregressive inference algorithms can be repurposed without retraining to solve supervised in-context learning tasks, thus explaining the aforementioned results within a single, unified framework.

It should be noted that our reverse-engineering findings are for now restricted to simple linear prediction tasks. More work is needed to understand how and if our findings translate to the nonlinear setting, and more generally to determine the conditions that lead some base optimization process to pick solutions corresponding to gradient-based in-context learning algorithms. It seems unlikely that the internal construction and gradient-based solution of least-squares problems is a universal mechanistic explanation of trained Transformers. An interesting future work direction is to attempt to reverse-engineer and describe through mesa-optimization models trained on problems of a radically different kind than those considered here, such as algorithmic reasoning [129].

The idea that a Transformer generates its predictions by solving one or more internal optimization problems has ties to many different lines of thinking in machine learning. One closely related line of work explores the concept of a declarative node: a differentiable layer whose output is defined implicitly as the solution of an optimization problem [90, 92, 93]. The mesa-layer is an example of such a node. Summarizing the operation of an entire chain of layers with thousands of parameters by a single declarative node is not only potentially more efficient, but also more interpretable. We thus join a line of interesting recent work exploring the advantages of including declarative nodes within attention-based models [116, 130].

Our reverse-engineering analyses brought a surprising revelation: gradient-based base-optimization of an autoregressive loss *discovered* such a declarative node, at least when the underlying sequence was generated by a linear dynamics. This discovery or selection of an optimization algorithm through learning has been termed mesa-optimization [105], a notion that we have adopted throughout this chapter. While we do not wish to comment here on the possible risks associated with mesa-optimization, we point out that our results may be of interest to the artificial intelligence safety community, by providing a simple mesa-optimization toy model.

The mesa-layer can also be seen as a locally-optimal fast weight programmer from the perspective of Schmidhuber [131]. In his seminal work, Schmidhuber [131] proposed to dynamically reprogram the weights of a feedforward neural network using a Hebbian rule. As pointed out by Schlag, Irie & Schmidhuber [110] and as can be seen from Eq. 3.2, this is precisely what a linear self-attention layer does: it generates predictions using an effective weight matrix that is learned during a forward pass by taking outer products of values and keys, a Hebbian associative rule [132]. In this work, we instead frame fast weight learning as an optimization problem, that is efficiently and optimally solved at every moment in time by the mesa-layer. This form of optimal fast learning is strictly superior to Hebb's rule, both in terms of generalization and memory capacity [133]. The mesa-layer is therefore also closely related to the Delta-Net of Schlag, Irie & Schmidhuber [110], which uses the delta rule [134] for fast weight learning. Unlike the mesa-layer which is optimal at every time step, this rule requires multiple steps to converge, but it is cheaper to implement.

When using mesa-layers in an autoregressive Transformer, the base-optimization process becomes explicitly a meta-learning algorithm [61]. This algorithm should however be distinguished from the end-to-end supervised meta-learning approaches that are currently highly popular in

machine learning [e.g., 67, 135, 136]. In our models, everything is ultimately driven by the pressure to predict the future, the signal that drives the slow autoregressive base-optimization process. This process ultimately dictates the objectives each layer must optimize. Moreover and also unusually for meta-learning, each mesa-layer is a greedy supervised local learner, which does not use backpropagation or any other kind of global error information. Instead, each mesa-layer has its own local objective functions specified through the corresponding key and value matrices.

Seen from this angle, our work has an unexpected connection to research on local learning rules, a question of great interest in theoretical neuroscience [14]. Decomposing a global supervised learning problem into a sequence of local quadratic optimization problems, as we do here, is at the heart of the target propagation [137], predictive coding [138] and control-based [139] theories of learning in the brain, and previous studies have proposed greedy layerwise learning algorithms that do not require global error information [140–144]. Our study introduces greedy local learning algorithms, which only use bottom-up information, to the fast timescale of inference. It is interesting that our models achieve strong performance in natural tasks without any top-down feedback at fast timescales, at odds with canonical predictive coding theories [145, 146].

We finish by sharing our excitement about future research directions that aim at analyzing simple autoregressively-trained sequence models like Transformers and in particular in-context learning within by reverse engineering. We hope our work motivates further studies trying to describe the emergence of single, multiple or mixture of expert models mesa-optimized in simple trained Transformers [147] which we hypothesize could illicit inference reminiscent to world models [148, 149]. Furthermore, the insights we gained in our controlled setting could motivate studying limitations and particularities of in-context learning [125, 126] and its powerful variants such as chain-of-thought prompting [37, 43, 150] as well as the fascinating interplay between in-weights and in-context learning [151].

3.7 APPENDIX

We present here some additional results to complement the main results discussed in the previous sections.

3.7.1 *Mesa layer with forgetting factors*

Here, we revisit the mesa-layer forward pass introduced in Section 3.4, with an added forget factor $\Gamma_{h,t} = (\gamma_{h,t'})_{t'=1}^{t}$, where $\gamma_{h,t'} \in (0, 1]$.

Although we leave an empirical investigation of the forget gate for future work, we hypothesize that a token-dependent forget gate can benefit the performance of the layer by allowing selective memory retention and forgetting. Nevertheless, we stress potential initialization and numerical issues as well as training instabilities when computing the necessary products of factors across time.

Given again a set of tokens E_t, the generalized mesa-layer changes the tokens as follows:

$$\Delta e_t^{\text{mesa}} = \sum_{h=1}^{H} P_h \hat{W}_{h,t}^{\text{mesa}} q_{h,t}, \tag{3.12}$$

$$\text{with} \qquad \hat{W}_{h,t}^{\text{mesa}} = \arg\min_{W} \left\{ \frac{1}{2} \sum_{t'=1}^{t} \left(\prod_{t''=t'+1}^{t} \gamma_{h,t''} \right) ||W k_{h,t'} - v_{h,t'}||^2 \right. \tag{3.13}$$

$$\left. + \frac{\prod_{t''=1}^{t} \gamma_{h,t''}}{2\lambda_h} ||W||_{\text{F}}^2 \right\}. \tag{3.14}$$

This is known as the recursive least squares problem with forgetting and is widely used in the online learning literature [152]. For notational simplicity we drop the subscript in h and ignore the sum over the heads in the following derivation. It can be shown that the analytical solution of the optimization problem is

$$\hat{W}_t^{\text{mesa}} = \left(\sum_{t'=1}^{t} \left(\prod_{t''=t'+1}^{t} \gamma_{t''} \right) v_{t'} k_{t'}^{\top} \right) \left(\sum_{t'=1}^{t} \left(\prod_{t''=t'+1}^{t} \gamma_{t''} \right) k_{t'} k_{t'}^{\top} + \frac{\prod_{t''=1}^{t} \gamma_{t''}}{\lambda} I \right)^{-1}$$

We will now see how Δe_t^{mesa} can be efficiently computed in a forward pass.

3.7.1.1 *Computing the inverse term within $\hat{W}_t^{\text{mesa}}$*

Computing the full-fledged inverse at every timestep is computationally too expensive. As in Section 3.4, we resort to using the Sherman-Morrison

formula to efficiently compute the inverse term for all timestep sequentially in time. We redefine

$$R_t = \left(\sum_{t'=1}^{t} \left(\prod_{t''=t'+1}^{t} \gamma_{t''} \right) k_{t'} k_{t'}^\top + \frac{\prod_{t''=1}^{t} \gamma_{t''}}{\lambda} I \right)^{-1}. \tag{3.15}$$

It satisfies the recursive formula

$$R_{t+1} = \left(\gamma_t R_t^{-1} + k_{t+1} k_{t+1}^\top \right)^{-1} \tag{3.16}$$

with $R_0 = \lambda I$, and the Sherman-Morrison formula thus gives

$$R_{t+1} = \gamma_{t+1}^{-1} \left(R_t^{-1} + \gamma_{t+1}^{-1} k_{t+1} k_{t+1}^\top \right)^{-1} \tag{3.17}$$

$$= \gamma_{t+1}^{-1} \left(R_t - \frac{\gamma_{t+1}^{-1} R_t k_{t+1} k_{t+1}^\top R_t}{1 + \gamma_{t+1}^{-1} k_{t+1}^\top R_t k_{t+1}} \right) \tag{3.18}$$

$$= \gamma_{t+1}^{-1} \left(R_t - \frac{R_t k_{t+1} k_{t+1}^\top R_t}{\gamma_{t+1} + k_{t+1}^\top R_t k_{t+1}} \right). \tag{3.19}$$

Note that we recover Eq. 3.9 by setting all γ_t to 1.

3.7.1.2 *Computing* Δe_t^{mesa}

Given $R_{h,t}$ for all heads, we can rewrite the token update as

$$\Delta e_t^{\text{mesa}} = \sum_{h=1}^{H} P_h \left(\sum_{t'=1}^{t} \left(\prod_{t''=t'+1}^{t} \gamma_{h,t''} \right) v_{h,t'} k_{h,t'}^\top \right) R_{h,t} q_{h,t} \tag{3.20}$$

$$= \sum_{h=1}^{H} P_h V_h \left(\left(\mathbb{I}_{t' \leq t} \prod_{t''=t'+1}^{t} \gamma_{h,t''} \right)_{t'=1}^{\top} \odot K_h^\top \tilde{q}_{h,t} \right) \tag{3.21}$$

$$= \sum_{h=1}^{H} P_h V_h \left(M_{:,t} \odot K_h^\top \tilde{q}_{h,t} \right) \tag{3.22}$$

where $\tilde{q}_{h,t} = R_{h,t} q_{h,t}$ and $M_{t',t} := \mathbb{I} r_{t' \leq t} \prod_{t''=t'+1}^{t} \gamma_{h,t''}$. Note that we apply some form causal masking here: we take the key $K_h \in \mathbb{R}^{D_a \times T}$ and value matrices $V_h \in \mathrm{R}^{D_a \times T}$ with all the sequence timesteps and select the entries occurring before time t. The main difference with the usual causal mask $(\mathbb{I}_{t' \leq t})_{t',t}$ is the inclusion of the forget factors. It can be efficiently computed leveraging partial products. We conclude by remarking that the same mask can be applied to softmax attention layers, applying it to the key-queries products before the softmax.

3.7.2 *Mesa layer backward computation*

3.7.2.1 *Mesa layer computation backward pass via Sherman-Morrison*

In this section, we detail how to compute the backward pass of the mesa layer with forget factor detailed in Section 3.7.1. Recall that the forward pass of the Mesa layer is computed recursively following

$$R_{h,t+1} = \gamma_{h,t+1}^{-1} \left(R_{h,t} - \frac{R_{h,t} k_{h,t+1} k_{h,t+1}^\top R_{h,t}}{\gamma_{h,t+1} + k_{h,t+1}^\top R_{h,t} k_{h,t+1}} \right) \tag{3.23}$$

$$\Delta e_{t,\text{mesa}} = \sum_{h=1}^{H} P_h V_h \left(M_{:,t} \odot K_h^\top \tilde{q}_{h,t} \right) \tag{3.24}$$

with $R_{h,0} = \lambda_h I$.

The forward pass can be decomposed into 3 steps:

1. First, the matrices $R_{t,h}$ are computed sequentially.
2. Then, for all t and h, the transformed queries $\tilde{q}_{h,t} = R_{h,t} q_{h,t}$ are computed.
3. Finally, using the transformed queries $\tilde{Q}_h = (\tilde{q}_{h,t})_t$ as the queries, a standard cross-attention operation is computed from $(V_h, K_h, \tilde{Q}_h)$ using the causal mask M that includes forgetting rates.

While the backward pass of 2 and 3 can be computed easily with automatic differentiation tools without much overhead compared to standard attention layers, the same thing cannot be said about 1. We will here discuss how the backward pass of the computation of $\tilde{Q}_h$ can be computed in a memory-efficient way. Without loss of generality, we drop the subscript h for notational simplicity.

THE ISSUE WITH AUTOMATIC DIFFERENTIATION OUT OF THE BOX. For all time t, $\tilde{q}_t = R_t q_t$ depends on q_t, but also K_t, Γ_t and λ through the variable R_t.

In the backward pass, we are given as input the gradient of the loss function w.r.t. $\tilde{Q}$, namely $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\tilde{q}_t}$ for all t. The goal is then to compute the gradient of the loss w.r.t. the input of $\tilde{Q}$, namely $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}k_t}$, $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\gamma_t}$, $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}q_t}$ and $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\lambda}$, which can be achieved via the chain rule.

While using automatic differentiation out of the box would take care of this computation, it would require in particular the storing of all intermediate variables R_t, which can be prohibitively expensive.

MEMORY EFFICIENT CUSTOM BACKWARD PASS. Instead, we will show that storing the matrices K, Γ, Q as well as R_T where T is the last time step of the training sequence, is sufficient to exactly compute the backward pass. Indeed, given the aforementioned inputs, all R_t can be recomputed in linear complexity w.r.t. T, which means we can reconstruct recursively the inputs of $\tilde{q}_t$ at all time steps.

By noticing that $R_{t-1} = \gamma_t(R_t^{-1} - k_t k_t^\top)^{-1}$, we can apply the Sherman-Morrison formula backwards to obtain R_{t-1} as

$$R_{t-1} = \gamma_t \left(R_t - \frac{R_t(-k_t)k_t^\top R_t}{1 + (-k_t)^\top R_t k_t} \right) \tag{3.25}$$

$$= \gamma_t \left(R_t - \frac{R_t k_t k_t^\top R_t}{k_t^\top R_t k_t - 1} \right) \tag{3.26}$$

We will now show how accumulating the right error signal and leveraging the vector-jacobian product trick together with automatic differentiation tools is sufficient for computing the full backward pass recursively.

Firstly, given the error signal and reconstructed R_t allows the computation of $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}q_t}$ via

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}q_t} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\tilde{q}_t}\frac{\mathrm{d}\tilde{q}_t}{\mathrm{d}q_t} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\tilde{q}_t} S_t \tag{3.27}$$

Secondly, we rewrite $\tilde{q}_t$ as a function of k_t, γ_t, R_{t-1} and q_t, i.e.

$$\tilde{q}_t = \mathcal{R}^{\text{forward}}(R_{t-1}, k_t, \gamma_t) q_t \tag{3.28}$$

Since $\mathcal{L}$ depends on k_t only via both $\tilde{q}_t$ and R_t, we can then rewrite

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}k_t} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\tilde{q}_t}\frac{\mathrm{d}\tilde{q}_t}{\mathrm{d}k_t} + \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_t}\frac{\mathrm{d}R_t}{\mathrm{d}k_t} \tag{3.29}$$

$$= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\tilde{q}_t}\frac{\partial\tilde{q}_t}{\partial k_t} + \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_t}\frac{\partial R_t}{\partial k_t} \tag{3.30}$$

where, provided R_{t-1}, k_t, γ_t and q_t, $\frac{\partial\tilde{q}_t}{\partial k_t}$ can be computed easily using e.g. automatic differentiation tools. Similarly, we have,

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\gamma_t} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\tilde{q}_t}\frac{\partial\tilde{q}_t}{\partial\gamma_t} + \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_t}\frac{\partial R_t}{\partial\gamma_t} \tag{3.31}$$

Notice that $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_t}$ can be computed recursively following the chain rule

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_{t-1}} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_t}\frac{\partial R_t}{\partial R_{t-1}} + \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\tilde{q}_t}\frac{\partial \tilde{q}_t}{\partial R_{t-1}} \tag{3.32}$$

where again, provided R_{t-1}, k_t, γ_t and q_t, both terms can be computed efficiently with standard automatic differentiation tools coupled with the well known vector-Jacobian product trick given the quantities $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_t}$ and $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\tilde{q}_t}$.

Thirdly, we can show that

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\lambda} = \mathrm{Tr}\left[\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_0}\right] \tag{3.33}$$

Combining everything, we can now implement the backward computation recursively via the following equations:

$$R_{t-1} = \gamma_t\left(R_t - \frac{R_t k_t k_t^\top R_t}{k_t^\top R_t k_t - 1}\right) \tag{3.34}$$

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_{t-1}} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_t}\frac{\partial R_t}{\partial R_{t-1}} + \frac{\partial \mathcal{L}}{\partial \tilde{q}_t}\frac{\partial \tilde{q}_t}{\partial R_{t-1}} \tag{3.35}$$

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}k_t} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\tilde{q}_t}\frac{\partial \tilde{q}_t}{\partial k_t} + \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_t}\frac{\partial R_t}{\partial k_t} \tag{3.36}$$

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\gamma_t} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\tilde{q}_t}\frac{\partial \tilde{q}_t}{\partial \gamma_t} + \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_t}\frac{\partial R_t}{\partial \gamma_t} \tag{3.37}$$

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}q_t} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\tilde{q}_t}R_t \tag{3.38}$$

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\lambda} = \mathrm{Tr}\left[\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_0}\right] \tag{3.39}$$

R_T is assumed to be given and $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_T} = 0$. The above equations only require the storage of $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_t}, \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}R_{t-1}}, R_t, R_{t-1}$ at all time, and computes the backward pass in a similar time and memory complexity as for the forward pass. The derivation is identical without forgetting factors, by setting all γ to 1.

Comment on runtime. We highlight that, although this implementation of the mesa-layer reduces the memory footprint of the forward and backward pass substantially, the layer still runs forward (and backward) in time. This prevents the computation of all mesa-layer outputs in parallelization during training, a crucial advantage of softmax as well as linear attention. On the other hand, during test time, the mesa-layer benefits from the same advantages of linear self-attention or RNNs and predicts the next token

without the necessity to store and attend to the past. In the next section, we present one potential avenue to improve the training time by approximating the necessary inversions by a Neumann series running in parallel.

3.7.2.2 *Alternative derivation through the implicit function theorem*

We here present an alternative way of deriving the gradients presented above that leverages the implicit function theorem. The key here is to remark that $\hat{W}_t^{\text{mesa}}$ satisfies that the gradient of the least-square regression loss L is 0. For simplicity, we restrict ourselves to the case in which the output dimension of $\hat{W}_t^{\text{mesa}}$ is one, that is $\hat{W}_t^{\text{mesa}} = \hat{w}_t^\top$ for $\hat{w}_t$ some column vector, and remark that we have to repeat the same operation over all rows of $\hat{W}_t^{\text{mesa}}$ to obtain the full gradient, as all output coordinates are independent in the least-square regression problem. Therefore, we w defined through the implicit function

$$\frac{\mathrm{d}L}{\mathrm{d}w}(\hat{w}_t) = \sum_{t'=1}^{t} M_{t',t}(\hat{w}_t^\top k_{t'} - v_{t'})k_{t'}^\top + \frac{M_{1,t}}{\lambda}\hat{w}_t^\top = 0. \tag{3.40}$$

We can then use the implicit function theorem and compute the derivative of w with respect to any quantity $\cdot$ through

$$\frac{\mathrm{d}\hat{w}_t}{\mathrm{d}\,\cdot} = -\left(\frac{\mathrm{d}^2 L_t}{\mathrm{d}w^2}(w_t)\right)^{-1} \frac{\mathrm{d}^2 L_t(\hat{w}_t)}{\mathrm{d}\,\cdot\,\mathrm{d}w} \tag{3.41}$$

$$= -R_t \frac{\mathrm{d}^2 L_t(\hat{w}_t)}{\mathrm{d}\,\cdot\,\mathrm{d}w}. \tag{3.42}$$

For example, this yields

$$\frac{\mathrm{d}\hat{w}_t}{\mathrm{d}v_{t'}} = M_{t',t} R_t k_{t'}. \tag{3.43}$$

Finally, we can recover the desired gradient by combining the previous equation with the chain rule.

3.7.2.3 *Parallel backward pass through Neumann series approximation*

Note: We present this section for the sake of completeness – no experiments presented in this chapter use this approximation.

Although the previous custom backward gradient computation allows for dramatic memory savings during training, the underlying recursive least squares computation still suffers from linear scaling in time, similar

to recurrent neural networks, as we cannot parallelize computation across time dimension.

Here, we discuss an alternative forward pass that can be used when one can afford storing all intermediate matrices $R_{h,t}$ in time. This forward pass leverages a K-step truncated Neumann series to approximate the inverses in parallel, and is compatible with automatic differentiation tools out of the box. Interestingly, we can do this by simply repeating (with the same weights) a slightly altered linear self-attention layer K times.

Our goal is now to efficiently compute the terms $\tilde{q}_t := R_t q_t = (K_t K_t^\top + \frac{1}{\lambda} I)^{-1} q_t$ for all time steps in parallel. Indeed, once give these vectors, one can leverage Equation 3.22 and efficient dot-product attention (DPA) layers implementations[2]. Note that we here ignore the forgetting factors, but their partial products can easily be integrated in one of the K_t in $K_t K_t^\top$ to recover the version with forget rates described above.

Given an invertible matrix X with operator norm less than 1, the truncated Neumann series approximates its inverse by

$$X^{-1} \approx \tilde{X}^{-1}_{(K)} := \sum_{k=0}^{K} (I - X)^k. \tag{3.44}$$

When multiplying a vector from the right, we see that

$$\begin{aligned}
\tilde{x}^{(K)} := \tilde{X}^{-1}_{(K)} x &= \sum_{k=0}^{K} (I - X)^k x && (3.45)\\
&= \sum_{k=1}^{K} (I - X)^k x + x && (3.46)\\
&= (I - X) \sum_{k=0}^{K-1} (I - X)^k x + x && (3.47)\\
&= (I - X)\tilde{x}^{(K-1)} + x && (3.48)
\end{aligned}$$

An advantage of the truncated Neumann series compared to other approximate inverse techniques such as Newton-Iteration is that we can compute more series elements without passing intermediate matrices across algorithmic steps – which in turn makes it memory efficient and straightforward to use in the light of automatic differentiation. We only need to keep the original matrix we wish to invert in memory at all times and store the intermediate vectors $\tilde{x}^{(k)}$ for the backward pass.

2 See `https://flax.readthedocs.io/en/latest/_modules/flax/linen/attention.html` for an implementation of DPA in JAX [153].

We now look at the quantities we wish to compute, that is $\tilde{q}_t = (K_t K_t^\top + \frac{1}{\lambda} I)^{-1} q_t$, and approximate it by $\tilde{q}_t^{(K)}$, obtained by multiplying q_t to the K-step truncated Neumann series approximating the inverse term $(K_t K_t^\top + \frac{1}{\lambda} I)^{-1}$. Note that a normalization by the operator norm of the matrix inside the inverse is necessary for the approximation to hold.

Then, $\tilde{q}_t^{(K)}$ can be computed recursively as

$$\tilde{q}_t^{(k+1)} = \left(I - \left(K_t K_t^\top + \frac{1}{\lambda} I \right) \right) \tilde{q}_t^{(k)} + q_t \tag{3.49}$$

$$= q_t + \left(1 - \frac{1}{\lambda} \right) \tilde{q}_t^{(k)} - K_t K_t^\top \tilde{q}_t^{(k)} \tag{3.50}$$

and thus by denoting $\tilde{Q}_t^{(k)} := (\tilde{q}_{t'}^{(k)})_{t'=1}^{t}$, we have

$$\tilde{Q}_{k+1}^{(k+1)} = Q_t + \left(1 - \frac{1}{\lambda} \right) \tilde{Q}_t^{(k)} - K_t K_t^\top \tilde{Q}_t^{(k)} \tag{3.51}$$

which is the sum of simple terms with a DPA computed between $K_t, K_t, \tilde{Q}_t^{(k)}$.

After obtaining $\tilde{Q}_t^{(K)}$ to approximate $\tilde{Q}_t$, we compute the approximate least-squares solution as described above. Note that other implementations could save us from effectively recomputing $(K_t K_t^\top)$ at every iteration of Equation 3.51 by simply pre-computing these terms before running the Neumann approximation. We nevertheless observe the former version to be faster when timing for forward and backward computation and speculate the reason being the highly optimized implementation of DPA as the backbone of the self-attention layer. Note that a simple byproduct of the derivations here is the insight that chaining linear self-attention layers can actually easily implement truncated Neumann series computation – especially if the goal is an inverse multiplied by a known vector. See Section 3.7.3.2 for a more in-depth analysis.

3.7.3 *Mechanistic interpretability of Transformers trained on linear dynaimcs*

3.7.3.1 *Single-layer mesa-gradient descent*

We state here for completion the training objective of Transformers trained on sequences with initial state $s_1 \sim \mathcal{N}(0, I)$ with subsequent states $t = 1, \ldots, T$ generated according to the rule $s_t = W^* s_{t-1} + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, \sigma_s^2 I)$ introduces uncorrelated Gaussian noise. We take W^* to be a

random orthogonal matrix. The Transformer models t_θ are trained by stochastic online minimization of the autoregressive loss $\mathcal{L}(\theta)$, cf. Eq. 3.11.

After training, we obtain structured matrix products $W_K^\top W_Q, PW_V$ per layer which we visualize in Figure 3.10 for a single linear self-attention Transformer and in Figure 3.11 for the multi-layer case. When inspecting the trained weight matrix products, one observes stable values across block-diagonals of the input size across all layers.

We start by analyzing the simpler single layer Transformer computation and reduce it to

$$\begin{aligned} e_t &\leftarrow e_t + \mathrm{LSA}(e_t; (e_{t'})_{t'=1}^t) \\ &= \sum_{h=1}^{H} \phi_1^t(d_h^{PV}, d_h^{K^TQ}) s_t + \phi_2^t(d_h^{PV}, d_h^{K^TQ}) s_{t-1}, \end{aligned} \tag{3.52}$$

where $\mathrm{LSA}(e_t; (e_{t'})_{t'=1}^t)$ denotes the linear self-attention operation with context $(e_{t'})_{t'=1}^t$ and query e_t, and with $d^\cdot_\cdot$ inputs to the functions ϕ defined as follows:

$$\begin{aligned} \phi_M^t(d^{PV}, d^{K^TQ}) = \sum_{t'=1}^{t} & d^{K^TQ_{M1}} \cdot \left(d^{PV_1} s_{t'} s_{t'}^\top + d^{PV_2} s_{t'-1} s_{t'}^\top \right) \\ & + d^{K^TQ_{M2}} \cdot \left(d^{PV_1} s_{t'} s_{t'-1}^\top + d^{PV_2} s_{t'-1} s_{t'-1}^\top \right). \end{aligned}$$

Here d^{PV}, d^{KQ} corresponds to the 4 (the lower right square of size $2d \times 2d$ of K^TQ) resp. 2 (the upper right rectangle of size $d \times 2d$ of PV) non-zero off-diagonal values that we observe in the trained weight products per head i.e.

$$W_k^\top W_q = \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & d^{K^TQ_{1,1}} I_s & d^{K^TQ_{1,2}} I_s \\ \cdot & d^{K^TQ_{2,1}} I_s & d^{K^TQ_{2,2}} I_s \end{pmatrix} \tag{3.53}$$

as well as

$$PW_v = \begin{pmatrix} \cdot & d^{PV_1} I_s & d^{PV_2} I_s \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}. \tag{3.54}$$

.

We extract the values from the trained models by computing the mean of the block diagonal matrices. Note that we allow for all combinations between temporally accumulated block matrices and the current token inputs $e_t = [0, 0, s_t, s_{t-1}]^\top$ with specific strengths controlled through the

parameters. In all of our experiments, we observe a performance increase when changing from single-head to two-head attention layers (more than two heads do not alter performance). This can be explained by the improved flexibility of scaling the different terms individually as can be seen by comparing the formulas for one and two heads.

Following equation 3.52, the prediction of a two-head single layer Transformer with our construction of $e_t = [0, s_t, s_{t-1}]$ is given by

$$\begin{aligned}\hat{s}_{t+1} &= 0 + \sum_{h=1}^{2} \phi_1^t(d_h^{PV}, d_h^{K^TQ}) s_t + \phi_2^t(d_h^{PV}, d_h^{K^TQ}) s_{t-1} \\ &= \underbrace{\left(\phi_1^t(d_1^{PV}, d_1^{K^TQ}) + \phi_1^t(d_2^{PV}, d_2^{K^TQ})\right)}_{A} s_t \\ &\quad + \underbrace{\left(\phi_2^t(d_1^{PV}, d_1^{K^TQ}) + \phi_2^t(d_2^{PV}, d_2^{K^TQ})\right)}_{B} s_{t-1}\end{aligned}$$

with

$$\begin{aligned}A = &\sum_{t'=1}^{t} d_1^{K^TQ_{11}} \cdot \left(d_1^{PV_1} s_{t'} s_{t'}^\top + d_1^{PV_2} s_{t'-1} s_{t'}^\top\right) \\ &+ d_1^{K^TQ_{12}} \cdot \left(d_1^{PV_1} s_{t'} s_{t'-1}^\top + d_1^{PV_2} s_{t'-1} s_{t'-1}^\top\right) \\ &+ d_2^{K^TQ_{11}} \cdot \left(d_2^{PV_1} s_{t'} s_{t'}^\top + d_2^{PV_2} s_{t'-1} s_{t'}^\top\right) \\ &+ d_2^{K^TQ_{12}} \cdot \left(d_2^{PV_1} s_{t'} s_{t'-1}^\top + d_2^{PV_2} s_{t'-1} s_{t'-1}^\top\right) \\ = &\sum_{t'=1}^{t} \lambda_{A,1} s_{t'} s_{t'}^\top + \lambda_{A,2} s_{t'-1} s_{t'}^\top + \lambda_{A,3} s_{t'} s_{t'-1}^\top + \lambda_{A,4} s_{t'-1} s_{t'-1}^\top. \qquad (3.55)\end{aligned}$$

after summarizing by combining factors in front of the same outerproducts. (B) is computed accordingly. Here, every outer product combination of $s_{t'}$ and $s_{t'-1}$, is weighted by a product of two factors, before summarizing, that are co-dependent within one head. Therefore we indeed need a minimum of two heads to obtain independent λ_i. However, adding further heads does not increase performance as no further expressivity is gained. See for a visualization of the corresponding weight matrix products and factors that we extract from the block diagonals Figure 3.10. The extracted mean values of our trained Transformer block diagonals are reported in Table 3.1.

We now aim to interpret this parametrized algorithm and motivate it by gradient descent on a particular regression loss. Since the parametrizations

Seed	λ	$\lambda_{\cdot,1}$	$\lambda_{\cdot,2}$	$\lambda_{\cdot,3}$	$\lambda_{\cdot,4}$	$\mathcal{L}$	$\mathcal{L}_{\text{reduc}}$	$\mathcal{L}_{\text{ablat}}$	$\mathcal{L}_{\text{GD}}$
1	λ_A	0.000620	0.000254	-0.033142	0.000149	0.892	0.901	1.728	0.915
1	λ_B	0.003644	0.000034	-0.000978	-0.000129				
2	λ_A	-0.001648	0.000089	-0.033428	0.001809	0.893	0.904	1.735	0.908
2	λ_B	0.003974	-0.000215	-0.000423	0.000023				
3	λ_A	-0.003381	0.000776	-0.033751	0.006270	0.894	0.941	1.764	0.908
3	λ_B	0.002756	-0.000444	-0.002846	0.000625				
4	λ_A	0.001905	0.000270	-0.033858	-0.002946	0.893	0.909	1.742	0.902
4	λ_B	0.003851	0.000285	0.000873	0.000225				
5	λ_A	-0.001676	0.000324	-0.033557	0.001557	0.893	0.905	1.722	0.914
5	λ_B	0.002450	-0.000028	-0.000027	0.000441				

TABLE 3.1: Understanding the algorithm parametrization of Transformers trained on linear dynamics. To test the significance of the λ values derived in Eq. 3.55, we set almost all values i.e. $\lambda_{\cdot,1} = \lambda_{\cdot,2} = \lambda_B = 0$ - we call this loss $\mathcal{L}_{\text{reduced}}$. This coincides as shown to gradient descent on a meta-learned initial prediction and learning rate (see Eq. 3.56). To show the influence of the λ values corresponding to GD, we compute the algorithms performance when only setting $\lambda_{A,3/4} = 0$ and observe a drastic loss increase, we denote this loss as $\mathcal{L}_{\text{ablation}}$ and also report the loss of one step of GD as $\mathcal{L}_{\text{GD}}$.

remain constant across a sequence, we speculate that the Transformer has two principles by which it aims to predict the next token: gradient descent, and past-token averaging. The latter becomes especially useful for quickly contracting dynamics after convergence since simply copying over the last token can be one optimal and simple-to-implement solution even when aiming to obtain low loss on the entire sequence. We thus hypothesize that past-token averaging is a simple way to overcome the sub-optimality of taking only one step of gradient descent.

Consider again the squared error loss from Eq. 3.4, which we hypothesize is internally optimized inside a single layer of self-attention

$$L_t^{\text{self-attention}}(W) = \frac{1}{2} \sum_{t'=1}^{t-1} \|s_{t'} - W s_{t'-1}\|^2,$$

We now compute and evaluate the gradient of the loss evaluated at the initial $W = \tilde{\lambda}_1 I$ leading to an initially scaled prediction of the current s i.e. $\hat{s}_{t+1} = \tilde{\lambda}_1 s_t$,

$$\nabla_W L_t^{\text{self-attention}}(\tilde{\lambda}_1 I) = -\sum_{t'=1}^{t-1}(s_{t'} - \tilde{\lambda}_1 s_{t'-1}) s_{t'-1}^\top.$$

The prediction after a gradient step can be computed by

$$\begin{aligned}\hat{s}_{t+1} &= (\tilde{\lambda}_1 I - \eta_1 \nabla_W L_t^{\text{self-attention}}) s_t \\ &= \tilde{\lambda}_1 s_t - \eta_1 \sum_{t'=1}^{t-1}(s_{t'} - \tilde{\lambda}_1 s_{t'-1}) s_{t'-1}^\top s_t \\ &= \tilde{\lambda}_1 s_t + (\sum_{t'=1}^{t-1} \lambda_{C,3} s_{t'} s_{t'-1}^\top + \lambda_{C,4} s_{t'-1} s_{t'-1})^\top s_t. \end{aligned} \tag{3.56}$$

Note that this is a stripped down version of the derivation above, see equation 3.55. We now simply compare the final MSE loss when setting $\lambda_{A,1} = \lambda_{A,2} = \lambda_B = 0$ and observe minimal loss degradation, see Table 3.1. Given this robustness, we are confident that almost all of the behavior and performance of the trained Transformer in this setting can be explained by simply descending the mesa-objective of Eq. 3.4 by gradient descent. We also see that empirically using an initial prediction and therefore a non-zero implicit initial weight has some influence on the final performance. Note that to realize full expressivity in $\lambda_{A,3} = \lambda_{A,4}$ it requires 6 parameters coming from both heads.

3.7.3.2 *Multi-layer accelerated mesa-gradient descent*

We now return to the mesa-optimization algorithms presented in Section 3.3 – stacked mesa-gradient descent layers, and preconditioned mesa-gradient descent – and present them in full detail, in the context of the linear dynamics prediction problems studied in the main text.

Review: d-layers of self-attention can implement d steps of gradient descent in the few-shot setting. We start by repeating the multi-layer construction provided in von Oswald *et al.* [48] which allows a Transformer, without causal masking, and applied to the few-shot regression setting. This construction performs a gradient descent step per layer while simultaneously constructing a prediction on some final input. Compared to Section 3.2, we slightly change notation to easily bridge the gap to the au-

toregressive case in the next paragraph. Recall the squared error regression loss given $T-1$ data pairs $(s_{t'}, s_{t'+1})$

$$L(W^{(0)}) = \frac{1}{2}\sum_{t'=1}^{T-1}(W^{(0)}s_{t'} - s_{t'+1})^2,$$

with the gradient given by

$$\nabla_W L(W^{(0)}) = \sum_{t'=1}^{T-1}(W^{(0)}s_{t'} - s_{t'+1})s_{t'}^\top$$

inducing a change in the weights $\Delta W^{(0)} = -\eta\nabla_W L(W^{(0)})$. We now evaluate the the loss again at $W^{(1)} = W^{(0)} + \Delta W^{(0)}$:

$$\begin{aligned} L(W^{(1)}) = L(W^{(0)} + \Delta W^{(1)}) &= \frac{1}{2}\sum_{t'=1}^{T-1}((W^{(0)} + \Delta W^{(1)})s_{t'} - s_{t'+1})^2 \\ &= \frac{1}{2}\sum_{t'=1}^{T-1}(W^{(0)}s_{t'} - (s_{t'+1} - \Delta W^{(1)}))^2 \\ &= \frac{1}{2}\sum_{t'=1}^{T-1}(W^{(0)}s_{t'} - \tilde{s}_{t'+1}^{(1)})^2 \end{aligned}$$

with $\tilde{s}_{t'+1}^{(1)} = s_{t'+1} - \Delta W^{(1)}$. Note that when repeating this algorithm, we descend the regression loss after d steps of gradient descent by *transforming* the targets instead of updating the weights. Note that we are here not learning weights which we could use for test predictions. Nevertheless, when using the induced target transformation on some novel data point s_{T+1} while simultaneously transforming the targets of our dataset, we see that after a -1 correction i.e. $\hat{s}_{T+1} = W^{(0)}s_{T+1} + -1\sum_{l=0}^{d-1} -\Delta W^{(d)})s_{T+1}) = W^{(d)}s_{T+1}$, we obtain an equivalent prediction to the one of standard gradient descent. Note that the linear self-attention weight matrices provided in the main text for the single-step case directly implement this multi-step case when we restrict the attention to the first $T-1$ datapoints.

To implement this d-step algorithm in a Transformer, if all bias terms are zero, $W_k^\top W_q = \begin{pmatrix} 0 & 0 \\ 0 & I_s \end{pmatrix}$, and $PW_v = \begin{pmatrix} -\eta I_s & \eta W_0 \\ 0 & 0 \end{pmatrix}$, and the token at initialization is $e_t^{(0)} = (s_{t+1}, s_t)$ for the training data and $e_{T+1}^{(0)} = (-W_0 s_{T+1}, s_{T+1})$ for the test point, after d such layers, the last token in which we compute the test data prediction is transformed into $e_{T+1}^{(d)} =$

$(-W_0 s_{T+1} + \sum_{l=0}^{d-1} -\Delta W^{(d)}) s_{T+1}, s_{T+1})$. The y-component of this token contains again the (negative) of the prediction obtained by a linear model that underwent d steps of gradient descent. Therefore, configured as such, our d self-attention layers take d steps of mesa-gradient descent toward solving a least-squares problem. Note that in this construction the lower half of the tokens $e_t^{(l)} = (\cdot, s_t)\ \forall l$ keep unchanged throughout the Transformer forward pass.

***d*-layers of causally-masked self-attention can implement d steps of *online* gradient descent.**

To transfer the previous multi-layer construction to the autoregressive setting, we make two observations: (*i*) we now wish to make a prediction at every time step which will require more token space, and (*ii*) introducing causal masking will affect the computations carried out by the layer of von Oswald *et al.* [48] reviewed above, leading to T different models learned in parallel by an unusual variant of gradient descent.

To see (*i*), we note that at every point in time the Transformer needs to keep in memory throughout its forward pass not only the inputs but also the targets, $(s_{t'}, s_{t'+1})$ for $t' \in \{1, \ldots, T\}$. This is the case as we need both to transform some target $s_{t'}$ through gradient descent dynamics to implicitly keep updating the learned linear model, but also to use that same target $s_{t'}$ now as input to the implicitly learned model to construct a final output. This observation motivates the token construction $e_t = (-W_0 s_t, s_t, s_t, s_{t-1})$, where the last two entries are kept unchanged throughout the Transformer

forward pass. This token construction suggests the following weight configuration for a 2-headed linear self-attention layer:

$$W_{k,1}^\top W_{q,1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & I_s & 0 \end{pmatrix},$$

$$P_1 W_{v,1} = \begin{pmatrix} 0 & -\eta I_y & 0 & \eta W_0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$W_{k,2}^\top W_{q,2} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_s \end{pmatrix},$$

$$P_2 W_{v,2} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & -\eta I_y & 0 & \eta W_0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The dynamics induced by these two heads are equivalent but are *evaluated* in head 1 at the current $s_{t'}$ which we use to make the next token prediction and at 2 at the training data input $s_{t'-1}$, to change the moving targets based on our previously discussed gradient-based transformation of the targets.

Given these weights in layer l, we obtain the following change in the tokens

$$\begin{aligned} e_{t+1} = e_t + \Delta e_t^{(l)} &= (\hat{s}_{t+1}^{(l)}, \tilde{s}_t^{(l)}, s_t, s_{t-1}) + (-\Delta W_t^{(l)} s_t, -\Delta W_t^{(l)} s_{t-1}, 0, 0) \\ &= (\hat{s}_{t+1}^{(l)}, \tilde{s}_t^{(l)}, s_t, s_{t-1}) + (\Delta \hat{s}_{t+1}^{(l)}, \Delta \tilde{s}_t^{(l)}, 0, 0) \end{aligned}$$

with $\Delta W_t^{(l)} = -\eta \sum_{t'=1}^{t} (W_0 s_{t'-1} - \tilde{s}_{t-1}^{(l)}) s_{t'-1}^\top$ and $\hat{s}_{t+1}^{(0)} = -W_0 s_t$, $\tilde{s}_t^{(0)} = s_t$. Note that now in the causally-masked setting, the sum only runs to element t for the token e_t and that therefore the transformed targets each follow their own dynamics instead of the gradient summed across the sequence, as in the few-shot regression case. The above construction is equivalent to

the few-shot setting, for which we would want to make a prediction based on d steps of gradient descent for the test as well as the training data, i.e., at all points in time.

Furthermore, we can motivate this token update as the gradient of the time-dependent loss $L_t(W_0) = \frac{1}{2}\sum_{t'=1}^{t}(W_0 s_{t'-1} - \bar{s}_{t-1}^{(l)})^2$ for which again the targets at time step t change throughout the layers based on their specific target transformation mechanism. This online gradient descent algorithm was proved to be sub-optimal w.r.t. conventional (full-batch) gradient descent by Ding *et al.* [114]. They show that, in the limit of infinitely many layers, this construction implements online gradient descent, which does not coincide with the optimal (recursive) least-squares solution. Additionally, stochastic gradient descent with non-vanishing learning rates does not converge so the effective weight does not converge to the optimal solution when given infinitely many samples.

In the next section we will show how, at least in theory, multi-layer self-attention can implement a different algorithm which can lead to the desired result, even in the causally-masked setting. We end by noting that the aforementioned weight construction, despite being potentially sub-optimal, motivates our token construction $e_t = (0, s_t, s_t, s_{t-1})$ which we use throughout all experiments when training deep Transformers, i.e., whenever the model has more than one self-attention layer. Note that $-W_0 s_t = 0$ since we assume an $W_0 = 0$. Finally, we stress that until now it is not clear how to incorporate common $L2$ regularization into the (online) gradient descent dynamics. This however is now in the autoregressive case of particular importance: In the beginning of the sequence the causally masked Transformer is forced to solve an under-constrained learning problem, dependent on the input data dimension and data generation. The following paragraph will again provide a simple solution to this problem.

***d*-layers of causally-masked self-attention can approximate optimal preconditioned gradient descent.** Our results are influenced by the GD++ algorithm presented by von Oswald *et al.* [48], which can lead to accelerated optimization by applying a whitening transform to input data. We restate the goal of the autoregressive Transformer, namely, to solve the underlying least-squares problem for all time steps simultaneously. This amounts to computing $S_t S_{t-1}^{\top}(S_{t-1}S_{t-1}^{\top} + \frac{1}{\lambda}I)^{-1}s_t \quad \forall t$, a (recursive) least squares solution, where time-shifted (by one) sequence elements play the role of inputs and desired outputs in a dataset, with inputs S_{t-1}, targets S_t, and test input s_t.

With the limited expressivity of one layer, we have already established that Transformers can, and do, in various settings, implement a single gradient step on the corresponding regression problem $\sum_{t'=1}^{t-1}(s_{t'+1} - Ws_{t'})^2$ both in theory and in practice. We now diverge from the previous section which generalized a single mesa-gradient descent step to the multi-layer case. Instead, we argue here that the Transformer could solve the problem differently. Our key observation is that given a preconditioning matrix $H_t = (S_{t-1}S_{t-1}^\top + \frac{1}{\lambda}I)^{-1}$ which changes the loss as $\sum_{t'=1}^{t-1} \|s_{t'+1} - WH_t s_{t'}\|^2$, a gradient descent dynamics would converge in a single step to the regularized least-squares solution. This way, we do not apply several gradient steps, thereby circumventing the potential problems arising from causally-masked gradient descent dynamics on the targets discussed in the previous section.

Based on these insights, we provide a theoretical construction that shows how Transformers can approximate $(S_{t-1}S_{t-1}^\top + \frac{1}{\lambda}I)^{-1}q_t$ layer by layer in their forward pass, leading to improved single-step gradient descent performance. To do so, we build on the derivations of Section 3.7.2.3, where we showed how to implement an approximation of $\tilde{s}_t := (S_{t-1}S_{t-1}^\top + \frac{1}{\lambda}I)^{-1}s_t$ efficiently, and for all time steps t in parallel. We achieved this result with the help of the dot-product-attention (DPA) operation at the heart of linear and softmax self-attention layers, and by resorting to the truncated Neumann series.

First, we recall from Section 3.7.2.3 that we can approximate $\tilde{s}_t$ by $\tilde{s}_t^K$, obtained when multiplying s_t to the K-step truncated Neumann series approximating the inverse term $(S_{t-1}S_{t-1}^\top + \frac{1}{\lambda}I)^{-1}$, modulo normalization of the matrix, see 3.7.2.3. Then, we observe that the $\tilde{s}_t^K$ satisfy the following recursive relationship:

$$\begin{aligned}\tilde{s}_t^{k+1} &= \left(I - (S_{t-1}S_{t-1}^\top + \frac{1}{\lambda}I)\right)\tilde{s}_t^k + s_t \\ &= s_t + (1 - \frac{1}{\lambda})\tilde{s}_t^k - S_{t-1}S_{t-1}^\top \tilde{s}_t^k. \end{aligned} \tag{3.57}$$

We now see how, if $(\tilde{s}_t^k, s_t, s_{t-1})$ is present in the activations at layer k at time point t, then $\tilde{s}_t^{k+1}$ can be computed for the next layer. In particular, the last term in eq 3.57, can be obtained by one head of linear self-attention from the input $(\tilde{s}_t^k, s_t, s_{t-1})$ when $W_k^\top W_q = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ I_x & 0 & 0 \end{pmatrix}$, and $PW_v = \begin{pmatrix} 0 & 0 & -I_x \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$. The other terms are simple scaled additions to $\tilde{s}_t^k$ of $\tilde{s}_t^k$ itself and $\tilde{s}_t$ for

which many constructions exist. The latter information is also available at all times if not overwritten otherwise. Note that this weight construction strictly speaking only requires 3 channels i.e. $e_t = (s_t, s_t, s_{t-1})$ in which we update the first by the just provided Neumann series computation. Nevertheless, in practice we still use $e_t = (0, s_t, s_t, s_{t-1})$, i.e. tokens with additional memory. We observe in practice that both constructions e.g. $e_t = (0, 0, s_t, s_{t-1})$ or $e_t = (0, s_t, s_{t-1})$ reach similar performance but observe more training difficulties and instabilities for the more compact one. We also stress that the derivation presented here is one out of possibly many ways of how Transformers could implement and approximate the desired inverses $(S_{t-1}S_{t-1}^\top + 1/\lambda I)^{-1}s_t$ in parallel. This is the main reason we resorted to the probing analyses presented in the main text.

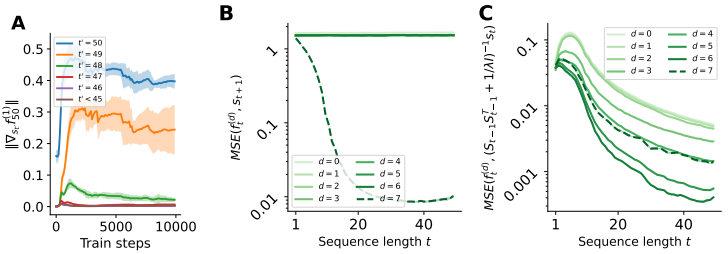


FIGURE 3.8: **Reverse-engineering full-fledged Transformers: Linear-Hybrid 1+6-layer model** (**A**) The first softmax layer groups together neighboring tokens. This can be seen in the high sensitivity to the current and previous tokens of the outputs of the first layer of a hybrid-linear Transformer. (**B** & **C**) We linearly regress the activations of each layer against final targets (**C**) as well as $(S_{t-1}S_{t-1}^\top + 1/\lambda I)^{-1}s_t$, the curvature-corrected inputs (**D**) predicted by the provided theory. We observe a harsh phase transition in the last layer when measuring target probing (**B**) while observing an intriguingly stable and gradual probing for curvature-corrected inputs (**C**), except for the last layer, where we hypothesize that the worse probing loss is explained by the computation of the actual predictions. Averages computed over 5 different seeds; shaded area represents standard deviation.

3.7.4 *Visualization of weights and attention maps of Transformers*

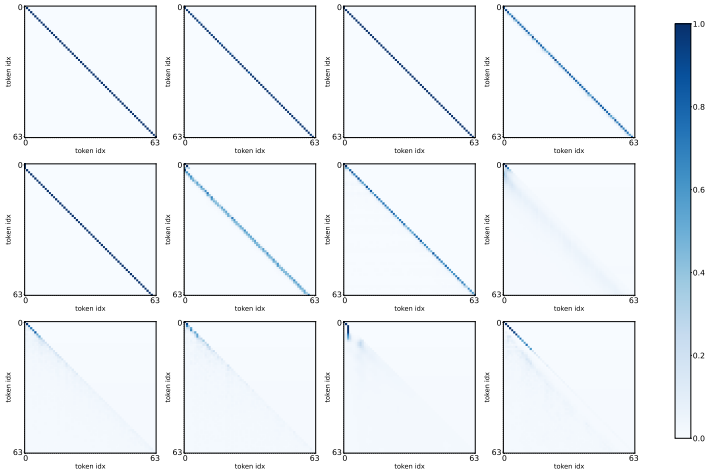


FIGURE 3.9: **Softmax attention maps of the 2-layer softmax-only Transformer trained on the Pile.** We average the attention maps of the first softmax-attention layer over a batch of size 256 and observe stable off diagonals with different offsets and widths indicating clean copying behavior based on positional encodings in multiple heads.

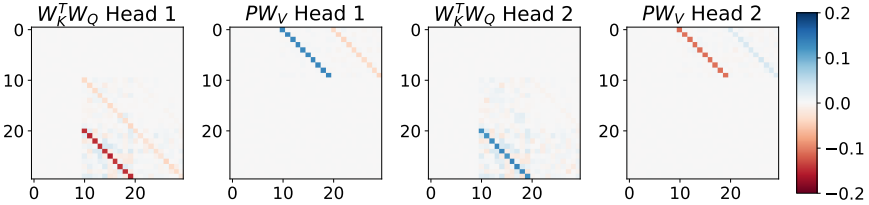


FIGURE 3.10: **Mesa-optimization in a trained linear self-attention layer.** We inspect the parameters of a two-headed, linear self-attention layer trained to predict the future state of a linear dynamical system. The dominant pattern obtained after learning corresponds to our mesa-gradient descent construction described in Section 3.3. The faint additional structure can be further reverse-engineered, and results from a modified mesa-objective function, Eq. 3.56, discovered by base-optimization of Eq. 3.11. Please compare to the similar structure of the weight matrix products of our construction.

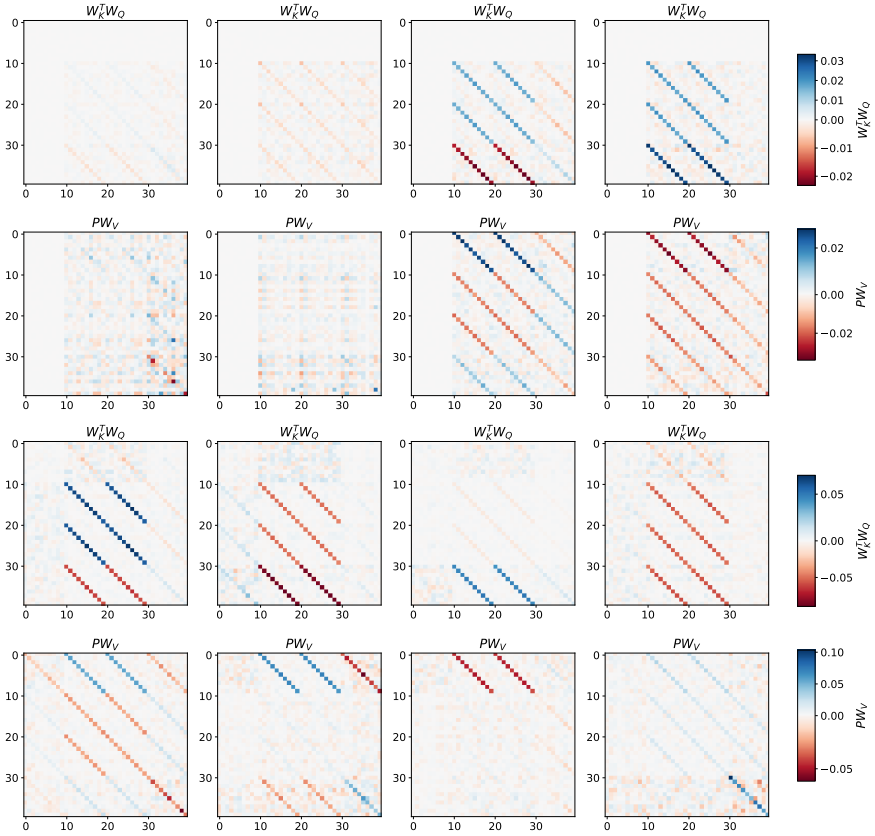


FIGURE 3.11: **Weights of the deep 6-layer linear Transformers trained on constructed tokens $e_t = (0, s_t, s_t, s_{t-1})$.** We observe clear structure in the trained Transformer weight products $W_K^\top W_Q$ as well as PW_V. Note that this structure seems to be sufficient to approximate $(S_{t-1}S_{t-1}^\top + 1/\lambda I)^{-1} s_t$, see probing experiment in the main text, Section 3.7.2.3 and Section 3.7.3. We show here all 4 heads (f.l.t.r.) of the first (top 2 rows) and the second (last 2 rows) linear layer.

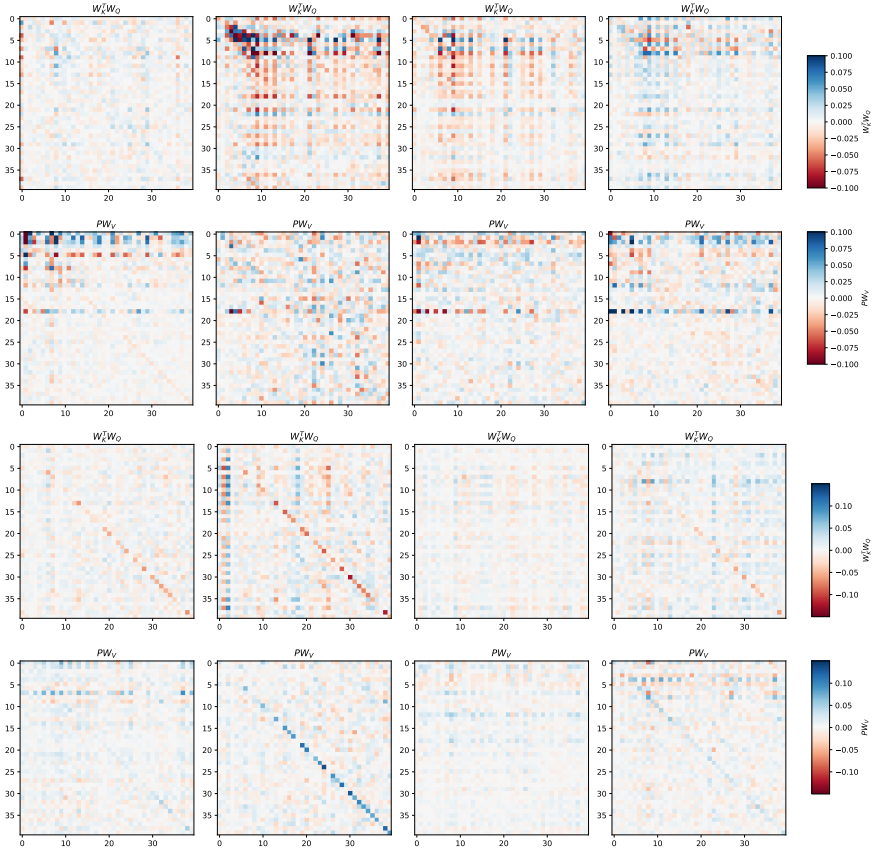


FIGURE 3.12: **Weights of the deep 1+6-layer linear-hybrid Transformers trained on unconstructed tokens $e_t = s_t$.** We observe some diagonal structure in the trained Transformer weight products $W_K^\top W_Q$ as well as PW_V. Note that this structure seems to be sufficient to approximate $(S_{t-1}S_{t-1}^\top + 1/\lambda I)^{-1}s_t$, see probing experiment in the main text, Section 3.7.2.3 and Section 3.7.3. We show here all 4 heads (f.l.t.r.) of the first (top 2 rows) and the second (last 2 rows) linear layer after the first softmax layer.

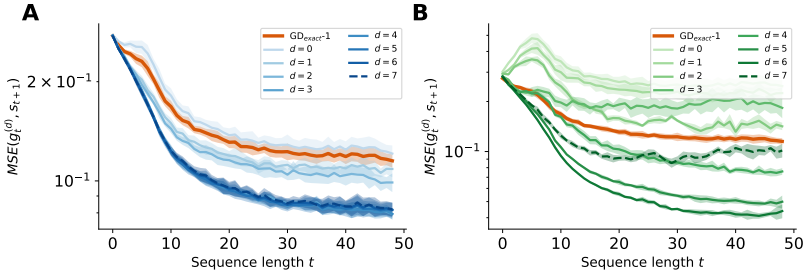


FIGURE 3.13: **Implicit target probing for full-fledged Transformers.** To further support the hypothesis that trained multi-layer Transformers first precondition constructed optimization problems by computing an, for example, an approximation of a truncated Neumann series of the required inverses before solving the optimization problems, we provide another probing analysis: For each layer, starting with the embedding layer at $d = 0$, we linearly regress the activations against the preconditioned inputs $(S_{t-1}S_{t-1}^\top + 1/\lambda I)^{-1}s_t$ and multiply these probes with $\eta S_t S_{t-1}^\top$ to compute a least-squares prediction approximation. We measure therefore measure the possibility to *implicitly* predict the target s_{t+1} from the hidden activations of the model. (**A**) For full-fledged softmax-only Transformers, we observe as expected, a gradual increase in probing performance across layers as expected at first, where we are able to outperform a step of gradient descent. (**B**) For the hybrid model, we find similar results: The probing performance gradually increases across layers and decreases for the last layer, were we hypothesize that the Transformer performs an update step of gradient descent to solve the well-conditioned optimization problem. Note that again, we are able to outperform a step of gradient descent. Averages computed over 5 different seeds; shaded area represents standard deviation.

4

LEARNING WHERE TO LEARN: GRADIENT SPARSITY IN META AND CONTINUAL LEARNING

This chapter's content was published in the Proceedings of the Conference on Neural Information Processing Systems (2021) and can be found online in an extended form including all experimental details which we omit here for clarity. The original publication is authored by Johannes von Oswald, Dominic Zhao*, Seijin Kobayashi, Simon Schug, Massimo Caccia, Nicolas Zucchet, João Sacramento [154].*

** These authors contributed equally.*

Finding neural network weights that generalize well from small datasets is difficult. A promising approach is to learn a weight initialization such that a small number of weight changes results in low generalization error. We show in the following that this form of meta-learning can be improved by letting the learning algorithm decide which weights to change, i.e., by learning where to learn. We find that patterned sparsity emerges from this process, with the pattern of sparsity varying on a problem-by-problem basis. This selective sparsity results in better generalization and less interference in a range of few-shot and continual learning problems. Moreover, we find that sparse learning also emerges in a more expressive model where learning rates are meta-learned. Our results shed light on an ongoing debate on whether meta-learning can discover adaptable features and suggest that learning by sparse gradient descent is a powerful inductive bias for meta-learning systems.

4.1 INTRODUCTION

Meta-learning holds the promise of discovering inductive biases that improve the performance of a primary learning process. Such a set of assumptions can materialize in various elements of the learner. The well-known model-agnostic meta-learning [MAML; 67] algorithm aims to learn a neural network initialization that generalizes well to new learning tasks. More sophisticated meta-learners augment this procedure by additionally modulating the inner-loop learning dynamics [79, 80, 82, 83, 155, 156].

It has been recently shown that applying MAML while adapting only last-layer weights leads to almost no decrease in performance in standard

few-shot learning benchmarks. Our study builds upon the surprising effectiveness of this form of meta-learning, known as almost no inner-loop training [ANIL; 47]. Here, instead of deciding which weights to freeze a priori, we endow the meta-learner with the possibility to explicitly stop changing certain weights in the inner-loop learning process. We do this by introducing an adjustable binary mask which is elementwise multiplied with gradient updates. This can be understood as a simple form of learned gradient modulation that induces sparsity. Overfitting can thus be prevented and learning sped up by focusing adaptation to a sparse parameter subset, discovered by meta-learning.

We find that our sparse-MAML algorithm recovers a behavior that is reminiscent of ANIL. It induces high gradient sparsity in earlier layers of the network while allowing for adaptation in deeper layers including the network's output. Despite this reduction in the number of adaptable parameters, the sparse learning patterns formed by sparse-MAML do not overly specialize to the family of tasks observed during meta-learning; both ANIL and MAML are outperformed by the resulting sparse learners in cross-adaptation problems involving a shift in task distribution [157, 158]. Furthermore, sparsity adapts intuitively to the number of inner-loop gradient steps as well as its learning rate, few-shot dataset size, and network specifications. This leads to a robust and interpretable variant of MAML that improves generalization by self-regularizing the parameters that the model should learn.

An exciting avenue of meta-learning research concerns continual learning. Learning tasks sequentially by gradient descent generally leads to poor results, as past tasks tend to be rapidly forgotten due to interfering weight updates. Such interference can be reduced with online meta-learning methods which optimize the base learning algorithm using both present and past data, kept in a replay buffer [159, 160]. Our findings translate to this setting. We analyze the state-of-the-art look-ahead MAML algorithm [La-MAML; 160] which introduces per-parameter meta-learned learning rates and find that sparse learning emerges, as a large fraction of learning rates drops to zero. Notably, similarly high performance can be reached when meta-learning binary gradient masks only. Moreover, performance improves after endowing a version of MAML adapted for online learning [161] with binary gradient masks. Thus, sparse learning can improve generalization, accelerate future learning, and reduce forgetting, and these benefits can be realized within online meta-learning.

4.2 FROM MAML TO SPARSE-MAML

MAML. The MAML algorithm seeks neural network weights θ from which only a few gradient descent steps suffice to reach high performance on a given task τ, that is assumed to be drawn from a certain distribution $p(\tau)$. Formally, a task is defined by an outer loss function L_τ^{out} and an inner loss function L_τ^{in}. We will later make explicit the form the two loss functions can take depending on the problem being solved. The result of the inner loss minimization is evaluated by the outer loss leading to the following optimization problem:

$$\min_\theta \mathbb{E}_{\tau\sim p(\tau)}\left[L_\tau^{\text{out}}(\phi_{\tau,K}(\theta))\right] \\ \text{s.t. } \phi_{\tau,k+1} = \phi_{\tau,k} - \alpha\,\nabla_\phi L_\tau^{\text{in}}(\phi_{\tau,k}) \text{ and } \phi_{\tau,0} = \theta,$$

with $\phi_{\tau,k}$ denoting the task-specific weights after k steps of gradient descent, α the inner-loop learning rate and K the inner-loop length. The initialization θ is then obtained by iterative updating, using

$$\theta \leftarrow \theta - \gamma_\theta\,\mathbb{E}_{\tau\sim p(\tau)}\left[\mathrm{d}_\theta\, L_\tau^{\text{out}}(\phi_{\tau,K}(\theta))\right], \tag{4.1}$$

with γ_θ the outer-loop learning rate. Note that we need the total derivative d_θ in Eq. 4.1 and not the partial derivative ∇_θ due to the complex relationship between $\phi_{\tau,k}$ and θ. In practice, the expectations over the task distribution that appear above are estimated by Monte Carlo integration. The updates in θ therefore correspond to stochastic gradient descent on the expected outer loss.

In MAML, the total derivative w.r.t. to θ is obtained by backpropagating through the inner optimization, a resource-intensive procedure. First-order MAML (FOMAML) drastically reduces the computational cost by setting to zero the second-order derivatives that appear when differentiating the inner-loop update.

LEARNING THE LEARNING RATES. Some variants of MAML focus on learning the learning rate and consider inner-loop updates of the following form:

$$\phi_{\tau,k+1} = \phi_{\tau,k} - \alpha\, M\, \nabla_\phi L_\tau^{\text{in}}(\phi_{\tau,k}), \tag{4.2}$$

for some learnable preconditioning matrix M, that is optimized similarly to the initialization θ. Through M, these algorithms learn some information on the geometry of the loss with the hope of faster inner-loop optimization.

Meta-SGD [79] considers a diagonal M, i.e., learnable learning rates, meta-curvature [81] considers a block matrix, while vanilla MAML corresponds to the $M = \mathrm{Id}$ case.

SPARSE-MAML. In line with these approaches, we introduce sparse-MAML. Together with an initial set of weights θ, our algorithm dynamically learns the parameters which will be updated and the ones that will not. Hence, sparse-MAML learns where to learn. To do so, we use a vector m (instead of a matrix M) that modulates the gradient in the inner-loop update in the following way:

$$\phi_{\tau,k+1} = \phi_{\tau,k} - \alpha \Big(\mathbb{1}_{m \geq 0} \circ \nabla_{\phi} L_{\tau}^{\text{in}}(\phi_{\tau,k}) \Big), \tag{4.3}$$

with $\mathbb{1}_{\cdot \geq 0} : \mathbb{R}^n \to \{0,1\}^n$ the step function that is applied elementwise to the underlying parameter vector $m \in \mathbb{R}^n$ and $\circ$ the pointwise multiplication. We differentiate the step function by considering it linear: this method is called the straight-through estimator [162] and it was recently used for similar large-scale masking [163]. Following FOMAML, we ignore second-order derivatives. This leads to the update

$$m \leftarrow m + \alpha \, \gamma_m \, \mathbb{E}_{\tau \sim p(\tau)} \left[\nabla_{\phi} L_{\tau}^{\text{out}}(\phi_{\tau,K}) \circ \sum_{k=0}^{K-1} \nabla_{\phi} L_{\tau}^{\text{in}}(\phi_{\tau,k}) \right]. \tag{4.4}$$

A detailed derivation of the mask update, alongside the presentation of the initialization update, can be found in the supplementary material (SM).

Our mask update depends on the alignment between the outer-loss gradient $g_{\tau}^{\text{out}} := \nabla_{\phi} L_{\tau}^{\text{out}}(\phi_{\tau,K})$ and the inner loss gradient $\overline{g}_{\tau}^{\text{in}} := \sum_{k=0}^{K-1} \nabla_{\phi} L_{\tau}^{\text{in}}(\phi_k)$ accumulated over the inner loop trajectory. Learning tends to be shut off on coordinates i for which these two quantities are of opposing sign, $\mathbb{E}_{\tau}\left[g_{\tau,i}^{\text{out}} \, \overline{g}_{\tau,i}^{\text{in}} \right] < 0$. Such freezing of learning when parameter updates are conflicting on the training and validation sets can decrease negative interference across tasks, which can in turn improve generalization performance [159, 164].

4.3 FEW-SHOT LEARNING

Finding a network that performs well when trained on few samples of unseen data can be formulated as a meta-learning problem. We study here the supervised few-shot learning setting where tasks comprise small labelled datasets. A loss function $\mathcal{L}(\phi, \mathcal{D})$ measures how much the predictions of a

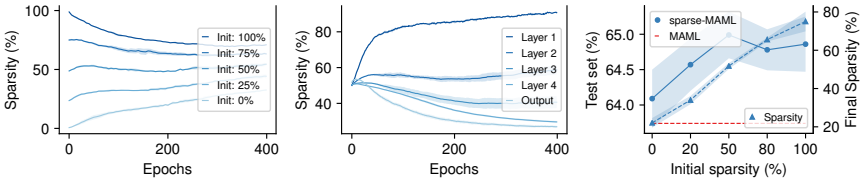


FIGURE 4.1: Gradient sparsity emerges in 5-shot, 5-way miniImageNet classification, standard ConvNet model. Results averaged over 5 seeds $\pm$ std. *Left*: Averaged gradient sparsity adapts for different sparsity initializations. *Center*: Different levels of gradient sparsity for convolutional and output layer weights emerge, with gradually less sparsity from earlier to deeper layers, while all being initialized at $\sim 50\%$ sparsity. *Right:* Sparse-MAML reaches higher test set accuracy for higher initial levels of gradient sparsity.

network parameterized by ϕ deviate from the ground truth labels on dataset $\mathcal{D}$. During meta-learning, the data of a given task τ is split into training and validation datasets, $\mathcal{D}_\tau^{\text{t}}$ and $\mathcal{D}_\tau^{\text{v}}$, respectively. The sparse-MAML formulation of few-shot learning then consists in optimizing the meta-parameters θ and m that, given the training set, in turn yield parameters ϕ that improve validation set performance:

$$\begin{aligned} &\min_{\theta} \ \mathbb{E}_{\tau\sim p(\tau)}[\mathcal{L}(\phi_{\tau,K}(\theta,m),\mathcal{D}_\tau^{\text{v}})] \\ &\text{s.t.} \quad \phi_{\tau,k+1} = \phi_{\tau,k} - \alpha\, \mathbb{1}_{m\geq 0} \circ \nabla_\phi \mathcal{L}\big(\phi_{\tau,k}, \mathcal{D}_\tau^{\text{t}}\big) \ \text{ and } \phi_{\tau,0} = \theta, \end{aligned} \tag{4.5}$$

This corresponds to setting the outer- and inner-loop loss functions introduced in Section 4.2 to $L_\tau^{\text{out}}(\phi) = \mathcal{L}(\phi, \mathcal{D}_\tau^{\text{v}})$ and $L_\tau^{\text{in}}(\phi) = \mathcal{L}(\phi, \mathcal{D}_\tau^{\text{t}})$.

We apply sparse-MAML to the standard few-shot learning benchmark based on the miniImageNet dataset [165]. Our main purpose is to understand whether our meta-learning algorithm gives rise to sparse learning by shutting off weight updates, and if the resulting sparse learners achieve better generalization performance. Furthermore, we analyze the patterns of sparsity discovered by sparse-MAML over a range of hyperparameter settings governing the meta-learning process.

Our experimental setup[1] follows refs. [67, 166] unless stated otherwise. In particular, by default, our experimental results are obtained using the standard 4-convolutional-layer neural network (ConvNet) model that has

1 Source code available at: `https://github.com/Johswald/learning_where_to_learn`

Method	1-shot	5-shot
MAML [67]	$48.07^{\pm 1.75}$	$63.15^{\pm 0.91}$
ANIL [47]	$46.70^{\pm 0.40}$	$61.50^{\pm 0.50}$
BOIL [158]	$49.61^{\pm 0.16}$	$66.45^{\pm 0.37}$
Meta-SGD [79]	$50.47^{\pm 1.87}$	$64.03^{\pm 0.94}$
MT-net [80]	$51.70^{\pm 1.84}$	—
MC (+data aug.) [81]	$54.23^{\pm 0.88}$	$68.47^{\pm 0.69}$
Shrinkage [156]	$47.7^{\pm 0.5}$	—
exp-MAML	$48.38^{\pm 0.45}$	$65.21^{\pm 0.62}$
sparse-ReLU-MAML	$49.84^{\pm 0.49}$	$66.80^{\pm 0.43}$
sparse-MAML	$50.35^{\pm 0.39}$	$67.03^{\pm 0.74}$
sparse-MAML^{+}	$51.04^{\pm 0.59}$	$68.05^{\pm 0.84}$

TABLE 4.1: 5-way few-shot classification accuracy (%) on miniImageNet, standard ConvNet model. We report mean $\pm$ std. over 5 seeds. All results except ours taken from the respective papers (we use the symbol '—' to indicate missing results). The results for meta-curvature (MC) are not directly comparable as additional data augmentation was used.

been intensively used to benchmark meta-learning algorithms. As is also conventional, we consider two data regimes: 5-shot 5-way, and 1-shot 5-way (the term 'shot' denotes the number of examples per class, and 'way' the number of classes). As we vary the hyperparameters of our algorithms, we monitor few-shot learning performance and the gradient sparsity level, defined for a parameter group or the entire network as $\|\mathbb{1}_{m<0}\|^2/\dim(m)$. All experimental details can be found in the SM.

4.3.1 *Gradient sparsity decreases with layer depth*

Our first finding validates and extends the phenomena described by Raghu et al. [47] and Chen et al. [156]. As shown in Figure 4.1, sparse-MAML dynamically adjusts gradient sparsity across the network, with very different values over the layers. As an example, we show the average gradient sparsity of the four convolutional weight matrices and the output layer during training. The same trend is observed for other parameter groups in the network except the output bias (for which sparsity is always high; see SM). Sparsity clearly correlates with depth and gradually increases towards the early layers of the network, despite the similar value before training (around 50%), i.e., sparse-MAML suppresses inner-loop updates of weights

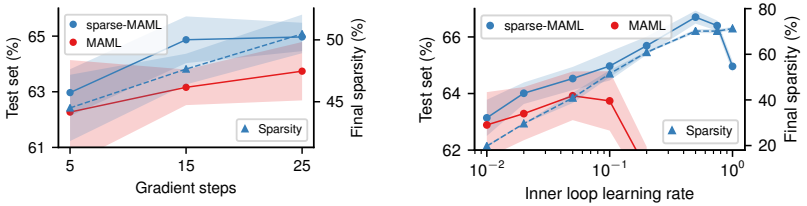


FIGURE 4.2: Sparse learning algorithms found by sparse-MAML work best in combination with highly-plastic models. Both gradient sparsity and generalization performance increase with number of inner-loop learning steps (*left*) and learning rate (*right*). Standard MAML, which does not employ sparse learning, requires more careful learning rate tuning and does not benefit as much from large learning rates. In all experiments, gradient sparsity is initially $\sim 50\%$. The inner-loop learning rate is set to 0.1 when varying the number of steps. Results are for 5-shot, 5-way miniImageNet, averaged over 5 seeds $\pm$ std.

in earlier layers while allowing deeper layers to adjust to new tasks. This effect is robust across different sparsity initializations, with final few-shot learning performance correlating with sparsity, cf. Figure 4.1.

These findings validate that our method can discover sparse learning algorithms. Moreover, they show that the level of sparsity is anti-correlated with depth. This result can be interpreted in the light of neural network models with human-engineered patterns of frozen features, which freeze layers of features based on their depth (in combination with MAML, see e.g. ANIL and BOIL, [47, 158]). Our method justifies these approaches, while outperforming them, cf. Table 4.1, suggesting that it might be preferable to meta-learn which features to freeze. We note that another related method for automatic discovery of task-shared weights based on learning per-parameter L_2 regularization strengths [Shrinkage, 156] yields a similar trend of high freezing for lower-level features, without however improving performance against standard MAML. Our findings hold when applying our method to a deeper and wider residual neural network (ResNet-12) model, see Tables 4.2 , where we observe the same trend of decreasing gradient sparsity with depth emerge.

4.3.2 *Sparse learning prefers highly-plastic models*

We hypothesize that restricting learning to an appropriate parameter subset allows for longer training and larger changes without overfitting, beyond meta-learning initial parameter values. To verify this hypothesis we scan over different inner-loop learning rates and lengths and compare the resulting test set performances of MAML and sparse-MAML.

First, we test three different inner-loop durations (5, 15 or 25 gradient steps, see Figure 4.2, left). We find that neither MAML nor sparse-MAML exhibit overfitting for the duration range considered here (for reference, the original study of MAML applied 5 inner-loop steps during meta-training). In contrast to MAML, the solutions found by sparse-MAML generalize significantly better for longer adaptation phases. This improvement in generalization performance is accompanied by an increase in gradient sparsity. Furthermore, applying sparse-MAML in the very-low data regime of 1-shot learning results in higher levels of gradient sparsity, even though the exact same model and training setup is used for both 1- and 5-shot learning experiments.

We further investigate if increasing the learning rate can result in improved generalization performance in combination with sparse learning. We scan the inner-loop learning rate over a large range, cf. Figure 4.2 (right), and find a clear trend towards gradient sparsity going along with better test-set accuracy for larger learning rates. Interestingly, similar effects have been reported in standard (non-meta-learned) neural network training where both freezing layers throughout training [167, 168] and the use of large learning rates [169] seem to improve generalization performance.

4.3.3 *Sparse learning vs. more expressive gradient modulation methods*

Sparse-MAML can be understood as a binary gradient modulation method. Second-order methods such as meta-curvature [81] modulate gradients by meta-learning pre-conditioning matrices; in meta-SGD [79], these matrices are restricted to be diagonal; sparse-MAML further restricts the diagonal values to be binary. From this point of view, sparse-MAML is the least expressive form of gradient modulation. Surprisingly, we find that despite its reduced expressiveness, sparse-MAML recovers the performance improvements achieved by the more sophisticated alternatives, significantly improving the performance of standard MAML (cf. Table 4.1). We point out that sparse-MAML uses a first-order update (Eq. 4.4), while all three

Method	1-shot	5-shot
MetaOptNet	51.13	70.88
MAML	$53.91^{\pm 0.61}$	$69.36^{\pm 1.23}$
ANIL	$55.25^{\pm 0.33}$	$70.03^{\pm 0.58}$
BOIL	—	$70.50^{\pm 0.28}$
sparse-MAML	$55.02^{\pm 0.46}$	$70.02^{\pm 1.12}$
sparse-ReLU-MAML	$56.39^{\pm 0.38}$	$73.01^{\pm 0.24}$

TABLE 4.2: 5-way few-shot classification accuracy (%) on miniImageNet with a ResNet-12 model. We report mean ± std. over 3 seeds. We report MetaOptNet [70] figures when no additional regularization techniques are applied. Results from BOIL and MetaOptNet are taken from the respective papers.

Method	1-shot	5-shot
sparse-ReLU-MAML	$77.53^{\pm 0.73}$	$73.53^{\pm 0.85}$
sparse-MAML	$79.04^{\pm 1.61}$	$74.98^{\pm 0.10}$
sparse-MAML^{+}	$78.05^{\pm 1.67}$	$76.66^{\pm 1.13}$

TABLE 4.3: Average gradient sparsity levels (%) after meta-learning on 5-way miniImageNet few-shot tasks, standard ConvNet model. Mean ± std. over 5 seeds.

gradient modulation methods we compare to (meta-SGD, meta-curvature and MT-nets) use second-order derivatives that are more costly to evaluate.

SPARSE LEARNING EMERGES WHEN META-LEARNING LEARNING RATES. We also implement a variant of meta-SGD which uses rectified learning rates (sparse-ReLU-MAML). Concretely, we replace the step function $\mathbb{1}_{m\geq 0}$ in the inner-loop dynamics (Eq. 4.3) by the positive part of m, $(m)_+ := \mathbb{1}_{m\geq 0}\, m$. Then, we learn the underlying learning rate parameter m using our first-order straight-through update of Eq. 4.4 to prevent learning rates from getting stuck at zero. Besides standard meta-SGD, which allows learning rates to go negative, we compare this method to an alternative exponential learning rate parameterization [170], $\exp m$, which like sparse-ReLU-MAML enforces non-negativity while avoiding permanently frozen updates (exp-MAML, Table 4.1). It is, however, harder to reach sparse

learning rate distributions under this parameterization, as the meta-gradient $\mathrm{d}_m \mathcal{L}$ becomes exponentially small as m approaches zero.

We analyze the distributions of learning rates that sparse-ReLU-MAML yields on miniImageNet and observe that gradient sparsity once more emerges, cf. Table 4.3. We find that the levels of gradient sparsity and generalization performance when meta-learning binary (sparse-MAML) or rectified learning rates (sparse-ReLU-MAML) are approximately the same, with both methods outperforming exp-MAML on both 1-shot and 5-shot tasks. These results support the hypothesis that shutting off weight updates is one of the essential gradient modulation operations in few-shot learning. We note that while sparse-MAML and sparse-ReLU-MAML quickly disable learning in a large fraction of weights, exp-MAML tends to push learning rates down, in particular for layers close to the input, but at a much slower pace; increasing the meta-learning rate γ_m cannot compensate for this slowdown as learning becomes unstable (data not shown).

This picture changes for the the deeper and larger ResNet-12 model, cf. Table 4.2. When using this more complex architecture, we find that sparse rectified learning rates (sparse-ReLU-MAML) are beneficial over binary gradient masks (sparse-MAML). In particular, the combination of sparse learning with learning rate modulation found by sparse-ReLU-MAML outperforms all other methods, including standard (dense-learning) MAML, as well as methods based on manually freezing layers in the inner-loop: BOIL [158], ANIL [47], and the closely-related MetaOptNet [70] method. Like ANIL, MetaOptNet only adapts the final classification layer in the inner-loop, but it uses a more sophisticated solver instead of a few steps of gradient descent to learn task-specific solvers. Thus, once more, learning by sparse gradient descent is an effective strategy to improve the generalization performance of a few-shot learner.

STOCHASTIC GRADIENT MASKING. We further investigate whether stochastic binary gradient masks can improve few-shot learning performance. Our interest in studying stochastic masks is two-fold: as a way to improve meta-optimization based on our straight-through estimator; and to determine if stochastic masking is beneficial at meta-test time. We thus investigate sparse-MAML$^+$, a variant of our algorithm in which gradient masks are generated from a low-dimensional Gaussian vector, with noise intensity determined by meta-learning (see SM). As before, we adjust meta-parameters using a first-order update. We find that this mask generation method does result in improved performance, cf. Table 4.1.

Problem	Method	TieredImageNet	CUB	Cars
1-shot	MAML	$51.61^{\pm 0.20}$	$40.51^{\pm 0.08}$	$33.57^{\pm 0.14}$
	ANIL	$52.82^{\pm 0.29}$	$41.12^{\pm 0.15}$	$34.77^{\pm 0.31}$
	BOIL	$53.23^{\pm 0.41}$	$\mathbf{44.20}^{\pm 0.15}$	$36.12^{\pm 0.29}$
	sparse-ReLU-MAML	$53.18^{\pm 0.52}$	$41.86^{\pm 0.95}$	$35.46^{\pm 0.67}$
	sparse-MAML	$53.47^{\pm 0.53}$	$41.37^{\pm 0.73}$	$35.90^{\pm 0.50}$
	sparse-MAML$^+$	$\mathbf{53.91}^{\pm 0.67}$	$43.43^{\pm 1.04}$	$\mathbf{37.14}^{\pm 0.77}$
5-shot	MAML	$65.76^{\pm 0.27}$	$53.09^{\pm 0.16}$	$44.56^{\pm 0.21}$
	ANIL	$66.52^{\pm 0.28}$	$55.82^{\pm 0.21}$	$46.55^{\pm 0.29}$
	BOIL	$69.37^{\pm 0.23}$	$60.92^{\pm 0.11}$	$50.64^{\pm 0.22}$
	sparse-ReLU-MAML	$69.06^{\pm 0.28}$	$59.55^{\pm 1.23}$	$51.21^{\pm 0.89}$
	sparse-MAML	$68.83^{\pm 0.65}$	$60.58^{\pm 1.10}$	$52.63^{\pm 0.56}$
	sparse-MAML$^+$	$\mathbf{69.92}^{\pm 0.21}$	$\mathbf{62.02}^{\pm 0.78}$	$\mathbf{53.18}^{\pm 0.44}$

TABLE 4.4: Few-shot classification accuracy (%) when meta-learning on miniImageNet but meta-testing on TieredImageNet, CUB and Cars. Mean ± std. over 5 seeds.

Interestingly, mask randomness is entirely suppressed by meta-learning; eventually, $\sigma \to 0$, and we recover a single deterministic mask m. The performance improvements observed on few-shot learning therefore stem from improvements to the meta-optimization process, likely related to the challenges of optimizing binary variables with (pseudo)gradient-based methods.

4.3.4 *Sparse learning improves performance in cross-domain adaptation tasks*

We now investigate whether the patterns of gradient sparsity discovered by our method overfit to the particular task family where they were obtained, namely, to few-shot miniImageNet classification tasks. This is an important question, since excessive parameter freezing may prevent adaptation to tasks that are too different from those presented during meta-learning.

We therefore move our analysis of few-shot learning to a cross-domain adaptation setting. In cross-domain adaptation problems, the family of tasks presented post-meta-learning to evaluate our algorithms is shifted by sampling classes from a different dataset. In particular, we train our meta-learner on the miniImageNet dataset and then evaluate learning performance on the TieredImageNet, CUB and Cars datasets. It has previously

been demonstrated that manually freezing either the head (BOIL) or the body (ANIL) during meta-testing improves performance in this setting [158], compared to letting all weights adapt (MAML). In Table 4.4 we compare the performance of our method to these baselines. We find that meta-learning the freezing pattern with sparse-MAML as opposed to manually selecting it consistently improves cross-domain adaptation.

4.4 CONTINUAL LEARNING

We now turn to a continual learning setting, where tasks must be learned sequentially. A successful continual learner is able to learn similar tasks faster, as in the few-shot learning case, while retaining high performance on previously seen tasks. We conjecture that sparse learning can improve memory retention and accelerate future learning by reducing interference with past updates.

4.4.1 *Gradient sparsity emerges when learning continually with Look-ahead MAML*

We investigate the benefits of sparse gradients in the recently proposed La-MAML algorithm [160]. This algorithm combines online meta-learning in conjunction with a small replay buffer which holds representative examples from the past in memory. Standard replay methods [171], define a joint objective using present and buffered data and directly optimize this objective. La-MAML follows a technique known as meta-experience replay [159] and introduces a bi-level optimization problem. The outer loss L^{out} is the multi-task objective optimized with standard replay methods, while the inner loss L^{in} is evaluated on the new incoming data only. Riemer *et al.* [159] have shown that such meta-learning promotes gradient alignment over tasks, which is a way to reduce interference [172].

Like the variants of MAML reviewed in Section 4.2, La-MAML introduces meta-learned per-parameter learning rates. We now briefly review a single iteration of the algorithm; complete pseudocode is provided in the SM. Each iteration of La-MAML consists of processing a new batch of data $\mathcal{B}$ as follows: (*i*) starting from $\phi_0 = \theta$ taking an inner-loop step on each sample k in $\mathcal{B}$, with $L_k^{\text{in}}(\phi_k) = \mathcal{L}(\phi_k, \mathcal{B}_k)$; (*ii*) defining an outer-loss $L^{\text{out}}(\phi_K, \mathcal{B} \cup \mathcal{R})$ on both new data $\mathcal{B}$ and a batch of past data $\mathcal{R}$ sampled from the replay buffer; (*iii*) taking an outer-loop step on the learning rate parameter using a first-order update followed by an outer-loop step (using the newly updated

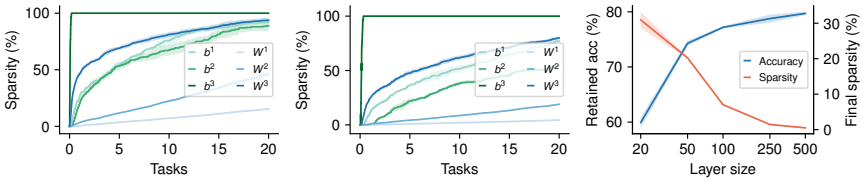


FIGURE 4.3: Gradient sparsity when learning MNIST rotations with the La-MAML and sparse-La-MAML algorithms. Results averaged over 3 seeds ± std. *Left*: Sparsity emerges on the original La-MAML algorithm across the three layer network and monotonically increases with the number of tasks and with depth for both weight (W_1, W_2, W_3) and bias parameters (b_1, b_2, b_3). *Center*: A similar behavior is observed when replacing meta-learned learning rates by meta-learned binary gradient masks (sparse-La-MAML). *Right:* Overall sparsity of sparse-La-MAML decreases with increased network capacity accompanied with higher retained accuracy (RA). Network capacity is varied by changing the number of neurons in the two hidden layers simultaneously.

learning rate) on the neural network parameters θ; (*iv*) re-populating the replay buffer with data in $\mathcal{B}$. The sequence of inner-loop updates is given by

$$\phi_{k+1} = \phi_k - (\alpha)_+ \circ \nabla_\phi L_k^{\text{in}}(\phi_k), \qquad \text{s.t. } \phi_0 = \theta, \tag{4.6}$$

where α is a vector of learning rates whose components are constrained to be non-negative by elementwise application of the positive part function. We note that while the main text of ref. [160] presents an inner-loop learning rate parameter that is allowed to go negative, the implementation for the experiments reported in ref. [173] uses rectified learning rates. In this implementation, a learning rate that is updated below zero will never recover, which can lead to dead coordinates and promote sparsity. The inner loss $L_k^{\text{in}}(\phi)$ is defined on a different data sample on each step k. A first-order update is applied to θ, again modulated by the adaptive learning rate:

$$\theta \leftarrow \theta - (\alpha)_+ \circ \nabla_\phi L^{\text{out}}(\phi_K). \tag{4.7}$$

SPARSE-LA-MAML. Our sparse-MAML can be readily applied to continual learning problems by modifying the inner- and outer-loop updates of La-MAML. We replace the meta-learned learning rates in equations 4.6-4.7 by meta-learned binary gradient masks, $\alpha = \alpha_0 \, \mathbb{1}_{m \geq 0}$, with $\alpha_0 \in \mathbb{R}_+$ some

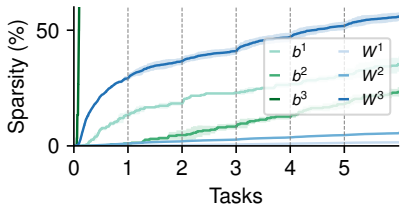


FIGURE 4.4: Structured sparsity emerges and tends to converge within a task in multi-pass continual learning. Results shown for La-MAML after training on MNIST rotations, averaged over 3 seeds $\pm$ std., for weight layers (W_1, W_2, W_3) and bias parameters (b_1, b_2, b_3). Sparsity increases with depth.

scalar (fixed) learning rate value. To learn the underlying parameter m, we again resort to our first-order update (equation 4.4).

SPARSE LEARNING IMPROVES CONTINUAL LEARNING. We hypothesize that a large fraction of learning rates approaches zero when the hyperparameters of La-MAML and sparse-La-MAML are tuned for best continual learning performance. To test this hypothesis, we follow the exact same setup as in the original study of La-MAML [160]. We perform a grid search over the learning rates α_0 and γ_m, and search for best continual learning performance, not sparsity (cf. SM). The remaining hyperparameters are kept to the values provided in [160].

We study the three MNIST [174] continual learning problems *rotations*, *permutations* and *many permutations* using a single-headed network using the code accompanying ref. [160]. Task information is not given to the network, and each data point is seen only once, unless noted otherwise. Full details as well as additional experiments using the CIFAR-10 [175] dataset are provided in the SM.

We verify that our initial hypothesis is correct: La-MAML shuts off learning in many coordinates (cf. Figure 4.3) reaching even higher levels of sparsity than sparse-La-MAML. This can be explained by the fact that dead coordinates can arise in La-MAML, which can lead to excess sparsity. By contrast, our straight-through update dynamically and continually adjusts the pattern of sparsity allowing previously frozen parameters to be unfrozen. This results in matching or slightly improved performances when using our binary gradient mask across all three MNIST variants, see Table 4.5, both in terms of final retained accuracy (RA) and backward-transfer

Method	Rotations		Permutations		Many permutations	
	RA	BTI	RA	BTI	RA	BTI
Baseline	$53.38^{\pm 1.53}$	$\mathbf{-5.44}^{\pm 1.70}$	$55.42^{\pm 0.65}$	$-13.76^{\pm 1.19}$	$32.62^{\pm 0.43}$	$-19.06^{\pm 0.86}$
GEM	$67.38^{\pm 1.75}$	$-18.02^{\pm 1.99}$	$55.42^{\pm 1.10}$	$-24.42^{\pm 1.10}$	$32.14^{\pm 0.50}$	$-23.52^{\pm 0.87}$
MER	$77.42^{\pm 0.78}$	$-5.60^{\pm 0.70}$	$73.46^{\pm 0.45}$	$-9.96^{\pm 0.45}$	$47.40^{\pm 0.35}$	$-17.78^{\pm 0.39}$
La-M	$77.42^{\pm 0.65}$	$-8.64^{\pm 0.40}$	$74.34^{\pm 0.67}$	$\mathbf{-7.60}^{\pm 0.51}$	$48.46^{\pm 0.45}$	$\mathbf{-12.96}^{\pm 0.07}$
sp-LaM	$\mathbf{77.77}^{\pm 0.58}$	$-8.16^{\pm 0.61}$	$\mathbf{76.88}^{\pm 0.72}$	$-8.39^{\pm 0.63}$	$\mathbf{50.81}^{\pm 0.79}$	$-13.73^{\pm 0.73}$

TABLE 4.5: Retained accuracy (RA) and backward-transfer and interference (BTI) for three different MNIST continual learning problems: rotations, permutations and many permutations. We report mean $\pm$ std. over 5 seeds. Negative BTI values closer to zero imply less forgetting and are therefore better. Results of related work are taken from [160]; for completeness we include the GEM [172] and MER [159] methods next to a stochastic gradient descent baseline. Although sparse-La-MAML (sp-LaM) is strictly less expressive than the original La-MAML algorithm, it shows competitive performance across all variants and both metrics. The lower baseline BTI values can be explained by lower overall accuracies achieved by La-MAML.

and interference (BTI; the change in accuracy measured at the end of the experiment minus just after learning a task, averaged over tasks). Moreover, the patterns of sparsity adjust to the capacity of the network, decreasing and eventually vanishing for larger models (Figure 4.3), as retained accuracy goes up, indicating that the task is not sufficiently difficult to create interference on large capacity models.

As in our few-shot learning experiments, structured sparsity emerges across the different parameter groups of the network (cf. Figure 4.3). We observe that now sparsity is highest closest to the output layer, the exact opposite of the trend found in our few-shot learning experiments. This provides evidence that online meta-learning can discover how to rewire low-level features without interference in order to accommodate different tasks that share high-level structure. We further investigate a multi-pass setting, where the examples from each task are visited multiple times (10 epochs instead of 1) before proceeding to the next task. In this setting, it can be seen that sparsity levels (displayed in Figure 4.4) tend to converge within tasks and then raise again when tasks switch, presumably to preserve past memories via gradient sparsification. Taken together, our results support the hypothesis that gradient sparsity is beneficial for continual learning and

Method	$p = 0.98$	$p = 0.9$
Online Adam [176]	$73.9^{\pm 2.2}$	$23.8^{\pm 1.2}$
Fine-tuning	$72.7^{\pm 1.7}$	$22.1^{\pm 1.1}$
MAML [67]	$84.5^{\pm 1.7}$	$75.5^{\pm 0.7}$
ANIL [47]	$75.3^{\pm 2.0}$	$69.1^{\pm 0.8}$
BGD [177]	$87.8^{\pm 1.3}$	$63.4^{\pm 0.9}$
MetaCOG [178]	$88.0^{\pm 1.0}$	$63.6^{\pm 0.9}$
MetaBGD [178]	$91.1^{\pm 2.6}$	$74.8^{\pm 1.1}$
C-MAML	$92.8^{\pm 0.6}$	$83.3^{\pm 0.4}$
sparse-C-MAML	$94.2^{\pm 0.4}$	$86.3^{\pm 0.4}$
sparse-ReLU-C-MAML	$93.5^{\pm 0.5}$	$86.1^{\pm 0.2}$

TABLE 4.6: Sparse learning improves continual-MAML performance. Cumulative online accuracy on Omniglot-MNIST-FashionMNIST benchmark. Tasks switch with probability $1 - p$. Results from previous work taken from [161]. Mean $\pm$ std. over 5 seeds.

that appropriate patterns of sparsity can be discovered by simple online gradient-based meta-learning.

4.4.2 *Sparse online learning*

We finally consider another online learning setting in which the underlying task is concealed from the learner and can randomly change at each step, potentially going back to previously seen tasks [161, 178, 179]. At each time step t, the data $\mathcal{D}_t$ is an i.i.d. sample from a stationary distribution that only depends on the current task. The learner, whose current state is denoted by ϕ_t, is evaluated whenever new data is presented and modifies its behavior accordingly. The goal is then to minimize the cumulative loss $\sum_{t=1}^{T} \mathcal{L}(\phi_t, D_t)$ measuring the performance before adaptation takes place.

This online learning protocol differs from the one adopted in the previous section, where tasks were visited only once and only the final loss $\sum_{t=1}^{T} \mathcal{L}(\phi_T, \mathcal{D}_t)$ evaluated at ϕ_T mattered. The cumulative loss criterion emphasizes fast learning and adaptability while memory is still needed to avoid re-learning, since tasks can be re-encountered. Recently, it has been shown that a simple modification of MAML [continual-MAML; 161] can outperform a number of algorithms specifically tailored for this setting as well as plain stochastic gradient descent [180]. Briefly, continual-MAML

extends MAML by introducing a task-switch detection mechanism based on changes in loss; data is buffered until a switch is detected. When this occurs, the buffered data is used to perform a meta-parameter update; the buffer is reset; and the inner-loop optimization restarts. Here, we merge continual-MAML with sparse-MAML, and modulate inner-loop gradients according to equation 4.3. We present complete pseudocode for the algorithm in the SM.

We reproduce the experiments of [161] in which a sequence of 10000 examples from the Omniglot [181], MNIST [174] and FashionMNIST [182] datasets is presented for online learning to a single-headed neural network, using the code provided by the authors. We carry out a grid search to tune the inner-loop learning rate α_0 and the mask learning rate γ_m introduced by sparse-MAML for best performance, not sparsity (see SM). We observe again structured (layer-dependent) gradient sparsity emerge when using this algorithm (sparse-C-MAML) and an increase in cumulative online learning accuracy over the original continual-MAML algorithm (cf. Table 4.6). Finally, we observe the same qualitative behavior (see SM for sparsity levels) and obtain similar performance when replacing our binary masks by rectified learning rates (sparse-ReLU-C-MAML) meta-learned with our straight-through update. These findings once more support the hypothesis that sparse learning, and not learning rate modulation, lead to the improved performance reported here.

4.5 DISCUSSION

We studied gradient-based meta-learning systems with the ability of learning where to learn. This was modeled by adding binary variables which masked gradients on a per-parameter basis, therefore determining which parameters are allowed to change. We observed gradient sparsity emerge in standard few-shot and continual learning problems, without introducing an explicit bias towards sparsity. This form of sparse learning, which may be understood as sparse gradient descent, was accompanied by overall improvements in generalization, as well as reduced interference and forgetting.

Previous work on gradient modulation has focused on estimating task-shared loss geometry to precondition the optimization procedure [79, 80, 82, 83, 155]. In addition, a stochastic variant of gradient masking was featured in the MT-net algorithm [80] as part of a more complex model. Our approach differs from these previous studies in its simplicity. We

restrict gradient modulation to be binary and deterministic and use an inexpensive first-order update to learn the gradient masks. In contrast to traditional methods for inducing sparsity via regularization [183] (here, gradient regularization) our approach does not require evaluating second derivatives, which would result from differentiating gradient regularizers. Despite these simplifications, we find competitive performance on our experiments. These results point towards sparse gradient descent as a powerful learning principle.

The idea of meta-learning learning rates can be traced back to the seminal work of Sutton [170], who proposed to estimate learning rate meta-gradients online using forward-mode automatic differentiation, and to use consecutive batches of data to define inner- and outer-loop loss functions. This approach, known as stochastic meta-descent (SMD), was extended to non-linear models by Schraudolph [184] using fast Hessian-vector product techniques. Using SMD to optimize neural network models is an ongoing area of research [185–188]. It is an interesting question whether gradient sparsity emerges when applying SMD to online learning problems that are not clearly structured in tasks, as considered here. Furthermore, this line of work suggests that it might be possible to obtain finer binary gradient mask updates in an online fashion using forward-mode automatic differentiation.

A recent study has put into question whether any useful adaptation still takes place when MAML few-shot learners are presented with a novel task after meta-learning [47]. Our findings shed light on this question, by demonstrating that few-shot learning performance can be improved when learning an adequate small subset of parameters. The additional plasticity of our meta-learned sparse learners led to a significant performance increase over handwired schemes based on frozen layers, in particular when encountering tasks drawn from a different family of problems than that used for meta-learning. This finding complements recent work showing that modular recurrent networks with sparse updating mechanisms hold great promise in improving out-of-distribution generalization performance [189, 190].

Our results may be of special interest to the design of neuromorphic hardware. Updating weights on-chip implies a significant power overhead whose cost scales with the number of plastic weights [191]. Reducing the number of plastic weights can therefore result in immediate improvements in energy efficiency and scalability. Likewise, synaptic plasticity is costly in biological neural networks. Given the high energy demands of the brain there has likely been selective pressure to reduce costs associated with

synaptic change [192]. It is therefore conceivable that the brain developed mechanisms to restrict learning to an appropriate subset of synapses to save energy. Our study presents further evidence in favor of sparse synaptic change, given its potential benefits in the biologically-relevant scenarios of few-shot and continual learning investigated here.

4.6 APPENDIX

We present here some additional results to complement the main results discussed in the previous sections.

4.6.1 *Derivation of the sparse-MAML update*

Here, we derive the sparse-MAML update rules on the initialization θ and on the underlying mask parameter m, that are given by

$$\theta \leftarrow \theta - \gamma_\theta \, \mathbb{E}_{\tau \sim p(\tau)} \left[\nabla_\phi \, L_\tau^{\text{out}}(\phi_{\tau,K}) \right] \tag{4.8}$$

$$m \leftarrow m + \alpha \, \gamma_m \, \mathbb{E}_{\tau \sim p(\tau)} \left[\nabla_\phi \, L_\tau^{\text{out}}(\phi_{\tau,K}) \circ \sum_{k=0}^{K-1} \nabla_\phi \, L_\tau^{\text{in}}(\phi_{\tau,k}) \right]. \tag{4.9}$$

UPDATE OF THE INITIALIZATION We first start by deriving the θ-update. To update θ with gradient descent we need the total derivative $\mathrm{d}_\theta \, L_\tau^{\text{out}}(\phi_{\tau,K})$. Using the chain rule, it is equal to

$$\mathrm{d}_\theta \, L_\tau^{\text{out}}(\phi_{\tau,K}) = \nabla_\phi \, L_\tau^{\text{out}}(\phi_{\tau,K}) \, \mathrm{d}_\theta \, \phi_{\tau,K}.$$

The last term of the right hand side of the previous equation requires back-propagating through the training procedure as modifying the initialization changes the entire trajectory of ϕ. By using the recursive formulation of $\phi_{\tau,K}$, we have

$$\begin{aligned} \mathrm{d}_\theta \, \phi_{\tau,K} &= \mathrm{d}_\theta \left[\phi_{\tau,K-1} - \alpha \, \mathbb{1}_{m \geq 0} \circ \nabla_\phi \, L_\tau^{\text{in}}(\phi_{\tau,K-1}) \right] \\ &= \mathrm{d}_\theta \, \phi_{\tau,K-1} - \alpha \, \mathbb{1}_{m \geq 0} \circ \left(\nabla_\phi^2 \, L_\tau^{\text{in}}(\phi_{\tau,K-1}) \, \mathrm{d}_\theta \, \phi_{\tau,K-1} \right). \end{aligned}$$

In sparse-MAML, we use a first-order approximation that consists in zeroing out all the second order derivatives to keep the computations as simple as possible, while keeping the benefits of meta-learning. It follows that

$$\begin{aligned} \mathrm{d}_\theta \, \phi_{\tau,K} &\approx \mathrm{d}_\theta \, \phi_{\tau,0} \\ &= \mathrm{d}_\theta \, \theta \\ &= \mathrm{Id} \end{aligned}$$

and

$$\mathrm{d}_\theta \, L_\tau^{\text{out}}(\phi_{\tau,K}) \approx \nabla_\phi \, L_\tau^{\text{out}}(\phi_{\tau,K}),$$

leading to the update presented in Eq. 4.8 once the derivative approximation is inserted in a gradient descent update.

In our online continual learning setting, we additionally apply the mask to the θ-update.

UPDATE OF THE MASK The derivation of the underlying mask parameter m update can be done similarly to the one of the θ-update. We first apply the chain rule and get

$$\mathrm{d}_m L_\tau^{\text{out}}(\phi_{\tau,K}) = \nabla_\phi L_\tau^{\text{out}}(\phi_{\tau,K})\, \mathrm{d}_m \phi_{\tau,K}.$$

We then compute the derivative of $\phi_{\tau,K}$ with respect to m:

$$\mathrm{d}_m \phi_{\tau,K} = \mathrm{d}_m \phi_{\tau,K-1} - \alpha\, \mathrm{d}_m \Big[\mathbb{1}_{m\geq 0} \circ \nabla_\phi L_\tau^{\text{in}}(\phi_{\tau,K-1}) \Big].$$

As for the θ-update, we do not take in account second-order derivatives, we thus consider first-order derivatives to be constant. The following terms remain

$$\mathrm{d}_m \phi_{\tau,K} \approx \mathrm{d}_m \phi_{\tau,K-1} - \alpha\, \mathrm{d}_m \left[\mathbb{1}_{m\geq 0}\right]\ \mathrm{diag}\Big(\nabla_\phi L_\tau^{\text{in}}(\phi_{\tau,K-1}) \Big).$$

We approximate $\mathrm{d}_m \mathbb{1}_{m\geq 0}$ using straight-through estimation, which consists in taking this derivative equal to the identity, thus having

$$\mathrm{d}_m \phi_{\tau,K} \approx \mathrm{d}_m \phi_{K-1} - \alpha\, \mathrm{diag}\Big(\nabla_\phi L_\tau^{\text{in}}(\phi_{\tau,K-1}) \Big)$$

and

$$\mathrm{d}_m \phi_{\tau,K} \approx -\alpha \sum_{k=0}^{K-1} \mathrm{diag}\Big(\nabla_\phi L_\tau^{\text{in}}(\phi_{\tau,k}) \Big).$$

Combining everything into a gradient descent update yields the update of Eq. 4.9.

Note that the updates for θ and m differ in their structure although both are obtained using first-order approximations. This is because θ only enters the first update step of ϕ, while m consistently appears along the whole trajectory of ϕ.

5

SUMMARY

In this thesis, I strive to use the tools of mechanistic interpretability to uncover the internal mechanism of how neural networks learn. With numerous tools pioneered in Neuroscience, mechanistic interpretability provided us with ideas and techniques to analyze, interpret, and whiten the artificial neural network black-box. This thesis can be positioned in a larger research field that works towards the goal of understanding how deep learning models work.

In particular, with a detailed analysis of the model's weights, the neuron's activations while processing information as well and computational interventions, we 1) shed light on learning mechanisms of deep neural networks and 2) used the insights obtained through mechanistic interpretability to develop novel algorithms. These novel algorithms while more powerful crucially offer more interpretability by design. In this thesis, I present two instances of an iterative framework which enables developing more interpretable models bottom-up. I argue that this repetitive procedure might be a path towards better controllability of AI models in the future.

In Chapters 2 & 3, I present MI work with particularly strong interpretability results focusing on in-context learning of Transformer models. Here, we were able to extract algorithms from the extremely sparse weights of Transformer models when trained on clean data and simplistic problems. In certain circumstances, our analyses led to a precise mathematical understanding of the inner workings of optimized models. Although this rigor seems unrealistic to obtain in models trained on real-world data, I believe these results present a fruitful path for interpretability research which first obtains a precise understanding of simplistic models and problems and translates these insights to developing capable but interpretable AI by design. Indeed, based on our MI work, in Chapter 3 we also present the mesa-layer which is a novel self-attention layer that is designed in support of Transformers implementing a least-squares solver in their forward pass observed in practice. This layer while leading to improved performance also offers better interpretability by design.

In Chapter 4, we propose a better interpretable variation of MAML, a prominent few-shot meta-learning algorithm. We build on previous MI work which identified that MAML, which seeks to find network weights that

allow quick adaptability to new tasks, finds a solution where only adapting the output layer of the model suffices, without significant accuracy loss. These findings suggest that MAML, without pressure to do so, finds a network initialization where only a few parameters should be updated when training on a novel task. Therefore, we propose to equip MAML with a learnable binary gradient mask and observe that indeed our new method termed sparse-MAML seeks to replicate MAML´s behaviour: sparse MAML induces high gradient sparsity for most network parameters while outperforming MAML in few-shot classification and continual learning benchmarks. We therefore again present a successful example of iterative interpretability. Here, we based our study on previous interpretability work [47] but again propose an algorithm change that leads to improved performance while offering better interpretability.

I end by pointing towards important shortcomings motivating future work and focus on more recent work presented Chapters 2 & 3.

- **Moving closer to LLMs with interpretable simplistic settings .** Various experimental decision in our simplistic problems studied deviates in various ways from how LLMs are set up and trained: LLMs are trained 1) to predict the correct next token i.e. on classification and 2) on highly complex data which requires to infer latent variables important to predict the most likely next token. Future work would address both of these shortcomings while, if possible, allow for interpretability.

- **Reverse engineering of LLMs.** It is unclear if our findings and interpretations, mostly obtained by studying small models trained on simplistic data, translate to the language domain. I believe that at least the mechanisms of in-context learning within small language models can be described to good extend by our insights. It remains nevertheless open if and how with larger size these mechanism might change. One direction could be to do extensively conduct interpretability work on pretrained open-source LLMs.

- **The tension between in-weights and in-context learning.** We focused in most settings on pure in-context learning behavior and not on tasks where global knowledge cloud be useful to be encoded in the weights. This tension is already discussed in [56] and poses the question how predictions, given a query, change due to the information given in-context. Although it is important to study this tension in simplistic setting, this direction has important implications for AI safety as it

is important and an open question if and how one can overwrite a models behaviour purely by information provided in context.

- **Studying other LLM characteristics.** The original motivation of the work presented in Chapters 2 & 3 is to understand the in-context learning capabilities of LLMs. This motivates the question if, with similar tools we used to study this LLM feature in isolation, other characteristics such as Chain-of-Thought prompting, reasoning abilities, instruction following and zero-shot performance can be studied.

- **More iterations of iterative interpretability.** In Chapter 3, we propose the mesa-layer based on our interpretability work. Note that this is not the only insight we could turn successful into an instance of iterative interpretability. For example, we could take our insights that Transformers implement fine-tuning algorithms literally. Here, future work could test how well simple deep networks fine-tuned with gradient descent on data given in-context behave and how similar these sequence dependent fine-tuned models to Transformers are.

Although the route of iterative interpretability seems a fruitful path to consider, it feels probable that we will quickly encounter limits to the simple interpretability techniques here [46, 193, 194]. Nevertheless, I am optimistic that mechanistic interpretability remains an exciting and highly valuable research direction for the coming years.

BIBLIOGRAPHY

1. Finn, C. & Levine, S. *Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm* in *International Conference on Learning Representations* (2018).
2. Olah, C. Interpretability dreams (2023).
3. Lipton, Z. C. The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability is Both Important and Slippery. (2018).
4. Grosse, R., Bae, J., Anil, C., Elhage, N., Tamkin, A., Tajdini, A., Steiner, B., Li, D., Durmus, E., Perez, E., Hubinger, E., Lukošiūtė, K., Nguyen, K., Joseph, N., McCandlish, S., Kaplan, J. & Bowman, S. R. Studying Large Language Model Generalization with Influence Functions. *arXiv* (2023).
5. Snodgrass, L. Sensation and Perception (2001).
6. Henning, C. *Knowledge uncertainty and lifelong learning in neural systems* PhD thesis (ETH Zurich, 2022).
7. Pearl, J. *Causality* 2nd ed. (Cambridge University Press, 2009).
8. Scherrer, N., Shi, C., Feder, A. & Blei, D. M. *Evaluating the Moral Beliefs Encoded in LLMs* 2023.
9. Anthropic. Core Views on AI Safety (2023).
10. Sussillo, D. & Barak, O. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Computation* **25**, 626 (3 2013).
11. Marschall, O. & Savin, C. Probing learning through the lens of changes in circuit dynamics. *bioRxiv* (2023).
12. Olah, C. Interpretability vs Neuroscience. *Colah's blog* (2021).
13. Jonas, E. & Kording, K. P. Could a Neuroscientist Understand a Microprocessor? *PLOS Computational Biology* **13**, 1 (2017).
14. Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J. & Hinton, G. Backpropagation and the brain. *Nature Reviews Neuroscience* **21**, 335 (2020).

15. Yamins, D. L. K. & DiCarlo, J. J. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience* (2016).

16. Hubel, D. H. & Wiesel, T. N. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology* **148**, 574 (1959).

17. Lyu, C., Abbott, L. F. & Maimon, G. Building an allocentric travelling direction signal via vector computation. *Nature* **601**, 92 (2021).

18. Khamsi, R. Jennifer Aniston strikes a nerve. *Nature* (2005).

19. Olah, C., Mordvintsev, A. & Schubert, L. Feature Visualization. *Distill* (2017).

20. Goh, G., †, N. C., †, C. V., Carter, S., Petrov, M., Schubert, L., Radford, A. & Olah, C. Multimodal Neurons in Artificial Neural Networks. *Distill* (2021).

21. Cammarata, N., Goh, G., Carter, S., Schubert, L., Petrov, M. & Olah, C. Curve Detectors. *Distill* (2020).

22. Holler, S., Köstinger, G., Martin, K. A., Schuhknecht, G. F. & Stratford, K. J. Structure and function of a neocortical synapse. *Nature* **591**, 111 (2021).

23. Krizhevsky, A., Sutskever, I. & Hinton, G. E. *ImageNet Classification with Deep Convolutional Neural Networks* in *Advances in Neural Information Processing Systems* (eds Pereira, F., Burges, C., Bottou, L. & Weinberger, K.) **25** (Curran Associates, Inc., 2012).

24. Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M. & Carter, S. Zoom In: An Introduction to Circuits. *Distill* (2020).

25. Voss, C., Goh, G., Cammarata, N., Petrov, M., Schubert, L. & Olah, C. Branch Specialization. *Distill* (2021).

26. Petrov, M., Voss, C., Schubert, L., Cammarata, N., Goh, G. & Olah, C. Weight Banding. *Distill* (2021).

27. Xu, C. S. *et al.* A Connectome of the Adult Drosophila Central Brain. *bioRxiv* (2020).

28. Deisseroth, K. Optogenetics: 10 years of microbial opsins in Neuroscience. *Nature Neuroscience* **18**, 1213 (2015).

29. Vaidya, A. R., Pujara, M. S., Petrides, M., Murray, E. A. & Fellows, L. K. Lesion studies in contemporary neuroscience. *Trends in Cognitive Sciences* **23**, 653 (2019).

30. Button, K. S., Ioannidis, J. P., Mokrysz, C., Nosek, B. A., Flint, J., Robinson, E. S. & Munafò, M. R. Power failure: Why small sample size undermines the reliability of neuroscience. *Nature Reviews Neuroscience* **14**, 365 (2013).

31. Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R. & Yu, B. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences* (2019).

32. Zhang, Y., Tino, P., Leonardis, A. & Tang, K. A Survey on Neural Network Interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence* (2021).

33. Schmidhuber, J. *Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook* Diploma Thesis (Technische Universitat Munchen, Germany, 1987).

34. Schmidhuber, J. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. *Neural Computation* **4**, 131 (1992).

35. Hospedales, T., Antoniou, A., Micaelli, P. & Storkey, A. Meta-Learning in Neural Networks: A Survey. *arXiv* (2020).

36. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. & Amodei, D. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165* (2020).

37. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q. & Zhou, D. *Chain-of-thought prompting elicits reasoning in large language models* in *Advances in Neural Information Processing Systems* **35** (2022).

38. Garg, S., Tsipras, D., Liang, P. S. & Valiant, G. *What can transformers learn in-context? A case study of simple function classes* in *Advances in Neural Information Processing Systems* **35** (2022).

39. Zhang, R., Frei, S. & Bartlett, P. L. Trained transformers learn linear models in-context. *arXiv preprint arXiv:2306.09927* (2023).

40. Mahankali, A., Hashimoto, T. B. & Ma, T. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. *arXiv preprint arXiv:2307.03576* (2023).

41. Ahn, K., Cheng, X., Daneshmand, H. & Sra, S. Transformers learn to implement preconditioned gradient descent for in-context learning. *arXiv preprint arXiv:2306.00297* (2023).

42. Li, Y., Ildiz, M. E., Papailiopoulos, D. & Oymak, S. *Transformers as algorithms: Generalization and stability in in-context learning* in *International Conference on Machine Learning* (2023).

43. Li, Y., Sreenivasan, K., Giannou, A., Papailiopoulos, D. & Oymak, S. Dissecting chain-of-thought: a study on compositional in-context learning of MLPs. *arXiv preprint arXiv:2305.18869* (2023).

44. Raventós, A., Paul, M., Chen, F. & Ganguli, S. Pretraining task diversity and the emergence of non-Bayesian in-context learning for regression. *arXiv* (2023).

45. Panigrahi, A., Malladi, S., Xia, M. & Arora, S. Trainable Transformer in Transformer. *arXiv* (2023).

46. Lillicrap, T. P. & Kording, K. P. What does it mean to understand a neural network? *arXiv* (2019).

47. Raghu, A., Raghu, M., Bengio, S. & Vinyals, O. *Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML* in *International Conference on Learning Representations* (2020).

48. Von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A. & Vladymyrov, M. *Transformers learn in-context by gradient descent* in *International Conference on Machine Learning* (2023).

49. Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S. & Olah, C. In-context Learning and Induction Heads. *arXiv preprint arXiv:2209.11895* (2022).

50. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. *Attention Is All You Need* 2017.

51. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. & Houlsby, N. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* in *International Conference on Learning Representations* (2021).

52. Yun, S., Jeong, M., Kim, R., Kang, J. & Kim, H. J. *Graph Transformer Networks* in *Advances in Neural Information Processing Systems* (eds Wallach, H., Larochelle, H., Beygelzimer, A., dÁlché-Buc, F., Fox, E. & Garnett, R.) (2019).

53. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A. & Zagoruyko, S. *End-to-End Object Detection with Transformers* in *Computer Vision – ECCV 2020* (Springer International Publishing, 2020).

54. Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y. & Pang, R. Conformer: Convolution-augmented Transformer for Speech Recognition. *arXiv preprint arXiv:2005.08100* (2020).

55. Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H. & Neubig, G. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *arXiv preprint arXiv:2107.13586* (2021).

56. Chan, S. C. Y., Santoro, A., Lampinen, A. K., Wang, J. X., Singh, A., Richemond, P. H., McClelland, J. & Hill, F. Data Distributional Properties Drive Emergent In-Context Learning in Transformers. *Advances in Neural Information Processing Systems* (2022).

57. Schmidhuber, J. *Evolutionary principles in self-referential learning, or on learning how to learn* Diploma thesis (Institut für Informatik, Technische Universität München, 1987).

58. Hinton, G. E. & Plaut, D. C. *Using fast weights to deblur old memories* in (1987).

59. Bengio, Y., Bengio, S. & Cloutier, J. *Learning a synaptic learning rule* tech. rep. (Université de Montréal, Département d'Informatique et de Recherche opérationnelle, 1990).

60. Chalmers, D. J. in *Connectionist Models* (eds Touretzky, D. S., Elman, J. L., Sejnowski, T. J. & Hinton, G. E.) 81 (Morgan Kaufmann, 1991).

61. Thrun, S. & Pratt, L. *Learning to learn* (Springer US, 1998).

62. Hochreiter, S., Younger, A. S. & Conwell, P. R. *Learning to Learn Using Gradient Descent* in *Artificial Neural Networks — ICANN 2001* (eds Dorffner, G., Bischof, H. & Hornik, K.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2001), 87.

63. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B. & de Freitas, N. *Learning to learn by gradient descent by gradient descent* in *Advances in Neural Information Processing Systems* (2016).

64. Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z. & Ionescu, C. *Using Fast Weights to Attend to the Recent Past* in *Advances in Neural Information Processing Systems 29* (2016).

65. Kirsch, L. & Schmidhuber, J. *Meta Learning Backpropagation And Improving It* in *Advances in Neural Information Processing Systems* (eds Beygelzimer, A., Dauphin, Y., Liang, P. & Vaughan, J. W.) (2021).

66. Schlag, I., Irie, K. & Schmidhuber, J. *Linear Transformers Are Secretly Fast Weight Programmers* in *ICML* (2021).

67. Finn, C., Abbeel, P. & Levine, S. *Model-agnostic meta-learning for fast adaptation of deep networks* in *International Conference on Machine Learning* (2017).

68. Finn, C. & Levine, S. *Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm* in *International Conference on Learning Representations* (2018).

69. Gordon, J., Bronskill, J., Bauer, M., Nowozin, S. & Turner, R. *Meta-Learning Probabilistic Inference for Prediction* in *International Conference on Learning Representations* (2019).

70. Lee, K., Maji, S., Ravichandran, A. & Soatto, S. *Meta-Learning With Differentiable Convex Optimization* in *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019).

71. Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S. & Hadsell, R. *Meta-Learning with Latent Embedding Optimization* in *International Conference on Learning Representations* (2019).

72. Von Oswald, J., Zhao, D., Kobayashi, S., Schug, S., Caccia, M., Zucchet, N. & Sacramento, J. *Learning where to learn: Gradient sparsity in meta and continual learning* in *Advances in Neural Information Processing Systems* (2021).

73. Hubinger, E., van Merwijk, C., Mikulik, V., Skalse, J. & Garrabrant, S. Risks from Learned Optimization in Advanced Machine Learning Systems. *arXiv [cs.AI]* (2019).

74. Kirsch, L., Harrison, J., Sohl-Dickstein, J. & Metz, L. *General-Purpose In-Context Learning by Meta-Learning Transformers* in *Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems* (2022).

75. Chan, S. C. Y., Dasgupta, I., Kim, J., Kumaran, D., Lampinen, A. K. & Hill, F. Transformers generalize differently from information stored in context vs in weights. *arXiv preprint arXiv:2210.05675* (2022).

76. Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Adler, T., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., Greiff, V., Kreil, D., Kopp, M., Klambauer, G., Brandstetter, J. & Hochreiter, S. Hopfield Networks is All You Need. *arXiv preprint arXiv:2008.02217* (2020).

77. Akyürek, E., Schuurmans, D., Andreas, J., Ma, T. & Zhou, D. *What learning algorithm is in-context learning? Investigations with linear models* in *The Eleventh International Conference on Learning Representations* (2023).

78. Garg, S., Tsipras, D., Liang, P. & Valiant, G. *What Can Transformers Learn In-Context? A Case Study of Simple Function Classes* in *Advances in Neural Information Processing Systems* (eds Oh, A. H., Agarwal, A., Belgrave, D. & Cho, K.) (2022).

79. Li, Z., Zhou, F., Chen, F. & Li, H. Meta-SGD: Learning to Learn Quickly for Few Shot Learning. *arXiv preprint arXiv:1707.09835* (2017).

80. Lee, Y. & Choi, S. *Gradient-based meta-learning with learned layerwise metric and subspace* in *International Conference on Machine Learning* (2018).

81. Park, E. & Oliva, J. B. *Meta-Curvature* in *Advances in Neural Information Processing Systems* (2019).

82. Zhao, D., von Oswald, J., Kobayashi, S., Sacramento, J. & Grewe, B. F. *Meta-Learning via Hypernetworks* in *NeurIPS Workshop on Meta-Learning* (2020).

83. Flennerhag, S., Rusu, A. A., Pascanu, R., Visin, F., Yin, H. & Hadsell, R. *Meta-Learning with Warped Gradient Descent* in *International Conference on Learning Representations* (2020).

84. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* 2014.

85. Benzing, F., Schug, S., Meier, R., von Oswald, J., Akram, Y., Zucchet, N., Aitchison, L. & Steger, A. Random initialisations performing above chance and how to find them. *OPT2022: 14th Annual Workshop on Optimization for Machine Learning* (2022).

86. Entezari, R., Sedghi, H., Saukh, O. & Neyshabur, B. The Role of Permutation Invariance in Linear Mode Connectivity of Neural Networks. *arXiv preprint arXiv:2110.06296* (2021).

87. Bertinetto, L., Henriques, J. F., Torr, P. H. S. & Vedaldi, A. *Meta-learning with differentiable closed-form solvers* in *International Conference on Learning Representations* (2019).

88. Widrow, B. & Hoff, M. E. *Adaptive Switching Circuits* in *1960 IRE WESCON Convention Record, Part 4* (IRE, New York, 1960), 96.

89. Zhmoginov, A., Sandler, M. & Vladymyrov, M. *HyperTransformer: Model Generation for Supervised and Semi-Supervised Few-Shot Learning* in *Proceedings of the 39th International Conference on Machine Learning* (eds Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G. & Sabato, S.) **162** (PMLR, 2022), 27075.

90. Amos, B. & Kolter, J. Z. *Optnet: Differentiable optimization as a layer in neural networks* in *International Conference on Machine Learning* (2017).

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

92. Gould, S., Hartley, R. & Campbell, D. J. Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

93. Zucchet, N. & Sacramento, J. Beyond backpropagation: bilevel optimization through implicit differentiation and equilibrium propagation. *Neural Computation* **34** (2022).

94. Dai, D., Sun, Y., Dong, L., Hao, Y., Ma, S., Sui, Z. & Wei, F. *Why Can GPT Learn In-Context? Language Models Implicitly Perform Gradient Descent as Meta-Optimizers* in *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models* (2023).

95. Nadaraya, E. A. On estimating regression. *Theory of Probability & its Applications* **9**, 141 (1964).

96. Watson, G. S. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, 359 (1964).

97. Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J. & Weller, A. *Rethinking Attention with Performers* in *International Conference on Learning Representations* (2021).

98. Zhang, A., Lipton, Z. C., Li, M. & Smola, A. J. Dive into Deep Learning. *arXiv preprint arXiv:2106.11342* (2021).

99. Irie, K., Schlag, I., Csordás, R. & Schmidhuber, J. Going Beyond Linear Transformers with Recurrent Fast Weight Programmers. *CoRR* **abs/2106.06295** (2021).

100. Von Oswald, J., Niklasson, E., Schlegel, M., Kobayashi, S., Zucchet, N., Scherrer, N., Miller, N., Sandler, M., y Arcas, B. A., Vladymyrov, M., Pascanu, R. & Sacramento, J. *Uncovering mesa-optimization algorithms in Transformers*

101. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. *Attention is all you need* in *Advances in Neural Information Processing Systems* **30** (2017).

102. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. & Amodei, D. *Language models are few-shot learners* in *Advances in Neural Information Processing Systems* **33** (2020).

103. Akyürek, E., Schuurmans, D., Andreas, J., Ma, T. & Zhou, D. *What learning algorithm is in-context learning? Investigations with linear models* in *International Conference of Learning Representations* (2023).

104. Kirsch, L., Harrison, J., Sohl-Dickstein, J. & Metz, L. *General-purpose in-context learning by meta-learning transformers* in *Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems* (2022).

105. Hubinger, E., van Merwijk, C., Mikulik, V., Skalse, J. & Garrabrant, S. Risks from learned optimization in advanced machine learning systems. *arXiv preprint 1906.01820* (2019).

106. Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J. & Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

107. Bahdanau, D., Cho, K. & Bengio, Y. *Neural machine translation by jointly learning to align and translate* in *International Conference of Learning Representations* (2015).

108. Katharopoulos, A., Vyas, A., Pappas, N. & Fleuret, F. *Transformers are RNNs: fast autoregressive transformers with linear attention* in *International Conference on Machine Learning* (2020).

109. Wang, S., Li, B. Z., Khabsa, M., Fang, H. & Ma, H. Linformer: self-attention with linear complexity. *arXiv preprint arXiv:2006.04768* (2020).

110. Schlag, I., Irie, K. & Schmidhuber, J. *Linear transformers are secretly fast weight programmers* in *International Conference on Machine Learning* (2021).

111. Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L. & Weller, A. *Rethinking attention with performers* in *International Conference of Learning Representations* (2021).

112. Fournier, Q., Caron, G. M. & Aloise, D. A practical survey on faster and lighter transformers. *ACM Computing Surveys* **55** (2023).

113. Treviso, M., Lee, J.-U., Ji, T., Aken, B. v., Cao, Q., Ciosici, M. R., Hassid, M., Heafield, K., Hooker, S., Raffel, C., Martins, P. H., Martins, A. F. T., Forde, J. Z., Milder, P., Simpson, E., Slonim, N., Dodge, J., Strubell, E., Balasubramanian, N., Derczynski, L., Gurevych, I. & Schwartz, R. Efficient methods for natural language processing: a survey. *Transactions of the Association for Computational Linguistics* **11** (2023).

114. Ding, N., Levinboim, T., Wu, J., Goodman, S. & Soricut, R. CausalLM is not optimal for in-context learning. *arXiv preprint arXiv:2308.06912* (2023).

115. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* in *Proceedings of NAACL-HLT* (2019).

116. Garnelo, M. & Czarnecki, W. M. Exploring the space of key-value-query models with intention. *arXiv preprint arXiv:2305.10203* (2023).

117. Sherman, J. & Morrison, W. J. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics* **21**, 124 (1950).

118. Gauss, C. F. *Theoria combinationis observationum: erroribus minimis obnoxiae* (Societas Regia Scientiarum Gottingensis, 1821).

119. Xie, S. M., Raghunathan, A., Liang, P. & Ma, T. *An explanation of in-context learning as implicit Bayesian inference* in *International Conference of Learning Representations* (2022).

120. Alain, G. & Bengio, Y. *Understanding intermediate layers using linear classifier probes* in *International Conference of Learning Representations* (2017).

121. Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S. & Olah, C. In-context learning and induction heads. *Transformer Circuits Thread* (2022).

122. Li, X. L. & Liang, P. *Prefix-tuning: optimizing continuous prompts for generation* in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics* (2021).

123. Lester, B., Al-Rfou, R. & Constant, N. *The power of scale for parameter-efficient prompt tuning* in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (2021).

124. Chan, S. C. Y., Dasgupta, I., Kim, J., Kumaran, D., Lampinen, A. K. & Hill, F. Transformers generalize differently from information stored in context vs in weights. *arXiv preprint arXiv:2210.05675* (2022).

125. Min, S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H. & Zettlemoyer, L. *Rethinking the role of demonstrations: what makes in-context learning work?* in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing* (2022).

126. Kossen, J., Rainforth, T. & Gal, Y. In-context learning in large language models learns label relationships but is not conventional learning. *arXiv preprint arXiv:2307.12375* (2023).

127. Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S. & Leahy, C. The pile: an 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027* (2020).

128. Fu, D. Y., Dao, T., Saab, K. K., Thomas, A. W., Rudra, A. & Ré, C. *Hungry hungry hippos: towards language modeling with state space models* in *International Conference of Learning Representations* (2023).

129. Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A. & Zhang, C. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749* (2023).

130. Martins, A., Farinhas, A., Treviso, M., Niculae, V., Aguiar, P. & Figueiredo, M. *Sparse and continuous attention mechanisms* in *Advances in Neural Information Processing Systems* **33** (2020).

131. Schmidhuber, J. Learning to control fast-weight memories: an alternative to dynamic recurrent networks. *Neural Computation* **4**, 131 (1992).

132. Hebb, D. O. *The Organization of Behavior: A Neuropsychological Theory* (Wiley, New York, 1949).

133. Hertz, J., Palmer, R. G. & Krogh, A. S. *Introduction to the Theory of Neural Computation* 1st (Perseus Publishing, 1991).

134. Widrow, B. & Hoff, M. E. *Adaptive switching circuits* in *IRE WESCON convention record* **4** (1960).

135. Ravi, S. & Larochelle, H. *Optimization as a model for few-shot learning* in *International Conference on Learning Representations* (2017).

136. Hochreiter, S., Younger, A. S. & Conwell, P. R. *Learning to learn using gradient descent* in *Artificial Neural Networks — ICANN 2001* (2001).

137. Lee, D.-H., Zhang, S., Fischer, A. & Bengio, Y. *Difference target propagation* in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2015).

138. Whittington, J. C. R. & Bogacz, R. An approximation of the error backpropagation algorithm in a predictive coding network with local Hebbian synaptic plasticity. *Neural Computation* **29**, 1229 (2017).

139. Meulemans, A., Zucchet, N., Kobayashi, S., von Oswald, J. & Sacramento, J. *The least-control principle for local learning at equilibrium* in *Advances in Neural Information Processing Systems* **35** (2022).

140. Hinton, G., Osindero, S. & Teh, Y. W. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation* **18**, 1527 (2006).

141. Nøkland, A. & Eidnes, L. H. *Training neural networks with local error signals* in *International Conference on Machine Learning* (2019).

142. Belilovsky, E., Eickenberg, M. & Oyallon, E. *Greedy layerwise learning can scale to ImageNet* in *International Conference on Machine Learning* (2019).

143. Löwe, S., O'Connor, P. & Veeling, B. *Putting an end to end-to-end: Gradient-isolated learning of representations* in *Advances in Neural Information Processing Systems* **32** (2019).

144. Hinton, G. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345* (2022).

145. Mumford, D. On the computational architecture of the neocortex. *Biological Cybernetics* **66**, 241 (1992).

146. Rao, R. P. N. & Ballard, D. H. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience* **2**, 79 (1999).

147. Bai, Y., Chen, F., Wang, H., Xiong, C. & Mei, S. Transformers as statisticians: provable in-context learning with in-context algorithm selection. *arXiv preprint arXiv:2306.04637* (2023).

148. Ha, D. & Schmidhuber, J. World models. *arXiv preprint arXiv:1803.10122* (2018).

149. Werbos, P. J. *Learning how the world works: Specifications for predictive networks in robots and brains* in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics, NY* (1987).

150. Giannou, A., Rajput, S., Sohn, J.-y., Lee, K., Lee, J. D. & Papailiopoulos, D. *Looped transformers as programmable computers* in *International Conference on Machine Learning* (2023).

151. Chan, S. C. Y., Santoro, A., Lampinen, A. K., Wang, J. X., Singh, A., Richemond, P. H., McClelland, J. & Hill, F. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems* **35** (2022).

152. Johnstone, R. M., Richard Johnson, C., Bitmead, R. R. & Anderson, B. D. O. Exponential convergence of recursive least squares with exponential forgetting factor. *Systems & Control Letters* **2**, 77 (1982).

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

154. Von Oswald, J., Zhao, D., Kobayashi, S., Schug, S., Caccia, M., Zucchet, N. & Sacramento, J. *Learning where to learn: Gradient sparsity in meta and continual learning* in *Advances in Neural Information Processing Systems* (2021).

155. Zintgraf, L., Shiarli, K., Kurin, V., Hofmann, K. & Whiteson, S. *Fast Context Adaptation via Meta-Learning* in *International Conference on Machine Learning* (2019).

156. Chen, Y., Friesen, A. L., Behbahani, F., Budden, D., Hoffman, M. W., Doucet, A. & de Freitas, N. *Modular Meta-Learning with Shrinkage* in *Advances in Neural Information Processing Systems* (2020).

157. Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C. F. & Huang, J.-B. *A closer look at few-shot classification* in *International Conference on Learning Representations* (2019).

158. Oh, J., Yoo, H., Kim, C. & Yun, S.-Y. *BOIL: Towards representation change for few-shot learning* in *International Conference of Learning Representations* (2021).

159. Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y. & Tesauro, G. *Learning to learn without forgetting by maximizing transfer and minimizing interference* in *International Conference on Learning Representations* (2019).

160. Gupta, G., Yadav, K. & Paull, L. *La-MAML: Look-ahead meta learning for continual learning* in *Advances in Neural Information Processing Systems* (2020).

161. Caccia, M., Rodriguez, P., Ostapenko, O., Normandin, F., Lin, M., Page-Caccia, L., Laradji, I. H., Rish, I., Lacoste, A., Vázquez, D. & Charlin, L. Online Fast Adaptation and Knowledge Accumulation (OSAKA): a New Approach to Continual Learning. *Advances in Neural Information Processing Systems* (2020).

162. Bengio, Y., Léonard, N. & Courville, A. Estimating or propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv preprint arXiv:1308.3432* (2013).

163. Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A. & Rastegari, M. *What's Hidden in a Randomly Weighted Neural Network?* in *IEEE Conference on Computer Vision and Pattern Recognition* (2020).

164. Nichol, A., Achiam, J. & Schulman, J. On First-Order Meta-Learning Algorithms. *CoRR* **abs/1803.02999**. _eprint: 1803.02999 (2018).

165. Ravi, S. & Larochelle, H. *Optimization as a model for few-shot learning* in *International Conference on Learning Representations* (2017).

166. Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K. & Wierstra, D. *Matching Networks for One Shot Learning* in *Advances in Neural Information Processing Systems* (2016).

167. Raghu, M., Gilmer, J., Yosinski, J. & Sohl-Dickstein, J. *SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability* in *Advances in Neural Information Processing Systems* (2017).

168. Brock, A., Lim, T., Ritchie, J. M. & Weston, N. FreezeOut: Accelerate Training by Progressively Freezing Layers. *arXiv preprint arXiv:1706.04983* (2017).

169. Lewkowycz, A., Bahri, Y., Dyer, E., Sohl-Dickstein, J. & Gur-Ari, G. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218* (2020).

170. Sutton, R. S. *Adapting bias by gradient descent: An incremental version of delta-bar-delta* in *AAAI Conference on Artificial Intelligence* (1992).

171. Robins, A. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science* **7**, 123 (1995).

172. Lopez-Paz, D. & Ranzato, M. *Gradient episodic memory for continual learning* in *Advances in Neural Information Processing Systems* (2017).

173. Gupta, G., Yadav, K. & Paull, L. *Official La-MAML repository* Downloaded from: `https://github.com/montrealrobotics/La-MAML/blob/main/model/lamaml.py`. Accessed: 2020-05-25. 2020.

174. LeCun, Y. The MNIST database of handwritten digits. *Available at http://yann. lecun. com/exdb/mnist* (1998).

175. Krizhevsky, A. *Learning multiple layers of features from tiny images* tech. rep. (2009).

176. Kingma, D. P. & Ba, J. *Adam: a method for stochastic optimization* in *International Conference on Learning Representations* (2015).

177. Zeno, C., Golan, I., Hoffer, E. & Soudry, D. Task Agnostic Continual Learning Using Online Variational Bayes with Fixed-Point Updates. *arXiv preprint arXiv:010.00373* (2020).

178. He, X., Sygnowski, J., Galashov, A., Rusu, A. A., Teh, Y. W. & Pascanu, R. Task agnostic continual learning via meta learning. *arXiv preprint arXiv:1906.05201* (2019).

179. Ritter, S., Wang, J., Kurth-Nelson, Z., Jayakumar, S., Blundell, C., Pascanu, R. & Botvinick, M. *Been there, done that: Meta-learning with episodic recall* in *International Conference on Machine Learning* (2018).

180. Bottou, L. in *On-line Learning in Neural Networks* 9 (Cambridge University Press, 1998).

181. Lake, B. M., Salakhutdinov, R., Gross, J. & Tenenbaum, J. B. *One shot learning of simple visual concepts* in *Proceedings of the Annual Meeting of the Cognitive Science Society* (2011).

182. Xiao, H., Rasul, K. & Vollgraf, R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

183. Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* **58**, 267 (1996).

184. Schraudolph, N. N. *Local gain adaptation in stochastic gradient descent* in *International Conference on Artificial Neural Networks* (1999).

185. Veeriah, V., Zhang, S. & Sutton, R. S. *Crossprop: Learning representations by stochastic meta-gradient descent in neural networks* in *Machine Learning and Knowledge Discovery in Databases* (2017).

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

187. Jacobsen, A., Schlegel, M., Linke, C., Degris, T., White, A. & White, M. *Meta-descent for online, continual prediction* in *AAAI Conference on Artificial Intelligence* (2019).

188. Kearney, A., Veeriah, V., Travnik, J., Pilarski, P. M. & Sutton, R. S. Learning feature relevance through step size adaptation in temporal-difference learning. *arXiv preprint arXiv:1903.03252* (2019).

189. Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y. & Schölkopf, B. *Recurrent independent mechanisms* in *International Conference on Learning Representations* (2021).

190. Madan, K., Ke, R. N., Goyal, A., Schölkopf, B. & Bengio, Y. *Fast and slow learning of recurrent independent mechanisms* in *International Conference on Learning Representations* (2021).

191. Park, J., Lee, J. & Jeon, D. A 65-nm neuromorphic image classification processor with energy-efficient training through direct spike-only feedback. *IEEE Journal of Solid-State Circuits* **55**, 108 (2019).

192. Li, H. L. & van Rossum, M. C. Energy efficient synaptic plasticity. *eLife* **9**, e50804 (2020).

193. Conmy, A., Mavor-Parker, A. N., Lynch, A., Heimersheim, S. & Garriga-Alonso, A. Towards Automated Circuit Discovery for Mechanistic Interpretability. *arXiv* (2023).

194. Gurnee, W., Nanda, N., Pauly, M., Harvey, K., Troitskii, D. & Bertsimas, D. Finding Neurons in a Haystack: Case Studies with Sparse Probing. *arXiv* (2023).

CURRICULUM VITAE

PERSONAL DATA

Name	Johannes von Oswald
Date of Birth	September 28, 1990
Place of Birth	Berlin, Germany
Citizen of	Germany

EDUCATION

2018 – 2023	Institute of Theoretical Computer Science, ETH Zürich, Switzerland *Final degree:* PhD
2014 – 2017	Technische Universität München München, Germany *Final degree:* Master of Science in Mathematics
2016	ETH Zürich Zürich, Switzerland *Semester Abroad*
2015	Hong Kong University of Science and Technology Hongkong *Semester Abroad*
2011 – 2014	Technische Universität Berlin Berlin, Germany *Final degree:* Bachelor of Science in Mathematics
2010	Werner-von-Siemens-Gymnasium Berlin, Germany *Final degree:* Abitur (university entrance diploma)

EMPLOYMENT (SELECTION)

2023 -	Research Scientist *Google* Zürich, Switzerland
Summer 2023	PhD Intern *Google* Seattle, USA
2022-2023	Research Student *Google* Zürich, Switzerland
2012-2014	Bartender & Co-Owner *Nice Bar* Berlin, Germany
2009-2011	Co-Founder & CTO *Rovo Agency* Berlin, Germany & Porto, Portugal

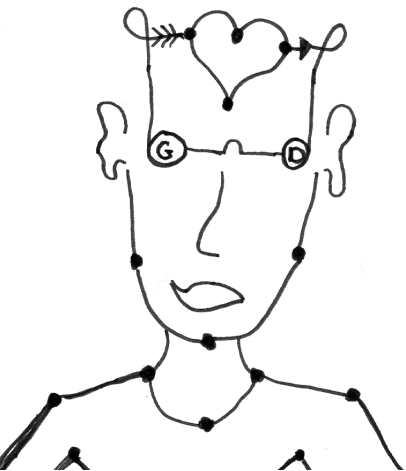
SELECTION OF RELEVANT OWN PUBLICATIONS

Preprints:

1. Von Oswald, J., Niklasson, E., Schlegel, M., Kobayashi, S., Zucchet, N., Scherrer, N., Miller, N., Sandler, M., y Arcas, B. A., Vladymyrov, M., Pascanu, R. & Sacramento, J. *Uncovering mesa-optimization algorithms in Transformers*

Conference and workshop contributions:

2. Von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A. & Vladymyrov, M. *Transformers Learn In-Context by Gradient Descent* in *Proceedings of the 40th International Conference on Machine Learning* (PMLR, 2023).
3. Von Oswald, J., Zhao, D., Kobayashi, S., Schug, S., Caccia, M., Zucchet, N. & Sacramento, J. *Learning where to learn: Gradient sparsity in meta and continual learning* in *Advances in Neural Information Processing Systems* (2021).
4. Zhao, D., von Oswald, J., Kobayashi, S., Sacramento, J. & Grewe, B. F. *Meta-Learning via Hypernetworks* in *NeurIPS Workshop on Meta-Learning* (2020).
5. Von Oswald, J., Henning, C., Grewe, B. F. & Sacramento, J. *Continual learning with hypernetworks* in *International Conference on Learning Representations* (2020).
6. Zucchet, N., Schug, S., Oswald, J. V., Zhao, D. & Sacramento, J. *A contrastive rule for meta-learning* in *Advances in Neural Information Processing Systems* (2022).


G
D