ARTIFICIAL MODELS OF BIOLOGICAL LEARNING

EXPLORING RECURRENCE AND DATA DIVERSITY

NICOLAS **ZUCCHET**

Diss ETH No. 31520

NICOLAS ZUCCHET

ARTIFICIAL MODELS OF BIOLOGICAL LEARNING: EXPLORING RECURRENCE AND DATA DIVERSITY

DISS. ETH NO. 31520

ARTIFICIAL MODELS OF BIOLOGICAL LEARNING: EXPLORING RECURRENCE AND DATA DIVERSITY

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by

NICOLAS ZUCCHET
MSc, ETH Zürich & École polytechnique

born on 16.01.1997

accepted on the recommendation of

Prof. Dr. Angelika Steger, examiner
Dr. J. Sacramento, co-examiner
Prof. Dr. B. Richards, co-examiner

2025

Nicolas Zucchet: *Artificial models of biological learning: exploring recurrence and data diversity,* © 2025

DOI: 10.3929/ethz-c-000787346

ABSTRACT

At the core of human intelligence lies the ability to learn and improve throughout life. Understanding the mechanisms underlying human learning remains one of science's most persistent challenges, complicated by the incremental, lifelong nature of learning processes and the difficulties of studying biological neural networks holistically and directly. This thesis addresses this fundamental question through artificial neural networks, which offer analytical tractability and experimental control while sharing core computational principles with biological systems.

We investigate learning through two complementary perspectives that illuminate different aspects of human learning. The first part examines the low-level mechanisms that could support learning in the brain, specifically focusing on how the recurrent connections essential for cognitive functions like working memory are learned. Our investigations reveal a previously unrecognized challenge we term the "curse of memory": networks that retain information longer become increasingly sensitive to parameter changes, making learning particularly unstable. We demonstrate that successful architectures used in practice overcome this challenge through modularity – relatively limited interactions between small components. Furthermore, we show that such modular organization makes biologically plausible online learning algorithms computationally tractable, suggesting why the brain's highly modular structure, exemplified in cortical columns, confers learning advantages.

The second part shifts focus on how data distribution shapes learning dynamics. We study how language models develop associative recall capabilities – the ability to store and retrieve cue-output pairs – finding that this capacity emerges suddenly during training. Surprisingly, repetition of specific associations accelerates this emergence, challenging conventional deep learning wisdom that emphasizes data diversity as paramount. Through theoretical analysis, we demonstrate that both the emergence and benefits of repetition stem from the development of selective attention, a mechanism needed across a wide range of cognitive tasks. Our findings thus suggest that reduced data diversity could be critical to rapid learning in artificial systems and humans.

These findings provide new insights into human learning efficiency while offering practical implications for artificial intelligence. Incorporating biophysical constraints brings computation closer to its physical substrate, making learning algorithms and architectures more efficient. Meanwhile, our data diversity findings suggest new training strategies for state-of-the-art deep learning systems. By bridging neuroscience and machine learning, this work demonstrates how studying artificial systems can illuminate biological intelligence while biological principles guide the development of more efficient artificial learning systems.

RÉSUMÉ

La capacité d'apprendre et de s'améliorer tout au long de la vie constitue le coeur de l'intelligence humaine. Comprendre les mécanismes qui sous-tendent cet apprentissage demeure l'un des défis scientifiques les plus importants de notre époque, rendu d'autant plus complexe par la nature progressive des processus d'apprentissage et la difficulté d'étudier directement les réseaux de neurones biologiques dans leur globalité. Cette thèse aborde cette question fondamentale par le prisme des réseaux de neurones artificiels, qui offrent à la fois une tractabilité analytique et un contrôle expérimental rigoureux, tout en partageant les mêmes principes computationnels que les systèmes biologiques. Notre approche explore l'apprentissage selon deux perspectives complémentaires, chacune éclairant des aspects distincts de l'intelligence humaine.

La première partie examine les mécanismes fondamentaux susceptibles de soutenir l'apprentissage cérébral, en se concentrant sur la façon dont s'acquièrent les connexions récurrentes, essentielles aux fonctions cognitives telles que la mémoire à court terme. Nos investigations révèlent un défi inédit que nous nommons la "malédiction de la mémoire" : les réseaux qui conservent l'information plus longtemps deviennent progressivement plus sensibles aux variations des paramètres appris, rendant l'apprentissage particulièrement instable. Nous démontrons que les architectures performantes utilisées en pratique surmontent ce défi en partie grâce à leur modularité – des interactions relativement limitées entre de petits composants spécialisés. Cette organisation modulaire rend également les algorithmes d'apprentissage en ligne biologiquement plausibles, computationnellement efficaces, suggérant pourquoi la structure hautement modulaire du cerveau, illustrée notamment par les colonnes corticales, confère de tels avantages adaptatifs.

La seconde partie se concentre sur la façon dont la distribution des données façonne la dynamique d'apprentissage. Nous étudions comment les modèles de langage développent leurs capacités de rappel associatif – l'aptitude à stocker et récupérer des paires indice-mémoire – et constatons que cette

capacité émerge de manière soudaine durant l'entraînement. De façon surprenante, la répétition d'associations spécifiques accélère cette émergence, remettant en question l'orthodoxie de l'apprentissage profond qui privilégie la diversité des données comme paramètre essentiel. Notre analyse théorique démontre que tant l'émergence que les bénéfices de la répétition découlent du développement de l'attention sélective, mécanisme fondamental requis dans un large éventail de tâches cognitives. Ces résultats suggèrent qu'une diversité de données réduite pourrait s'avérer critique pour un apprentissage rapide, tant dans les systèmes artificiels que biologiques.

Ces découvertes apportent un éclairage nouveau sur l'efficacité remarquable de l'apprentissage humain tout en offrant des implications pratiques concrètes pour l'intelligence artificielle. L'incorporation de contraintes biophysiques rapproche le calcul de son substrat physique, rendant les algorithmes et architectures d'apprentissage plus performants. Par ailleurs, nos résultats concernant la diversité des données ouvrent la voie à de nouvelles stratégies d'entraînement pour les systèmes d'apprentissage profond de pointe. En établissant des ponts entre neurosciences et apprentissage automatique, ce travail illustre comment l'étude de systèmes artificiels peut éclairer l'intelligence biologique, tandis que les principes biologiques guident le développement de systèmes d'apprentissage artificiel toujours plus efficaces.

ACKNOWLEDGEMENTS

No dissertation is written alone, and mine owes much to the many people who have supported, challenged, and inspired me.

First and foremost, I want to thank João for taking a chance on me when I was a Master student with no research experience. Over the last six years, you have taught me the arts and crafts of research that have made me the researcher I am today, and you have successfully transmitted your passion for research to me. I am grateful that what started as a research project has become a genuine friendship. I thank Angelika for building such a welcoming group where every moment is enjoyable. Your curiosity and willingness to always learn new things never cease to inspire me. I am grateful to Blake for serving as a committee member. Your interdisciplinary perspective and ability to connect topics across AI and neuroscience are always invaluable. I also want to thank Walter for the many passionate conversations about learning theories that we had. Though you were unable to serve as an official committee member, your intellectual contributions to this thesis are no less significant.

This journey was made particularly enjoyable by the many friends I made along the way, as well as by all the great scientists with whom I had the chance to collaborate. I thank all past and present group members for the passionate journal club discussions and fun times off, from our old J-floor terrace to the Stels and Buchboden retreats. Beyond this, I especially thank my office mates: Robert for the good times pushing ourselves on the bike, and Simon for being the ideal PhD companion, always one year ahead and showing me the way forward at each turn. I am also grateful to Soham for hosting my internship at Google DeepMind. You taught me how to focus on the most important and information-rich research paths.

Finally, I thank my family for cultivating my interest in science and engineering, for always nudging me toward tackling harder challenges, and for supporting me through each challenge. And Lisa, for making my life so much better.

CONTENTS

1 Introduction 1
1.1 Scope of the thesis 3
1.2 Biologically plausible learning of recurrent neural networks . 4
1.3 How does data shape learning? 6
1.4 A machine learning perspective on the thesis' results 8
2 Background 11
2.1 Neural architectures 11
2.2 Learning algorithms for neural networks 21

I Biologically plausible learning in recurrent neural networks
3 Vanishing and exploding gradients are not the end of the story 37
3.1 Introduction 37
3.2 Vanishing and exploding gradients 38
3.3 The curse of memory 39
3.4 Mitigating the curse of memory 45
3.5 A linear teacher-student analysis 48
3.6 Signal propagation in deep recurrent networks at initialization 53
3.7 Discussion 56
4 Online learning of long-range dependencies 57
4.1 Introduction 57
4.2 Background 59
4.3 Online learning of networks of independent recurrent modules 63
4.4 Experiments 67
4.5 Discussion 73

II How data shapes learning? A case study on associative memories and the role of data diversity
5 How do language models learn facts? 77
5.1 Introduction 77
5.2 An experimental setup to track knowledge 78
5.3 How language models acquire knowledge during learning . . 81
5.4 Data distributional properties drive knowledge acquisition . . 86
5.5 Hallucinations hinder the integration of new knowledge . . . 89
5.6 Discussion 91

6 The emergence of sparse attention 95
6.1 Introduction 95
6.2 Motivation 96
6.3 Theoretical insights on a linear regression task 97
6.4 Emergence of in-context learning 108
6.5 Discussion 113

Bibliography 117

III Appendix
A Vanishing and exploding gradients are not the end of the story 145
A.1 Definition of the recurrent networks we use 145
A.2 Theory 149
A.3 Linear teacher-student task 160
A.4 Signal propagation in deep RNNs 181
B Online learning of long-range dependencies 189
B.1 Network architecture and algorithm 189
B.2 Experimental details 192
C How do language models learn facts? 197
C.1 Related work 197
C.2 Experimental setup 200
C.3 Additional analysis of the learning dynamics (Section 5.3.1) . 205
C.4 Details of the mechanistic study 206
C.5 Additional analysis for the impact of data distribution 210
C.6 Details of the fine-tuning analysis 214
C.7 Hyperparameter configurations 225
D The emergence of sparse attention 231
D.1 Details of the theoretical results 231
D.2 Details and additional analysis for the linear regression task . 248
D.3 Details and additional experiments for the in-context associative recall task 254
D.4 Methods 257

NOTATION

SYMBOL	MEANING
h	hidden state (eventually indexed by layer l or time t)
x	input given to the network (eventually at time t)
y	output of the network (eventually at time t)
$\hat{y}$	predicted output of the network (eventually at time t)
θ	model parameters
W, A, B, C	weight matrices (eventually indexed by layer l)
b	bias vector (eventually indexed by layer)
λ	eigenvalue or recurrent parameter (for recurrent networks)
ρ	activation function
α	attention weight
L	number of layers (for deep networks)
T	sequence length
$\mathbb{E}$	expectation
$\mathcal{L}(\theta)$	expected loss function, as a function of the parameters
$\ell(y, \hat{y})$	loss measuring the discrepancy between a target and the output
∇_θ	gradient with respect to θ (column vector)
$\frac{d}{d\theta}$	total derivative with respect to θ (row vector)
$\frac{\partial}{\partial\theta}$	partial derivative with respect to θ (row vector)
$\delta = \frac{\mathrm{d}\ell}{\mathrm{d}h}^\top$	error term for backpropagation (eventually through time)
$s = \frac{\mathrm{d}h}{\mathrm{d}\theta}$	sensitivity matrix, for real-time recurrent learning

1

INTRODUCTION

How do we acquire new skills, adapt to changing environments, and refine our behavior throughout life? This fundamental question about learning has captivated researchers across disciplines [e.g., 1–3], yet understanding how humans learn remains a persistent challenge.

Part of what makes learning so difficult to study is its very nature: it consists in iteratively improving behavior through interactions with the external world [4]. These interactions are unique to each individual and exhibit tremendous variation depending on factors such as age, environment, and historical context. This diversity makes it particularly challenging to understand how interactions with the external environment shape learning. Moreover, learning occurs throughout an individual's entire lifespan, making it particularly time-consuming to experimentally test scientific hypotheses in real life. Even mammals with shorter lives than humans, such as rats, still require a few weeks of training before being able to perform cognitively relevant tasks.

At the mechanistic level, behavior changes through progressive refinement of neural computation, primarily through synaptic plasticity [2, 5, 6], though other mechanisms exist such as changes in dendritic morphology [7] or neuromodulation [8]. Neural computation is extremely complex, and despite more than a century of research ranging from studying the physiology of individual neurons [9–12] to large-scale brain imaging, a holistic understanding remains far from being achieved. Understanding how such a system changes is thus particularly challenging. While recent major advances in recording technologies have dramatically improved our ability to monitor neural activity [13–15], simultaneously recording all neurons while measuring the strength of synaptic connections connecting them remains beyond our technical reach. Even if such comprehensive recordings were available, we would require sophisticated computational models to make sense of this gigantic amount of data – models that we currently lack.

Given these fundamental challenges in studying biological learning directly, we pursue an alternative approach in this thesis: the study of artificial neural networks. This strategy aligns with the emerging field of NeuroAI [16, 17], which sits at the intersection of neuroscience and artificial intelligence. Rather than viewing these domains as separate, NeuroAI recognizes their potential for mutual enrichment – using biological insights to improve artificial intelligence systems while employing artificial intelligence as a lens to understand neural computation. In our case, we harness artificial neural networks to model biological learning and generate hypotheses about how these processes unfold in the brain. They are particularly valuable for our purposes as they trade biological realism for the scientific and mathematical tractability that biological networks lack.

Several properties make artificial neural networks particularly suited for this research agenda. Most notably, they demonstrate remarkable success across diverse cognitive tasks, including strategy game playing [18–20], natural language processing [21–23], and computer vision [24–26], often matching or surpassing human performance. These networks therefore display some form of intelligence acquired through learning that deserves scientific investigation.

Beyond their cognitive capabilities, their distributed architecture of interconnected processing units mirrors the organizational principles of biological neural networks. Though modern architectures have diverged considerably from their original biological inspiration, the field encompasses a spectrum of abstractions, including spiking neural networks [27] that more closely replicate biological neural interactions. This range allows researchers to select the appropriate level of biological fidelity for tractable analysis.

Perhaps most importantly for scientific study, their artificial nature provides unprecedented experimental control. Complete access to internal states and parameters enables mechanistic analysis [28–31], while researchers can precisely manipulate learning conditions by controlling environmental interactions. The ability to train networks from scratch – sometimes nearly instantaneously for smaller architectures – dramatically accelerates the scientific method.

1.1 SCOPE OF THE THESIS

This thesis uses artificial neural networks to understand how humans learn in two complementary ways. In the first part, we consider the following question:

What are the mechanisms that support learning in the brain?

One of the overarching goals of our approach in answering this question is to provide algorithms that satisfy, ideally all, the constraints that we know the brain has so that it could, in principle, learn following these algorithms. One convincing way of establishing that is to build physical devices, for example, neuromorphic chips [32, 33], that are akin to biological neural networks and train them on-device with those algorithms. Another goal of this research, which is shared with the broader fields of theoretical and computational neuroscience, is to provide hypotheses and experimental predictions that experimentalists could test in biological systems.

Here, we focus on a specific subquestion related to the learning of a specific connectivity pattern within neural networks: recurrence. Such a pattern is ubiquitous in the brain [34] and plays an important role in cognitive abilities like working memory [35]. Its learning is particularly complicated: learning such networks is difficult in general because they can be extremely sensitive to parameter changes, and because biophysically plausible learning algorithms have to satisfy constraints – specifically, causality – that the best algorithms we know do not satisfy. Our investigations reveal some of these difficulties and that the most successful artificial recurrent architectures are well equipped to solve this problem. They also show that the organization of the network into submodules, which the brain features, could greatly simplify learning. Throughout our investigations, we highlight that there is still much progress needed before being able to solve our leading question, and that we are still far from the ultimate goals stated above.

In the second part of this thesis, we focus on another aspect of learning, that is the interaction between the learning agent and its environment, specifically the data it receives from it. We study the question:

How does data shape learning?

The focus shifts from understanding the low-level mechanisms supporting learning akin to the work of a neuroscientist, to investigating the high-level effect of data distribution properties on behavior along the learning dynamics, as a psychologist would do. From this perspective, the internal specifics of the neural network we study matter less than the fact that it is a statistical learning machine that iteratively learns from data and processes information in a distributed manner, as we humans do. As we can easily control the properties of the data when training artificial neural networks, we can tweak and identify specific properties that control learning speed or generalization. In this thesis, we mostly focus on understanding how reduced data diversity, in modern language models first and then on a toy setting, can speed up learning without hurting generalization abilities. These results provide concrete examples in which reducing data diversity when learning progress stalls can be a powerful lever to master the task to be learned.

The following sections provide more context and detail about the two parts of the thesis, highlight their specific contributions, as well as highlight the relevance of these results from a pure machine learning perspective.

1.2 BIOLOGICALLY PLAUSIBLE LEARNING OF RECURRENT NEURAL NETWORKS

The renaissance of artificial neural networks in the 1980's was motivated by a key insight: instead of constraining learning algorithms to mimic known biological mechanisms like Hebb's rule [2], we should think from first principles about how learning could be optimally achieved, then later determine the physical implementation. This approach led to the development of backpropagation [36], which has powered the early successes of deep learning and provides a computational blueprint for how learning could be achieved from first principles. Biologically plausible implementations of this algorithm have been extensively investigated in the last decade [16, 37–45], leading to ideas about how error signals could be implemented biologically through dendritic compartments [40, 41], burst firing patterns [44], or changes in neural activity [39, 45]. These concepts are now being actively explored by experimental neuroscientists [e.g., 46].

However, most of this research focuses on memory-less feedforward neural networks, partly because spatial credit assignment of errors through the hierarchy of layers is a natural first problem to solve, and partly because early deep learning successes (mostly in computer vision) were primarily achieved with such architectures. But the brain is fundamentally a stateful machine, with neural activity, molecular concentrations, and synaptic states maintaining memory of recent events. Recurrent neural networks are well-suited to modeling these processes and can, for example, be understood as rough approximations of population-level neural dynamics [47].

Learning neural networks with memories is significantly more challenging because the temporal credit assignment problem must be solved in addition to spatial credit assignment. Indeed, to learn such networks, one must understand how updates to the network's current state will affect future behavior. Backpropagation-through-time [48] addresses this by sending error signals in reverse time, but this approach is unrealistic for any physically constrained process. This is not the only reason: even with arbitrary learning algorithms, it is difficult to learn recurrent networks. In this part of the thesis, we make two contributions: better understand the learning difficulties linked to these networks (Chapter 3) and show that specific neural architectures are particularly suited to physically plausible online learning (Chapter 4).

CHAPTER 3 – RECURRENT NEURAL NETWORKS: VANISHING AND EXPLODING GRADIENTS ARE NOT THE END OF THE STORY.

While vanishing and exploding gradients are well-known problems in recurrent networks [49–52], we reveal an additional issue that arises when training networks with long memories, which we call the curse of memory. The longer a network retains information in memory, the more sensitive it becomes to parameter changes, making learning increasingly unstable. Successful machine learning architectures (such as LSTMs [53] or deep state-space models [54]) mitigate this problem by careful architectural design: modularity, that is, relatively independent relationships between different network components, and reparameterization. This analysis emphasizes that progress in recurrent architectures in deep learning has been mostly achieved through careful architectural design and that we still do not really know how to optimize recurrent systems without these strong architectural constraints, brains being one such system.

CHAPTER 4 – ONLINE LEARNING OF LONG-RANGE DEPENDENCIES.
We then investigate methods to solve the temporal credit assignment problem in physically plausible ways. We build upon real-time recurrent learning [55, 56], a gradient estimation technique that operates fully online. The main limitation of this method is its computational and memory complexity. We demonstrate that modularity makes this form of gradient estimation much more tractable.

Both chapters suggest the importance of modularity to facilitate (online) learning. This resonates with the modular organization of the brain, exemplified by cortical columns [57] that are densely connected internally but exhibit sparse column-to-column communication. Further research is needed to better understand the conditions under which this form of modularity helps learning and whether brain-inspired architectures satisfy them.

1.3 HOW DOES DATA SHAPE LEARNING? A CASE STUDY ON ASSOCIATIVE MEMORIES AND THE ROLE OF DATA DIVERSITY

The artificial neural network renaissance mentioned in the previous section spurred an equally important revolution in cognitive psychology. Instead of the symbolic approach of the cognitive revolution, in which psychologists understood brain computations through computer metaphors, the parallel distributed processing [58] approach proposed that information is distributed and processed throughout the entire system rather than stored in discrete locations. This framework emphasized the importance of pattern recognition over pure symbolic manipulation. Learning shifted from being conceptualized as acquiring and storing discrete rules or facts, much like a computer loading new software modules, to being understood as a progressive statistical process in which connection strengths across networks of simple processing units are gradually adjusted. This connectionist framework proved particularly successful in naturally accounting for characteristic features of human cognition: the gradual improvement curves seen in human learning, our ability to recognize degraded or partial inputs, why similar memories tend to interfere with each other, and how abstract patterns can be extracted from specific examples without explicit rule instruction.

This approach has proven especially valuable for capturing learning dynamics and characteristic developmental errors, making it well-suited for investigating how data influences development. For example, analyses of deep linear networks have accounted for the progressive hierarchical differentiation between semantically different concepts observed in human development, while their mathematical tractability allows for precise theoretical understanding of how data distribution affects learning [59, 60]. Part ii contributes to this line of inquiry by studying the impact of data diversity on development through investigations of language models and simplified sequence models. Our results demonstrate an intuitive but previously unobserved phenomenon: when progress stalls on a task, reducing data diversity, and thus complexity, can facilitate learning breakthroughs. Chapter 5 examines such effects in how language models acquire factual knowledge, while Chapter 6 develops a simplified experimental setup that reproduces these findings and provides theoretical explanations for why reduced diversity can enhance learning.

Chapter 5 – How do language models learn facts? Dynamics, curricula and hallucinations

As a case study, we focus in this chapter on the learning dynamics underlying associative memory, a capability critical to intelligence that involves storing cue-output pairs so that the associated output can be retrieved given the cue. This basic form of intelligence has been extensively studied (e.g., Pavlovian conditioning experiments [61]) and modeled through computational frameworks like Hopfield networks [62]. We examine how mechanisms supporting parameter-based associative memory and recall develop during large language model training. Our analysis reveals that knowledge emerges after a phase transition – the network must encounter many cue-output pairs before developing the ability to recall associations. This setting additionally serves as a testbed for understanding how data shapes learning. We demonstrate that repetition (seeing some cue-output pairs more frequently than others) can accelerate the initial phase where no memory is acquired, speeding up overall learning. This finding is particularly interesting as it challenges much of the deep learning literature that emphasizes data diversity as paramount.

Chapter 6 – The emergence of sparse attention: impact of data distribution and benefits of repetition

We link the surprising effect of repetition from the last chapter to the learning of sparse attention. In this context, attention layers in Transformers learn to focus on small subsets of tokens within sequences, capturing common scenarios where desired outputs depend only on specific events in recent history. The associative recall studied in the previous chapter is an example of this: when a language model learns associative memory, it must extract relevant cues while ignoring surrounding text. In our simplified setting, we are able to reproduce the phase transition occurring during learning that we observe in Chapter 5 and can theoretically quantify the speedup provided by repetition.

Together, these chapters highlight the importance of strategic repetition in accelerating learning. Interestingly, human development involves progressively increasing environmental complexity as infants gradually broaden their range of actions and accessible experiences [63]. For example, newborns primarily see their parents' faces, which gradually diversifies to include other faces, toys, and the broader world. Understanding the tradeoffs in such diversification strategies could help partially explain why humans achieve remarkable sample efficiency.

1.4 A MACHINE LEARNING PERSPECTIVE ON THE THESIS' RESULTS

While motivated by the desire to understand how humans learn, the contributions of this thesis can also be interpreted from a pure machine learning perspective. The research papers forming it were actually mainly targeted to a machine learning audience and were published in machine learning conferences.

This alignment is to be expected given the intimate relationship between neuroscience and machine learning. Both fields aim to emulate intelligence through neural network models, though with different emphases. When deciphering how brains learn, neuroscience researchers ultimately seek to match known properties of biological neural networks and their corresponding physical constraints. When building artificial intelligence systems, the emphasis shifts to whatever works best given current hardware constraints. However, brains remain far more efficient in terms of training data requirements and energy consumption, suggesting that architectures and learning approaches that more closely match human behavior could significantly benefit artificial intelligence development.

The neural networks we study in the first part of the thesis – recurrent neural networks [64]– are among the most efficient architectures for handling sequential data, and are in particular more efficient than the Transformer architecture [21] that powers most modern deep neural networks [65, 66]. These properties have motivated recent research on deep state-space models [54, 67–71], which Chapters 3 and 4 directly study. The first of these chapters provides an optimization-centric explanation for their success: deep state-space models are relatively easy to learn due to their ability to solve the curse of memory while addressing vanishing and exploding gradient problems. The second chapter shows their particular suitability for online learning due to their organizational structure, which may prove valuable in applications like robotics, where learning agents must learn while operating in the real world – just as humans do.

The second part of the thesis focuses on understanding how language models, one of the most actively studied topics in contemporary deep learning, learn. The learning dynamics of these models remain poorly understood, yet opening this black box to understand how learning proceeds and how training data affects final model behavior represents a crucial step toward building trustworthy systems. As society becomes increasingly reliant on this technology, scientific understanding of these models has become a pressing concern. Chapter 5 focuses on associative memories, an area where large language models excel – sometimes to the point of being called "stochastic parrots." [72] By studying the mechanisms underlying the development of this ability and these associative memories, and revealing how data distribution influences this process (particularly demonstrating that strategic repetition can accelerate learning), this work contributes to our overall understanding of learning in language models while suggesting strategies for more efficient training of large language models. Chapter 6 identifies a simple setting in which the surprising benefits of repetition manifest. Deep learning theory often centers around surprising empirical puzzles like this one, such as grokking [73] or double descent [74], and this result may provide theorists with an angle of attack for demonstrating the benefits of non-i.i.d. training.

2

BACKGROUND

This section reviews the technical deep learning background needed for the thesis. It focuses on two aspects: neural architectures and learning algorithms.

2.1 NEURAL ARCHITECTURES

Artificial neural networks are, from their inception, closely related to biological systems, as they were originally introduced to model information processing in the brain. The networks we know today can be traced back to McCulloch and Pitts, who proposed in 1943 the first mathematical model of a neuron [75], demonstrating that networks of simple binary threshold units could, in principle, compute many logical functions (aside from the infamous `XOR` function [76]). This foundational work established the possibility that intelligence could emerge from the collective behavior of simple computational elements. With the perceptron, which was later introduced by Rosenblatt in 1958 [77], such a network could automatically adjust its parameters to improve performance on a given task. Since then, neural architectures have significantly evolved from these early brain-inspired models to become machines that excel at capturing the statistical regularities of the data they are exposed to. This section provides an introduction to some of the key artificial neural architectures that will be used within this thesis, highlighting, when possible, the links that still exist with their biological counterparts.

2.1.1 *Feedforward networks*

At its core, a feedforward neural network (sometimes called a multi-layer perceptron) is elegantly simple: a sequence of linear transformations interleaved with element-wise nonlinearities, such as the sigmoid or the ReLU [78] function. Mathematically, a feedforward neural network with L hidden layers is defined recursively as follows:

$$h_0 = x \tag{2.1}$$

$$h_l = \rho(W_l h_{l-1} + b_l) \quad \text{for } l = 1, \ldots, L \tag{2.2}$$

where W_l is the weight matrix and b_l is the bias vector of layer l, and ρ is a nonlinear activation function applied element-wise. The output of the network $\hat{y}$ is typically computed as a linear transformation of the final hidden state, often followed by a softmax for classification tasks.

Despite their simplicity, feedforward networks can express a wide range of functions. The seminal universal approximation theorems of Hornik, Stinchcombe & White [79] and Cybenko [80] established that such networks with a single hidden layer only can approximate any continuous function on a compact set, given sufficient hidden neurons. This theoretical foundation provided crucial validation for the neural network approach, but it left open the question of why depth should matter. Later analyses revealed that depth provides exponential advantages in expressivity. Montufar *et al.* [81] demonstrated that deep networks with ReLUs express functions whose complexity grows exponentially with depth, while [82] found that certain functions require exponentially many neurons in shallow networks but only a constant number in deep ones. These results formalized the intuition that hierarchical representations – where each layer builds increasingly abstract features from the outputs of previous layers – are fundamentally more efficient in terms of parameters than shallow representations.

Yet, deep networks remained largely impractical for decades mainly due to training difficulties linked to signal propagation. This core challenge is known as the vanishing and exploding gradient problem [49]: depending on the scale of the linear weights and the non-linearity, remote layers, i.e. the ones far from the output layer, tend to either receive close to no learning signals or exponentially large ones. This makes gradient-based optimization, the only viable optimization algorithm for optimizing networks with millions or more parameters, struggle with deep architectures. Proper initialization schemes, introduced by Glorot & Bengio [83] and refined by He *et al.* [84], ensured that gradients could flow effectively through deep networks from the start of training. The introduction of residual connections [26, 85], which allow information to bypass layers, and normalization layers [86, 87] further mitigates these issues and enables the training of very deep networks. This issue is not entirely solved and there is still active research on how to initialize best deep neural architectures so that the different training parameters can be transferred from small to large models [88–92].

The combination of these architectural and algorithmic innovations (not to forget convolutional neural networks [93, 94]), coupled with the availability of large datasets [95] and computational resources [96], unlocked the potential of deep feedforward networks [97]. Yet, these networks suffer from a fundamental limitation: they can only process fixed-size inputs, making them unsuitable for the variable-length sequences that characterize much of real-world data, such as text.

2.1.2 *Recurrent neural networks*

The inability of feedforward neural networks to deal with inputs of varying size motivated the development of recurrent neural networks, which introduced an additional memory into neural computation through the use of recurrent connections [98, 99]. This architectural choice has strong biological ties, as recurrent connections are ubiquitous in the brain [100, 101]. They are believed to support key cognitive mechanisms such as working memory [35, 102], attention modulation [103, 104], and memory retrieval [62, 105].

The mathematical insight behind recurrent networks is to use the same computational module repeatedly, maintaining a hidden state that serves as the network's memory. In its canonical form, a recurrent network is defined through the equation

$$h_{t+1} = \rho(Wh_t + b + Ux_{t+1}) \tag{2.3}$$

where h_t is the hidden state at time t, W represents the synaptic weights, b is the vector of biases, and x_t is the external input received by the network at time t. The output of the network at time t is

$$\hat{y}_t = Vx_t. \tag{2.4}$$

Such networks are akin to feedforward neural networks in which time replaces depth. However, there are a few notable differences that enable such networks to deal with arbitrarily long sequences: the parameters are tied between each update, and the network receives a new input at every time step. This will be particularly important for Chapter 3: we will see how these two minimal changes significantly modify signal propagation and the optimization properties of these networks.

In the following, we introduce the main classes of recurrent neural networks that we will consider in this thesis, starting from linear recurrent neural networks before moving to non-linear and gated networks. The following order of exposition focuses on increasing complexity and not necessarily on chronological order.

2.1.2.1 *Linear recurrent networks*

The simplest recurrent networks are linear. In this case, the hidden state evolves according to.

$$h_{t+1} = Ah_t + Bx_{t+1} \tag{2.5}$$

and the output is

$$\hat{y}_t = Ch_t + Dx_t \tag{2.6}$$

with A being the recurrent connectivity, B the input, C the readout and D the skip connection matrix. When the matrix A can be diagonalized, which is possible for almost all matrices[1] [106], the dynamics decouple into independent modes. More precisely, if A can be diagonalized as $A = P\Lambda P^{-1}$ with $\Lambda = \mathrm{diag}(\lambda_1, \cdots, \lambda_n)$ a matrix whose diagonal elements are the eigenvalues of A and P the change of basis matrix, the different modes h' evolve as

$$h'_{t+1} = Ph_t = \Lambda h'_t + PBx_t \tag{2.7}$$

and the output is

$$\hat{y}_t = CP^{-1}h_t. \tag{2.8}$$

The eigenvalues λ_i control the memory timescales of the network as well as its stability. When all the eigenvalues of the system have magnitude smaller than 1, the system is stable and it will converge when receiving to a constant input signal. Otherwise, it will be unstable and ultimately diverge. The precise magnitude of the eigenvalues controls the memory timescale: values close to 1 create long-term memory, while smaller values lead to rapid forgetting. Complex eigenvalues introduce oscillatory dynamics, enabling the network to capture periodic patterns in the data in a similar way to discrete Fourier transform [107].

---

[1] More formally, the set of complex-diagonalizable matrices of $\mathbb{R}^{n\times n}$ is dense in $\mathbb{R}^{n\times n}$.

Despite their simplicity, such networks do have practical applications, as the recent deep state-space models [54, 68, 69, 108] line of research highlights. There are two main reasons for this: parallelization and optimization. Parallelization is a prerequisite for any modern neural architecture due to the significant speed-up it brings and the widespread availability of parallel matrix multiplication hardware with GPUs. Recurrent linear networks can be parallelized, either by remarking that their transfer function[2] is a convolution or by leveraging associative scans [68, 109, 110]. Diagonalization simplifies the entire process as it reduces the computational footprint. Regarding optimization, most of the benefits of linear layers can be attributed to the fact they can be diagonalized. Indeed, by keeping eigenvalues to have their magnitude below 1, one can keep the dynamics stable and avoid exploding gradients. Additionally, they enable simple reparametrization that makes optimization much more well-behaved . We do not provide more details on these points here given that Chapter 3 will be dedicated to understanding the optimization properties of this type of neural networks in detail.

Linear recurrent networks are not a particularly expressive class of networks on their own. However, when paired with a multi-layer perceptron, they can in theory express any continuous function on a finite number of time-steps or when the target mapping satisfies a fading memory property [111–113]. This universality result holds both for complex diagonal matrices and, perhaps more surprisingly, for networks with diagonal real-valued connectivity[3]. However, the use of complex states can be particularly important in terms of numerical stability [113] and therefore simplifies learning.

Before concluding this section on linear recurrent networks, we note that such models have a rich history in control theory, a field in which they are known as state-space models and whose ideas have influenced the deep state-space model line of research mentioned above. These models, while not always perfectly accurate, enable researchers and engineers to systematically design optimal controllers through linear quadratic regulator methods [115], analyze fundamental system properties such as controllability and observability [116] and implement robust control strategies [117]. This framework additionally enables one to naturally deal with multiple

---

[2] The transfer function is the linear mapping between inputs x and the hidden states h encoded by the recurrent linear layers.

[3] In this case, we have what are called leaky integrator neurons in neuroscience [114].

inputs and outputs, or, as we shall see below, with time-varying systems, a feat that frequency-based approaches do not benefit from.

In this thesis, we will be interested in this type of network both from a practical perspective (Chapter 4), as they are the state-of-the-art recurrent architectures to model long-range dependencies, but also from a theoretical perspective (Chapter 3), as they are undoubtedly the simplest recurrent architecture that one can study.

2.1.2.2 *Non-linear recurrent networks*

The linear recurrent networks introduced in the previous section have seen surprisingly limited adoption in the machine learning literature until recently, primarily due to their restricted expressivity when used in isolation. The canonical recurrent architecture is the nonlinear recurrent network, which incorporates nonlinear activation functions as shown in equation 2.3. This architectural form demonstrates substantially greater expressiveness than its linear counterpart. Notably, nonlinear recurrent networks are Turing complete [118], meaning they can emulate any Turing machine, and serve as universal approximators for dynamical systems [119, 120].

Beyond their computational capabilities, these networks maintain strong connections to neuroscience, as they can be interpreted as discretizations of first-order differential equations of rate-based neural models. These models, first established by Wilson & Cowan [47], characterize the population-level or temporally-averaged dynamics of spiking neural networks. Given this biological grounding, it is unsurprising that such models – and particularly the diverse dynamics they can exhibit – have been extensively studied in the theoretical neuroscience literature [62, 104, 121–127].

However, this enhanced expressive power comes at a considerable cost in terms of optimization difficulty, ultimately leading the machine learning community to largely abandon this architecture. Training nonlinear recurrent networks presents notorious challenges, primarily stemming from the vanishing and exploding gradient problems [50–53]. This optimization challenge proves more intractable than in linear networks because the eigenvalues of the recurrent Jacobian – which determine whether gradients vanish or explode – are significantly more difficult to control. We provide a detailed review of these considerations in Section 2.2.4.1. Despite decades of research, no universally accepted solution to this fundamental problem exists for nonlinear recurrent architectures. Resolving this optimization challenge represents one of the major obstacles toward mechanistically understanding how biological neural networks learn.

2.1.2.3 *Gated recurrent networks*

Another approach to incorporating non-linearity into linear recurrent neural networks involves making the recurrent and input parameters dependent on the input and/or hidden state. This principle forms the foundation of gating mechanisms, which were first pioneered by the LSTM (Long Short-Term Memory) architecture [53]. Recently, these foundational ideas have been revisited and refined in light of advances in deep state-space model architectures [70, 71, 128–130]. While the LSTM architecture represents a sophisticated system with numerous components that have evolved considerably over the years [131], we focus here on introducing gating in its simplest form. The core gating mechanism can be expressed as:

$$h_{t+1} = f(h_t, x_{t+1}) \odot h_t + i(h_t, x_{t+1}) \odot x_{t+1} \tag{2.9}$$

where f represents the forget gate and i represents the input gate. Both gates can depend on the previous hidden state h_t, the current input x_{t+1}, and additional parameters (omitted from our notation for simplicity). Note that these two gates can be coupled, as demonstrated in the GRU (Gated Recurrent Unit) architecture [132].

Conceptualizing gated recurrent networks as enhanced linear diagonal recurrent networks provides valuable theoretical insight. When both the forget gate and input gate operate independent of the input and output, the architecture reduces to a diagonal real-valued network. This relationship is significant for several reasons. First, it enables these architectures to avoid exploding gradients through mechanisms similar to those in linear networks. Second, as we will demonstrate in Chapter 3, this connection allows us to transfer learning dynamics insights from linear recurrent networks to gated architectures. This perspective also illuminates the parallelization capabilities of these models. While exact parallelization remains challenging for the general case[4], parallelization becomes feasible when gates depend solely on the input rather than the hidden state. The contemporary recurrent

---

[4] Current research investigates efficient approaches to this problem [133, 134].

architectures scaled for language modeling tasks we mentioned above leverage precisely this property to achieve computational efficiency.

2.1.3 *Attention and Transformers*

The Transformer architecture [21] represented a fundamental shift in sequence modeling. Attention mechanisms were originally introduced as components within recurrent neural networks to help models focus on relevant parts of the input sequence [135], but Transformers took the radical step of removing recurrence entirely and relying solely on attention. Instead of processing sequences step by step, attention mechanisms allow every token to directly access information from every other token. This architectural change has a significant impact on two critical aspects of deep learning: parallelization and optimization. Since there are no sequential updates to a hidden state to perform, training can be largely parallelized and thus sped up, while vanishing and exploding gradients are out of the picture.

A causal self-attention head[5] as used in autoregressive Transformers dynamically interpolates the previous inputs $(x_t')t' \leq t$ it has received depending on how similar they are to the current input x_t. The process unfolds in three steps: First, it computes the attention scores A by comparing the current query $q_t = W_Q x_t$ with the keys $k_{t'} = W_K x_{t'}$ $(t' \leq t)$ associated to the previous tokens through

$$A_{t,t'} = \frac{q_t^\top k_{t'}}{\sqrt{d}} \tag{2.10}$$

with d the dimension of the queries and keys. Second, it normalizes these attention scores to form the attention patterns α

$$\alpha_{t,t'} = \text{softmax}(A_t)_{t'} = \frac{\exp(A_{t,t'})}{\sum_{t''=1}^{t} \exp(A_{t,t''})} \tag{2.11}$$

---

[5] The version of self-attention we use is a causal one, that is the output at time t only depend on inputs at previous time steps $t' \leq t$. This is not the most general version of attention but it is the one we consider in this thesis and the one that powers the vast majority of current large language models.

which specify where the model attends when processing x_t. Finally, these attention patterns are used to linearly interpolate the previous inputs:

$$y_t = W_V \sum_{t'=1}^{t} \alpha_{t,t'} x_{t'}. \tag{2.12}$$

with W_V the value matrix. Note that $v_t = W_V x_t$ is commonly referred to as values; we here move the value matrix outside the sum to emphasize that attention is just a linear transformation of the input once the attention scores are computed. This view is particularly useful when analyzing the internals of Transformer models [29], which we will do in Part ii.

Transformers employ multiple such heads in parallel, which are then linearly combined. This multi-head design allows for greater flexibility as a single layer can capture different types of relationships simultaneously—some heads might focus on syntactic dependencies while others capture semantic associations.

By design, the attention mechanism is permutation-invariant: shuffling previous inputs would produce an identically shuffled output. For sequence modeling tasks where order matters, this poses a challenge – the order of words in a sentence clearly affects meaning. Transformers address this limitation through positional encoding, which adds learned or deterministic positional information to the input embeddings [21, 22, 136, 137].

The power of Transformers comes with computational costs. The attention mechanism scales quadratically with sequence length: processing a sequence of length T token per token requires $O(T^2)$ memory and computation. For very long sequences, this becomes prohibitive, motivating ongoing research into more efficient attention mechanisms, sparse attention patterns, as well as the recent renewed interest in efficient recurrent networks we mentioned in the last section. Despite this limitation, the combination of parallelizability, long-range connectivity, and strong empirical performance has made Transformers the dominant architecture for sequence modeling across domains from natural language processing [23, 138] to computer vision [139, 140] and beyond.

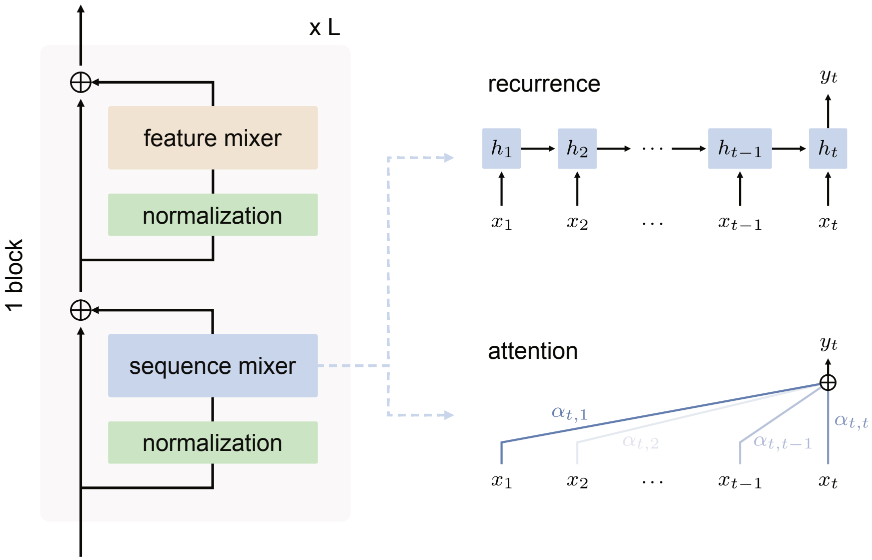


FIGURE 2.1: STRUCTURE OF MODERN DEEP LEARNING ARCHITECTURES. (left) Modern sequential deep learning models are typically built as follows: the representations of the input tokens x are progressively enriched by the L blocks that constitute the entire network. Each block typically has a sequence mixer, often attention and sometimes recurrence, whose main role is to move information between tokens, and a feature mixer (i.e., a multi-layer perceptron) which will process the representations of each token individually. Each mixer is preceded by a normalization layer. (right) Zoom in the two main types of sequence mixers: recurrence and attention. Recurrence maintains a hidden state h which summarize the past input sequence, whereas attention directly interpolates between all past tokens.

2.1.4 *The deep learning stack*

We are now equipped to introduce the architectures behind most modern deep learning architectures. At a high level, such architectures interleave sequence mixers, which are in charge of moving around information between tokens, and feature mixers, which transform each token independently, cf. Figure 2.1 (left). Sequence mixers are sequence modeling layers such as attention, or sometimes recurrent layers. Feature mixers are typically 1-hidden layer multi-layer perceptrons with a hidden layer that is 4-times larger than its input and output dimension. This architecture is the result of many years of architectural evolution, with tiny but important changes being progressively adopted by the community[6].

This structure ends up being surprisingly similar between different modalities, from text to vision and speech, which highlights the generality of this design pattern for sequence modeling. Effective neural computation therefore emerges from the interplay between mechanisms that route information through time (sequence mixers) and mechanisms that transform representations in space (feature mixers). This duality will prove central to understanding how neural networks learn, generalize, and ultimately solve complex problems. This separation will be particularly useful in Part ii as we aim to understand which solutions some attention-based models learn, and how they are acquiring it.

2.2 LEARNING ALGORITHMS FOR NEURAL NETWORKS

Training large neural networks to perform complex tasks presents a fundamental challenge: how do you adjust millions or billions of interconnected parameters to progressively improve behavior? Early approaches to neural network training, such as Hebbian learning [2], tried to mimic how the brain learns and were attempts to ground artificial learning in biological reality. Instead of constraining learning algorithms to mirror biological processes, researchers like Hinton, Rumelhart, and their collaborators [36] focused directly on optimizing performance metrics. This shift – from constraint-driven to objective-driven learning – unlocked the potential of neural networks and established the foundation for modern deep learning.

This section covers the mathematical foundations underlying neural network training that will be needed for the rest of the thesis. We start by formalizing learning as an optimization problem and examine the algorithms that make large-scale training possible. We then explore backpropagation, the algorithm enabling efficient gradient computation in neural networks. Finally, we dive into the mathematical details underlying the vanishing and exploding gradient problem we have mentioned above.

---

[6] For example, the normalization layer was one the residual stream in the original Transformer architecture [21], but now appears in each residual block (as shown in Figure 2.1) in more recent architectures [141].

2.2.1 *Learning as optimization*

Learning, at its core, typically involves interacting with an environment and using outcomes of these interactions to progressively improve behavior. In neural networks, this idea is usually formalized as an optimization problem in which we seek parameters that minimize a loss function measured over data. For this thesis, we focus on supervised and self-supervised learning scenarios in which the learning agent does not actively choose how it interacts with the environment. In particular, we exclude reinforcement learning despite its relevance for biological learning [142], as action selection introduces additional complexity such as non-stationarity of the training data distribution. Yet, the supervised setting already provides sufficient richness for our investigation.

The learning problem is defined by a loss function ℓ that measures how well the model performs. Learning is achieved by minimizing expected loss between predicted and target outcomes:

$$\mathcal{L}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(\hat{y}, y)] \tag{2.13}$$

where $\mathcal{D}$ represents the true data distribution, $\theta \in \mathbb{R}^d$ represents the model parameters, x is the input, $\hat{y}$ the model prediction, and y the target output. This framework is quite general. In image classification, x might be a high-dimensional image and y a categorical label, with cross-entropy loss measuring classification accuracy. In language modeling, x can be a sequence of tokens and y the next token, allowing models to learn language structure.

In general, the training distribution is not known. Machine learning algorithms typically optimize a surrogate objective that replaces the true data distribution with a finite number of samples drawn from that distribution. This creates a discrepancy between the training objective and the true objective of interest, sometimes referred to as the generalization gap. Given that the focus of this thesis is rather on optimization than generalization, we do not thoroughly discuss this kind of consideration.

2.2.2 *Stochastic gradient descent and adaptive optimizers*

Optimizing such a loss function is not an easy task. First, both parameter spaces and datasets are enormous[7], making classical optimization approaches computationally intractable. Second, neural network loss functions are highly non-convex, full of local minima, saddle points, and flat regions that can trap optimization algorithms [143]. Third, unlike convex problems, neural networks lack structure that algorithms can systematically exploit.

One simple algorithm has proven surprisingly successful to solve these challenges: stochastic gradient descent [144, 145]. The parameters of the model are updated according to

$$\theta_{t+1} = \theta_t - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta \, \ell(\hat{y}(x_i, \theta_t), y_i) \tag{2.14}$$

where η is the learning rate and (x_i, y_i) are sampled from a random batch $\mathcal{B}$ of training examples and $\hat{y}_i$ is the output of the network on input x_it, using the current parameters θ_t of the network.

While stochastic gradient descent provides the conceptual foundation, practical training often requires more sophisticated algorithms. Vanilla gradient descent has several limitations such as slow convergence on ill-conditioned problems and sensitivity to the choice of learning rate. Modern optimizers, the most widely used being Adam [146], partially solve this issue by incorporating momentum [147, 148], which averages past update directions, and per-parameter adaptive learning rates [149, 150]. While feedforward and convolutional networks often train well with stochastic gradient descent and momentum, Transformers and recurrent networks frequently require Adam [151]; Chapters 3 and 6 will provide concrete examples.

Despite its extensive practical use, the effectiveness of stochastic gradient descent and its variants remains one of the theoretical mysteries of deep learning [152, 153].

---

[7] Current state-of-the-art language models have hundred of billions parameters and are trained on trillions of examples

2.2.3 *Backpropagation: computing gradients efficiently*

The gradient-based methods we have discussed require efficiently computing gradients with respect to network parameters. The backpropagation algorithm makes it possible. Backpropagation is the application of the chain rule [144] to neural networks. Werbos [154] first applied these ideas to neural networks, though the approach gained recognition following Rumelhart & McClelland [36]. Below, we derive the backpropagation equations and use this mathematical exposition as an occasion to discuss vanishing and exploding gradients more formally.

Before doing so, we introduce the notation for derivatives that we will use consistently in the thesis. We will use d for total derivatives and ∂ for partial derivatives. We use total derivatives to emphasize that the derivative is taken over multiple operations and partial derivatives when the differentiated quantity directly depends on the variable of interest. For example, we use total derivatives when differentiating the loss with respect to the parameters of a remote layer ($\mathrm{d}_{W^l}\mathcal{L}$) and partial derivatives when differentiating the activation of one layer with respect to the ones of the previous layer ($\partial_{h_{l-1}} h_l$). We find this convention helps distinguish between local computations and global ones. Finally, we use the row convention for derivatives, which means that the derivative of the loss with respect to some parameters is a row vector. The gradient ∇ is then the transpose of this derivative and is therefore a column vector (Equation 2.14 is thus consistent notation-wise).

2.2.3.1 *Backpropagation for feedforward networks*

Recall the definition of a feedforward network with L layers:

$$h_0 = x \tag{2.15}$$

$$h_l = \rho(W_l h_{l-1} + b_l) \quad \text{for } l = 1, \dots, L \tag{2.16}$$

Introducing $z_l := W_l h_{l-1} + b_l$ and using the chain rule, we get that the gradient of the loss with respect to weights in layer l is:

$$\frac{\mathrm{d}\ell}{\mathrm{d}W_l} = \frac{\mathrm{d}\ell}{\mathrm{d}z_l}\frac{\partial z_l}{\partial W_l} = h_{l-1}\frac{\mathrm{d}\ell}{\mathrm{d}z_l} \tag{2.17}$$

The key insight is to consider the error term δ^l, defined for each sample as $\delta^l := \frac{\mathrm{d}\ell}{\mathrm{d}z^l}$ and computing it recursively, starting from the output and updating it backwards through

$$\delta^l = \frac{\mathrm{d}\ell}{\mathrm{d}z_l}^\top \tag{2.18}$$

$$= \frac{\partial z_{l+1}}{\partial z_l}^\top \frac{\mathrm{d}\ell}{\mathrm{d}z_{l+1}}^\top \tag{2.19}$$

$$= \left(W_{l+1}^\top \delta^{l+1}\right) \odot \rho'(z^l) \tag{2.20}$$

where $\odot$ denotes element-wise multiplication. We initialize the recursion at the output layer with $\delta^L = \frac{\partial \ell}{\partial z^L}^\top$. For example, $\delta^L = \mathbb{E}[\hat{y} - y]$ for the mean square error loss. With these error terms, the gradient formula becomes:

$$\nabla_{W_l} \ell = \delta_l h_{l-1}^\top, \quad \nabla_{b_l} \ell = \delta_l. \tag{2.21}$$

Intuitively, gradient descent will modify the weights so that the outputs corresponding to inputs that are similar to h_k move towards the direction given by δ_t.

Computationally, this implies that one can efficiently calculate gradients by first performing a forward pass to compute activations, keeping them in memory, and then backpropagating errors through the layers. The complexity of the backward pass is the same as that of the forward pass, making backpropagation particularly efficient.

2.2.3.2 *Backpropagation through time*

The derivation detailed above for feedforward neural networks extends to recurrent neural networks, with time playing a similar role to depth. The resulting algorithm is called backpropagation through time (BPTT). The main differences with backpropagation on feedforward networks are that an input is fed to each time step and that the loss is potentially measured at all time steps (whereas it is typically measured in the last layer in feedforward networks).

Let us consider recurrent networks whose hidden state updates are governed by the equation

$$h_{t+1} = f_\theta(h_t, x_{t+1}) \tag{2.22}$$

and whose output at time t is

$$\hat{y}_t = g_\theta(h_t). \tag{2.23}$$

Note that this notation captures all the recurrent networks we have introduced in Section 2.1.2. For sequence modeling tasks, the total loss is a sum of per time-step losses:

$$\mathcal{L}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\sum_{t=1}^{T} \ell_t(\hat{y}_t, y_t)\right]. \tag{2.24}$$

Let us now derive the backpropagation through time equations. First, we define, as before, the error terms δ_t:

$$\delta_t := \frac{\mathrm{d}\ell}{\mathrm{d}h_t}^\top = \sum_{t'=t}^{T} \frac{\mathrm{d}\ell_{t'}}{\mathrm{d}h_t}^\top. \tag{2.25}$$

It can be computed recursively from the last to the first time step:

$$\begin{aligned}
\delta_t^\top &= \frac{\mathrm{d}\ell}{\mathrm{d}h_t} \\
&= \sum_{t'=t+1}^{t} \frac{\mathrm{d}\ell_{t'}}{\mathrm{d}h_t} + \frac{\partial \ell_t}{\partial h_t} \\
&= \sum_{t'=t+1}^{t} \frac{\mathrm{d}\ell_{t'}}{\mathrm{d}h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial \ell_t}{\partial h_t} \\
&= \delta_{t+1}^\top \frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial \ell_t}{\partial h_t}.
\end{aligned} \tag{2.26}$$

In the second line, we split the loss ℓ into the current term ℓ_t and future terms $\ell_{t'}$ ($t' > t$). The future terms all depend on the Jacobian $\partial_{h_t} h_{t+1}$ (third line), so we factorize it and remark that the remaining term is $\mathrm{d}_{h_{t+1}}\ell = \delta_{t+1}$ in the last line. Finally, we get the following formula for the gradient:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\theta} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \theta} + \delta_t^\top \frac{\partial h_t}{\partial \theta} \tag{2.27}$$

Note that $\partial_\theta \ell_t$ is essentially computing the derivative through the readout function g_θ and $\partial_\theta h_t$ corresponds to the derivative of the (last) recurrent update function f_θ with respect to its parameters. Importantly, it is possible to compute it with only information currently available at time t. For vanilla recurrent networks ($h_{t+1} = \rho(Wh_t + Ux_{t+1} + b)$), the update rule for the recurrent weights W takes a similar form to the one for feedforward networks (see Equation 2.21), except that the error term has now been backpropagated through time and thus depends on future losses.

As for the feedforward case, the compute time for the backward pass scales linearly with the number of layers. However, as the number of time steps eventually grows to infinity when processing longer sequences, the memory requirements can become a bottleneck. To circumvent that, some techniques involve truncating the backward pass by splitting the sequence into multiple chunks and not letting gradients flow between them[8]. Alternatively, real-time recurrent learning offers an appealing alternative method to compute the gradient whose memory complexity does not scale with sequence length; we review it in the next section.

2.2.3.3 *Real-time recurrent learning*

Backpropagation through time is an example of backward-mode differentiation [156], whereas real-time recurrent learning (RTRL) is forward-mode differentiation applied to recurrent neural networks. Instead of computing error terms, which depend on future losses, real-time recurrent learning keeps track of a *sensitivity matrix*, which captures how the hidden states depend on the parameters θ of the network. This quantity effectively acts as a summary statistic that is enough to compute the gradient of the loss at any given time step, without having to do any backpropagation.

To derive the real-time recurrent learning equations, we first need to introduce the sensitivity matrix s_t:

$$s_t := \frac{\mathrm{d}h_t}{\mathrm{d}\theta}. \tag{2.28}$$

As for the error terms, the sensitivity matrix is defined for each sample. Note that, however, it does not depend on any loss term so it only depends

---

[8] Such a technique is called truncated backpropagation through time, e.g. [155].

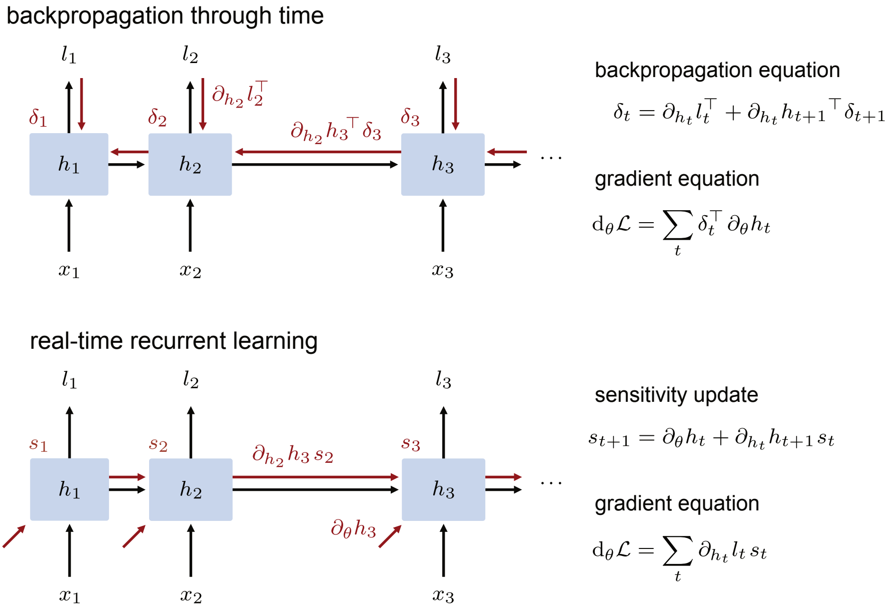

backpropagation through time
l_1
l_2
l_3
δ_1
δ_2
δ_3
$\partial_{h_2} l_2^\top$
$\partial_{h_2} h_3^\top \delta_3$
h_1
h_2
h_3
$\dots$
x_1
x_2
x_3
backpropagation equation
$\delta_t = \partial_{h_t} l_t^\top + \partial_{h_t} h_{t+1}{}^\top \delta_{t+1}$
gradient equation
$\mathrm{d}_\theta \mathcal{L} = \sum_t \delta_t^\top \partial_\theta h_t$
real-time recurrent learning
l_1
l_2
l_3
s_1
s_2
s_3
$\partial_{h_2} h_3 \, s_2$
$\partial_\theta h_3$
h_1
h_2
h_3
$\dots$
x_1
x_2
x_3
sensitivity update
$s_{t+1} = \partial_\theta h_t + \partial_{h_t} h_{t+1} s_t$
gradient equation
$\mathrm{d}_\theta \mathcal{L} = \sum_t \partial_{h_t} l_t \, s_t$

FIGURE 2.2: COMPARISON OF BACKPROPAGATION THROUGH TIME AND REAL-TIME RECURRENT LEARNING. In backpropagation through time, an error vector δ which captures information about all future losses is backpropagated starting from the last time step ($\delta_t = \mathrm{d}_{h_t} \ell^\top$). The current error δ_t is the sum of the backpropagated next error ($\partial_{h_t} h_{t+1}^\top \, \delta_t$) and the instantaneous error the hidden state receives ($\partial_{h_t} \ell_t$), as highlighted by the red arrows. The gradient at time t is the multiplication of the current error δ_t with how the parameters directly affect the current hidden state ($\partial_\theta h_t$). See Section 2.2.3.2 for derivations and additional details. In real-time recurrent learning, a sensitivity matrix s which captures how the parameters how the parameters affect the current hidden state ($s_t = \mathrm{d}_\theta h_t$). The next sensitivity is the sum of the current dependency of the hidden states on the parameters ($\partial_\theta h_T$) and the propagated previous sensitivity ($\partial_{h_t} h_{t+1} \, s_t$). See Section 2.2.3.3 for derivations and additional details. In backpropagation through time, the gradient is given by the multiplication between the current sensitivity and the total error, whereas in real-time recurrent learning it is between the total sensitivity and the current error. This backward- versus forward-looking difference is conceptually important, as backpropagation requires storing the trajectory of hidden states and revisiting in reverse time during the backpropagation phase. Real-time recurrent learning does not suffer from this issue, but requires storing a potentially large matrix in memory.

on the input x and not the target y. The sensitivity matrix satisfies the recursive formula

$$\begin{aligned} s_{t+1} &= \frac{\mathrm{d}h_{t+1}}{\mathrm{d}\theta} \\ &= \frac{\partial h_{t+1}}{\partial \theta} + \frac{\partial h_{t+1}}{\partial h_t}\frac{\mathrm{d}h_t}{\mathrm{d}\theta} \\ &= \frac{\partial h_{t+1}}{\partial \theta} + \frac{\partial h_{t+1}}{\partial h_t} s_t \end{aligned} \tag{2.29}$$

meaning that it can be updated simultaneously with the hidden state, as highlighted in Figure 2.2. We can then combine it with spatially backpropagated errors to compute the total gradient:

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \theta} + \frac{\partial \ell_t}{\partial h_t} s_t \tag{2.30}$$

Compared to backpropagation through time, whose second term is the combination of total errors, which contain information about the future, and the current sensitivity of the hidden state on the parameters, real-time recurrent learning leverages the current error and the total sensitivity of the hidden state on the parameters, aggregated over all previous updates. This property makes it possible to run the latter totally online, that is, one time step after the other. This property makes the algorithm particularly appealing when it comes to physically or biologically plausible learning.

The online property of real-time recurrent learning comes at a cost: memory and compute complexity. The sensitivity s is a matrix of size number of hidden neurons times number of parameters. When the parameters include dense recurrent connections, as it is the case in vanilla recurrent networks or gated ones, this scales cubically with the number of hidden neurons[9]. The memory footprint of backpropagation through time scales linearly with the number of hidden neurons and the sequence length, and is thus smaller for short sequences. The number of operations needed to update the sensitivity matrix (Equation 2.29) is typically quartic in the number of hidden neurons. For these reasons, plain real-time recurrent learning is infeasible in practice and has either to be approximated [157, 158], sampled [159–161] or applied to specific architectures [162–165] to be tractable. Chapter 4 will consider recurrent layers in which real-time recurrent learning is tractable, as well as approximate it to deal with deep recurrent networks.

---

[9] Such a weight matrix connects all hidden neurons to all hidden neurons, and thus has n^2 parameters with n the number of hidden neurons

Finally, one can wonder why we did not introduce the equivalent of real-time recurrent learning for feedforward neural networks. Such an algorithm would have appealing properties as gradients could be computed in one single (large) forward pass. However, feedforward networks do not have the weight-sharing features in recurrent networks. As a consequence, one needs to keep track of the sensitivity of the hidden state with respect to all upstream parameters, which is even less tractable than the recurrent version we discussed above. Yet, recent work has investigated randomized versions of these ideas for feedforward neural networks [166, 167].

2.2.4 *Signal propagation and its effects on optimization*

As we mentioned earlier, stochastic gradient descent and its variants can only efficiently optimize neural networks in certain regions of the parameter space, and different architectures could facilitate learning. Now that we have introduced the mathematics behind gradient calculations, we are equipped to understand these limitations in more depth. The core challenge lies in how gradients propagate through deep networks or long sequences and fundamentally determines the structure of the loss landscape and the efficiency of gradient-based learning.

2.2.4.1 *The vanishing and exploding gradient problem*

The vanishing and exploding gradient problem [49–53] arises in deep feedforward networks and recurrent networks processing long sequences. It arises due to the inherently sequential nature of signal processing in these networks. The dependency of outputs on remote neurons tends to increase or decrease exponentially with the number of steps between the output and remote layers. To formalize this mathematically, we can look at how changes to the hidden state h_i influence a downstream hidden state $h_{i+\Delta i}$[10]. The chain rule gives

$$\frac{\mathrm{d}h_{i+\Delta i}}{\mathrm{d}h_i} = \prod_{j=1}^{\Delta i} \frac{\partial h_{i+j}}{\partial h_{i+j-1}} \tag{2.31}$$

---

[10] We use i to index the hidden state to emphasize that it can both be indexing the layer l in a feedforward network or the time t in a recurrent network.

This product of Jacobians is particularly important as it directly determines how error signals backpropagate to remote hidden states and thus deep credit assignment. For example, in feedforward neural networks, we have

$$\delta_l^\top = \frac{\partial h_L}{\partial h_l} \delta_L^\top. \tag{2.32}$$

The behavior of this product critically depends on the eigenvalues of the individual Jacobians $\frac{\partial h_{i+\Delta i}}{\partial h_{i+\Delta i - 1}}$. If the largest eigenvalue $\lambda_{\max}$ across all Jacobians satisfies $\lambda_{\max} < 1$, then the product converges exponentially to zero as Δi increases, leading to vanishing gradients. Conversely, if $\lambda_{\max} > 1$, the product (can) grow exponentially, causing exploding gradients. This analysis directly connects to the concept of Lyapunov exponents from dynamical systems theory, which characterize the rate of convergence or divergence of nearby trajectories [168]. The vanishing gradient regime corresponds to Lyapunov exponents smaller than 0, whereas the exploding gradient one corresponds to exponents greater than 0 (and thus chaotic systems). A corollary of this analysis is that chaotic systems are particularly hard to learn [169, 170].

For recurrent neural networks, the situation is more nuanced than this simple analysis suggests. It does emphasize that learning long-range dependencies, as they will either contribute very little to the overall learning signal or will entirely dominate. However, due to parameter sharing across time steps, the total gradient with respect to the recurrent weights does not reduce to a single contribution or a single path in the computation graph, as it is the case for feedforward networks. In recurrent networks, vanishing and exploding gradients are therefore not the end of the story. Understanding these behaviors in linear networks and their impacts on learning dynamics is the main contribution of Chapter 3.

2.2.4.2 *Signal propagation analysis*

To develop a more precise understanding of these effects in neural networks, we can examine how signals, both hidden states and error terms, propagate throughout the network. Ideally, we would like these quantities to have similar constant magnitude, regardless of the width and depth of the network, in order to let every layer / time dependency contribute equally to the gradient. This property is closely related to the vanishing and exploding gradient analysis above: maintaining constant gradient magnitudes requires the product of Jacobians to remain bounded despite deeper or wider networks.

Consider a simple feedforward network with no non-linearity, assuming that the inputs and the weights are independently drawn from normal distributions with mean 0 and, for simplicity, that inputs and hidden neurons have the same dimension n. First, we can remark that all the hidden states are 0-mean and normally distributed due to the properties of the product of independent normally distributed random variables. We can leverage this property to compute the expected norm of the hidden state of the different layers through the recursive formula

$$\begin{aligned}
\mathbb{E}_x\left[\|h_{l+1}\|^2\right] &= \sum_{i=1}^{n} \mathbb{E}\left[h_{l+1,i}^2\right] \\
&= \sum_{i=1}^{n} \mathbb{E}\left[\left(\sum_{j=1}^{n} W_{l+1,ij}\, h_{l,j}\right)^2\right] \\
&= \sum_{i,j=1}^{n} \mathbb{E}\left[W_{l+1,ij}^2\right] \mathbb{E}\left[h_{l,j}^2\right] \\
&= n\mathbb{E}\left[w^2\right] \sum_{j=1}^{n} \mathbb{E}\left[h_{l,j}^2\right] \\
&= n\mathbb{E}\left[w^2\right] \mathbb{E}_x\left[\|h_l\|^2\right].
\end{aligned} \tag{2.33}$$

In the third line, we have used the independence of the weights W_{l+1} and the hidden states h_l and in the fourth line that all entries of W_{l+1} are i.i.d. One can do a similar analysis for the backward pass and show, assuming that the weights are independent to the backpropagated error[11], that

$$\mathbb{E}\left[\|\delta_l\|^2\right] = n\mathbb{E}[w^2]\mathbb{E}\left[\|\delta_{l+1}\|^2\right] \tag{2.34}$$

As a consequence, the variance of the weight distribution should scale as $1/n$ for gradients to be well-behaved. Saxe, McClelland & Ganguli [59] showed that when this condition is satisfied, a deep linear network can learn in finite time.

---

[11] This is technically not true, but this heuristic derivation enables to quickly estimate how this quantity will evolve.

This analysis reveals the importance of adapting the initialization scheme for the network parameters depending on some of the properties of the network, here width. Similar ideas can be used to derive such schemes for non-linear networks and for layers of varying width [26, 83].

2.2.4.3 *Connection to loss geometry and optimization*

A question remains: why is maintaining consistent gradient magnitudes crucial for optimization? Part of the answer lies in the geometry of the loss landscape, particularly the conditioning of the optimization problem. An important quantity in understanding gradient-based optimization dynamics is the Hessian matrix $H = \nabla^2 \mathcal{L}$, which describes the local curvature of the loss function. Effective optimization requires small learning rates in sharp directions (large eigenvalues of H) to avoid overshooting, but large learning rates in flat directions (small eigenvalues) to make progress. The condition number $\kappa(H) = \lambda_{\max}(H)/\lambda_{\min}(H)$ quantifies this trade-off: the larger the condition number, the slower gradient descent converges [171].

For neural networks, the Hessian is closely related to the gradient covariance matrix, particularly when the model is overfitting the training data [172]. Indeed, the Hessian generally rewrites as

$$H = \mathbb{E}\left[\frac{\mathrm{d}\hat{y}}{\mathrm{d}\theta}^\top \frac{\partial^2 \ell}{\partial \hat{y}^2} \frac{\mathrm{d}\hat{y}}{\mathrm{d}\theta} + \frac{\partial \ell}{\partial \hat{y}} \frac{\mathrm{d}^2 \hat{y}}{\mathrm{d}\theta^2}\right] \tag{2.35}$$

and the second term vanishes when the model fits the training data and the loss gradient $\partial_{\hat{y}} \ell$ is close to 0. Recall that $\hat{y}$ is the output of the model, that the expectation is taken over the training data, and that ℓ is the loss that measures the discrepancy between the produced outputs and a target y. Up to some rescaling induced by the specific choice of the loss function ℓ, the Hessian is thus the gradient covariance matrix

$$\mathbb{E}\left[\frac{\mathrm{d}\hat{y}}{\mathrm{d}\theta}^\top \frac{\mathrm{d}\hat{y}}{\mathrm{d}\theta}\right]. \tag{2.36}$$

The trace of this matrix, which equals $\mathbb{E}[||\nabla_\theta\, \hat{y}||^2]$, provides insight into the eigenvalue structure[12]. The analysis we reviewed in the last section therefore

---

[12] Mathematically, the norm is equal to the trace of the Hessian, which is itself equal to the sum of the Hessian eigenvalues

directly provides insights into the structure of the loss landscape by looking at the gradient magnitude. When gradient magnitudes vary dramatically across different parts of the network – as happens with vanishing and exploding gradients – the resulting Hessian becomes poorly conditioned. Sharp and flat direction can have a specific structure, as it is the case with vanishing gradients[13], but this structure can be more intricate. Maintaining consistent gradient magnitudes therefore helps in ensuring that gradient-based optimization remains well-behaved, and in enabling efficient learning across all parameters of the network.

---

[13] In this case, flat directions correspond to the parameters of the first layers, and sharp directions to the one of the last layers.

Part I

BIOLOGICALLY PLAUSIBLE LEARNING IN RECURRENT NEURAL NETWORKS

3

RECURRENT NEURAL NETWORKS: VANISHING AND EXPLODING GRADIENTS ARE NOT THE END OF THE STORY

The contents of this chapter were published as a conference paper at the annual conference on neural information processing systems in 2024 [173] and authored by Nicolas Zucchet and Antonio Orvieto.

3.1 INTRODUCTION

Recurrent neural networks [RNNs; 99, 174] have long been the canonical architecture for modeling temporal data [53, 175]. However, they are notoriously difficult to train on long sequences, as error signals flowing backward in time tend to either vanish or explode [49–52]. Attention mechanisms [135], as featured in transformers [21], address these issues by enabling direct token-to-token communication, considerably simplifying signal propagation across long time intervals. Yet, their superior performance comes with increased computational and memory costs, due to their quadratic scaling in the sequence length. This limitation has motivated significant research aimed at making transformers more efficient [176–180].

A promising line of research in this direction involves a new type of linear recurrent networks known as deep state-space models [SSMs; 54, 67–71, 108]. These models trade expressivity for faster training speed, and they have been shown to be particularly effective at capturing long-range dependencies. In this paper, we wonder whether this effectiveness can be solely attributed to their ability to avoid vanishing and exploding gradients. The simplicity of such models presents an opportunity for in-depth theoretical analysis. We focus on signal propagation within these models.

After reviewing classical results on recurrent neural networks in Section 3.2, we demonstrate that they can suffer from an understudied problem: as the recurrent network encodes longer memories, the network's activity becomes increasingly sensitive to changes in its parameters, even when its dynamics remains stable. In Section 3.4, we then show that SSMs, as well as other architectures such as LSTMs, are well equipped to mitigate this issue. We then analyze a simple teacher-student task (Section 3.5). This task already reveals the remarkable complexity underlying the learning of linear recurrent networks and enables us to verify empirically our theory. Finally, we discuss how our findings extend to more realistic scenarios (Section 3.6), both in terms of architectures and data. Overall, our paper provides theoretical insights into the training of recurrent neural networks, an area where such analysis is rare. While vanishing and exploding gradients are well-known challenges, our results demonstrate that this is not the end of the story – there exists an additional layer of complexity beyond them.

3.2 VANISHING AND EXPLODING GRADIENTS

Let us recall the notations we use and the vanishing and exploding gradient problem. Further detail can be found in the background section.

We consider a recurrent neural network with hidden state h_t, update function f_θ parametrized by θ, and input sequence (x_t). The average performance of the network is measured by a loss $\mathcal{L}$. We have

$$h_{t+1} = f_\theta(h_t, x_{t+1}) \text{ and } \mathcal{L} = \mathbb{E}\left[\sum_{t=1}^{T} \ell_t(h_t)\right]. \tag{3.1}$$

The gradient of the instantaneous loss ℓ_t with respect to the parameters θ is then equal to

$$\frac{\mathrm{d}\ell_t}{\mathrm{d}\theta} = \frac{\partial \ell_t}{\partial h_t}\frac{\mathrm{d}h_t}{\mathrm{d}\theta} = \frac{\partial \ell_t}{\partial h_t}\sum_{t' \leq t}\frac{\mathrm{d}h_t}{\mathrm{d}h_{t'}}\frac{\partial f_\theta}{\partial \theta}(h_{t'-1}, x_{t'}) \tag{3.2}$$

Recall that we used ∂ to denote partial derivatives and d for total derivatives. Using this notation enables us to distinguish between $\partial_{h_t}\ell_t$, which corresponds to the error backpropagated from the current loss term to the hidden state through the readout function, and $\mathrm{d}_{h_t}\mathcal{L}$, which accumulates the errors that are backpropagated through the future hidden state values.

In particular, $\partial_{h_t}\mathcal{L} = \partial_{h_t}\ell_t$ and $\mathrm{d}_{h_t}\mathcal{L} = \partial_{h_t}\ell_t(h_t) + \sum_{t'>t} \mathrm{d}_{h_t}\ell_{t'}(h_{t'})$. When stacking several recurrent layers on top of each other, $\partial_{h_t}\mathcal{L}$ corresponds to the current error being backpropagated to the hidden state h_t through the hierarchy of the network and $\mathrm{d}_{h_t}\mathcal{L}$ to future error signals backpropagated through the recurrence.

Early work [49] highlighted the difficulty for gradient descent to make recurrent neural networks remember past inputs that will later be useful to produce a desired behavior. This is due to the fact that error signals flowing backward in time tend to either explode or vanish. The key quantity is

$$\frac{\mathrm{d}h_t}{\mathrm{d}h_{t'}} = \prod_{i=t'}^{t-1} \frac{\partial h_{i+1}}{\partial h_i} = \prod_{i=t'}^{t-1} \frac{\partial f_\theta}{\partial h}(h_i, x_{i+1}). \tag{3.3}$$

One can remark that this quantity exponentially converges to 0 when the spectral radius of the Jacobian $\partial_h f_\theta$ is upper bounded by a constant strictly smaller than 1, and can exponentially explode if there exists some component bigger than 1. The error signal at time t backpropagated to time t' behaves similarly, as $\mathrm{d}_{h_{t'}}\ell_t = \partial_{h_t}\ell_t \, \mathrm{d}_{h_t'} h_t$. Gradient-based learning of long-term memories is thus difficult: the contribution of past hidden states to the current loss becomes either negligible or predominant as the time span considered increases.

Since then, the analysis has been refined [50–52] and the development of recurrent architectures has mostly been driven by the desire to solve this pathological issue. Most famously, the LSTM [53] unit, and later on the GRU [132], solve this problem by using memory neurons that facilitate direct information storage and retrieval, and thus facilitate error backpropagation. Other approaches to solving this problem, to name a few, involve gradient clipping [52, 181], activity normalization [86, 87, 182], careful weight initialization [183, 184] or enforcing architectural constraints such as hierarchical processing [185, 186], orthogonal weight matrices [187–189] and oscillations [190–192].

3.3 THE CURSE OF MEMORY

According to common deep learning wisdom, it is often believed that solving the vanishing and exploding gradients problem enables recurrent neural networks to learn long-term dependencies. We challenge this view and question:

> *Is solving those issues really enough to ensure well-behaved loss landscapes and enable learning long-range dependencies?*

We answer negatively by showing that gradients can explode as the memories of the network are kept for longer, even when the dynamics of the network remains stable.

3.3.1 *Intuition*

Recurrent neural networks have something special: the very same update function f_θ is applied over and over. Therefore, modifying the parameters θ will not only influence one update, as changing the weights of a given layer in a feedforward neural network would, but all of them. As the memory of the network gets longer, the hidden states keep a trace of the effects of more updates. Hidden states thus become increasingly sensitive to parameter changes. This is the *curse of memory*. We borrow the term from [193, 194], although we use it in a different context, and note that Martens & Sutskever [195] hypothesized that such a phenomenon could arise in RNNs and hinder their optimization.

Let us formalize our intuition by considering the sensitivity of the hidden state h_t on the parameters θ:

$$\frac{\mathrm{d}h_t}{\mathrm{d}\theta} = \sum_{t' \leq t} \frac{\mathrm{d}h_t}{\mathrm{d}h_{t'}} \frac{\partial f_\theta}{\partial \theta}(h_{t'-1}, x_{t'}). \tag{3.4}$$

When information stays in the network's memory for longer, the number of non-negligible Jacobian $\mathrm{d}_{h_{t'}} h_t$ terms increases. As a result, the magnitude of this sensitivity increases when the network encodes longer-term dependencies, and learning θ becomes trickier. It is crucial to observe that this phenomenon arises even when exploding gradients are removed from the picture by constraining the eigenvalues of the recurrent Jacobian to be smaller than one and ensuring that the network dynamics remains stable. The rest of this section will be dedicated to studying this behavior quantitatively.

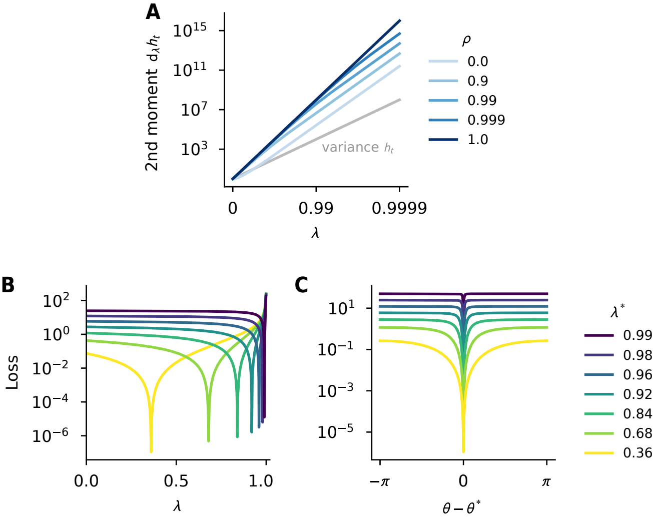


FIGURE 3.1: OPTIMIZATION OF RECURRENT NEURAL NETWORKS GETS HARDER AS THEIR MEMORY INCREASES. **A.** Evolution of the second moment of $\mathrm{d}_\lambda h_t$ as a function of the recurrent parameter λ and of the input x auto-correlation decay rate ρ, when $h_{t+1} = \lambda h_t + x_t$. As the memory of the network increases ($\lambda \to 1$), h_t becomes more sensitive to changes in λ, particularly as the elements in the input sequence are more correlated ($\rho \to 1$). The explosion of $\mathrm{d}_\lambda h_t$ is faster than the one of h_t, as highlighted with the grey line obtained for $\rho = 1$. See Section 3.3.2 for more detail. **B, C.** Illustration of the phenomenon on the toy one-dimensional teacher-student task of Section 3.5.1, in which the teacher is parametrized by a real number λ^* and the student by a complex number λ. In B, λ varies on the real axis, and it varies on the circle of radius λ^* parametrized by θ in C. The loss becomes sharper as information is kept longer in memory, making gradient-based optimization nearly impossible.

3.3.2 *Signal propagation in linear diagonal recurrent neural networks*

We study how hidden state and gradient magnitudes evolve as the network encodes longer-term dependencies. Ideally, these quantities do not vanish or explode, as it improves the conditioning of the loss landscape [91] and eases optimization [88, 90]. We operate under the following assumptions:

a) **Linear diagonal recurrent neural networks**. We restrict ourselves to update functions of the form $f_\theta(h_t, x_{t+1}) = \lambda \odot h_t + x_{t+1}$ with λ a vector of the size of h_t and $\odot$ the element-wise product. For ease of exposition, we present results for real-valued λ here; see Appendix A.2.2 for the complex-valued setting. While this assumption is strong, it allows us to identify some crucial mechanisms and models like S4 [54], S5 [68] and LRUs [69] satisfy it. We later show that our analysis can model some features of more sophisticated networks. Note that we do not consider the input and readout mappings usually featured in recurrent layers as they are feedforward layers and signal propagation within them is already well understood [e.g., 83, 84].

b) **Infinite time horizon**. We consider infinite sequences and initialize the network dynamics at $t_0 = -\infty$. It simplifies our calculations while being a reasonable assumption when the sequences considered are longer than the characteristic timescales of the dependencies we want to learn.

c) **Wide-sense stationarity**. We assume the different quantities that the network receives, which include the inputs x_t, to be wide-sense stationary (WSS). A random process X_t is said to be WSS if its auto-correlation function is independent of time, that is, for all $t \in \mathbb{Z}$ and $\Delta \in \mathbb{Z}$, $\mathbb{E}_X\left[X_{t+\Delta} X_t\right] =: R_X(\Delta)$, where $\mathbb{E}_X$ denotes the expectation over the data. It corresponds to assuming that the statistics of the data are invariant to time shifts. This is a standard assumption when analyzing stochastic processes [196]. It keeps our calculations concise and does not qualitatively affect our conclusions (cf. Section 3.6). We discuss how to relax it in Appendix A.2.2.4.

We are now equipped to analyze signal propagation in one recurrent layer, both in the forward and backward passes. We show that both hidden states and backpropagated errors explode as $|\lambda| \to 1$.

FORWARD PASS. Here, we are interested in understanding how the hidden state second moment $\mathbb{E}[h_t^2]$ evolves as a function of λ and of the input auto-correlation function R_x. After a calculation that we defer to Appendix A.2.2, we obtain

$$\mathbb{E}\left[h_t^2\right] = \frac{1}{1-\lambda^2}\left(R_x(0) + 2\sum_{\Delta\geq 1}\lambda^\Delta R_x(\Delta)\right). \tag{3.5}$$

Importantly, this quantity goes to infinity as longer-term dependencies are encoded within the network, that is $|\lambda| \to 1$. Additionally, the divergence speed depends on the input data distribution: it increases as consecutive time steps in the input distribution become more correlated (i.e., less of the $R_x(\Delta)$ terms are negligible). This behavior already highlights potential difficulties of gradient-based learning of deep neural networks containing linear recurrent layers as the variance of neural activity can become arbitrarily large, hindering learning abilities of deeper layers.

BACKWARD PASS. Let us first derive the gradient of the loss with respect to λ. Using the chain rule we have $\mathrm{d}_\lambda \mathcal{L} = \sum_t \partial_{h_t}\ell_t \, \mathrm{d}_\lambda h_t$. We thus seek to understand how $\mathrm{d}_\lambda h_t$ behaves. We remark that $\mathrm{d}_\lambda h_{t+1} = \lambda \mathrm{d}_\lambda h_t + h_t$ so that $\mathrm{d}_\lambda h_t$ is a low pass filtered version of the hidden state, which is itself a low pass filter version of the inputs. It therefore comes as no surprise that the second moment of $\mathrm{d}_\lambda h_t$ diverges faster than the one of h_t when $|\lambda| \to 1$. More precisely, we get

$$\begin{aligned}\mathbb{E}\left[\left(\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right)^2\right] = \frac{1+\lambda^2}{(1-\lambda^2)^3}&\left(R_x(0) + 2\sum_{\Delta\geq 1}\lambda^\Delta R_x(\Delta)\right) \\ &+ \frac{2}{(1-\lambda^2)^2}\left(\sum_{\Delta\geq 1}\Delta\lambda^\Delta R_x(\Delta)\right).\end{aligned} \tag{3.6}$$

We plot the exact behavior of this quantity when the auto-correlation of x satisfies $R_x(\Delta) = \rho^{|\Delta|}$ on Figure 3.1 and refer the interested reader to Appendix A.2.2 for a derivation of Equation 3.6. More generally, the hidden state of the network, and thus its final output, becomes increasingly sensitive to changes in recurrent parameters as the network reaches the edge of dynamical stability ($|\lambda| \to 1$).

The last quantity we need to consider is the error that is backpropagated to the inputs x of the recurrent layer. It can be observed that the backward pass is dual to the forward pass in the sense that it is a recurrent process that receives backpropagated errors $\partial_{h_t}\mathcal{L}$ and it runs in reverse time:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}x_t} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}h_t}\frac{\partial h_t}{\partial x_t} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}h_{t+1}}\frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial \mathcal{L}}{\partial h_t} = \lambda\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}h_{t+1}} + \frac{\partial \ell_t}{\partial h_t}, \tag{3.7}$$

in which we made use of $\partial_{x_t} h_t = 1$. It follows that the analysis we did for the forward pass also holds here. Crucially, this implies that the explosion behavior will be most significant for the recurrent parameters rather than for potential input or readout weights.

3.3.3 *Extending the analysis to the non diagonal case*

We now generalize our results to fully connected linear recurrent neural networks of the form $h_{t+1} = Ah_t + x_t$. For the sake of the analysis, we assume that A is complex diagonalizable, that is there exists a complex-valued matrix P and a complex-valued vector λ such that $A = P\mathrm{diag}(\lambda)P^{-1}$. Note that this occurs with probability one under random initialization of A [69]. In this case,

$$h_t = Ph_t^{\mathrm{diag}} \ \text{ with } h_{t+1}^{\mathrm{diag}} = \mathrm{diag}(\lambda)h_t^{\mathrm{diag}} + P^{-1}x_{t+1} \tag{3.8}$$

and

$$\frac{\mathrm{d}h_t}{\mathrm{d}A} = \frac{\partial h_t}{\partial P}\frac{\partial P}{\partial A} + \frac{\partial h_t}{\partial h_t^{\mathrm{diag}}}\frac{\mathrm{d}h_t^{\mathrm{diag}}}{\mathrm{d}\lambda}\frac{\partial \lambda}{\partial A} + \frac{\partial h_t}{\partial h_t^{\mathrm{diag}}}\frac{\mathrm{d}h_t^{\mathrm{diag}}}{\mathrm{d}P^{-1}}\frac{\partial P^{-1}}{\partial A}. \tag{3.9}$$

From the analysis above, we know that the dominating term in the limit $|\lambda| \to 1$ among $\partial_P h_t$, $\mathrm{d}_\lambda h_t$ and $\mathrm{d}_P^{-1} h_t$ is $\mathrm{d}_\lambda h_t$, as P and P^{-1} act as readout and input weights. Given that all other terms do not directly depend on the magnitude of λ, we have that $\mathrm{d}_A h_t \simeq \partial_{h_t^{\mathrm{diag}}} h_t \, \mathrm{d}_\lambda h_t^{\mathrm{diag}} \, \partial_A \lambda$; cf. Appendix A.2.2.3 for formal statements. This has two consequences: First, the sensitivity of h_t on A will explode as longer memories are encoded and this directly comes from the eigenvalues of A. Second, as each entry of A typically impacts all eigenvalues of the matrix, the explosion behavior will be distributed across all entries, whereas it was concentrated on the eigenvalues for the diagonal case. We will later observe that this has significant practical consequences and partly explains why fully connected linear RNNs are difficult to train. As a side note, we remark that enforcing the matrix A to be orthogonal solves vanishing and exploding gradient issues

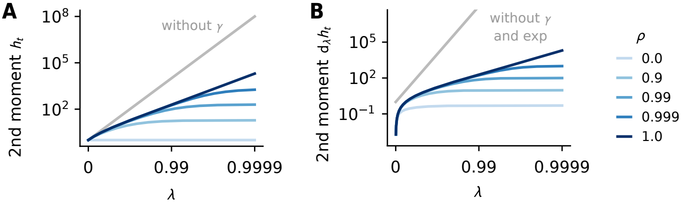


Figure 3.2: Illustration of the effects of normalization and reparametrization. It can effectively control the magnitude of **A.** $\mathbb{E}[h_t^2]$ and **B.** $\mathbb{E}[(\mathrm{d}_\lambda h_t)^2]$ over all λ values when the input auto-correlation satisfies $R_x(\Delta) = \rho^{|\Delta|}$ with $\rho = 0$, but does not manage do to so for other type of distributions ($\rho \neq 0$). Here, we use $\gamma(\lambda) = \sqrt{1-\lambda^2}$, decouple it from λ when differentiating, and take $\lambda = \exp(-\exp(\nu))$, as in [69]. The grey line indicates the value the two quantities take without any normalization and reparametrization, when $\rho = 1$.

but these weights may remain sensitive to learn because of the curse of memory.

3.4 MITIGATING THE CURSE OF MEMORY

We have discussed the sensitivity of recurrent networks to parameter updates. Given this problem, how can it be mitigated? We show that recurrent networks with diagonal connectivity are particularly well-suited for this purpose. Besides enabling control over the Jacobian and avoiding exploding gradients, they facilitate the mitigation of the curse of memory. We additionally highlight that deep state-space models and gated RNNs inherently incorporate such mechanisms.

3.4.1 *A solution: normalization and reparametrization*

Both forward and backward passes explode as the network encodes longer memories. When $h_{t+1} = \lambda h_t + x_{t+1}$, we argue that it is relatively straightforward to mitigate this effect. We aim to keep $\mathbb{E}[h_t^2]$, $\mathbb{E}[(\mathrm{d}_\lambda h_t)^2]$ and $\mathbb{E}[(\mathrm{d}_{x_t} h_t)^2]$ independent of λ, similar to initialization schemes that maintain the magnitude of neural activity constant in deep networks [83, 84], regardless of the layer width [88–90].

INPUT NORMALIZATION. A simple way to enforce $\mathbb{E}[h_t^2]$ to stay constant is to introduce a scaling factor $\gamma(\lambda)$ applied to the inputs a neuron receives, that satisfies $\gamma(\lambda)^2\mathbb{E}[h_t^2] = \Theta(1)$. Given that the backward propagation of output errors to inputs is dual to the forward pass, the role of γ in the backward pass will be similar. The value γ needs to take therefore both depends on the input distribution to normalize the forward pass, as well as on the output error distribution to normalize the backward pass. Perfect normalization is likely unrealistic, but some normalization can help, as shown in Figure 3.2.A.

EIGENVALUE REPARAMETRIZATION. We are now left with keeping the gradient of the loss with respect to λ under control. Input normalization partly reduces the memory-induced exploding effect, but not entirely as the variance of $\mathrm{d}_\lambda h_t$ is much larger than the one of h_t (cf. Fig.3.1.A). Reparametrization can close that gap. Indeed, if λ is parametrized by ω, we have that $\mathrm{d}_\omega h_t = \mathrm{d}_\lambda h_t \mathrm{d}_\omega \lambda$. Choosing a parameterization that is more and more granular as λ goes to 1 thus helps in keeping the magnitude of $\mathrm{d}_\omega h_t$ constant. Assuming γ is independent of λ for simplicity, achieving $\mathbb{E}[(\mathrm{d}_\omega h_t)^2] = \Theta(1)$ requires solving the differential equation $\gamma(\lambda)^2\lambda'(\omega)^2\mathbb{E}[(\mathrm{d}_\lambda h_t)^2] = 1$. While deriving a universal optimal parametrization is again unrealistic due to dependency on the input distribution, reparametrization definitely helps, as shown in Figure 3.2.B. Figure A.1 illustrates how it affects the loss landscape.

THE CASE OF COMPLEX NUMBERS. We have not yet discussed the case $\lambda \in \mathbb{C}$, relevant for SSMs such as S4 [54]. We extend our analysis to complex-valued λ in Appendix A.2.3.2. Briefly, it reveals that changes in the magnitude of λ have a similar impact as in the real case, but this similarity does not extend to the angle. To keep the sensitivity on the angle constant, its parametrization must depend on the magnitude of λ. However, doing so hurts learning, particularly far from optimality, as we exemplify in Appendix A.2.3.2. A key implication of this analysis is that the sensitivity of the hidden state on the angle of λ explodes as its magnitude approaches 1.

3.4.2 *Several RNN architectures implicitly alleviate the curse of memory*

Deep state-space models, as well as gated RNNs, feature some form of normalization and reparametrization which helps keep signal propagation under control. We discuss how below.

DEEP STATE-SPACE MODELS. Deep SSMs were originally motivated as discretizations of the differential equation $\dot{h} = Ah + Bx$ [108]. Naïve discretization of the differential equation yields $h_{t+1} = (\mathrm{Id} + \mathrm{d}tA)h_t + \mathrm{d}tBx_{t+1}$ which already provides some input normalization. More elaborate discretization schemes, such as the zero-order hold, effectively reparametrize the A matrix, e.g. with $\exp(\mathrm{d}tA)$. Here, diagonalization arises from computational efficiency and simplicity reasons [67]. While such models can approximate any smooth mappings [111, 113], their expressivity remains limited [197]. The next generation of these models, including Mamba [70], incorporates input-dependent gates which modulate $\mathrm{d}t$ depending on the input x_t. The theory we developed above does not strictly apply to this setting as $\mathrm{d}t$ is no longer constant. However, since the recurrence Jacobian remains diagonal, we expect the qualitative behaviors we analyzed to remain.

GATED RNNS. While the original motivation behind gated RNNs such as LSTMs [53] or GRUs [132] largely differs from the one of SSMs, they share similar mechanisms. In these networks, the memories stored in hidden neurons can be erased through a forget gate, and incoming inputs can selectively be written in memory through an input gate. Mathematically, this corresponds to hidden state updates of the form $h_{t+1} = f_{t+1} \odot h_t + i_{t+1} \odot x_{t+1}$, with the forget f_{t+1} and input i_{t+1} gates being independent non-linear functions of x_{t+1} and h_t. The forget gate is akin to λ and usually involves a sigmoid non-linearity, which has a similar effect as reparametrizing λ in the backward pass. The input gate can act as an input normalization depending on the initialization of the network or if it is coupled to the forget gate as in the GRU ($f_t = 1 - i_t$) [184]. Importantly, the gates here depend on the hidden states and thus make the Jacobian $\partial_{h_t} h_{t+1}$ non-diagonal. Yet, we argue that these architectures still have a bias towards diagonality. Indeed, the contributions of the hidden state through the forget and input gates are indirect, and they can be ignored when the weights connecting the hidden states to the gates are small. Empirically, we find that GRUs lie in this regime at initialization, cf. Section A.4.2, so that our theory accurately captures signal propagation in GRUs. We additionally confirm that signal propagation is well behaved in gated RNNs in Section 3.6. In regimes in which this approximation does not hold, studying signal propagation requires a much more sophisticated analysis than the one we have done here [198].

3.5 A LINEAR TEACHER-STUDENT ANALYSIS

We start our empirical analysis with a teacher-student task using linear recurrent networks [199]. This is arguably the simplest setting in which one can train recurrent networks. Yet, as we shall see, it is already remarkably complex and captures some of the differences between different architectures observed in more realistic settings [69]. It additionally makes it possible to control the characteristic time constants of the data, which is only possible with synthetic data.

Our investigation starts with the one-dimensional setting to provide an intuitive illustration of the consequences of the curse of memory on the loss landscape. We then address the general setting and observe that linear networks indeed suffer from the curse of memory, and that the remedies we studied in the last section are effective. We additionally find that diagonality greatly modifies the structure of the loss landscape and helps optimizers with adaptive learning rates to compensate for eventual increased sensitivities.

3.5.1 *The one-dimensional case*

We first consider a student and a teacher following the one-dimensional dynamics $h_{t+1} = \lambda h_t + x_{t+1}$, with complex-valued parameter λ for the student and λ^* for the teacher. For simplicity, we independently draw each x_{t+1} from a unit normal distribution (its autocorrelation function is $R_x(\Delta) = \delta_{\Delta=0}$) and note that other input distributions do not qualitatively change the results. The performance of the student is measured by a loss

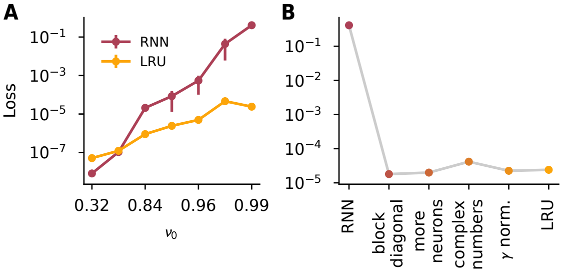


Figure 3.3: LRUs are better at replicating a teacher's behavior than linear RNNs. **A.** As the teacher encodes longer dependencies ($\nu_0 \to 1$), the linear RNN struggles to reproduce it, but not the LRU. **B.** An ablation study ($\nu_0 = 0.99$) reveals that this gap mainly comes from having a close to diagonal recurrent connectivity matrix. See Section 3.5.2 for more detail.

$\mathcal{L}$ that averages the per time-step losses $\ell_t := \frac{1}{2}|h_t - h_t^*|^2$ over the entire sequence.

This simple model already captures two key difficulties of gradient-based learning of recurrent neural networks. In Figure 3.1, we plot the resulting loss landscape for different λ^* values, when λ evolves on the positive part of the real axis (Fig. 3.1.B) and when it evolves on the circle of radius $|\lambda^*|$ in the complex plane (Fig. 3.1.C). We restrict λs to have absolute values smaller than one: exploding gradients are out of the picture. Still, two difficulties for gradient-based learning appear here. On one side, vanishing gradients lead to flat loss regions that are hard to escape. On the other side, the loss sharpens as the student encodes longer memories because of the curse of memory. As a consequence, gradient-based optimization is extremely tedious, already in this simple example.

3.5.2 *Diagonal connectivity simplifies optimization*

We now move to the general case in which the teacher evolves according to

$$h_{t+1} = Ah_t + Bx_{t+1} \quad \text{and} \quad y_t = Ch_t + Dx_t. \tag{3.10}$$

with $h_t \in \mathbb{R}^n$, $x_t \in \mathbb{R}$ drawn i.i.d. from $\mathcal{N}(0,1)$, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times 1}$, $C \in \mathbb{R}^{1 \times n}$ and $D \in \mathbb{R}^{1 \times 1}$. Here both inputs and outputs are scalars.

Given the intuition we have developed so far, we expect fully connected linear recurrent neural networks to struggle solving the task when the teacher encodes longer memories, not only because of exploding gradients but also due to the curse of memory. Conversely, diagonality facilitates the eigenvalue reparametrization needed to avoid exploding gradients and make them better behaved. We run the following experiment to verify this intuition. We draw random teachers with hidden dimension $n = 10$ and transform the complex eigenvalues of the recurrent matrix A to have magnitudes close to a value ν_0 that we control[1]. The larger ν_0 is, the longer the memories encoded by the teacher are. We train a linear RNN, as well as an LRU [69], with hidden dimension 64 on this task. The students are therefore largely overparametrized. We chose the LRU architecture to represent deep SSMs due to its simplicity. This architecture uses input normalization and an exponential reparametrization of the eigenvalues, similar to what we analyze in Section 3.4. Both networks are trained using the Adam optimizer [146] and cosine annealing schedule for 10k steps, on batches of size 128. To ensure that we are in the infinite sequence length regime, we take the sequences to be of length 300, that is three times longer than the characteristic time scale of the teacher. Learning rates are tuned separately for each method and training distribution. The results, which we plot in Figure 3.3.A, confirm our intuition: LRUs significantly outperform linear RNNs when long memories have to be learned, despite having 10 times fewer parameters.

Next, we wonder which design choices behind the LRU architecture are crucial to this performance improvement. To this end, we interpolate between a linear RNN and an LRU in the following way: First, we restrict the weight matrix of the linear RNN to a block diagonal with blocks of size 2. Each block can represent a complex number, so the network can represent 32 complex numbers in total. We additionally double the number of hidden neurons. Second, we change those 2×2 blocks (and their input and output weights) to be complex numbers. Finally, we add the γ input normalization and the exponential parametrization to obtain the final LRU architecture. We report the results of this experiment in Figure 3.3.B. Surprisingly, we find the gap comes from making the weight matrix block diagonal (4×4 blocks are here enough, cf. Figure A.6 in the Appendix). Interestingly, this change reduces the number of parameters the model has and slightly reduces the model expressivity. An explanation of this behavior is therefore likely to

---

[1] We draw each entry of A from $\mathcal{N}(0, 1/\sqrt{n})$, complex diagonalize it, and apply the transformation $x \mapsto \nu_0 + (1 - \nu_0)\tanh(x)$ to the absolute values of the eigenvalues.

be related to the optimization properties of those models. We confirm this hypothesis in the next section.

3.5.3 *On the importance of adaptive learning rates*

So far, our results highlight the importance of having a close to diagonal recurrent connectivity matrix. In this section, we show that this parametrization alone does not mitigate any exploding behavior but modifies the structure of the loss landscape, making it possible for optimizers with adaptive learning rates to compensate for these behaviors.

To demonstrate this, we consider the Hessian of the loss:

$$\frac{\mathrm{d}^2\mathcal{L}}{\mathrm{d}\theta^2} = \sum_t \mathbb{E}_x \left[\frac{\mathrm{d}h_t}{\mathrm{d}\theta} \frac{\partial^2 \ell_t}{\partial h_t^2} \frac{\mathrm{d}h_t}{\mathrm{d}\theta}^\top + \frac{\partial \ell_t}{\partial h_t} \frac{\mathrm{d}^2 h_t}{\mathrm{d}\theta^2} \right]. \tag{3.11}$$

If the network can perfectly fit the target data, which is the case in the experiments above, the second term vanishes at optimality. We plot the Hessian at optimality in Figure 3.4.A and B for a standard linear recurrent network and one with complex diagonal parametrization, both with 4 hidden neurons ($\nu_0 = 0.99$). We observe that the eigenvalue spectra are similar for the two architectures, both exhibiting large terms that are characteristic of the curse of memory, which makes learning with stochastic gradient descent almost impossible[2]. However, their structures differ. For the fully connected linear RNN, the top eigenvectors are distributed over many coordinates, whereas they are concentrated on a few coordinates for the complex diagonal one. This feature aids adaptive optimization [e.g., 203]: adapting to large curvature is much easier for Adam when the pathological directions are aligned to the canonical basis. This is what we observe in practice.

In Figure 3.4.C and D, we compare the effective learning rate used by Adam, which we compute by providing a vector of ones to the optimizer. For the dense linear RNN, the adaptive learning rates cannot compensate for the intricate coupling between components, resulting in small learning rates

---

[2] The gradient Lipschitz constant L of the loss equals the maximum Hessian eigenvalue [200]. This quantity sets a bound $2/L$ for the maximum globally stable learning rate. While convergence might happen in a subspace, it is generally aligned with the top Hessian eigenspace near the solution [201, 202].

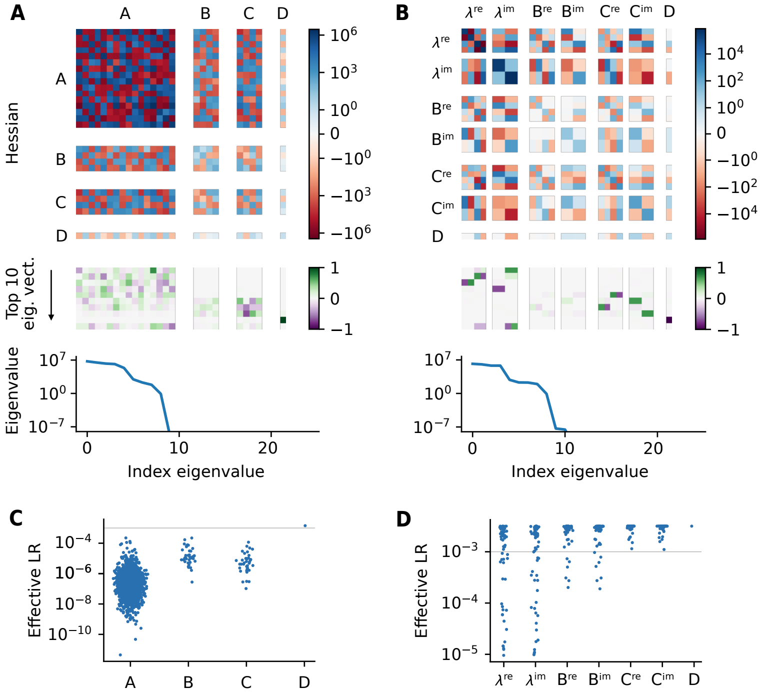


Figure 3.4: Differences in learning abilities between fully connected and complex diagonal linear RNNs are due to a better structure of the loss landscape. **A, B.** Hessian of the loss at optimality, its 10 eigenvectors with greatest eigenvalues and its eigenspectra for a fully connected RNN (A) and a complex diagonal one (B). The spectra are almost the same. However, the top eigenvectors are concentrated on few coordinates for the complex diagonal one but not for the fully connected one. **C, D.** This structure makes it possible for Adam to efficiently deal with the extra sensitivity, as shown with the effective learning rates that it uses at the end of learning. For the fully connected one (C), Adam uses small learning rates to compensate for the sensitivity, whereas it can use larger ones for the complex diagonal one without hindering training stability. The horizontal grey line shows the learning rate used, which is here 10^{-3}.

overall. Conversely, the sensitive directions of complex diagonal RNNs are concentrated on few parameters, which adaptive learning rates can compensate for. This leads to more targeted and overall larger learning rates, significantly speeding up learning. As a side note, the complex eigenvalues of the teacher come in conjugate pairs. However, during training, the complex values of the complex RNN are not conjugates of each other, thereby increasing Hessian diagonality. Finally, performing this analysis for the LRU, we find that the Hessian spectrum is similar to the diagonal setting and that the exploding dimensions of the Hessian are almost exclusively due to the angle parameter, consistently with our theoretical analysis; see Figure A.4.A and C. The loss landscape for S4 model [54] can be qualitatively similar to the complex diagonal RNN or to the LRU, depending on which regime it is in; see Figure A.4.B and D.

Before concluding this section, we investigate whether certain eigenvalue distributions can break the diagonal structure of the Hessian, thereby complicating optimization and increasing the need for eigenvalue reparametrization. In Appendix A.3.2, we provide a theoretical quantification of the intuitive result that more concentrated eigenvalues lead to less diagonal Hessian. Consequently, the performance gap between complex-valued diagonal networks and LRUs widens, although the former still greatly outperform their fully-connected counterpart (see Figure A.5). An important corollary is that increasing the number of hidden neurons breaks the diagonal structure of the loss landscape, thus reducing the effectiveness of optimizers with adaptive learning rates in mitigating the curse of memory. This observation may explain why Orvieto *et al.* [69] reported a more substantial performance improvement from eigenvalue reparametrization than what we observe in our study (cf. block diagonal vs. LRU in Figure 3.3.B).

3.6 SIGNAL PROPAGATION IN DEEP RECURRENT NETWORKS AT INITIALIZATION

The ultimate goal of our theoretical quest is to gain insights into the training of practical recurrent network architectures. Specifically, we aim to verify whether the trends established theoretically and in controlled experiments hold in practice, by studying signal propagation at initialization on a realistic next-token prediction natural language processing task.

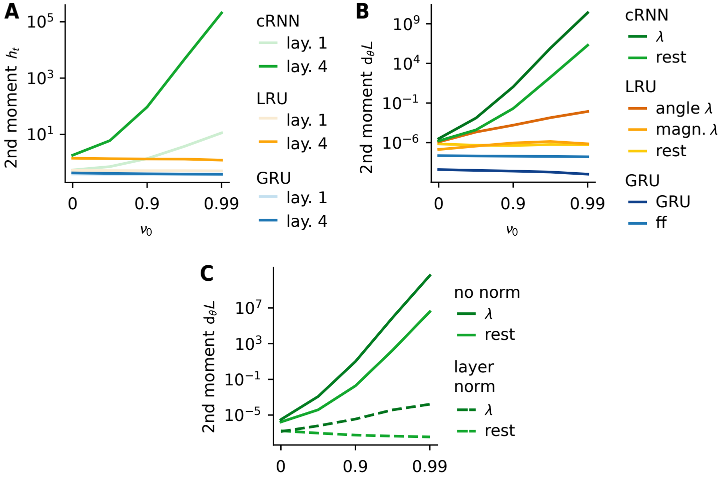


Figure 3.5: Signal propagation in deep recurrent networks at initialization is consistent with our theory. **A.** $\mathbb{E}[h_t^2]$ after the first and the fourth layer, as a function of the exponential decay parameter ν_0, for complex-valued diagonal RNN (cRNN), LRU, and GRU recurrent layers. The input normalization present in the LRU and in the GRU effectively keeps neural activity constant across ν_0 values. **B.** Comparison of the evolution of the loss gradient $\mathbb{E}[(\mathrm{d}_\theta \mathcal{L})^2]$ for the different recurrent layers and specific groups of parameters. For the complex diagonal RNN, the gradients of all parameters explode and in particular the ones of the recurrent parameters, whereas only the ones of the angle of λ explode for the LRU, consistently with the theory. Error signal propagation in GRUs is under control: the magnitude of the gradients is independent of ν_0. The GRU-specific parameters exhibit smaller gradients than the feedforward (ff) ones. **C.** Layer normalization keeps the overall gradient magnitude under control in cRNNs. Batch normalization yields similar results.

To that matter, we provide sentences as input to deep recurrent networks that contain four blocks and use a next-token prediction loss to measure their performance. Each block consists of a recurrent layer followed by a feedforward gated linear unit [204]. By default, there are no normalization layers in this architecture. More details can be found in Appendix A.4.1. We empirically study how $\mathbb{E}[h_t^2]$ and $\mathbb{E}[(\mathrm{d}_\theta \mathcal{L})^2]$ evolve when the characteristic time scale of the recurrent layers, controlled through ν_0, increases. We compare three different recurrent layers: a complex-valued diagonal RNN (cRNN), a LRU, and a GRU initialized with the Chrono initialization [184].

The results are consistent with our theory. Complex-valued RNNs suffer from the curse of memory and recurrent parameters grow faster to infinity than the rest as ν_0 goes to 1. Perhaps more surprisingly, a finer grain analysis reveals that the gradient magnitude is independent of the layer; see Figure A.8. LRUs almost perfectly mitigate exploding behaviors in the forward pass (Figure 3.5.A) as well as in the backward pass (Figure 3.5.B), except for the angle parameter, consistent with our previous analysis. We also wonder whether layer normalization can replace the input normalization and reparametrization of the LRU. We find that it mitigates the memory-induced gradient explosion at the macroscopic level (Figure 3.5.C), but it likely kills any learning signal for the smallest eigenvalues [69]. Finally, the GRU manages to keep the gradient magnitude constant over different characteristic time constants, consistent with the intuition we developed in Section 3.4.2. Preliminary experiments revealed that same trends also hold for LSTMs.

The results presented above for the GRU align qualitatively with the intuition developed throughout the paper. We now consider how well our theory can quantitatively explain this behavior. The primary difference between our simple model and GRUs is that the λ values, referred to as forget gates in the GRU terminology, depend on both inputs and hidden states, and are therefore not constant. Interestingly, we find that GRUs almost behave like the diagonal linear RNNs we have focused on in this paper, particularly for slowly decaying recurrent neurons with high ν_0 values (see Appendix A.4.2). Consequently, applying our theory as if this context-dependency does not exist only introduces minor approximation errors, which we confirm empirically in Appendix A.4.3. Given the similarity of the Chrono initialization to those used in modern architectures like Mamba [70] and Hawk [71], we expect our theory to also serve as a good proxy for studying signal propagation in these models at initialization.

3.7 DISCUSSION

Vanishing and exploding gradients complicate the learning of recurrent networks, but solving these problems is not enough. We uncovered yet another difficulty of training such networks, which is rooted in their iterative nature and arises at the edge of dynamical stability. Reparametrizations and adaptive learning rates can effectively mitigate this behavior in practice, and diagonalizing the recurrence simplifies both. Our analysis additionally reveals the complexity of learning the angle of complex eigenvalues, which may explain why complex numbers were not found to be useful in most recent state-space model architectures [70, 71].

A side finding of our study is the symbiosis between independent modules, which are here neurons and can be more generally small heads, with adaptive learning rate optimizers in linear recurrent networks. Such a design pattern has promising properties: it facilitates online learning [164] and compositional generalization [205], allows for a high level of parallelization [71], and matches, at a high level, the modular organization of the cortex in cortical columns [57]. Understanding how to increase the expressivity of small linear modules while keeping their great optimization properties constitutes a promising avenue for future research.

4

ONLINE LEARNING OF LONG-RANGE DEPENDENCIES

The contents of this chapter were published as a conference paper at the annual conference on neural information processing systems in 2023 [164] and authored by Nicolas Zucchet*, Robert Meier*, Simon Schug*, Asier Mujika, and João Sacramento.

---

* Shared first authorship.

4.1 INTRODUCTION

How can the connections between neurons in a neural network be adjusted to improve behavior? This question, known as the credit assignment problem, is central in both neuroscience [16] and machine learning [36], owing to its fundamental importance for elucidating learning mechanisms in the brain and constructing intelligent artificial systems. However, the complex and nonlinear nature of neural network processing makes the precise allocation of credit an intricate task.

Deep learning provides a compelling solution to the credit assignment problem via gradient descent, which refines network parameters along the locally most promising direction. For networks processing temporal sequences, gradient computation is made possible by backpropagation-through-time [BPTT; 36, 48, 206]. BPTT stores and revisits neural activity in reverse-time order to understand how infinitesimal changes to neural activity, and thus to network parameters, would have impacted the objective function. One important drawback of this algorithm is its requirement to store the entire activity trajectory in memory, which constrains the sequence length for exact gradient computation, impairing the learning of long-term interactions. This constraint becomes a critical bottleneck when working within memory-limited systems, such as neuromorphic hardware [207] and presumably the brain [208].

Alternatives to BPTT for gradient computation do exist. One such approach, forward-mode differentiation [55, 209], involves computing gradients online as the input sequence is processed, by keeping track of the sensitivity of neural activity with respect to each of the network parameters. This marks a qualitative departure from BPTT, as it prepares for all potential future trajectories simultaneously; by contrast, BPTT focuses on improving the activity of a past trajectory. Importantly, the memory footprint of this approach does not depend on sequence length. Still, it remains intractable for real-world applications and likely infeasible in the brain due to its cubic memory scaling and quartic computational complexity in the number of neurons. Recent research focused on approximation strategies to make online gradient estimation more tractable for general-purpose recurrent networks [157–161, 163, 166, 167, 210, 211]. Our work takes a fundamentally different approach: instead of tailoring the learning algorithm to the neural network architecture, we fix the learning algorithm and seek an architecture that makes it tractable.

We build upon recent advances in linear state space models, a class of recurrent neural networks (RNNs) employing linear recurrent blocks [54, 67–69, 212]. These blocks are stacked and interconnected through nonlinear networks. The key insights from this line of research are that linear recurrent connections simplify temporal credit assignment and enable parallel temporal processing, while nonlinearities between recurrent blocks ensure that network expressiveness remains comparable to that of densely-connected nonlinear RNNs. Much, if not all, of the state-of-the-art performance of those models on long-range temporal tasks [213] can be maintained by transitioning from real-valued to complex-valued neural activities [67–69], and restricting the recurrent connectivity matrix to be diagonal. Recurrent neurons within a given layer are now independent of each other. This greatly improves the tractability of online gradient estimation, as the recurrent parameters of a given neuron do not impact other neurons. We leverage this property to achieve exact online gradient computation within a single layer with as little as twice the memory and compute requirements needed for inference. Further, we demonstrate how this leads to improved gradient estimation compared to existing online learning algorithms when recurrent layers are stacked.

This paper is organized as follows. We start by briefly reviewing existing gradient-based online learning methods in Section 4.2.1. Next, we introduce the concept of independent recurrent modules, showing how some of the recent high-performance models mentioned above fit in this framework in Section 4.2.2. Deriving our learning rule requires complex differentiation; we give a concise overview of those tools in Section 4.2.3. In Section 4.3, we detail our online learning algorithm that combines exact differentiation within a layer of recurrent independent modules with spatial backpropagation across layers. Finally, in Section 4.4, we analyze our algorithm and relevant baselines on a synthetic copy task and show that it can learn sequential tasks with sequence lengths up to over 4000 steps.

4.2 BACKGROUND

4.2.1 *Online gradient-based RNN learning*

We study gradient-based learning of recurrent neural networks, which process input data $x_1, \ldots, x_T$ sequentially while maintaining an internal (hidden) state h_t. The objective of learning is to minimize a cumulative loss $\mathcal{L}(\theta) = \sum_{t=1}^{T} \ell_t(\theta)$ which measures performance on a task at hand as a function of network parameters θ. The standard algorithm for computing the gradient $\nabla \mathcal{L}(\theta)$ is the offline backpropagation-through-time method, which requires storing the entire input $x_{1:T}$, loss $\ell_{1:T}$ and internal activity $h_{1:T}$ sequences, and then revisiting them proceeding backwards in time. Here, we focus on online algorithms which carry the information needed to compute or estimate $\nabla \ell_t(\theta)$ forward in time. This enables simultaneous processing of inputs and learning for RNNs, without storing past data and network states. In principle, online algorithms can learn arbitrarily long temporal dependencies as well as seamlessly handle sequences of arbitrary length T.

The classical alternative to BPTT for forward-in-time gradient computation is known as real-time recurrent learning [RTRL; 55] in the context of RNNs[1], a method which has its roots in control theory [215]. While RTRL enables

---

[1] More generally, BPTT can be seen as a special case of reverse-mode automatic differentiation, and RTRL of forward-mode automatic differentiation [214], applied to the problem of RNN learning.

online gradient-based learning, it requires storing $\mathrm{d}_\theta h_t$ in memory and updating it as the RNN processes its inputs. The size of this auxiliary variable is $\mathcal{O}(n^3)$, where $n = |h|$ is the number of hidden units in the RNN. For comparison, the memory requirements of BPTT are $\mathcal{O}(nT)$. In practice, this precludes the usage of RTRL for all but the smallest of models.

There has been much effort in developing memory-efficient alternatives to RTRL. We now briefly discuss these prior efforts while referring to a recent review by Marschall, Cho & Savin [211] for a more detailed treatment. We divide prior work into three broad categories. One class of algorithms relies on neglecting terms in $\mathrm{d}_\theta h_t$ to reduce its size, thereby creating a biased gradient estimator. Typically, this requires introducing crude approximations, which allow going from $\mathcal{O}(n^3)$ to a tractable $\mathcal{O}(n^2)$ size. Despite such approximations, in many cases, performance still holds in non-trivial tasks [157, 158, 163]. At the end of this spectrum sits instantaneous (spatial) backpropagation, which neglects all temporal dependencies in the hidden state when approximating the gradient. A second class of algorithms relies on stochastic estimation; this allows retaining unbiased gradient estimates at the expense of introducing variance [159–161, 166, 167]. Finally, a third class of methods introduces gradient models [critics; 216–218] to produce gradient estimates online. The critics themselves are then either trained separately offline, making such methods hybrid on/offline; or fully online, using temporal difference techniques [219]. We note that despite their widespread use in reinforcement learning, it is not yet well understood whether temporal difference methods can reliably and efficiently improve the performance of a gradient critic on real-world RNN learning problems.

As noted by Irie, Gopalakrishnan & Schmidhuber [165], the RTRL literature mostly focuses on single-layer recurrent networks and remains scarce for deeper networks. Recently, Javed *et al.* [220] developed a greedy learning algorithm where a growing network is progressively frozen and trained one layer at a time. Existing approximations such as e-prop [157] and SnAp-1 [158] do not prescribe how to learn the parameters of remote layers, and the multi-layer case is not considered in the respective papers. Introducing a powerful RTRL algorithm that scales to networks of multiple layers is the main algorithmic contribution of this paper. This property is of great empirical relevance given the power of depth to learn the temporal structure [e.g. 69, 212].

4.2.2 *Linear recurrent units and independent recurrent modules*

Instead of developing approximate, general-purpose forward-mode differentiation methods, our goal shifts towards seeking an expressive architecture allowing exact, tractable online gradient calculation. We propose that networks with linear recurrent units [LRU; 69] and, more generally, networks with independent recurrent modules, are particularly well-suited for this purpose.

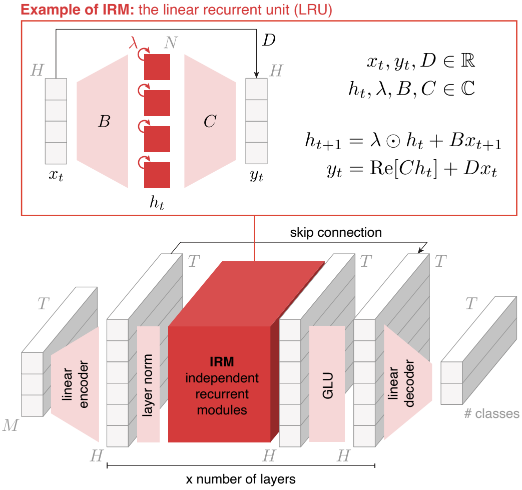

Example of IRM: the linear recurrent unit (LRU)
H
λ
N
D
H
B
C
x_t
h_t
y_t
$x_t, y_t, D \in \mathbb{R}$
$h_t, \lambda, B, C \in \mathbb{C}$
$h_{t+1} = \lambda \odot h_t + Bx_{t+1}$
$y_t = \mathrm{Re}[Ch_t] + Dx_t$
skip connection
T
T
T
T
T
M
linear encoder
layer norm
IRM
independent recurrent modules
GLU
linear decoder
# classes
H
H
H
x number of layers

FIGURE 4.1: OVERVIEW OF THE CLASS OF NEURAL NETWORKS WE CONSIDER. We stack layers of independent recurrent modules (IRMs), augmented with layer norm [87] and gated linear units [GLU; 204]. Light red indicates instantaneous spatial processing, dark red temporal processing. When we examine networks with fully connected recurrent layers, only the dark red block is modified. The linear recurrent unit is the instantiation of a layer of IRMs we use in our experiments.

A linear recurrent unit, depicted in Figure 4.1, is defined as

$$h_{t+1} = \lambda \odot h_t + Bx_{t+1}, \quad y_t = \mathrm{Re}[Ch_t] + Dx_t, \tag{4.1}$$

with $\odot$ the element-wise product. Here, $x_t \in \mathbb{R}^H$ represents the input received by the LRU at time t, $h_t \in \mathbb{C}^N$ denotes its internal state, and $y_t \in \mathbb{R}^H$ its output. The parameters of the unit include $\lambda \in \mathbb{C}^N$, $B \in \mathbb{C}^{N\times H}$, $C \in \mathbb{C}^{H\times N}$ and $D \in \mathbb{R}^{H\times H}$. The version of the LRU we use in our experiments includes an element-wise normalization factor for the input

Bx and uses an exponential parametrization of λ for network stability. We omit these details in the main text for conciseness; see Appendix B.1.1 for more details.

LRUs differ from traditional recurrent layers in deep learning: they have linear neural dynamics, a complex-valued state h_t, and a diagonal connectivity pattern. Orvieto *et al.* [69] found that the absence of temporal linearity in networks that stack those units through nonlinear connections (see Fig. 4.1) does not alter expressivity and eases gradient-based learning, a notoriously difficult process for nonlinear RNNs [50, 52]. In addition, the diagonal structure of the recurrence matrix provides several benefits over fully connected ones. First, it affords an easy way to control the eigenvalues of the Jacobian of the system and thus ensure that neural dynamics remain stable. Due to the linearity, it also enables processing the input sequence in parallel [109], significantly accelerating the training of such models on modern computers [68]. Importantly, despite its diagonal parametrization, the LRU remains functionally equivalent to a linear recurrent layer with a dense recurrence matrix A, as A can be approximated as accurately as needed by a complex-diagonalizable matrix.

Each complex neuron in an LRU is an *independent recurrent module*, meaning its current state does not impact the dynamics of other modules. This property greatly simplifies online credit assignment (see Section 4.3). We focus on this specific architecture due to its simplicity and great empirical performance, but our theoretical insights also apply to networks of independent recurrent modules with low-dimensional state vectors per module.

4.2.3 *A primer on complex differentiation*

The use of complex-valued networks, such as the LRU, and hence complex differentiation remains relatively scarce. In the following, we provide a concise review of the tools of complex differentiation integral to the derivation of our online learning rule. We use f and g to denote complex-valued functions that take the complex variable z as input.

The Wirtinger derivatives of f are defined through

$$\frac{\mathrm{d}f}{\mathrm{d}z} := \frac{1}{2}\left(\frac{\mathrm{d}f}{\mathrm{dRe}[z]} - i\frac{\mathrm{d}f}{\mathrm{dIm}[z]}\right), \ \frac{\mathrm{d}f}{\mathrm{d}\bar{z}} := \frac{1}{2}\left(\frac{\mathrm{d}f}{\mathrm{dRe}[z]} + i\frac{\mathrm{d}f}{\mathrm{dIm}[z]}\right). \tag{4.2}$$

Using them for complex differentiation allows using similar calculus rules as for real functions. Note that we use the row convention for derivatives, that is $\mathrm{d}_z f$ is a row vector of size $|z|$. The following formula holds in general $\overline{\mathrm{d}_z f} = \mathrm{d}_{\bar{z}} \bar{f}$.

The complex derivative of a complex function is similar to a 2×2 real-valued matrix as both $\mathrm{d}_z f \in \mathbb{C}$ and $\mathrm{d}_{\bar{z}} f \in \mathbb{C}$ are necessary to characterize it. Yet, there exists a subclass of functions, called holomorphic functions, for which it can be reduced to a 2 dimensional real-valued vector, leading to a more compact representation of derivatives. A continuous function f is *holomorphic* if it satisfies the Cauchy-Riemann equations

$$\frac{\mathrm{dRe}[f]}{\mathrm{dRe}[z]} = \frac{\mathrm{dIm}[f]}{\mathrm{dIm}[z]} \text{ and } \frac{\mathrm{dRe}[f]}{\mathrm{dIm}[z]} = -\frac{\mathrm{dIm}[f]}{\mathrm{dRe}[z]}, \text{ i.e. } \frac{\mathrm{d}f}{\mathrm{d}\bar{z}} = 0. \tag{4.3}$$

Any affine function, as well as the composition of two holomorphic functions, is itself holomorphic.

The chain rule of complex differentiation, crucial for automatic differentiation, is

$$\frac{\mathrm{d}(f \circ g)}{\mathrm{d}z} = \frac{\mathrm{d}f}{\mathrm{d}g}\frac{\mathrm{d}g}{\mathrm{d}z} + \frac{\mathrm{d}f}{\mathrm{d}\overline{g}}\frac{\mathrm{d}\overline{g}}{\mathrm{d}z}. \tag{4.4}$$

When either f or g is holomorphic, the second term vanishes as $d_{\overline{g}} f = 0$ or $\mathrm{d}_z \bar{g} = \overline{\mathrm{d}_{\bar{z}} g} = 0$. When f is a real-valued function, it can be optimized through gradient descent by iteratively updating its input variable z through $\Delta z \propto -\mathrm{d}_{\mathrm{Re}[z]} f^\top - i \mathrm{d}_{\mathrm{Im}[z]} f^\top = -2\mathrm{d}_{\bar{z}} f^\top$.

4.3 ONLINE LEARNING OF NETWORKS OF INDEPENDENT RECURRENT MODULES

Based on the foundations laid down in the preceding section, we now derive our online gradient-based learning algorithm for multi-layer networks of independent recurrent modules. We first focus on a single layer, demonstrating that exact forward-mode differentiation is tractable. This insight then guides the derivation of our rule for multi-layer networks.

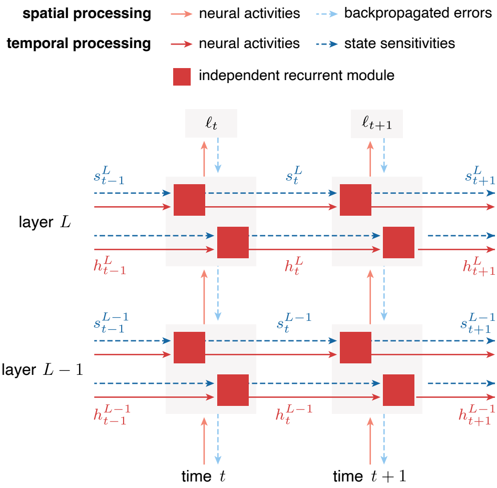


FIGURE 4.2: OVERVIEW OF OUR LEARNING RULE. As an input sequence is processed, hidden states h_t and their sensitivities s_t to the parameters are updated. Learning ensues by combining the sensitivities s_t with spatially backpropagated error signals. No information flows in reverse time; our rule is fully online.

4.3.1 *Single-layer networks*

We focus on parameters θ that influence the hidden states; computing the gradient of any other parameter does not require temporal credit assignment. For the LRU, we have $\theta = \{\lambda, B\}$. Recall that $\mathcal{L}(\theta) = \sum_{t=1}^{T} \ell_t(y_t(\theta))$ denotes the loss function that measures how good the outputs $y_{1:T}$ of a network parametrized by θ are. Its derivative $\mathrm{d}_\theta \mathcal{L}(\theta)$ can be calculated using forward-mode (complex) differentiation:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\theta} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial h_t}\frac{\mathrm{d}h_t}{\mathrm{d}\theta} + \frac{\partial \mathcal{L}}{\partial \overline{h_t}}\frac{\mathrm{d}\overline{h_t}}{\mathrm{d}\theta} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial h_t}\frac{\mathrm{d}h_t}{\mathrm{d}\theta}. \tag{4.5}$$

As mentioned in Section 4.2.3, the last equality holds as h_{t+1} is a holomorphic function of h_t and of θ, hence h_t is a holomorphic function of θ by recursive composition, and $\mathcal{L}$ only directly depends on h_t through ℓ_t. The term $\delta_t := \mathrm{d}_{h_t}\mathcal{L}^\top$ that is here equal to $\mathrm{d}_{h_t}\ell_t^\top$ can easily be computed by spatial backpropagation, as the output y_t at time t only depends on the current hidden state h_t. We are left with computing the sensitivities $\mathrm{d}_\theta h_t$ of the states to the parameters.

Independent recurrent modules do not influence each other. The parameters θ_i that directly influence the state $h_{t,i}$ of module i never impact the state $h_{t',j}$ of another module. As a consequence, the number of non-zero entries of the sensitivity $\mathrm{d}_\theta h_t$ grows linearly with the size of θ whenever the number of recurrent neurons within each module is fixed. Applying this to the LRU, $\mathrm{d}_\theta h_t$ is entirely characterized by $s_t^\lambda := (\mathrm{d}_{\lambda_i} h_{t,i})_i$ and $s_t^B := (\mathrm{d}_{B_{ji}} h_{t,j})_{ji}$. Differentiating Equation 4.1 using the product rule gives the corresponding updates:

$$s_{t+1}^\lambda = \lambda \odot s_t^\lambda + h_t, \;\; s_{t+1}^B = \mathrm{diag}(\lambda) s_t^B + 1 x_{t+1}^\top, \tag{4.6}$$

with 1 a vector of size $|h|$ filled with ones. More detail on the derivation of Equation 4.6 and on how to efficiently simulate this update is given in Appendix B.1.2. Keeping track of those quantities only requires considering an additional hidden state of size $|\theta|$.[2] Finally, the λ and B updates can be obtained by following the gradient, as calculated in Equation 4.5:

$$\Delta\lambda \propto \sum_{t=1}^{T} \delta_t \odot s_t^\lambda, \;\; \Delta B \propto \sum_{t=1}^{T} \mathrm{diag}(\delta_t) s_t^B. \tag{4.7}$$

Interestingly, all the e-updates are local to the neuron or synapse in question, and no approximations were required to accomplish this. This feature makes the algorithm particularly promising for neuroscience and neuromorphic engineering, where localized computation is highly desirable. The parameter update for λ and B is also fully local, as it combines a parameter-specific sensitivity, sometimes considered as an eligibility trace [157], and a postsynaptic error term.

The idea that element-wise recurrence simplifies RTRL precedes our work. It can be found in early work by Mozer [162] and Gori, Bengio & De Mori [221], and has been revisited recently [165, 220]. In this paper, we extend this insight to complex numbers and thus do not lose expressivity, unlike previous work. We also note that some approximations to RTRL such as e-prop [157] or SnAp-1 [158] end up being exact when applied to networks with independent recurrent modules.

---

[2] If h_t is not a holomorphic function of θ, one would need to keep $2|\theta|$ instead of $|\theta|$ states.

4.3.2 *Multi-layer networks*

The derivation in the last section presumes that the loss $\mathcal{L}$ only directly depends on h_t through ℓ_t. This assumption no longer holds when layers are stacked, which is crucial to the expressiveness of the model. In the following, we explain how we can extend our rule to the multilayer case. Let us consider layer l of the network where we aim to compute the gradient of the loss $\mathcal{L}$ with respect to its parameters θ^l. The sensitivity $\mathrm{d}_{\theta^l} h_t^l$ can be computed as before, assuming independent recurrent modules, as θ^l does not influence the behavior of the inputs it receives from previous layers. Hence, we are left with computing $\delta_t^l = \mathrm{d}_{h_t^l} \mathcal{L}^\top$. The simplification $\delta_t^l = \mathrm{d}_{h_t^l} \ell_t^\top$ we made in the previous section still holds for the last layer but is violated for the other layers. This is because $\mathcal{L}$, taken as a function of the hidden states h^l of layer l, now has an internal memory through the subsequent recurrent layers. The hidden state h_t^l at time t will thus directly affect all future losses $\ell_{t'}$ for $t' \geq t$. As a consequence, one has to resort to backpropagation-through-time to compute $\mathrm{d}_{h_t^l} \mathcal{L}^\top$ exactly, which breaks causality and rules out the possibility of learning online. To circumvent this issue, we approximate the error signal each layer receives by $\delta_t^l \approx \mathrm{d}_{h_t^l} \ell_t^\top$ so that it can be computed instantaneously with spatial backpropagation. We emphasize that the only source of approximation of this algorithm is the one above. Given that there is no approximation for the last layer, we will always compute the exact gradient for that layer.

We summarize our learning rule in Figure 4.2. It prescribes augmenting the hidden state of each recurrent layer l with the sensitivity s^l. For each input/output sample $(x_t, y_t^{\text{target}})$, we first update the full entire hidden state (h_t, s_t) using the previous one (h_{t-1}, s_{t-1}) and current input x_t. We then spatially backpropagate the error signal obtained at the last layer by comparing the prediction of the network to its desired value y_t^{target}. Finally, we combine s_t^l and δ_t^l available at each layer using Equation 4.7 to compute the current update. So far, we have only described how to update parameters that directly influence the hidden neurons of the recurrent layer. We update the rest of the parameters with the gradient estimate obtained with spatial backpropagation. Importantly, the size of the extended hidden state is upper bounded by the number of neurons plus the number of parameters and is in practice much smaller.

Next, we aim to understand the factors that may degrade the quality of the gradient estimate as a result of the approximation $\delta_t^l \approx \mathrm{d}_{h_t} \ell_t^\top$ which introduces bias in the gradient estimate. In the LRU, neural activities, and thus error signals, are processed temporally through dynamics similar to the one of Equation 4.1. When the norm of each λ_i approaches one, neural activity preserves past information, and correspondingly, error signals backpropagated over time contain more information about the future. This suggests that our approximation becomes less accurate as the distribution of $|\lambda_i|$ narrows around 1, since it discards future error information. Moreover, δ worsens as it is backpropagated through more layers. At each layer, backpropagation mixes errors from the next layer and the future state of the layer. Since we neglect future information, only part of the error signal is backpropagated, resulting in a less accurate approximation. We delve deeper into these two approximation sources in a memory task in Section 4.4.1.

4.4 EXPERIMENTS

In the following, we analyze the properties of our proposed online learning method empirically and explore how independent recurrent modules aid the learning of long-range dependencies. To this end, we first conduct an extensive analysis on the copy task [53], a well-established test bed to study temporal credit assignment in recurrent neural networks. Comparisons to truncated online versions of BPTT and to an extension of SnAp-1 to deep recurrent networks reveal that independent recurrent modules are generally beneficial for learning long-range dependencies, as they excel in both the online and the offline setting. Finally, we evaluate our method on three tasks of the Long Range Arena [213]: a sequential version of CIFAR [97], ListOps [222] and IMDB [223], scaling online learning to sequence lengths of over 4000 time steps and to deep recurrent neural networks. For additional experimental details and hyperparameter configurations, we refer to Appendix B.2.

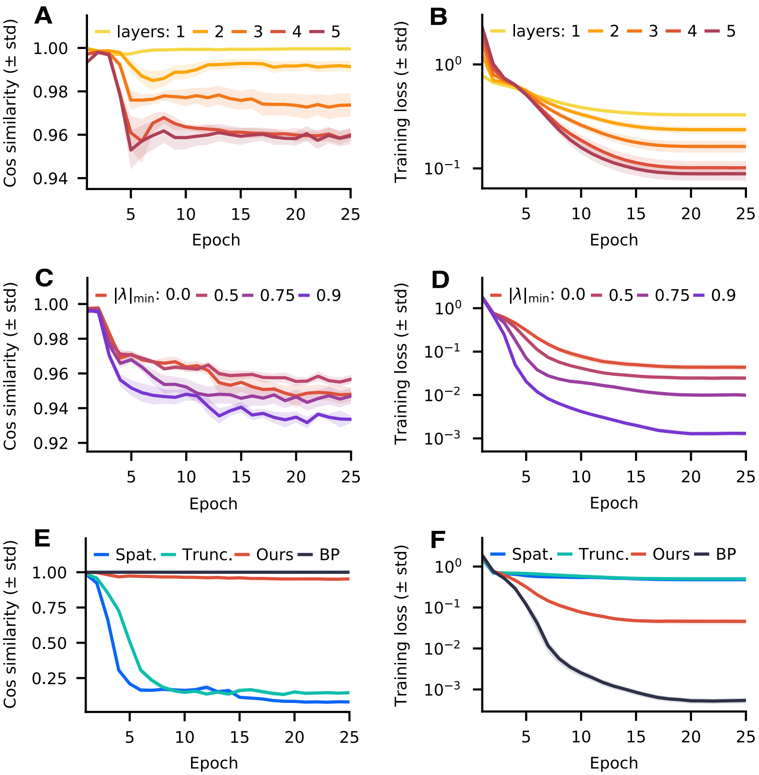


FIGURE 4.3: IMPACT OF DEPTH OF THE NETWORK (LEFT), EIGENVALUES OF THE NETWORK (MIDDLE), AND TYPE OF APPROXIMATION ON THE QUALITY OF ONLINE LEARNING (RIGHT). The cosine similarity measures the alignment between the estimated gradient (with our learning rule for all panels, and with spatial backpropagation (Spat.) and 1-step truncated backpropagation (Trunc.) for panels E and F) and the true gradient, computed with backpropagation-through-time (BP). It is computed per layer and then averaged across layers to make a quantitative comparison possible. The encoder and decoder alignments do not influence this metric. This task is solved (100% accuracy) for losses lower than 0.05 and 70% accuracy roughly corresponds to a loss of 0.5. See Section 4.4.1 for more details.

4.4.1 *Understanding the approximations behind online learning in networks of LRUs*

In this first series of experiments, we investigate the approximation introduced by our online algorithm in detail. We recall that our learning rule only approximates the full gradient when the network has two recurrent layers or more, as it ignores the temporal component of backpropagated error signals. Therefore, we expect that the learning signal becomes less accurate as we increase network depth and shift the eigenvalue distribution towards a norm of 1. To explore the effects of our approximation in a controlled setting, we consider a copy task [53, 158, 160, 166] in which the network observes a length-20 sequence of 7-bit patterns that it must recall when presented with an output token. Intuitively, temporal credit assignment is critical in this task to, among other things, convert forgetting neurons ($|\lambda| \ll 1$) into perfect memory units ($|\lambda| \approx 1$) that can solve the task. To ensure that these perfect memory units are scarce at the beginning of training, necessitating learning to create some, we initialize λ uniformly at random in the complex unit disk and set the number of recurrent units per layer to $N = 64$. The default architecture used in this section has four layers.

As remarked in Section 4.3.2, the updates prescribed by our learning rule match the gradient exactly for all parameters in the last LRU layer. We confirm that empirically for a network of one layer in Figure 4.3.A. While the approximation quality deteriorates with increasing depth (Fig. 4.3.A), alignment remains high, noticeably better than for all baseline online learning rules (Fig. 4.3.E). Moreover, despite alignment decreasing with depth, performance enhances significantly (Fig. 4.3.B). BPTT exhibits similar improvements with depth, suggesting that this is likely due to the enhanced capabilities of the model. Our rule can learn useful hierarchical temporal representations online whereas baseline methods, 1-step truncated and spatial backpropagation, which ignore most of temporal dependencies, fail (c.f. Fig. 4.3.F). Additionally, we found that, despite its bias, our learning rule can decrease the loss to 0 when training a 4 layers network on a simpler memory task for long enough.

Exact error terms are backpropagated through recurrent units by weighting the current prediction error by 1 and the future error by λ. In order to maintain an online algorithm, we ignore this dependency on the future. We expect alignment with the gradient to decrease as the distribution of $|\lambda|$ shifts towards 1. To test that, we initialize the λ uniformly at random in the complex ring of radius $[|\lambda|_{\min}, 1]$. Interestingly, alterations in the initial eigenvalue distribution only slightly affect estimation quality in the beginning of training (Fig. 4.3.C). The key factor seems to be the degradation associated with learning progress, rather than degradation due to

Layer	LRU	Linear RNN	GRU	GRU
Number layers	4	4	1	4
SPATIAL	4.66×10^{-1}	6.20×10^{-1}	6.26×10^{-1}	6.55×10^{-1}
TRUNCATED	2.62×10^{-1}	5.81×10^{-1}	6.20×10^{-1}	6.49×10^{-1}
OURS / SNAP-1	8.44×10^{-3}	5.82×10^{-1}	3.16×10^{-1}	3.27×10^{-1}
BPTT	7.59×10^{-6}	1.07×10^{-4}	2.61×10^{-1}	1.94×10^{-1}

TABLE 4.1: COMPARISON OF FINAL TRAINING LOSSES OF DIFFERENT ONLINE LEARNING ALGORITHMS ON THE COPY TASK OF SECTION 4.4.2. The independent recurrent modules design improves online learning performance. Performance greatly degrades when the LRU is replaced with a dense recurrent matrix (Linear RNN). Comparison with the SnAp-1 algorithm applied to the GRU architecture highlights that online learning of multilayer networks is difficult without element-wise recurrence. Results are averaged over 5 seeds.

larger eigenvalues. Smaller initial $|\lambda|_{\min}$ values slow down training, as more perfect memory neurons have to be recruited, but all initializations eventually lead to solving the task (Fig. 4.3.D).

4.4.2 *Independent recurrent modules improve online learning performance*

After dissecting the learning dynamics of our algorithm, we next show the importance of the independence of recurrent modules for online gradient-based learning. To this end, we compare linear recurrent units to densely-connected linear recurrent layers which do not have independent recurrent modules. To make the comparison fair, we ensure all models have the same number of parameters and recurrent neurons. In addition to the baselines we considered in the previous section, we include an extended version of the SnAp-1 algorithm that combines spatially backpropagated errors with the SnAp-1 sensitivity approximation. This algorithm reduces to ours when applied to networks of independent recurrent modules. Therefore, it enables us to isolate the impact of the independent recurrent module design on online gradient-based learning. We report final training losses in Table 4.1 and refer the reader to Appendix B.2 for experimental details.

Our findings confirm the benefits of independent recurrent modules for online learning, in particular for multi-layer networks. To demonstrate that, we first compare our algorithm on the LRU architecture with the SnAp-1 algorithm applied to an equivalent linear recurrent network. The diagonal approximation of the sensitivity tensor in SnAp-1 introduces an additional bias when learning linear RNNs. We found that this additional bias hurts performance: when moving from offline BPTT to online training, the performance drop is significantly higher for linear RNNs. Interestingly, sensitivity approximation does not bring any performance gain, in this setting, compared to the cruder approximations that are 1-step truncated BPTT and spatial backpropagation.

Additionally, we run experiments on another RNN architecture, the gated recurrent unit [GRU; 132], to better understand the impact of depth in online and offline RNN training, and to confirm the importance of element-wise recurrence for online learning. In the single layer case, consistent with Menick *et al.* [158], we find that the SnAp-1 approximation performs competitively with offline BPTT. However, it suffers from depth in contrast to BPTT that benefits from it. This result highlights the importance of depth in this memory task, as well as the difficulty learning over depth poses for existing online learning algorithms.

4.4.3 *Scaling online learning to the long-range arena benchmark*

While the approximations typically employed to enable online learning prohibit scaling to tasks with extended temporal structure, the results from our previous section have demonstrated the potential of independent linear units for online learning of long-range dependencies. We therefore move to tasks from the challenging long-range arena benchmark [213] specifically designed to evaluate this ability. Transformers excel in almost any benchmark today. However, they perform surprisingly subpar in this setting [178] in which deep state-space models [54] and LRUs [69] achieve impressive results.

We run experiments on three tasks, sequential CIFAR, IMDB and ListOps. In sequential CIFAR, the network receives the 32×32 image as a pixel sequence and has to perform a classification task. In line with Orvieto *et al.* [69], we use the colored version of sCIFAR instead of the grayscale version

Method	sCIFAR	IMDB	ListOps	sCIFAR
Spatial	58.20 ± 0.70	83.50 ± 0.20	32.02 ± 0.27	50.63 ± 0.23
Trunc.	60.01 ± 1.26	84.04 ± 0.47	31.88 ± 0.59	50.53 ± 0.43
Ours	79.59 ± 1.01	86.48 ± 0.41	37.62 ± 0.68	63.71 ± 0.33
BPTT	83.40 ± 1.54	87.69 ± 0.39	39.75 ± 0.17	65.23 ± 0.56

Table 4.2: (left) Test accuracy on three tasks of the LRA benchmark [213] for spatial backpropagation, 1-step truncated backpropagation, our algorithm, and full backpropagation through-time. While we always use per time step local losses during training, we accumulate logits over the sequence during inference. We report the mean and std. for three seeds each. (right) Test accuracy of a linear RNN on the CIFAR task. Instead of our learning rule, we apply the SnAp-1 learning rule extended to the multilayer case, as described in Section 4.4.2.

originally proposed. In the IMDB task, the network is given a text encode in bytes of length at most 4000, and has to perform binary classification. In ListOps, the input is a sequence of numbers, brackets and operators like Max which the model needs to evaluate to determine a classification target in the range from 1 to 10. We do not use the three other tasks of the LRA benchmark: the performance gap between different models is usually small in the Retrieval task (c.f. [69]) and, in our preliminary experiments, we could not reach above chance performance in the PathFinder tasks with BPTT and the modifications we made to make the loss causal, as described in the next paragraph.

In order to make the sequence models employed on this benchmark compatible with the causality requirement in online learning, we remove the time pooling operation during training and consider a local loss term at every time step instead. During inference, we then evaluate our models using the average of the last layer logits in time which respects causality. Moreover, we replace batch normalization with layer normalization to avoid sending batch statistics backwards in time and consider smaller models to lower the computational burden for online learning. For further experimental details, please refer to Appendix B.2.

We report results comparing online learning to spatial backpropagation, truncated BPTT and the BPTT upper bound in Table 4.2 (left). Our online learning algorithm outperforms other online learning approximations, significantly reducing the gap towards BPTT. As in the last section, replacing the LRU with a linear RNN layer in the CIFAR experiment leads to worth online learning performance, c.f. Table 4.2 (right), providing further evidence for the effectiveness of independent recurrent modules for capturing long-term dependencies.

4.5 DISCUSSION

We have demonstrated that long-range dependencies can be learned online, allowing recurrent neural networks to reach strong performance on a set of tasks from the long-range arena benchmark. Moreover, a detailed analysis of a memory problem revealed that our method significantly outperforms both spatial (online) backpropagation as well as prior approaches based on approximate real-time recurrent learning, coming close to full backpropagation-through-time. These findings may inform the design of new neuromorphic hardware with on-chip learning capabilities, an application where approximate real-time recurrent learning is garnering significant attention [224–226].

While most prior related work focused on developing generic gradient approximation schemes, we asked which architecture would simplify online gradient computations. In high-level terms, our philosophy draws from seminal work on long short-term memory networks [LSTMs; 53] or neural Turing machines [227], which established the importance of architecture design for the success of gradient descent. We build on this insight, moving to the harder setting of online learning. This led us to consider networks built of recurrent independent modules: decoupled units with low-dimensional state vectors, for which exact real-time recurrent learning is cheap. Importantly, this design underlies recent models such as deep linear recurrent units [69] and S4-related models [67, 68] which achieve strong performance in a wide array of challenging problems, including language modeling at scale [228] and the long-range arena benchmark [68].

We conclude by noting that modularity, the overarching principle behind our approach, is at the very heart of the influential columnar hypothesis in neuroscience [229]. This hypothesis states that the architecture of the neocortex is modular, with the cortical column as an elementary (or canonical, [230]) building block one level of abstraction above neurons. We thus speculate that modularity could be a key neural network design principle discovered by evolution, that considerably simplifies the temporal credit assignment problem. This is in line with our finding that a modular architecture enables learning complicated temporal dependencies through simple local temporal credit assignment mechanisms, letting spatial backpropagation take care of assigning credit over the network hierarchy. We stress this point because numerous biological implementations and alternatives for spatial backpropagation have been proposed [e.g., 37–39, 42, 44, 45, 231–237], while essentially none exist yet for backpropagation-through-time [238]. Our findings provide a starting point for understanding how the brain deals with the fundamental problem of learning the temporal structure behind its sensory inputs.

Part II

HOW DATA SHAPES LEARNING? A CASE STUDY ON ASSOCIATIVE MEMORIES AND THE ROLE OF DATA DIVERSITY

5

HOW DO LANGUAGE MODELS LEARN FACTS? DYNAMICS, CURRICULA AND HALLUCINATIONS

The contents of this chapter were published as a conference paper at the conference on language modeling in 2025 [239] and were authored by Nicolas Zucchet, Jörg Bornstein, Stephanie Chan, Andrew Lampinen, Razvan Pascanu, and Soham De.

5.1 INTRODUCTION

Large language models offer a powerful and intuitive interface to access the vast amounts of knowledge on which they were trained. The learning process acts as a lossy compression algorithm turning training data into neural network parameters, thereby implicitly determining what information is preserved within the final model. The extent and nature of this information loss depend on numerous factors, including architecture, training objective, and data distribution, dependencies that remain poorly understood. Deciphering the mechanisms underlying knowledge compression is increasingly important as these models are becoming our main gateway to human knowledge. Our work addresses this challenge by systematically investigating how language models acquire factual knowledge.

Inspired by a long history of work in neuroscience focusing on associative recall to study [61] and model [62] intelligence, we focus on a synthetic factual recall task to study in depth the learning dynamics of language models. Our contributions are:

- We adapt the synthetic task of Allen-Zhu & Li [240], which generates artificial biographies for testing factual recall, to make it suitable for studying the formation of associative memories (Section 5.2).

- We find that language models learn in multiple phases on this task. They first learn overall distribution statistics, then their performance plateaus, and, finally, they acquire individual-specific knowledge (Section 5.3.1). Leveraging a novel attention patching technique, we demonstrate that attention-based recall circuits develop during this plateau (Section 5.3.2).

- We analyze the impact of data distribution properties on these dynamics, revealing that imbalanced distributions accelerate the transition through the intermediate plateau phase (Section 5.4) but lead to overfitting. We further demonstrate that data scheduling strategies can exploit this accelerated transition while mitigating overfitting, providing a rare example in which data curricula benefit (self-)supervised learning.

- We highlight the ineffectiveness of fine-tuning to incorporate new knowledge in the model (Section 5.5). This stems from two related factors: First, models hallucinate (overconfident predictions on unseen individuals) as soon as they acquire individual-specific knowledge. Second, associative memories stored in feed-forward layers are rapidly corrupted when training on new individuals.

5.2 AN EXPERIMENTAL SETUP TO TRACK KNOWLEDGE OVER THE COURSE OF LEARNING

This study investigates the learning dynamics underlying factual recall and knowledge acquisition in language models. This presents two main methodological challenges: First, we need to isolate the model's knowledge from other abilities. Second, we want to evaluate the models over the course of learning, which requires computationally efficient measurement techniques. Following Allen-Zhu & Li [240], we train language models on synthetic biographies that feature key properties of datasets used to train large language models. By carefully designing these synthetic biographies, we can attribute the model's ability to predict specific tokens solely to its acquired knowledge and efficiently measure its knowledge through its performance on these tokens. In this section, we first define knowledge and contrast it with memory, then describe our synthetic dataset and introduce the metrics we use to track knowledge during training, and finally, describe training details.

5.2.1 *Knowledge, and how it differs from memory*

For our analysis, we define knowledge to be the information a model has internally stored about its training data, abstracted from the specific form in which it was encountered. It is important to distinguish it from memorization. While knowledge involves accessing and applying information flexibly across different contexts, memorization is tied to particular training instances. In this view, knowledge can be understood as a form of memorization in a latent semantic space. For instance, knowing that Paris is France's capital represents knowledge, while remembering the exact sentence 'Paris is the capital of France' would constitute memorization. From a practical perspective, this distinction carries significant implications. Memorization can be problematic due to potential data leakage [e.g., 241], while knowledge is a more desirable property that enables models to ground their outputs in factual reality, making their responses both accurate and generalizable.

5.2.2 *Synthetic biographies as a framework for studying knowledge*

An important methodological challenge in our study is to isolate knowledge from other model capabilities. To address this, we adapt the synthetic biography dataset of Allen-Zhu & Li [240], which offers several appealing properties. First, all facts in the dataset are atomic, meaning no fact can be derived from others, thus allowing us to disambiguate knowledge recall from other abilities like reasoning. Second, it is fully synthetic, which allows for precise control of the data distributional properties, while employing natural language with realistic statistics. For example, sufficient textual variation is crucial for enabling genuine knowledge acquisition rather than mere memorization [240], cf. Figure C.1 in the appendix. Third, and most importantly, previous work has established that relatively small language models trained on this dataset [240] store and use knowledge in a similar way to large language models [242–245].

We summarize the data generation process in Figure 5.1. The dataset contains a population of N randomly sampled individuals, each with a unique name and six attributes: birthdate, birthplace, university, major, company, and location. To generate a biography, we first sample an individual from

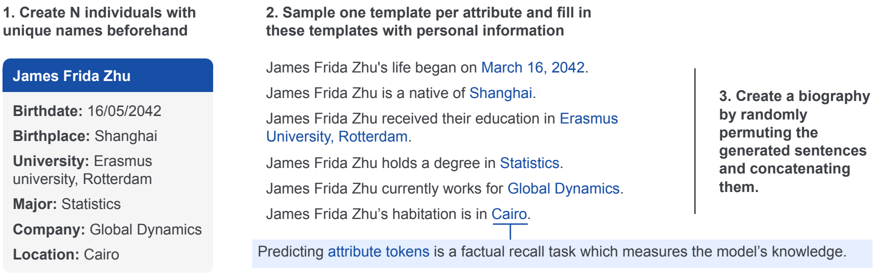

1. Create N individuals with unique names beforehand
James Frida Zhu
Birthdate: 16/05/2042
Birthplace: Shanghai
University: Erasmus university, Rotterdam
Major: Statistics
Company: Global Dynamics
Location: Cairo
2. Sample one template per attribute and fill in these templates with personal information
James Frida Zhu's life began on March 16, 2042.
James Frida Zhu is a native of Shanghai.
James Frida Zhu received their education in Erasmus University, Rotterdam.
James Frida Zhu holds a degree in Statistics.
James Frida Zhu currently works for Global Dynamics.
James Frida Zhu's habitation is in Cairo.
3. Create a biography by randomly permuting the generated sentences and concatenating them.
Predicting attribute tokens is a factual recall task which measures the model's knowledge.

FIGURE 5.1: DATA GENERATION PROCESS UNDERLYING THE SYNTHETIC BIOGRAPHY DATASET WE TRAIN ON. We measure the knowledge stored within these models through the loss they achieve when predicting the attribute tokens, highlighted in blue. See Section 5.2 for more details.

the population. For each attribute, we then sample a template from a finite pool, fill it with the individual's information, and concatenate the resulting sentences in random order to form a complete biography. It should be noted that to make the dataset suitable for our study, our template generation and manipulation differ from the original setup. We detail these modifications in the next section and provide additional details in Appendix C.2.

5.2.3 *Measuring knowledge at scale*

Measuring knowledge in language models typically relies on factual question-answering tasks [246]. However, this approach is computationally intractable in our setup, as it requires repeatedly fine-tuning models to handle question-answer prompt formats throughout the training process. Our framework offers a simple solution to this challenge: by consistently placing information about the individual and attribute type before the attribute value, we transform each attribute value prediction into a factual recall task. This lets us quantify knowledge via the model's loss and accuracy on predicting attribute value tokens. We denote them as *attribute loss* and *attribute accuracy*. Formal definitions are provided in Appendix C.2. The attribute accuracy metric is arguably more intuitive, as it serves as a proxy for the accuracy with which the model, once fine-tuned, would correctly answer recall questions. However, we focus most of our analysis on the attribute loss due to its greater interpretive power and as it generally exhibits more consistent progression across various scales [247].

Two important considerations remain to be solved: how to distinguish knowledge from memorization, and what is a meaningful baseline for the attribute loss. To address the first point, we evaluate models on unseen biographies of previously encountered individuals. We do so by randomly splitting the template pool into training and evaluation templates for each individual, with this split varying across individuals. This ensures that models are evaluated on entirely new biography formulations, thereby ruling out the possibility that strong performance stems from memorization. For the second point, we introduce a theoretical baseline corresponding to the best possible performance achievable by a model with no individual-specific knowledge. Such a model would only know the overall distribution of attribute values, and its loss would be equal to the entropy of the attribute value distribution. We refer to this value as the *no knowledge* baseline. Any model performing better than this baseline must necessarily have acquired some knowledge about specific individuals in the training distribution.

5.2.4 *Language models are trained following standard recipes*

While we use synthetic data in our experiments, we keep the architecture and the training protocol as close as possible to standard practices. By default, we train an 8-layer decoder-only Transformer (44M non-embedding parameters) with the same structure as in Hoffmann *et al.* [248] using the AdamW optimizer [249] and a cosine learning rate schedule without warm-up. We tune the learning rate in all our experiments. Additional details are provided in Appendix C.2.

5.3 HOW LANGUAGE MODELS ACQUIRE KNOWLEDGE DURING LEARNING

Our findings reveal that language models initially learn generic statistics before acquiring individual-specific knowledge. When individuals appear less frequently, such as when the population size grows, performance can plateau for extended periods between these two phases. Mechanistically,

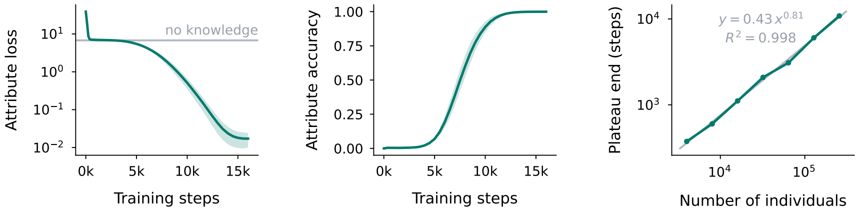


Figure 5.2: Knowledge acquisition occurs in three phases. (left) In a very short first phase, the model learns generic attribute value statistics. In the second phase, performance plateaus at a level achievable by an ideal model lacking individual-specific knowledge (this corresponds to the *no knowledge* baseline defined in Section 5.2.3 and a near-zero knowledge accuracy). This plateau's duration is nearly proportional to the number of individuals (right). Finally, the model learns associations between subjects and attributes: knowledge emerges as the model is trained longer (middle). Results are averaged over 5 seeds (± std). See Section 5.3.1 for details.

we attribute this plateau to the development of attention-based circuits that enable the recall of knowledge stored within the network's parameters and modulate the learning speed of the rest of the model.

5.3.1 *The three phases underlying knowledge acquisition*

The temporal evolution of the attribute loss (Figure 5.2) exhibits a consistent three-phase pattern:

1. Initial language understanding. Early on during learning, the network learns the overall attribute value distribution. At the end of this phase, it performs like an optimal model without individual-specific knowledge, matching the no knowledge baseline.

2. Performance plateaus at the edge of knowledge acquisition. The model's performance then plateaus at the no knowledge baseline. This could be both explained from an optimization argument – the network parameters are at a saddle point of the loss landscape – or from a statistical argument – the model needs to observe multiple biographies from the same individual to discern that attribute values are specific

to each individual. The plateau length grows almost linearly with the number of individuals (Figure 5.2, right panel), supporting the statistical hypothesis.

3. KNOWLEDGE EMERGENCE. The model finally enters a knowledge acquisition phase, in which it progressively learns association between individuals and their attributes. Its knowledge accuracy progressively leaves the near-zero regime in this phase: knowledge emerges.

The pattern described above is robust across a range of hyperparameter configurations. Specifically, qualitative results remain consistent when varying learning rate, weight decay, batch size, number of individuals, model size, and even the type of sequence mixing block (attention-based vs. recurrent), cf. Figure C.3. This is consistent with the findings of [250], who observed a similar pattern in a theoretically tractable version of the task learned by a linear attention layer. These findings suggest that the data structure, particularly the high individual-to-attribute ratio, plays a critical role in driving this multi-phase learning behavior. [251] found a similar behavior in a setup closely related to ours, and that, below some critical model size, the model does not learn any individual-specific information at all despite extensive training. We additionally discuss connections to existing work studying Transformer learning dynamics through the lens of N-grams in Appendix C.1.

5.3.2 *The attention-based circuits supporting recall are created during the loss plateau*

Although knowledge retrieval performance remains constant during the plateau phase, we observe the development of attention-based circuits supporting individual-specific attribute recall. We argue that learning is considerably stalled when these circuits are under development, as errors are not properly backpropagated from attribute value tokens to name tokens, explaining the existence of the plateau. The following evidence supports this claim.

Previous work has analyzed how Transformer-based language models solve factual recall tasks similar to the one we are interested in this study. It can be summarized as follows (cf. Figure C.5 for a visual summary). The

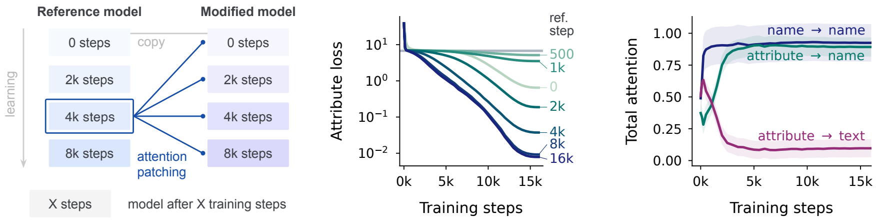


Figure 5.3: The attention-based circuits supporting recall are created during the loss plateau. (left) We design an attention patching experiment, in which we take a snapshot of a reference model at some time during its training, and use its attention patterns as a replacement for the ones of a modified model throughout its training. (middle) The more trained the reference model is, the better its attention patterns are for the modified model, and these changes mainly happen during the plateau. The very beginning of learning is an exception to this trend. This correlates with the fact that, at this time during training, the name tokens (compared to the rest of the text, which contains information about the attribute type) receive reduced attention when the first attribute value token is predicted (cf. right panel). See Section 5.3.2 for more details.

first attention layers aggregate name tokens together to form a representation of the individual's name at the position of the last name token. This representation serves as a query for subsequent layers, particularly the multi-layer perceptrons, which effectively act as a key-value database [242, 243]. Individual-specific knowledge thus typically appears within the final name token's residual stream [240, 244, 245]. The final attention layer then selects the relevant information conditioned on the attribute type and makes it available for predicting attribute value tokens when needed [244, 245]. With this last circuit established, attribute value prediction errors directly backpropagate to relevant tokens, here the name tokens. However, without it, errors are spread across irrelevant tokens, hindering learning efficiency. We therefore hypothesize that the plateau phase corresponds to the formation of these attention-based recall circuits, with learning speed directly linked to their development stage.

A consequence of that hypothesis is that a model with learned attention patterns should learn faster than one with pre-learning patterns. To test this, we design the following *attention patching* experiment (Figure 5.3, left). We first train a reference model and save reference checkpoints from it at various stages of training. We then restart training a model from scratch, but replace the model's attention patterns[1] with those produced from one of the reference checkpoints. This means that all the token-to-token interactions are specified by the reference model and that the modified model only learns token-wise feedforward transformations. We expect the quality of the attention patterns, measured by how easy it is to learn the task with them, provided by the reference model to progressively improve as they are taken closer to the plateau end. This is what we observe empirically (Figure 5.3, middle), and the plateau disappears when providing the model with learned attention patterns (i.e. post-plateau patterns). Interestingly, the attention patterns acquired early during learning (e.g., at steps 500 and 1k) are significantly worse than the pre-learning ones, likely due to initial focus on predicting the attribute value distribution conditioned on the attribute type only. This attention on attribute type tokens rather than on name tokens slows down learning, as per our argument from the last paragraph.

So far, our results show the formation of an attention-based circuit during the plateau phase, though not necessarily the specific extraction circuit we hypothesized. To investigate this further, we examine the evolution of the network's attention patterns during training for signatures of such a circuit. Specifically, we analyze the attention paid to name tokens (relative to general text tokens containing attribute type information) when predicting the first attribute value token, the primary position testing factual recall. We observe this attention increasing throughout the plateau, after being significantly lower during the initial phase where predictions align with the generic attribute value distribution (Figure 5.3, right panel). This offers an intuitive explanation for the qualitative differences in attention patterns observed in Figure 5.3 (middle). A similar analysis for the name tokens grouping circuit, characterized by high attention to name tokens from the last name token in the first layer, reveals its emergence within a few hundred steps. This may be specific to our setup, in which such a circuit is useful to better predict which template comes next. Details of this analysis are provided in Appendix C.4.2. Together with our attention patching results, this result suggests that the attention-based extraction circuit responsible for recall develops during the plateau phase, consistent with the findings of [252]. Note that our results do not rule out that other qualitative changes may occur during this phase, and they do not provide any mechanistic insights regarding how models acquire individual-specific knowledge.

---

[1] We call attention patterns the softmax of the attention logits.

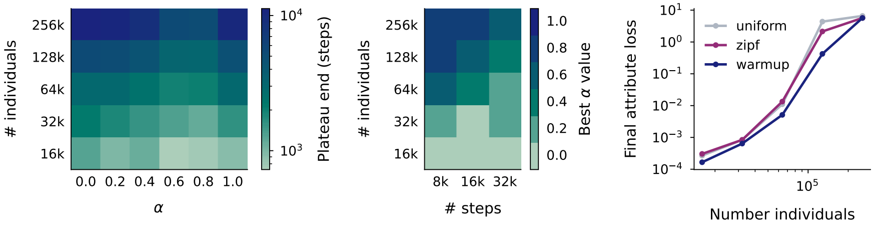


FIGURE 5.4: DATA DISTRIBUTIONAL PROPERTIES CAN SPEED UP KNOWLEDGE ACQUISITION. (left) The length of the plateau significantly decreases when some individuals appear more frequently (up to some extent) than other, which is here achieved by increasing α. (middle) As a result, it is beneficial to train the model on more imbalanced distributions (higher α values), particularly as the number of training steps decreases or as the total number of individuals increases. This panel reports the α value that minimizes the final attribute loss for each number of steps and individuals. (right) Such a strategy, improves the final amount of knowledge contained in the network (purple vs. grey line). Dynamically adapting the data distribution yields even larger benefits (blue line). See Section 5.4 for more detail.

5.4 DATA DISTRIBUTIONAL PROPERTIES DRIVE KNOWLEDGE ACQUISITION

While each individual appears with equal frequency in the preceding analysis, real-world data can be significantly less balanced. This section investigates the impact of such imbalances on the learning dynamics. We find that plateau length scales with the frequency of the most prevalent individuals (minimized in highly imbalanced distributions), whereas knowledge acquisition speed primarily depends on the frequency of the least prevalent ones (maximized in uniform distributions). This observation has two implications we confirm empirically: First, the training distribution that maximizes the final knowledge stored within the model becomes more imbalanced as the amount of knowledge within the data increases, or the network is trained for a shorter amount of time. Second, dynamically adjusting the data distribution during training allows for simultaneous minimization of plateau length and maximization of knowledge acquisition speed.

5.4.1 *The trade-off underlying imbalances in the data distribution*

Building on the analysis presented in the preceding section, we can already develop an intuition of how data distribution impacts the plateau and knowledge acquisition phases. Assuming that the circuits that the network creates during the plateau transfer to other individuals, reducing the number of individuals should minimize time spent in this phase, as shown in Figure 5.2 (right). Naturally, this assumption will break when an excessively small number of individuals is over-represented, as the model will likely overfit. As a consequence, imbalanced distributions with a small group of over-represented individuals should generally shorten the plateau duration. However, this does not hold for the knowledge acquisition phase. Each individual contributes equally to the total amount of knowledge the model has, so the attribute loss on the entire population will asymptotically behave like the one on the group which is learned slowly, that is, the least frequent one. For that reason, a uniform distribution should be optimal for knowledge acquisition. As a consequence, there is a trade-off: imbalanced distributions speed up the plateau and slow down knowledge acquisition, whereas the reverse holds for more uniform distributions. The distribution that optimizes the final amount of knowledge stored within the network weights should therefore become increasingly more imbalanced as the plateau takes a larger portion of training time, i.e., when the number of training steps decreases or when the number of individuals increases.

To empirically test this intuition, we modify the individual occurrence probability to follow an inverse power law with exponent α. This exponent controls the balance of the distribution: $\alpha = 0$ recovers the uniform distribution whereas $\alpha = 1$ corresponds to the highly imbalanced Zipf law. Figure 5.4 (left) reports how the number of training steps needed to escape the plateau evolves with different numbers of individuals and α values, when the total training budget is fixed to 16k steps. As expected, increasing α reduces the plateau length. However, excessive increases are detrimental, likely due to overfitting. We find that the optimal α minimizing plateau length lies between 0.6 and 0.8, irrespective of the population size, while the plateau length itself increases with the total number of individuals. We expect this improvement to be particularly significant when the plateau consumes a large portion of training – i.e., with large population sizes or limited training budgets. Figure 5.4 (middle) confirms this: the α minimizing final attribute loss follows the anticipated trend. Overall, the properties of the data distribution significantly impact final knowledge retrieval performance, as demonstrated in Figure 5.4 (right), especially when individuals appear infrequently, as is likely the case during the pre-training of large-scale models.

We are aware of three related findings. First, Allen-Zhu & Li [240] demonstrated the benefits of "celebrities" in a similar setup to ours. Our analysis provides a partial explanation for this phenomenon. While we find that celebrities offer comparable benefits to our inverse power law distribution (cf. Appendix C.5), their effect may be amplified in their setup due to their use of a single biography for non-celebrities, whereas we never repeat biographies. Second, Charton & Kempe [253] showed that repetition benefits Transformer training on arithmetic tasks. Their learning curves exhibit numerous plateaus, and sample-level repetition reduces time spent on these plateaus. Our analysis may generalize to this task, suggesting a broader principle underlying our findings. Finally, Park *et al.* [254] trained Transformers on a synthetic mixture of Markov chains and found that low task diversity shortens plateau length, albeit leading to solutions that do not generalize as well out-of-distribution. The next chapter will be dedicated to theoretical analysis of this behavior.

5.4.2 *Data schedulers increase the final amount of acquired knowledge*

The previous section revealed a trade-off in choosing a data distribution, depending on the learning phase we wish to optimize. That analysis assumed a fixed distribution throughout training. However, we can envision data distribution schedulers that dynamically adapt the distribution to the current learning phase. We confirm that this strategy can yield substantial improvements.

We implement a "warm-up" strategy: the model initially trains on a subset of individuals for a fixed number of steps, and on all individuals afterwards. Starting with a subset of individuals should shorten the plateau, while the subsequent uniform distribution maximizes knowledge acquisition once the recall circuits are well established. Indeed, we observe significant gains, particularly when the number of individuals is large and the model starts acquiring knowledge (Figure 5.4, right), which is a practically relevant regime. Experimental details and additional analysis are available in Appendix C.5.

While our experiments used a fixed warm-up duration, it could be dynamically adjusted based on observed plateau termination. This training strategy shares goals with curriculum learning [255], but differs in that data complexity remains constant. Although preliminary and task-specific, this promising result suggests a novel strategy for accelerating training in neural networks that exhibit similar loss plateaus on learned sub-tasks. Further investigation is needed to better understand the precise conditions under which data repetition can effectively reduce time spent on performance plateaus during training.

5.5 HALLUCINATIONS HINDER THE INTEGRATION OF NEW KNOWLEDGE POST-TRAINING

This final section examines the challenge of expanding language models' parametric knowledge through post-training procedures like fine-tuning. We find that certain types of hallucinations (overconfident predictions on unseen individuals) emerge simultaneously with knowledge about individuals within the training distribution and can be detected at the population level. These hallucinations significantly impact learning dynamics on new data, requiring numerous training steps to overcome miscalibration, during which pre-existing knowledge is significantly degraded, a phenomenon reminiscent of catastrophic forgetting [256]. Adding replay of existing knowledge to the fine-tuning data mix only partially mitigates this issue. Overall, our findings offer an explanation for the infrequent use of fine-tuning [257, 258] for incorporating new knowledge in the model's parameters.

5.5.1 *Models start hallucinating as soon as they acquire knowledge*

Before investigating fine-tuning per se, we first analyze the evolution of the model's performance on unseen individuals over the course of (pre-)training, focusing on whether it predicts attribute values following the relation-based distribution, i.e., whether it matches the no-knowledge baseline. We find that the model hallucinates, confidently predicting incorrect attribute values (Figure 5.5, left), but with lower overall confidence compared to predictions for individuals seen during training (Figure C.13). Furthermore, hallucinations emerge simultaneously with knowledge acqui-

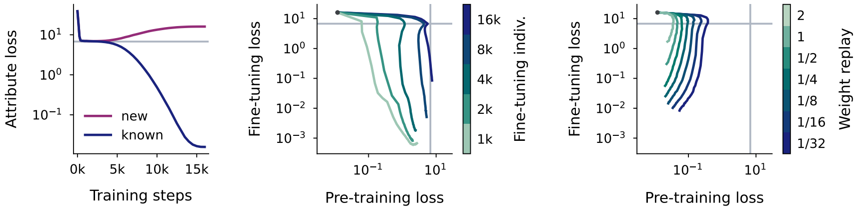


Figure 5.5: Hallucinations hinder the integration of new knowledge post-training. (left) Hallucinations (overconfidence in inaccurate predictions) appear concurrently with the knowledge acquisition, hindering subsequent adaptation to new knowledge. (middle) Fine-tuning on new individuals causes a rapid drop in performance on individuals learned during pre-training, with new knowledge acquisition being a slower process. (right) Incorporating replay of pre-training data partially mitigates the final performance drop, but not the initial decline. Grey dots in the middle and right panels indicate performance at the beginning of fine-tuning. The pre-training (resp. fine-tuning) losses are attribute losses measured on pre-training (resp. fine-tuning) individuals. See Section 5.5 for details.

sition about the training individuals, suggesting they may be a counterpart to accurate predictions in current language models.

5.5.2 *A large portion of the pre-training knowledge is erased during early fine-tuning*

Given this uncalibrated starting point, fine-tuning on new individuals is expected to be challenging. We observe a rapid collapse in performance on pre-training data during the initial stages of fine-tuning (first few hundred steps), with very little corresponding acquisition of new knowledge (Figure 5.5, middle). A larger number of fine-tuning individuals intensifies this effect. Extended fine-tuning eventually changes this trend, and the model starts learning about new individuals. Incorporating replay data about pre-training individuals in the fine-tuning set partially mitigates the initial collapse and facilitates the restoration of corrupted knowledge (Figure 5.5, right). Finally, we do not observe such degradation when fine-tuning on subsets of the pre-training data (see the corresponding analysis in Appendix C.6.3). These results are consistent with the findings of Gekhman *et al.* [259], who found that fine-tuning on new knowledge is slow and leads to hallucinations. We complement our analysis of fine-tuning dynamics with one in which the training distribution changes regularly (Appendix C.6.5). These experiments confirm the findings reported here, as well as highlight that it becomes easier to learn new individuals as training progresses.

We now turn to explaining this behavior. The first hypothesis we consider is related to attention patterns. Indeed, introducing new individuals may disrupt attention patterns, requiring time to restore recall ability, while keeping parametric knowledge relatively untouched, and further training may restore them. We analyze the network's attention patterns, in both our fine-tuning (Figure C.15) and switching distribution (Figure C.24) setups, and find them to be remarkably stable, therefore mostly ruling out this hypothesis. Our second hypothesis is that introducing novel individuals creates additional key-value pairs that corrupt existing ones when being learned. To test this hypothesis, we train a one hidden layer multi-layer perceptron on a toy associative recall task (Appendix C.6.4). With such a model, which has no attention layers at all, we are able to reproduce the main qualitative findings of the experiments above, suggesting that changes in the feed-forward associative memories of the model we consider are the main factor explaining the model's behavior. Getting a finer understanding of these dynamics constitutes an interesting future research direction. For example, investigating how introducing independent knowledge topics affects this early catastrophic forgetting could provide some practically relevant insights.

5.6 DISCUSSION

ON THE LEARNING DYNAMICS OF LANGUAGE MODELS. This work reveals the existence of phase transitions, the formation of induction heads being the canonical example [260–262], in tasks as simple as factual recall. While the results presented above were obtained on an isolated task, we speculate that even with natural text and its multi-task nature, task-specific learning dynamics retain the same abrupt transitions and plateaus appear when the formation of a new circuit is the main bottleneck for solving a specific task. We found the time spent in the transition phase to depend more on the data distribution than the model size. Emergent abilities [263] can thus result from increased training time as models are scaled. On the data distribution side, our results have two consequences: First, they suggest the benefits of using synthetic data early in pre-training, since data used before the end of the plateau is not retained in the final model. Second, data schedulers, potentially even adaptive ones that reduce diversity when performance plateaus on a task, appear to be a promising direction to improve learning speed. Finally, our identification of changes in feedforward associative memories leading to rapid performance drops during early fine-tuning provides a simple explanation for the practically observed ineffectiveness of fine-tuning for new knowledge (e.g., [257, 258]).

On the learning dynamics of neural networks more broadly. This analysis reveals that, in a simple but relevant setting, attention-based recall circuits emerge before the formation of associative memories in feedforward layers. We hypothesize that this occurs as established circuits amplify the correlation between the inputs and backpropagated errors received by feedforward layers. Prior to circuit formation, task learning remains possible (e.g., by artificially increasing name token values), but progress is slow, performance plateaus, and generalization likely suffers. Phenomena like grokking [73, 245] may indicate that learning dynamics initially find this shortcut, before abandoning it due to regularization. The proposed credit-assignment argument for plateau formation generalizes to other architectures, as evidenced by ablations in which attention is replaced with recurrence (Figure C.3). Decoupling the token-mixing role of attention from other computations, as advocated by [29] for mechanistic understanding of trained Transformers, also proves powerful for analyzing learning dynamics. This perspective offers a promising avenue for future investigations into neural network learning dynamics.

ON THE ROLE OF NON-UNIFORMITY AND CONNECTIONS TO DEVELOPMENTAL PSYCHOLOGY. A key finding of this work is the precise analysis of how imbalances in the training distribution enable faster escape from performance plateaus. While not explicitly connected to the formation of attention patterns, similar phenomena have been observed in diverse settings [253, 254], though sometimes at the cost of limited generalization. Our data scheduling results, by adapting the data to the network's current learning phase, offer a promising path towards accelerated learning without sacrificing generalization. Intriguingly, these strategies mirror the implicit curriculum experienced by developing infants [63], arising from factors like limited mobility or frequent exposure to familiar faces, and similar strategies have proven successful in reinforcement learning [264, 265]. As argued earlier in the discussion, repetition strengthens the signal-to-noise ratio in correlations between events (tokens in our case), facilitating faster identification of key relationships and accelerating learning. Crucially, however, (progressive) diversification remains essential for optimal generalization in later stages of learning. Our findings could therefore serve as a building block for a statistical theory of development.

LIMITATIONS

In our study, we train relatively small language models on a synthetic factual recall task to investigate knowledge acquisition dynamics. While this approach enables precise experimental control and mechanistic insights, it presents several important limitations.

SCALE AND ARCHITECTURE. The 44M parameter models we train are substantially smaller than the billion-parameter models used in practice. The learning dynamics we observe may manifest differently at larger scales or with different architectural choices. However, our ablations across model sizes, architectures (including recurrent variants), and hyperparameters suggest the core phenomena are robust to these variations.

DATA. The synthetic factual recall task, while designed to capture essential properties of knowledge acquisition, only represents one specific type of knowledge language models acquire during pretraining. Different types of knowledge could interact in sophisticated ways that are not modeled in our task. This may create extra redundancy that language models could leverage to accelerate learning. Nevertheless, our fine-tuning results already align with practically observed inefficiencies of fine-tuning for integrating new knowledge, suggesting some degree of transferability.

Despite these limitations, we believe our work provides a necessary foundation for understanding knowledge acquisition in neural language models. The mechanistic insights we derive offer concrete hypotheses that future work can validate in more realistic scenarios. We hope these findings will guide the design of more effective training strategies and data scheduling approaches for large-scale language models.

6

THE EMERGENCE OF SPARSE ATTENTION: IMPACT OF DATA DISTRIBUTION AND BENEFITS OF REPETITION

The contents of this chapter have been published as a conference paper at the annual conference on neural information processing in 2025 [266] and have been authored by Nicolas Zucchet, Francesco d'Angelo, Andrew Lampinen and Stephanie Chan.

6.1 INTRODUCTION

Scaling has been central to the recent success of large language models [23, 65, 66, 267, 268], with scaling laws [248, 269] describing how increased model size, data, and training consistently improve average performance. However, beneath this macroscopic predictability, model performance on specific tasks often reveals capabilities that appear suddenly beyond critical scaling thresholds – a phenomenon known as emergence [263, 270–272]. While recent studies have begun to characterize how emergence can appear after a critical training time [253, 260, 273–277], a comprehensive scientific understanding remains elusive.

This work explores sparse attention as a lens to understand emergence during training. The formation of sparse attention – where Transformers' attention layers focus on a small subset of critical tokens – coincides with sudden performance improvements in many emergent behaviors, including in-context learning abilities [260, 275] and factual recall [277]. We investigate why the development of sparse attention can lead to abrupt performance improvements and reveal how characteristics of the training data influence the speed of emergence. We make two key contributions:

- We design a variant of linear regression that specifically requires Transformers to learn to focus on a few tokens within the context (Section 6.3). This analytically tractable task allows us to mathematically characterize, in a toy model, the mechanics behind the emergence of sparse attention and precisely quantify how shorter sequences and repetition accelerate emergence.
- We apply our sparse attention framework to explain the emergence of in-context learning in an associative recall task (Section 6.4). Our theoretical predictions successfully account for how data influences the emergence speed of the induction head that solves this task.

Overall, our results suggest that sparse attention may provide a unifying perspective for understanding seemingly diverse emergent phenomena in large language models. They additionally highlight the potential practical benefits of repetition for accelerating the formation of specific neural circuits.

6.2 MOTIVATION

CAN EMERGENCE BE PREDICTED? Emergence in large language models not only challenges our scientific understanding of how they acquire new skills but also poses AI safety issues [278] due to its unpredictable nature, while additionally complicating frontier model development. To address this unpredictability, researchers have proposed several progress metrics under which scaling has more predictable consequences, including validation loss [279], metrics that reward partial progress [247, 280], and those that employ mechanistic interpretability-motivated measures [245]. However, a significant limitation is that these metrics typically can only be derived post-emergence [281]. An alternative approach demonstrates that fine-tuned models' performance on certain tasks can predict whether the ability to solve the task will emerge in larger models [276]. Our work explores predictability from a different angle: understanding how the data distribution influences the learning time at which emergence occurs.

IS THERE A LINK BETWEEN EMERGENCE AND SPARSE ATTENTION? The driving hypothesis underlying this work is that learning of sparse attention patterns is particularly prone to produce sharp transitions in behavior during training – i.e., the sudden emergence of new capabilities. There is a statistical argument behind this intuition: when the target a Transformer has to predict depends only on a few tokens within the context, these tokens initially have very low weight in the prediction of the model, as attention typically starts uniform. Initial progress is therefore slow and the more targeted (thus sparse) attention is, the faster learning becomes. Multiple empirical observations strengthen our hypothesis. The induction head [260], whose formation coincides with the sudden emergence of certain in-context learning abilities, fundamentally relies on the combination of two attention layers with sparse patterns. Similarly, the mechanisms underlying factual recall in large language models [244, 245] demonstrate sparse attention properties and show emergent learning dynamics [277]. These specific emergent phenomena appear to be linked to the development of sparse attention mechanisms, suggesting the existence of a potential causal relationship between the two.

THE INTRIGUING INTERPLAY BETWEEN REPETITION AND EMERGENCE. While data diversity is generally considered a gold standard in machine learning, a growing body of evidence suggests that repetition can actually accelerate emergence over training. Chan *et al.* [273] demonstrated that showing some tasks more often favors the emergence of in-context learning. Charton & Kempe [253] revealed that repeating a subset of examples more frequently than others dramatically accelerates Transformers' ability to solve certain arithmetic tasks. This pattern extends beyond mathematical reasoning, as Zucchet *et al.* [277] (and to some extent Allen-Zhu & Li [240]) showed that repeating biographies of specific individuals speeds up the development of circuits critical for factual recall from model parameters. Collectively, these results establish repetition as a fundamental property of data distributions that can systematically influence emergence timing, thus justifying integrating it in our model to better understand its role.

6.3 THEORETICAL INSIGHTS ON AN ATTENTION-BASED LINEAR REGRESSION TASK

To illustrate how emergence can arise from sparse attention learning, we introduce a variant of linear regression that additionally requires selecting a relevant token from the input sequence. We introduce this task alongside a minimal attention-based toy model for solving it (Section 6.3.1), theoretically analyze its learning dynamics both without (Section 6.3.2) and with repetition (Section 6.3.3), and empirically show that our findings extend to more realistic Transformers (Section 6.3.4).

6.3.1 *The single-location linear regression task*

We consider the following supervised learning task. We are given an input sequence $(x_t)_{t=1}^T$ of length T in which each token $x_t \in \mathbb{R}^d$ is drawn i.i.d. from a zero-mean normal distribution with variance $1/d$ and aim to predict the target y^* given by

$$y^* = W^* x_T \tag{6.1}$$

Here, the target weight matrix $W^* \in \mathbb{R}^{d\times d}$ is a predetermined matrix and $y \in \mathbb{R}^d$. To successfully solve this task, an attention-based model must learn to attend to the last token only and learn the ground-truth target weights W^*. We deliberately incorporate a sparse attention target mechanism to study the relationship between sparse attention and emergence. This task shares similarities with the single-location regression task of Marion *et al.* [282], although in our case, the relevant token location is always the same. Figure 6.1 summarizes this task.

We model two forms of repetition that are ubiquitous in natural language:

- **In-context repetition.** This occurs when specific token groups (e.g., a person's name) repeatedly appear within a single context window. In our framework, we model this property by repeating the relevant token x_T multiple times in the sequence $(x_t)_{t=1}^T$, always at the same positions for our simplified model to be able to use it. Following Chan *et al.* [273], who termed this property "burstiness", we denote B as the number of times the task-relevant token appears in the context.

- **Cross-sample repetition.** This form of repetition comes from having certain information (such as biographical details of specific individuals) overrepresented in the overall training data. We implement this by first sampling the input sequence normally, but then occasionally (with probability p) replacing the relevant token x_T with a special predefined token $\tilde{x}$.

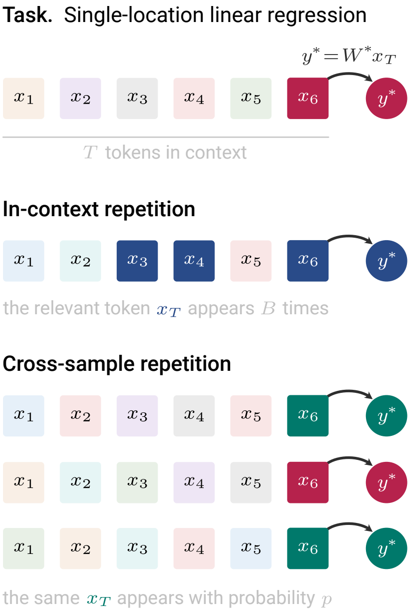

Task. Single-location linear regression
$y^* = W^* x_T$
x_1 x_2 x_3 x_4 x_5 x_6 y^*
T tokens in context
In-context repetition
x_1 x_2 x_3 x_4 x_5 x_6 y^*
the relevant token x_T appears B times
Cross-sample repetition
x_1 x_2 x_3 x_4 x_5 x_6 y^*
x_1 x_2 x_3 x_4 x_5 x_6 y^*
x_1 x_2 x_3 x_4 x_5 x_6 y^*
the same x_T appears with probability p

FIGURE 6.1: A SIMPLE TASK TO STUDY THE EMERGENCE OF SPARSE ATTENTION. We introduce a variant of linear regression task that is analytically tractable and in which Transformers-like models need to learn sparse attention. The model must identify which token (here the last one, x_T) is relevant for the target output y^*. We incorporate two realistic forms of repetition in the data: in-context repetition, where the relevant token appears multiple times within the context, and cross-sample repetition, where an input sequence contains a special token $\tilde{x}$ (here colored in green) at the relevant position with probability p. See Section 6.3.1 for details.

For the purpose of our theoretical analysis, we introduce a simplified attention layer, defined by

$$y = W \sum_{t=1}^{T} \mathrm{softmax}(a)_t \, x_t \tag{6.2}$$

with $a \in \mathbb{R}^T$ the attention scores vector and $W \in \mathbb{R}^{d \times d}$ the weight matrix. Unlike in standard Transformer attention [21], our model does not use any semantic information to determine where to attend. This simplification facilitates theoretical analysis and implicitly assumes that the attention layer has already learned to filter irrelevant contextual information. The model's parameters (a, W) are learned by minimizing the expected mean square error between predictions y and targets y^*.

6.3.2 *The learning dynamics of the simplified Transformer exhibit emerging abilities*

The performance of our simplified model on this task exhibits a characteristic learning pattern: an initial plateau where the loss minimally decreases, followed by a sharp phase transition towards significantly lower loss values. The analysis we detail below reveals that this emergent behavior arises from the interaction between feedforward weights and attention during learning, and that the duration of the initial plateau increases with the sequence length T and data dimensionality d.

REDUCED LEARNING DYNAMICS. We analyze the gradient flow dynamics of the simplified model. The assumptions and mathematical details of this analysis can be found in Appendix D.1. Under these mild assumptions, the learning dynamics of the entire model reduces to two key scalar variables: $\Delta a := a_T - a_t$ for any $t < T$ (all attention scores to non-relevant tokens stay the same under our assumptions) and w the scalar projection[1]

---

[1] w is formally defined as $w := \langle W^*, W \rangle_F / \|W^*\|_F \in \mathbb{R}$ with $\langle \cdot, \cdot \rangle_F$ the Froebenius inner product.

of W on W^*. These variables are initially both equal to 0 and they evolve according to the following system of ordinary differential equations:

$$\dot{w} = \frac{\alpha(\sqrt{d} - \alpha w)}{d} - \frac{(1-\alpha)^2 w}{d(T-1)} \tag{6.3}$$

$$\dot{\Delta a} = \alpha(1-\alpha)\left(\frac{w(\sqrt{d} - \alpha w)}{d} + \frac{(1-\alpha)w^2}{d(T-1)}\right), \tag{6.4}$$

with $\alpha := (1 + (T-1)\exp(-\Delta a))^{-1}$ the attention given to the final token. In these two equations, the first term is the relevant signal and the second term is the noise coming from non-relevant tokens.

These equations enable us to elucidate the roots of emergence in this task. Initial learning is slow, as Δa does not receive any teaching signal as $w = 0$ and w slowly increases as attention is initially uniform ($\alpha = 1/T$). As a consequence, the feedforward weight W must first align with the target weights W^* before attention can learn. A positive feedback loop is then progressively established: increased attention improves the learning signal for w, and a better-learned w further reinforces the correct attention pattern. This dynamic, similar to the one found in deep linear networks [59], leads to a sharp decrease in loss and the sudden emergence we observe in Figure 6.2.a.

PREDICTING WHEN EMERGENCE ARISE. To estimate how long it takes to escape the initial loss plateau, we linearize the dynamics around the initial conditions and obtain

$$\dot{\begin{pmatrix} w \\ \Delta a \end{pmatrix}} = \begin{pmatrix} \frac{1}{\sqrt{d}T} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & \frac{1}{\sqrt{d}T} \\ \frac{1}{\sqrt{d}T} & 0 \end{pmatrix} \begin{pmatrix} w \\ \Delta a \end{pmatrix}. \tag{6.5}$$

This linearization provides two key insights: First, it confirms that feedforward weight learning precedes and drives attention learning, as evidenced by the initial gradient and the top-right entry of the matrix in the equation above, and corroborated by the simulations of Figure 6.2.b. Second, it enables us to estimate the escape time from initial conditions, defined as the time required to reach $(1-\varepsilon)$ of the initial loss value. It is equal to

$$T_\varepsilon = \frac{\sqrt{d}T}{2} \ln\left(\varepsilon\sqrt{d}T\right). \tag{6.6}$$

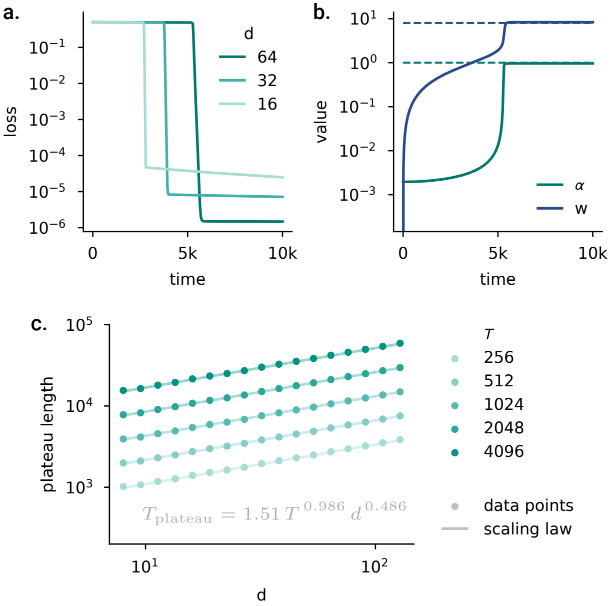


Figure 6.2: Sparse attention in the single-location linear regression task. a. As desired, the learning dynamics of our simplified Transformer (Eq. 6.2) exhibit a multi-phase behavior including an initial plateau, on the task without repetition ($T = 512$). b. Mechanistically, the weights w begin learning before attention to the relevant token α ($T = 512$, $d = 64$). Dashed lines represent optimal values. c. The duration of the initial plateau increases as a function of sequence length T and input/output dimension d, closely following a power law scaling relationship ($R^2 = 0.999$). See Section 6.3.2 for details.

This formula succinctly demonstrates that both longer sequences and higher-dimensional inputs increase the time spent on the plateau and delay emergence. This theoretically derived scaling closely matches the one obtained in simulations, as depicted in Figure 6.2.c ($\varepsilon = 0.8$ in these simulations).

6.3.3 *Repetition speeds up emergence*

Now that we have thoroughly examined the vanilla case, we focus on understanding the effects of repetition. Our analysis reveals that in-context repetition makes the attention pattern to be learned less sparse, thereby simplifying the task, while cross-sample repetition accelerates feedforward weight learning in specific directions, which subsequently increases overall learning speed by increasing the attention of the model to relevant tokens earlier. We present the key findings below.

IN-CONTEXT REPETITION. The role of in-context repetition is rather simple. At initialization, the attention layer allocates B times more weight to the relevant tokens compared to the case without any repetition. This accelerates both initial learning of the weights and the increase of attention to the relevant tokens. Theoretically, we can show, cf. Appendix D.1.3, that the escape time from the initial loss plateau becomes proportional to $T\sqrt{d}/B$. Replacing T by T/B in the scaling law we obtained in Figure 6.2.c yields an almost perfect empirical fit, cf. Figure 6.3.b. This result highlights that B reduces the sparsity of the target attention pattern and thus accelerates emergence.

This analysis ignores, by design, any ability of the attention mechanism to flexibly use semantic or positional information, as we directly parameterized the attention scores. We argue that our findings will extend to the more general case. Indeed, both the attention scores, which would now be the output of some function, and the feedforward weights receive larger gradients with in-context repetition, and thus learning will overall be faster. We will confirm that the same conclusion holds empirically for more realistic architectures and optimizers in the next sections.

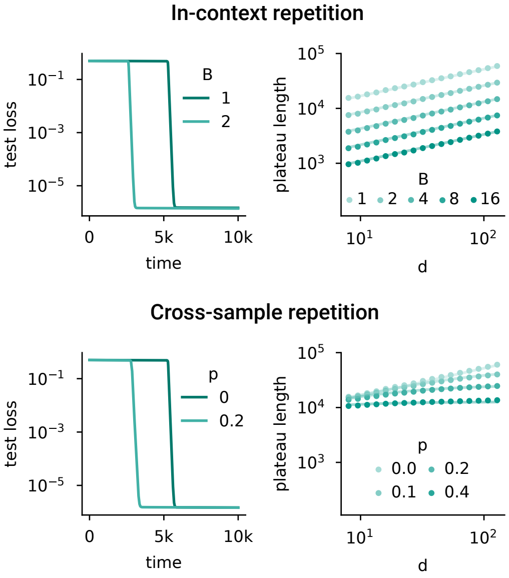


FIGURE 6.3: REPETITION SPEEDS UP EMERGENCE IN THE LINEAR REGRESSION TASK. (top) Increasing in-context repetition through B reduces the initial plateau, and the length of the plateau is well captured by the power law $T_{\text{plateau}} = 1.51\, T^{0.99}\, B^{-0.99}\, d^{0.49}$ ($R^2 = 0.999$). (bottom) Cross-sample repetition, modulated by the repetition probability p, exhibits similar effects, even when evaluating the model on a test loss without repetition (i.e., $p = 0$). The length of the plateau follows $T_{\text{plateau}} = 2.15(\sqrt{d}T/\sqrt{p^2 d + (1-p)^2})^{1.02}$ ($R^2 = 0.992$). See Section 6.3.3 for details.

CROSS-SAMPLE REPETITION. The role of cross-sample repetition is more intricate. Repeating one token more frequently, that is increasing p, causes the input covariance matrix of the relevant token, $\mathbb{E}[x_T x_T^\top]$, to become anisotropic. The different components of the weight matrix W are then learned at different speeds. While this difference in learning speed traditionally leads to slower convergence rates in vanilla linear regression [171], it turns out to be beneficial in our attention-based version of the task. Indeed, following similar principles to the ones detailed in the previous section, learning weights in the repeated direction will lead to the attention to the relevant tokens increasing faster, and this then speeds up the learning of the non-repeated dimensions. As a consequence, the model escapes the initial learning plateau faster. Importantly, this also holds when measuring the model's performance on non-repeated data, as seen in Figure 6.3. The effect of cross-sample repetition can also be understood as increasing the signal-to-noise ratio. Repeating the same token increases the amount of signal coming from the relevant token while keeping the amount of noise received from other tokens fixed, at the price of losing some information about the relevant signal.

Theoretical analysis requires the introduction of an additional variable, so that the learning speed in both the repeated dimension and the other dimensions can be tracked independently. This model, which we introduce in Appendix D.1.4, enables us to justify the mechanistic insights mentioned above, as well as to derive that the plateau length scales as $\sqrt{d}T / \sqrt{p^2 d + (1-p)^2}$, which accurately describes empirical behavior (cf. Figure 6.3). The repetition probability p thus implicitly interpolates between the d-dimensional case and the 1-dimensional case, highlighting that the learning of the feedforward mapping becomes less of a bottleneck.

However, cross-sample repetition is not a free lunch: it only provides a temporary advantage. First, these dynamics only have a smaller test loss in the medium term. In the long run, learning is slower for similar reasons as for the standard linear regression. Second, there is an additional overfitting problem. While it does not appear in this simplified setting as anisotropic input covariance does not bias the final solution, it starts appearing for more realistic architectures. This phenomenon has been observed in practice: examples include [253, 254, 277] for synthetic tasks that have properties similar to the one we focus on here, and [283, 284] for the pretraining of large language models.

6.3.4 *The theory qualitatively captures learning dynamics of full-fledged Transformers*

We conclude our linear regression analysis by examining how our theoretical predictions extend to more realistic task versions, optimizers, and models – particularly those with standard attention that varies with inputs. Our findings indicate that learning dynamics behave qualitatively similarly, showing sharp phase transitions, with emergence time maintaining similar dependencies on data properties. However, the precise dependencies differ, which we investigate in more depth.

For this investigation, we train a standard 2-layer 4-heads Transformer with the Adam optimizer [146], using a constant learning rate of 10^{-4}. Our only deviation from standard architectures is removing layer normalization, which makes solving the task more challenging. To enhance task realism, we randomly sample the positions of relevant tokens and incorporate an additional feature into the input to indicate whether a token is task-relevant. Further experimental details are provided in the appendix.

The learning dynamics of Transformers align qualitatively with our theoretical predictions. First, the loss evolution exhibits a sharp phase transition (see Figure D.6 in the appendix for an example). Second, emergence time depends similarly on the data properties identified in our theory, with repetition accelerating emergence. However, this result comes with several nuances.

For scenarios with no repetition or with in-context repetition, power laws still accurately capture the dependency of emergence time on sequence length T, dimension d and burstiness B, though with significantly different exponents. Ablations (see Appendix D.2.3) reveal that optimizer, architecture, and task specifics all influence these exponents. Notably, replacing Adam with SGD substantially slows emergence, both in absolute terms and by increasing dependency on task difficulty. This finding illustrates a broader observation: Adam is crucial for efficient Transformer learning [151, e.g.,].

Regarding cross-sample repetition, power laws no longer accurately describe the empirical relationship, partly because measuring plateau length becomes more difficult (see the learning dynamics under cross-sample repetition in Figure D.7 in the appendix for an example). Nevertheless, the

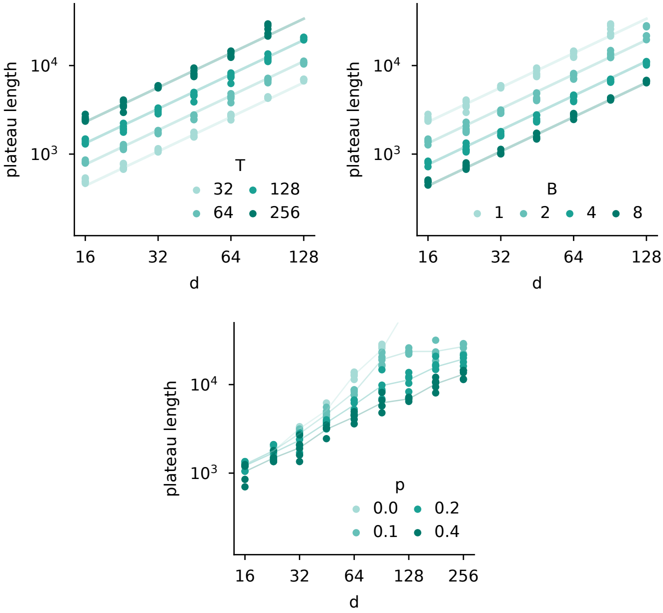


Figure 6.4: Theoretical insights on the linear regression task transfer to more realistic versions of the task, the model, and the optimizer. Transformer-based architectures exhibit similar phase transitions as our toy model and its corresponding plateau length follows similar trends to the ones derived in theory. (left and middle) Evolution of the plateau length as a function of d, when varying T and B (by default $T = 256$ and $B = 1$). The lines corresponds to the power law $T_{\text{plateau}} = 0.76\, d^{\,1.29}\, T^{\,0.80}\, B^{\,-0.80}$ ($R^2 = 0.995$). (right) Same plot, this time varying the cross-sample repetition probability p. The lines correspond to the evolution of the average plateau length as d increases, for the different p values. See Section 6.3.4 for details.

trends identified in our theory still hold: this form of repetition accelerates emergence, with the effect becoming more pronounced as d increases.

6.4 EMERGENCE OF IN-CONTEXT LEARNING, THROUGH THE LENS OF SPARSE ATTENTION

We conclude by investigating to what extent the insights developed on our simple linear regression task extend to more realistic learning problems, in particular one in which in-context learning emerges. To this end, we examine an in-context associative recall task, which can be solved by learning an induction head – a well-studied circuit strongly implicated in in-context learning [260, 275], which necessitates at least two attention layers with sparse attention patterns. Our theory qualitatively captures both the learning dynamics and the impact of the training distribution on learning speed.

6.4.1 *The in-context associative recall task*

The in-context associative recall task serves as a standard benchmark for testing language models' ability to access information in their context and to perform a simple form of one-shot learning. This task requires abilities that correlate with models' capacity for in-context learning [260], and variations of it have been extensively studied to better understand how in-context learning emerges [262, 273, 275, 285–288]. It has also been used as a testbed for recently proposed linear recurrent architectures [e.g., 70, 71, 129], as it acts as an important differentiator between attention-based and recurrent architectures [289].

We implement this task as follows: each sequence consists of N_{pairs} key-value token pairs (5 such pairs in the example below) followed by a query token (`Z` below), as shown:

```
Y I   A X   U R   Z Y   C A   Z ?
```

The query corresponds to one of the keys in the context, and the model must output the corresponding value – `Y` in the example above (since the query `Z` matches the key in the pair `Z Y`). In practice, we work with a total

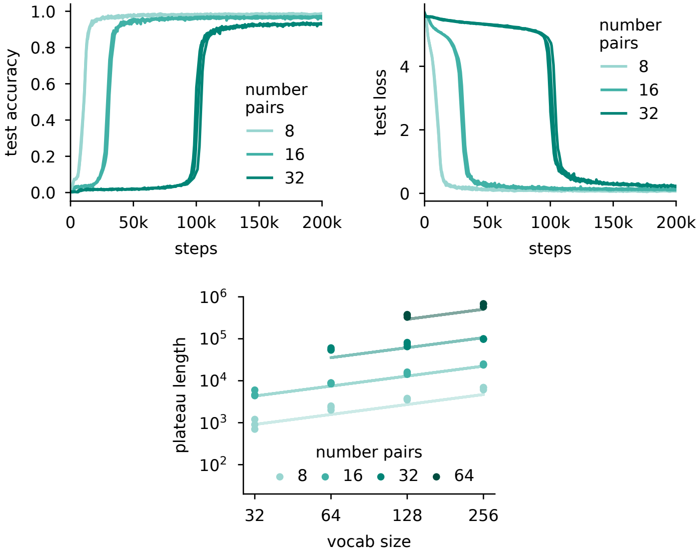


Figure 6.5: In-context learning emerges in the associative recall task, and emergence time grows with the number of pairs in the context and the vocabulary size. (left and middle) The ability to solve the task emerges through training, with emergence time increasing as the task gets harder (by increasing the number of pairs here, the vocabulary size being fixed to 256). This is qualitatively consistent with the theory developed for sparse attention. (right) Systematically investigating the relationship between emergence time and data properties reveals that it follows the power law $T_{\text{plateau}} = 0.55\, N_{\text{tokens}}^{0.79}\, N_{\text{pairs}}^{2.25}$ ($R^2 = 0.982$). Results are only shown when the number of pairs is larger than the vocabulary size, to ensure that the query does not appear multiple times in the context (we do not consider repetition here). More details, in particular a hypothesis for why the N_{pairs} exponent is so high, can be found in Section 6.4.2.

of N_{tokens} unique tokens provided to the network via one-hot encodings. We train the model using a cross-entropy loss comparing the model output to the target value.

The number of pairs N_{pairs} controls the sequence length T as $T = 2N_{\text{pairs}} + 1$ (accounting for the query token), and the vocabulary size N_{tokens} plays a role comparable to the input dimension d in the linear regression task. To model in-context repetition, we ensure that the query token appears on average B times in the context (as a key). To model cross-sample repetition, we select a subset of 2 tokens that will appear more often[2] and vary p, the probability to sample the query from this subset. The precise description of the task is provided in the appendix. The results we report in the main text are testing the learned models on data without any repetition, thereby testing the generalization abilities of models learned on repeated data.

Transformers typically solve this type of task by implementing an induction head – a circuit that combines an attention layer responsible for concatenating the representation of the current token with the previous token, and a selection attention layer responsible for retrieving the relevant information from the context. Both attention layers focus on a sparse subset of tokens (usually just one), making this task particularly well-suited to test our theory. Unlike the linear regression task where sparse attention was hard-coded in the target input-output mapping, there is no inherent constraint forcing models to develop sparse attention here. This task thus serves as the perfect testbed to evaluate our theory on more realistic, yet still finely controlled, learning scenarios.

6.4.2 *The emergence of in-context learning depends on data as in the theory*

As with the linear regression task, we investigate the learning dynamics of Transformers on this task and compare the qualitative findings with those of our developed theory. The experimental setup being closely related to the one in Section 6.3.4, we defer its thorough description to the appendix and note that we use layer normalization and MLPs for this set of experiments.

---

[2] 2 is an arbitrary choice that we have not ablated.

Overall, we find that our theory accurately describes both the learning dynamics (in-context learning emerges in this task) and the dependency of emergence timing on data distribution properties.

LONGER SEQUENCES AND LARGER VOCABULARY SIZE DELAY EMERGENCE. We begin by verifying that the intuition described above applies and that the learning curve exhibits the sharp phase transitions characteristic of emergence. This is indeed the case for sufficiently long sequences and/or large vocabularies, as illustrated in Figure 6.5. Importantly, there is only one phase transition despite two attention layers being needed to learn sparse patterns; Figure D.12 shows that they are learned simultaneously. This observation is consistent with the results of Reddy [262] and Singh *et al.* [275].

In Figure 6.5 (right), we report how the emergence time, arbitrarily defined as the time needed to reach 5% accuracy, evolves as a function of sequence length and data dimension. It accurately follows a power law, as in all our other experiments, albeit with exponents that are relatively low for N_{tokens} (0.79) and extremely high for N_{pairs} (2.25). We hypothesize that the lower exponent for N_{tokens} stems from the fact that most of the feedforward processing can be handled by residual connections, and given that the dimension of the data primarily influences the speed of feedforward learning, data dimension does not have such a large effect. For the higher exponent for N_{pairs}, our hypothesis is that this occurs as two sparse attention patterns must be learned jointly. In the toy model we analyze theoretically, a similar coupling exists between the attention and the feedforward weights, resulting in a multiplicative interaction. We posit that a similar mechanism may be at work here, with each attention layer contributing additively to the N_{pairs} exponent. We leave a more thorough investigation of how different circuits interact to influence emergence time to future work.

REPETITION SPEEDS UP EMERGENCE BUT COMES WITH OVERFITTING RISKS. We next investigate the benefits of in-context and cross-sample repetition. From the high sensitivity of the emergence time on sequence length revealed by previous analysis, we expect in-context repetition to have a stronger effect than the cross-sample one. This is what we observe. The results reported in Figure 6.6, which evaluate the model performance on test data that has no repetition, demonstrate that repetition can signifi-

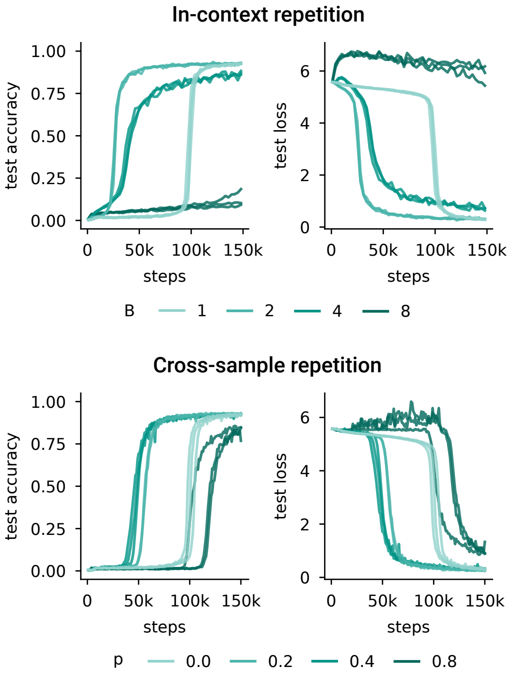


Figure 6.6: Repetition speeds up emergence in the in-context associative recall task but comes with overfitting risks. (left) We vary the amount of in-context repetition B, that is the number of times the query appears as a key in the context on average, and find significant benefits of small amount of repetition. Larger amounts of repetition lead to overfitting, but learning for long enough eventually leads to grokking. (right) Cross-sample repetition, more precisely the probability p that the query is one of the 2 repeated tokens, has similar effects. Results are obtained for $N_{\text{pairs}} = 32$ and $N_{\text{tokens}} = 256$.

cantly speed up emergence, by a factor of 4 for in-context repetition and a factor of 2 for cross-sample repetition. The attention sparsity perspective thus explains why in-context repetition has been found to favor in-context learning vs. in-weight learning [262, 273, 290]: the induction head needed to solve this kind of task is formed earlier with such a form of repetition.

However, this benefit comes with an overfitting risk: while repetition consistently accelerates learning (repetition always speeds up learning on the train loss, cf. Figure D.11 in the appendix), too much repetition leads to learning strategies that do not generalize well. For example, selecting the most frequent value is a valid strategy for this task whenever the query appears two or more times in the context. Interestingly, we also observe some grokking-like patterns [73] as the test accuracy eventually starts to increase late in training for large amounts of repetition. We argue that this occurs because the no-repetition case is still represented in the training data, albeit with very little weight. This trade-off between learning speed and generalization is consistent with what Park *et al.* [254] reported on another in-context learning task. The overfitting we observe here may appear inconsistent with our theory but is not: our theory addresses learning speed (performance on the training loss) rather than generalization ability.

6.5 DISCUSSION

CONNECTION WITH OTHER THEORETICAL WORK. Through the lens of sparse attention, we provide a unifying perspective on a set of empirical results highlighted in the Motivation (Section 6.2). While our focus on sparse attention and the effect of repetition is unique (to the best of our knowledge), there are multiple closely related works. First, the interaction between the feedforward weights and attention we study is closely related to the one between two consecutive feedforward linear layers [59, 60] [59, 60, 291]. Linear networks also display incremental learning with abrupt transitions characterizing each phase change [292–295]. This line of research on linear networks has since then been extended to linear Transformers [290, 296], in particular to study the emergence of in-context learning. On the more conceptual side, there exist other theories of how emergence could arise from longer training, such as grokking [73, 274] or singular learning theory [297, 298]. While rarer, other theories focus on emergence from increased model size [299]. Compared to these, our theory focuses on learning dynamics (emergence over the course of training) and is directly tied to the internal mechanisms of the Transformers.

HOW MUCH EMERGENCE COMES FROM SPARSE ATTENTION? One key result of our paper is that the learning of sparse attention is prone to producing emergence over the course of training. Given that sparse or concentrated attention is a common emergent feature of Transformers (e.g., induction heads [260], task/function vectors [300, 301], or high-norm tokens in vision transformers [302]), one may ask how much currently known emergent behaviors can be attributed to the learning of sparse attention patterns, and whether there are many more emergent behaviors than what is currently reported. However, these points must be weighed. As noted above, even simpler MLPs can give rise to abrupt emergence over training [e.g., 59]; thus, learning in other circuits within Transformers might contribute to sudden transitions in their performance. Furthermore, emergence results at large scale mostly show a sharp phase transition of the model as the number of FLOPs [e.g., 263], which roughly corresponds to the model size times the number of training steps, reaches a certain threshold. It is therefore unclear whether these examples of emergence are rooted in longer training, models with higher capacity, or a complex interplay between the two. More empirical and theoretical work is needed to better understand the causes of emergence in large models and the possible complex relationship between training-based and size-based emergence.

DATA DIVERSITY AS A PATH TOWARDS ACTIVE LEARNING? Another key result of our work is that low data diversity can actually improve performance. This contrasts with classic machine learning principles, which state that low diversity hurts generalization. These two apparently conflicting statements are actually compatible. As our results highlight, low diversity accelerates the learning of sparse attention, but high data diversity is better as training time goes to infinity. We hypothesize that data diversity might be a powerful lever towards enabling active learning [303], with the ultimate goal of letting the learner decide what it wants to learn on and reaching human-level sample efficiency that current deep learning systems crucially lack [18, 304, 305]. Humans also encounter varying levels of data diversity over development; e.g., infants receive progressively increasing data diversity during their early years [63]. Such principles could already be at play in the current training pipeline of large language models, for instance when code, which typically has lower syntactic diversity than other types of text [306], is included at specific moments during training. While promising, this thread requires more extensive analysis, both at a larger scale and on more realistic data distributions.

BIBLIOGRAPHY

1. James, W. The principles of psychology. *Henry Holt* (1890).
2. Hebb, D. O. *The organization of behavior: a neuropsychological theory* (Wiley, 1949).
3. Jordan, M. I. & Mitchell, T. M. Machine learning: Trends, perspectives, and prospects. *Science* **349** (2015).
4. Sutton, R. S. & Barto, A. G. Introduction to reinforcement learning. **135** (1998).
5. Bliss, T. V. P. & Lømo, T. Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *The Journal of Physiology* **232** (1973).
6. Kandel, E. R. The molecular biology of memory storage: a dialogue between genes and synapses. *Science* **294** (2001).
7. Spruston, N. Pyramidal neurons: dendritic structure and synaptic integration. *Nature Reviews Neuroscience* **9** (2008).
8. Bargmann, C. I. Beyond the connectome: How neuromodulators shape neural circuits. *Bioessays* **34** (2012).
9. Du Bois-Reymond, E. *Untersuchungen über thierische Elektricität* **2** (G. reimer, 1884).
10. Y Cajal, S. R. *Histologie du système nerveux de l'homme & des vertébrés: Cervelet, cerveau moyen, rétine, couche optique, corps strié, écorce cérébrale générale & régionale, grand sympathique* (A. Maloine, 1911).
11. Hodgkin, A. L. & Huxley, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology* **117** (1952).
12. Abbott, L. F. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin* **50**. Publisher: Citeseer (1999).
13. Tsien, R. Y. New calcium indicators and buffers with high selectivity against magnesium and protons: Design, synthesis, and properties of prototype structures. *Biochemistry* **19** (1980).

14. Jun, J. J., Steinmetz, N. A., Siegle, J. H., Denman, D. J., Bauza, M., Barbarits, B., Lee, A. K., Anastassiou, C. A., Andrei, A., Aydın, Ç., Barbic, M., Blanche, T. J., Bonin, V., Couto, J., Dutta, B., Gratiy, S. L., Gutnisky, D. A., Häusser, M., Karsh, B., Ledochowitsch, P., Lopez, C. M., Mitelut, C., Musa, S., Okun, M., Pachitariu, M., Putzeys, J., Rich, P. D., Rossant, C., Sun, W.-l., Svoboda, K., Carandini, M., Harris, K. D., Koch, C., O'Keefe, J. & Harris, T. D. Fully integrated silicon probes for high-density recording of neural activity. *Nature* **551** (2017).

15. Steinmetz, N. A., Zatka-Haas, P., Carandini, M. & Harris, K. D. Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings. *Science* **372** (2021).

16. Richards, B. A., Lillicrap, T. P., Beaudoin, P., Bengio, Y., Bogacz, R., Christensen, A., Clopath, C., Costa, R. P., de Berker, A., Ganguli, S., Gillon, C. J., Hafner, D., Kepecs, A., Kriegeskorte, N., Latham, P., Lindsay, G. W., Miller, K. D., Naud, R., Pack, C. C., Poirazi, P., Roelfsema, P., Sacramento, J., Saxe, A., Scellier, B., Schapiro, A. C., Senn, W., Wayne, G., Yamins, D., Zenke, F., Zylberberg, J., Therien, D. & Kording, K. P. A deep learning framework for neuroscience. *Nature Neuroscience* **22** (2019).

17. Zador, A., Escola, S., Richards, B., Ölveczky, B., Bengio, Y., Boahen, K., Botvinick, M., Chklovskii, D., Churchland, A., Clopath, C., DiCarlo, J., Ganguli, S., Hawkins, J., Körding, K., Koulakov, A., LeCun, Y., Lillicrap, T., Marblestone, A., Olshausen, B., Pouget, A., Savin, C., Sejnowski, T., Simoncelli, E., Solla, S., Sussillo, D., Tolias, A. S. & Tsao, D. Catalyzing next-generation artificial intelligence through NeuroAI. *Nature Communications* **14** (2023).

18. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. Human-level control through deep reinforcement learning. *Nature* **518** (2015).

19. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529** (2016).

20. Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., *et al.*

Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **575** (2019).

21. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. *Attention is all you need* in *Advances in Neural Information Processing Systems* (2017).
22. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. *BERT: Pre-training of deep bidirectional transformers for language understanding* in *Conference of the Association for Computational Linguistics* (2019).
23. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., *et al.* Language models are few-shot learners. *Advances in Neural Information Processing Systems* (2020).
24. Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86** (1998).
25. Krizhevsky, A., Sutskever, I. & Hinton, G. E. *Imagenet classification with deep convolutional neural networks* in *Advances in Neural Information Processing Systems* (2012).
26. He, K., Zhang, X., Ren, S. & Sun, J. *Deep residual learning for image recognition* in *Conference on Computer Vision and Pattern Recognition* (2016).
27. Yamazaki, K., Vo-Ho, V.-K., Bulsara, D. & Le, N. Spiking neural networks and their applications: A review. *Brain Sciences* **12**, 863 (2022).
28. Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M. & Carter, S. Zoom in: An introduction to circuits. *Distill* **5** (2020).
29. Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., *et al.* A mathematical framework for transformer circuits. *Transformer Circuits Thread* (2021).
30. Sharkey, L., Chughtai, B., Batson, J., Lindsey, J., Wu, J., Bushnaq, L., Goldowsky-Dill, N., Heimersheim, S., Ortega, A., Bloom, J., *et al.* Open problems in mechanistic interpretability. *arXiv preprint arXiv:2501.16496* (2024).

31. Lindsey, J., Gurnee, W., Ameisen, E., Chen, B., Pearce, A., Turner, N. L., Citro, C., Abrahams, D., Carter, S., Hosmer, B., Marcus, J., Sklar, M., Templeton, A., Bricken, T., McDougall, C., Cunningham, H., Henighan, T., Jermyn, A., Jones, A., Persic, A., Qi, Z., Thompson, T. B., Zimmerman, S., Rivoire, K., Conerly, T., Olah, C. & Batson, J. On the biology of a large language model. *Transformer Circuits Thread* (2025).
32. Mead, C. Neuromorphic electronic systems. *Proceedings of the IEEE* **78** (1990).
33. Schuman, C. D., Kulkarni, S. R., Parsa, M., Mitchell, J. P., Date, P. & Kay, B. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science* **2** (2022).
34. Douglas, R. J. & Martin, K. A. Neuronal circuits of the neocortex. *Annual Reviews in Neuroscience* **27** (2004).
35. Goldman-Rakic, P. S. Cellular basis of working memory. *Neuron* **14** (1995).
36. Rumelhart, D. E. & McClelland, J. L. in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations* (1986).
37. 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).
38. Lillicrap, T. P., Cownden, D., Tweed, D. B. & Akerman, C. J. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications* **7** (2016).
39. Scellier, B. & Bengio, Y. Equilibrium propagation: bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience* **11** (2017).
40. Guerguiev, J., Lillicrap, T. P. & Richards, B. A. Towards deep learning with segregated dendrites. *eLife* **6** (2017).
41. Sacramento, J., Costa, R. P., Bengio, Y. & Senn, W. *Dendritic cortical microcircuits approximate the backpropagation algorithm* in *Advances in Neural Information Processing Systems* (2018).
42. Whittington, J. C. R. & Bogacz, R. Theories of error back-propagation in the brain. *Trends in Cognitive Sciences* **23** (2019).
43. Meulemans, A., Carzaniga, F., Suykens, J., Sacramento, J. & Grewe, B. F. *A theoretical framework for target propagation* in *Advances in Neural Information Processing Systems* (2020).

44. Payeur, A., Guerguiev, J., Zenke, F., Richards, B. A. & Naud, R. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *Nature Neuroscience* **24** (2021).

45. 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* (2022).

46. Francioni, V., Tang, V. D., Toloza, E. H., Brown, N. J. & Harnett, M. T. Vectorized instructive signals in cortical dendrites during a brain-computer interface task. *bioRxiv* (2025).

47. Wilson, H. R. & Cowan, J. D. Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical Journal* **12** (1972).

48. Werbos, P. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* **78** (1990).

49. Hochreiter, S. *Untersuchungen zu dynamischen neuronalen Netzen* Diploma thesis (Technische Universität München, 1991).

50. Bengio, Y., Simard, P. & Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* **5** (1994).

51. Hochreiter, S., Bengio, Y., Frasconi, P. & Schmidhuber, J. in *A field guide to dynamical recurrent networks* (IEEE, 2001).

52. Pascanu, R., Mikolov, T. & Bengio, Y. *On the difficulty of training recurrent neural networks* in *International Conference on Machine Learning* (2013).

53. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Computation* **9** (1997).

54. Gu, A., Goel, K. & Ré, C. *Efficiently modeling long sequences with structured state spaces* in *International Conference on Learning Representations* (2022).

55. Williams, R. J. & Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* **1** (1989).

56. Williams, R. J. & Zipser, D. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science* **1** (1989).

57. Mountcastle, V. B. The columnar organization of the neocortex. *Brain* **120** (1997).

58. *Parallel distributed processing: Explorations in the microstructure of cognition* (eds Rumelhart, D. E. & McClelland, J. L.) (MIT Press, 1986).

59. Saxe, A. M., McClelland, J. L. & Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear networks. *International Conference on Learning Representations* (2014).
60. Saxe, A. M., McClelland, J. L. & Ganguli, S. A mathematical theory of semantic development in deep neural networks. *Proceedings of the National Academy of Sciences* **116** (2019).
61. Pavlov, P. I. Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex. *Oxford University Press* (1927).
62. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences* **79** (1982).
63. Smith, L. B., Jayaraman, S., Clerkin, E. & Yu, C. The developing infant creates a curriculum for statistical learning. *Trends in Cognitive Sciences* **22** (2018).
64. Jaeger, H. A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach (2002).
65. OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., *et al.* GPT-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).
66. Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., *et al.* Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).
67. Gupta, A., Gu, A. & Berant, J. *Diagonal state spaces are as effective as structured states spaces* in *Advances in Neural Information Processing Systems* (2022).
68. Smith, J. T., Warrington, A. & Linderman, S. W. *Simplified state space layers for sequence modeling* in *International Conference on Learning Representations* (2023).
69. Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gülçehre, Ç., Pascanu, R. & De, S. *Resurrecting recurrent neural networks for long sequences* in *International Conference on Machine Learning* (2023).
70. Gu, A. & Dao, T. *Mamba: linear-time sequence modeling with selective state spaces* in (Conference on Language Modeling, 2024).

71. De, S., Smith, S. L., Fernando, A., Botev, A., Cristian-Muraru, G., Gu, A., Haroun, R., Berrada, L., Chen, Y., Srinivasan, S., *et al.* Griffin: mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427* (2024).

72. Bender, E. M., Gebru, T., McMillan-Major, A. & Shmitchell, S. *On the dangers of stochastic parrots: can language models be too big?* in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency* (2021).

73. Power, A., Burda, Y., Edwards, H., Babuschkin, I. & Misra, V. Grokking: generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177* (2022).

74. Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B. & Sutskever, I. Deep Double Descent: Where Bigger Models and More Data Hurt. *arXiv:1912.02292 [cs, stat]*. arXiv: 1912.02292 (2019).

75. McCulloch, W. S. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* **5** (1943).

76. Minsky, M. & Papert, S. A. *Perceptrons: an introduction to computational geometry* (1969).

77. Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* **65** (1958).

78. Nair, V. & Hinton, G. E. *Rectified linear units improve restricted boltzmann machines* in *International Conference on Machine Learning* (2010).

79. Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural networks* **2** (1989).

80. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* **2** (1989).

81. Montufar, G. F., Pascanu, R., Cho, K. & Bengio, Y. On the number of linear regions of deep neural networks. *Advances in Neural Information Processing Systems* (2014).

82. Telgarsky, M. *Benefits of depth in neural networks* in *Conference on Learning Theory* (2016).

83. Glorot, X. & Bengio, Y. *Understanding the difficulty of training deep feedforward neural networks* in *Artificial Intelligence and Statistics* (2010).

84. He, K., Zhang, X., Ren, S. & Sun, J. *Delving deep into rectifiers: surpassing human-level performance on ImageNet classification* in *International Conference on Computer Vision* (2015).

85. Srivastava, R. K., Greff, K. & Schmidhuber, J. *Highway networks* in *ICML Deep learning workshop* (2015).
86. Ioffe, S. & Szegedy, C. *Batch normalization: accelerating deep network training by reducing internal covariate shift* in *International Conference on Machine Learning* (2015).
87. Ba, J. L., Kiros, J. R. & Hinton, G. E. *Layer normalization* in *Neural Information Processing Systems - Deep Learning Symposium* (2016).
88. Yang, G. & Hu, E. J. Feature learning in infinite-width neural networks. *arXiv preprint arXiv:2011.14522* (2020).
89. Yang, G., Hu, E. J., Babuschkin, I., Sidor, S., Liu, X., Farhi, D., Ryder, N., Pachocki, J., Chen, W. & Gao, J. Tensor programs V: tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466* (2022).
90. Yang, G., Simon, J. B. & Bernstein, J. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813* (2023).
91. Noci, L., Meterez, A., Hofmann, T. & Orvieto, A. *Why do learning rates transfer? Reconciling optimization and scaling limits for deep learning* in *Advances in Neural Information Processing Systems* (2024).
92. Dey, N., Zhang, B. C., Noci, L., Li, M., Bordelon, B., Bergsma, S., Pehlevan, C., Hanin, B. & Hestness, J. Don't be lazy: CompleteP enables compute-efficient deep transformers. *arXiv preprint arXiv:2505.01618* (2025).
93. Fukushima, K. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics* **5** (1969).
94. LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E. & Jackel, L. D. *Handwritten digit recognition with a back-propagation network* in *Advances in Neural Information Processing Systems* (1990).
95. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. *Imagenet: a large-scale hierarchical image database* in *Conference on Computer Vision and Pattern Recognition* (2009).
96. Sevilla, J., Heim, L., Ho, A., Besiroglu, T., Hobbhahn, M. & Villalobos, P. *Compute trends across three eras of machine learning* in *International Joint Conference on Neural Networks* (2022).

97. Krizhevsky, A. & Hinton, G. *Learning multiple layers of features from tiny images* tech. rep. (2009).

98. Jordan, M. I. in *Neural-network models of cognition* (1986).

99. Elman, J. L. Finding structure in time. *Cognitive science* **14** (1990).

100. Lorente de Nó, R. Vestibulo-ocular reflex arc. *Archives of Neurology and Psychiatry* **30** (1933).

101. Douglas, R. J., Koch, C., Mahowald, M., Martin, K. A. C. & Suarez, H. H. Recurrent excitation in neocortical circuits. *Science* **269** (1995).

102. Wang, X. J. Synaptic reverberation underlying mnemonic persistent activity. *Trends in Neurosciences* **24** (2001).

103. Desimone, R. & Duncan, J. Neural mechanisms of selective visual attention. *Annual Review of Neuroscience* **18** (1995).

104. Mante, V., Sussillo, D., Shenoy, K. V. & Newsome, W. T. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature* **503** (2013).

105. Horner, A. J., Bisby, J. A., Bush, D., Lin, W. J. & Burgess, N. Evidence for holistic episodic recollection via hippocampal pattern completion. *Nature Communications* **6** (2015).

106. Axler, S. *Linear algebra done right* (Springer Nature, 1997).

107. Cooley, J. W. & Tukey, J. W. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation* **19** (1965).

108. Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A. & Ré, C. *Combining recurrent, convolutional, and continuous-time models with linear state-space layers* in *Advances in Neural Information Processing Systems* (2021).

109. Blelloch, G. E. *Prefix sums and their applications* 1990.

110. Martin, E. & Cundy, C. *Parallelizing linear recurrent neural nets over sequence length* in *International Conference on Learning Representations* (2018).

111. Boyd, S. & Chua, L. Fading memory and the problem of approximating nonlinear operators with Volterra series. *IEEE Transactions on Circuits and Systems* **32** (1985).

112. Wang, S. & Xue, B. State-space models with layer-wise nonlinearity are universal approximators with exponential decaying memory. *Advances in Neural Information Processing Systems* **36** (2023).

113. Orvieto, A., De, S., Gülçehre, Ç., Pascanu, R. & Smith, S. L. *Universality of linear recurrences followed by non-linear projections: finite-width guarantees and benefits of complex eigenvalues* in *International Conference on Machine Learning* (2024).

114. Gerstner, W., Kistler, W. M., Naud, R. & Paninski, L. *Neuronal dynamics: from single neurons to networks and models of cognition* (Cambridge University Press, 2014).

115. Kalman, R. E. Contributions to the theory of optimal control. *Boletín de la Sociedad Matemática Mexicana* **5** (1960).

116. Gilbert, E. G. Controllability and observability in multivariable control systems. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control* **1** (1963).

117. Åström, K. J. & Murray, R. *Feedback systems: an introduction for scientists and engineers* (Princeton University Press, 2008).

118. Siegelmann, H. T. & Sontag, E. D. Turing computability with neural nets. *Applied Mathematics Letters* **4** (1991).

119. Funahashi, K.-i. & Nakamura, Y. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks* **6** (1993).

120. Schäfer, A. M. & Zimmermann, H.-G. *Recurrent neural networks are universal approximators* in *International Conference on Artificial Neural Networks* (2006).

121. Van Vreeswijk, C. & Sompolinsky, H. Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science* **274** (1996).

122. Amit, D. J. & Brunel, N. Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex. *Cerebral Cortex* **7** (1997).

123. Seung, H. S. How the brain keeps the eyes still. *Proceedings of the National Academy of Sciences* **93** (1996).

124. Zhang, K. Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory. *Journal of Neuroscience* **16** (1996).

125. Ben-Yishai, R., Bar-Or, R. L. & Sompolinsky, H. Theory of orientation tuning in visual cortex. *Proceedings of the National Academy of Sciences* **92** (1995).

126. Compte, A., Brunel, N., Goldman-Rakic, P. S. & Wang, X.-J. Synaptic mechanisms and network dynamics underlying spatial working memory in a cortical network model. *Cerebral Cortex* **10** (2000).
127. Sussillo, D. & Abbott, L. F. Generating coherent patterns of activity from chaotic neural networks. *Neuron* **63** (2009).
128. Beck, M., Pöppel, K., Spanring, M., Auer, A., Prudnikova, O., Kopp, M., Klambauer, G., Brandstetter, J. & Hochreiter, S. *xLSTM: Extended Long Short-Term Memory* arXiv:2405.04517 [cs, stat]. 2024.
129. Yang, S., Wang, B., Zhang, Y., Shen, Y. & Kim, Y. *Parallelizing linear transformers with the delta rule over sequence length* in *Advances in Neural Information Processing Systems* (2024).
130. Von Oswald, J., Scherrer, N., Kobayashi, S., Versari, L., Yang, S., Schlegel, M., Maile, K., Schimpf, Y., Sieberling, O., Meulemans, A., *et al.* MesaNet: sequence modeling by locally optimal test-time training. *arXiv preprint arXiv:2506.05233* (2025).
131. Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R. & Schmidhuber, J. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems* **28** (2016).
132. Cho, K., van Merrienboer, B., Bahdanau, D. & Bengio, Y. *On the properties of neural machine translation: encoder-decoder approaches* in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation* (2014).
133. Lim, Y. H., Zhu, Q., Selfridge, J. & Kasim, M. F. *Parallelizing non-linear sequential models over the sequence length* in *International Conference on Learning Representations* (2024).
134. Gonzalez, X., Warrington, A., Smith, J. T. & Linderman, S. W. *Towards scalable and stable parallelization of nonlinear RNNs* in *Advances in Neural Information Processing Systems* (2024).
135. Bahdanau, D., Cho, K. & Bengio, Y. *Neural machine translation by jointly learning to align and translate* in *International Conference on Learning Representations* (2015).
136. Wang, Y.-A. & Chen, Y.-N. *What do position embeddings learn? an empirical study of pre-trained language model positional encoding* in *Conference on Empirical Methods in Natural Language Processing* (2020).
137. Su, J., Lu, Y., Pan, S., Wen, B. & Liu, Y. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing* **568** (2023).

138. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., *et al.* Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
139. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., *et al. An image is worth 16x16 words: Transformers for image recognition at scale* in *International Conference on Learning Representations* (2021).
140. Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., *et al. Segment anything* in *International Conference on Computer Vision* (2023).
141. DeepSeek-AI *et al.* DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437* (2024).
142. Schultz, W., Dayan, P. & Montague, P. R. A neural substrate of prediction and reward. *Science* **275** (1997).
143. Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S. & Bengio, Y. *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization* in *Advances in Neural Information Processing Systems* (2014).
144. Cauchy, A. Méthode générale pour la résolution des systemes d'équations simultanées. *Comptes Rendus Hebdomadaire Séances Académie des Sciences Paris* **25** (1847).
145. Robbins, H. & Monro, S. A stochastic approximation method. *The Annals of Mathematical Statistics* (1951).
146. Kingma, D. P. & Ba, J. *Adam: a method for stochastic optimization* in *International Conference on Learning Representations* (2015).
147. Polyak, B. T. Some methods of speeding up the convergence of iteration methods. *USSR computational mathematics and mathematical physics* **4** (1964).
148. Nesterov, Y. A method for unconstrained convex minimization problem with the rate of convergence O(1/k2). *Doklady Akademii Nauk SSSR* **269** (1983).
149. Duchi, J., Hazan, E. & Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* **12** (2011).

150. Tieleman, T. & Hinton, G. Lecture 6.5-RMSProp: divide the gradient by a running average of its recent magnitude (2012).

151. Zhang, Y., Chen, C., Ding, T., Li, Z., Sun, R. & Luo, Z. Why transformers need Adam: a Hessian perspective. *Advances in Neural Information Processing Systems* (2024).

152. Du, S., Lee, J., Li, H., Wang, L. & Zhai, X. *Gradient descent finds global minima of deep neural networks* in *International Conference on Machine Learning* (eds Chaudhuri, K. & Salakhutdinov, R.) (2019).

153. Orvieto, A. & Gower, R. In search of Adam's secret sauce. *arXiv preprint arXiv:2505.21829* (2025).

154. Werbos, P. J. *Beyond regression: new tools for prediction and analysis in the behavioral sciences* PhD Thesis (Harvard University, 1974).

155. Tallec, C. & Ollivier, Y. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209* (2017).

156. Scardapane, S. Alice's adventures in a differentiable wonderland – volume I, a tour of the land. *arXiv preprint arXiv:2404.17625* (2024).

157. Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R. & Maass, W. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature Communications* **11** (2020).

158. Menick, J., Elsen, E., Evci, U., Osindero, S., Simonyan, K. & Graves, A. *A practical sparse approximation for real time recurrent learning* in *International Conference on Learning Representations* (2021).

159. Tallec, C. & Ollivier, Y. *Unbiased online recurrent optimization* in *International Conference on Learning Representations* (2018).

160. Mujika, A., Meier, F. & Steger, A. *Approximating real-time recurrent learning with random Kronecker factors* in *Advances in Neural Information Processing Systems* **31** (2018).

161. Benzing, F., Gauy, M. M., Mujika, A., Martinsson, A. & Steger, A. *Optimal Kronecker-sum approximation of real time recurrent learning* in *International Conference on Machine Learning* (2019).

162. Mozer, M. C. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems* **3** (1989).

163. Zenke, F. & Vogels, T. P. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural Computation* **33** (2021).

164. Zucchet, N., Meier, R., Schug, S., Mujika, A. & Sacramento, J. *Online learning of long-range dependencies* in *Advances in Neural Information Processing Systems* (2023).

165. Irie, K., Gopalakrishnan, A. & Schmidhuber, J. *Exploring the promise and limits of real-time recurrent learning* in *International Conference on Learning Representations* (2024).

166. Silver, D., Goyal, A., Danihelka, I., Hessel, M. & van Hasselt, H. *Learning by directional gradient descent* in *International Conference on Learning Representations* (2022).

167. Baydin, A. G., Pearlmutter, B. A., Syme, D., Wood, F. & Torr, P. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587* (2022).

168. Strogatz, S. H. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering (studies in nonlinearity)* (Westview press, 2001).

169. Metz, L., Freeman, C. D., Schoenholz, S. S. & Kachman, T. Gradients are not all you need. *arXiv preprint arXiv:2111.05803* (2021).

170. Mikhaeil, J. M., Monfared, Z. & Durstewitz, D. *On the difficulty of learning chaotic dynamics with RNNs* in *Advances in Neural Information Processing Systems* (2022).

171. Bach, F. *Learning theory from first principles* (2024).

172. Sagun, L., Evci, U., Guney, V. U., Dauphin, Y. & Bottou, L. Empirical analysis of the hessian of over-parametrized neural networks. *International Conference on Learning Representations Workshop Track* (2017).

173. Zucchet, N. & Orvieto, A. Recurrent neural networks: vanishing and exploding gradients are not the end of the story. *Advances in Neural Information Processing Systems* (2024).

174. Rumelhart, D. E., Smolensky, P., McClelland, J. L. & Hinton, G. Sequential thought processes in PDP models. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition* **2** (1986).

175. Sutskever, I., Vinyals, O. & Le, Q. V. *Sequence to sequence learning with neural networks* in *Advances in Neural Information Processing Systems* (2014).

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

177. Katharopoulos, A., Vyas, A., Pappas, N. & Fleuret, F. *Transformers are RNNs: fast autoregressive Transformers with linear attention* in *International Conference on Machine Learning* (2020).
178. Dao, T., Fu, D. Y., Ermon, S., Rudra, A. & Ré, C. *FlashAttention: fast and memory-efficient exact attention with IO-awareness* in *Advances in Neural Information Processing Systems* (2022).
179. Fedus, W., Zoph, B. & Shazeer, N. Switch Transformers: scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research* (2022).
180. Ma, S., Wang, H., Ma, L., Wang, L., Wang, W., Huang, S., Dong, L., Wang, R., Xue, J. & Wei, F. The era of 1-bit LLMs: all large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764* (2024).
181. Mikolov, T. *Statistical language models based on neural networks* PhD thesis (Brno University of Technology, 2012).
182. Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç. & Courville, A. *Recurrent batch normalization* in *International Conference on Learning Representations* (2017).
183. Le, Q. V., Jaitly, N. & Hinton, G. E. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941* (2015).
184. Tallec, C. & Ollivier, Y. *Can recurrent neural networks warp time?* in *International Conference on Learning Representations* (2018).
185. Hihi, S. E. & Bengio, Y. *Hierarchical recurrent neural networks for long-term dependencies* in *Advances in Neural Information Processing Systems* (1995).
186. Mujika, A., Meier, F. & Steger, A. *Fast-slow recurrent neural networks* in *Advances in Neural Information Processing Systems* (2017).
187. Arjovsky, M., Shah, A. & Bengio, Y. *Unitary evolution recurrent neural networks* in *International Conference on Machine Learning* (2016).
188. Vorontsov, E., Trabelsi, C., Kadoury, S. & Pal, C. *On orthogonality and learning recurrent networks with long term dependencies* in *International Conference on Machine Learning* (2017).
189. Helfrich, K., Willmott, D. & Ye, Q. *Orthogonal recurrent neural networks with scaled Cayley transform* in *International Conference on Machine Learning* (2018).

190. Rusch, T. K. & Mishra, S. *Coupled oscillatory recurrent neural network (coRNN): an accurate and (gradient) stable architecture for learning long time dependencies* in *International Conference on Learning Representations* (2021).
191. Keller, T. A. & Welling, M. *Neural wave machines: learning spatiotemporally structured representations with locally coupled oscillatory recurrent neural networks* in *International Conference on Machine Learning* (2023).
192. Park, I. M., Ságodi, Á. & Sokół, P. A. Persistent learning signals and working memory without continuous attractors. *arXiv preprint arXiv:2308.12585* (2023).
193. Li, Z., Han, J., E, W. & Li, Q. Approximation and optimization theory for linear continuous-time recurrent neural networks. *Journal of Machine Learning Research* **23** (2022).
194. Wang, S., Li, Z. & Li, Q. *Inverse approximation theory for nonlinear recurrent neural networks* in *International Conference on Learning Representations* (2024).
195. Martens, J. & Sutskever, I. *Learning recurrent neural networks with hessian-free optimization* in *International Conference on Machine Learning* (2011).
196. Pavliotis, G. A. *Stochastic processes and applications* (2014).
197. Merrill, W., Petty, J. & Sabharwal, A. *The illusion of state in state-space models* in *International Conference on Machine Learning* (2024).
198. Chen, M., Pennington, J. & Schoenholz, S. S. *Dynamical isometry and a mean field theory of RNNs: gating enables signal propagation in recurrent neural networks* in *International Conference on Machine Learning* (2018).
199. Hardt, M., Ma, T. & Recht, B. Gradient descent learns linear dynamical systems. *Journal of Machine Learning Research* **19** (2018).
200. Nesterov, Y. *et al. Lectures on convex optimization* (Springer, 2018).
201. Cohen, J. M., Kaur, S., Li, Y., Kolter, J. Z. & Talwalkar, A. *Gradient descent on neural networks typically occurs at the edge of stability* in *International Conference on Learning Representations* (2021).
202. Gur-Ari, G., Roberts, D. A. & Dyer, E. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754* (2018).
203. Pan, Y. & Li, Y. Toward understanding why Adam converges faster than SGD for Transformers. *arXiv preprint arXiv:2306.00204* (2023).

204. Dauphin, Y. N., Fan, A., Auli, M. & Grangier, D. *Language modeling with gated convolutional networks* in *International Conference on Machine Learning* (2017).

205. Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y. & Schölkopf, B. *Recurrent independent mechanisms* in *International Conference on Learning Representations* (2021).

206. Linnainmaa, S. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics* **16** (1976).

207. Frenkel, C., Bol, D. & Indiveri, G. *Bottom-up and top-down neural processing systems design: neuromorphic intelligence as the convergence of natural and artificial intelligence* in *Proceedings of the IEEE* (2023).

208. Ganguli, S., Huh, D. & Sompolinsky, H. Memory traces in dynamical systems. *Proceedings of the National Academy of Sciences* **105** (2008).

209. Wengert, R. E. A simple automatic derivative evaluation program. *Communications of the ACM* **7** (1964).

210. Murray, J. M. Local online learning in recurrent networks with random feedback. *eLife* **8** (2019).

211. Marschall, O., Cho, K. & Savin, C. A unified framework of online learning algorithms for training recurrent neural networks. *Journal of Machine Learning Research* **21** (2020).

212. Gu, A., Dao, T., Ermon, S., Rudra, A. & Ré, C. *HiPPO: recurrent memory with optimal polynomial projections* in *Advances in Neural Information Processing Systems* (2020).

213. Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S. & Metzler, D. Long Range Arena: a benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006* (2020).

214. Griewank, A. & Walther, A. *Evaluating derivatives: principles and techniques of algorithmic differentiation* (Society for Industrial and Applied Mathematics, 2008).

215. McBride, L. & Narendra, K. Optimization of time-varying systems. *IEEE Transactions on Automatic Control* **10** (1965).

216. Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D. & Kavukcuoglu, K. *Decoupled neural interfaces using synthetic gradients* in *International Conference on Machine Learning* (2017).

217. Wayne, G. D. *Self-modeling neural systems* PhD Thesis (Columbia University, 2013).

218. Schmidhuber, J. *Recurrent networks adjusted by adaptive critics* in *International Joint Conference on Neural Networks* **1** (1990).
219. Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine Learning* **3** (1988).
220. Javed, K., Shah, H., Sutton, R. S. & White, M. Scalable real-time recurrent learning using columnar-constructive networks. *Journal of Machine Learning Research* **24** (2023).
221. Gori, M., Bengio, Y. & De Mori, R. *BPS: a learning algorithm for capturing the dynamic nature of speech* in *International Joint Conference on Neural Networks* (1989).
222. Nangia, N. & Bowman, S. R. ListOps: A diagnostic dataset for latent tree learning. *arXiv preprint arXiv:1804.06028* (2018).
223. Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y. & Potts, C. *Learning word vectors for sentiment analysis* in *Proceedings of the Annual Meeting of the Association for Computational Linguistics* (2011).
224. Zenke, F. & Neftci, E. O. Brain-inspired learning on neuromorphic substrates. *Proceedings of the IEEE* **109** (2021).
225. Frenkel, C. & Indiveri, G. *ReckOn: a 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales* in *2022 IEEE International Solid- State Circuits Conference (ISSCC)* **65** (2022).
226. Demirag, Y., Frenkel, C., Payvand, M. & Indiveri, G. Online training of spiking recurrent neural networks with phase-change memory synapses. *arXiv preprint arXiv:2108.01804* (2021).
227. Graves, A., Wayne, G. & Danihelka, I. Neural Turing machines. *arXiv preprint arXiv:1410.5401* (2014).
228. 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. *International Conference on Learning Representations* (2023).
229. Mountcastle, V. B. Modality and topographic properties of single neurons of cat's somatic sensory cortex. *Journal of Neurophysiology* **20** (1957).
230. Douglas, R. J., Martin, K. A. & Whitteridge, D. A canonical microcircuit for neocortex. *Neural Computation* **1** (1989).
231. Roelfsema, P. R. & Holtmaat, A. Control of synaptic plasticity in deep cortical networks. *Nature Reviews Neuroscience* **19** (2018).

232. Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J. & Hinton, G. Backpropagation and the brain. *Nature Reviews Neuroscience* **21** (2020).
233. Zucchet, N., Schug, S., von Oswald, J., Zhao, D. & Sacramento, J. *A contrastive rule for meta-learning* in *Advances in Neural Information Processing Systems* (2022).
234. Zucchet, N. & Sacramento, J. Beyond backpropagation: bilevel optimization through implicit differentiation and equilibrium propagation. *Neural Computation* **34** (2022).
235. Laborieux, A. & Zenke, F. *Holomorphic equilibrium propagation computes exact gradients through finite size oscillations* in *Advances in Neural Information Processing Systems* (2022).
236. Senn, W., Dold, D., Kungl, A. F., Ellenberger, B., Jordan, J., Bengio, Y., Sacramento, J. & Petrovici, M. A. A neuronal least-action principle for real-time learning in cortical circuits. *eLife* (2024).
237. Zucchet, N., Feng, Q., Laborieux, A., Zenke, F., Senn, W. & Sacramento, J. Teaching signal synchronization in deep neural networks with prospective neurons. *arXiv preprint arXiv:2511.14917* (2025).
238. Lillicrap, T. P. & Santoro, A. Backpropagation through time and the brain. *Current Opinion in Neurobiology* **55** (2019).
239. Zucchet, N., Bornschein, J., Chan, S., Lampinen, A., Pascanu, R. & De, S. How language models learn facts? Dynamics, curricula and hallucinations. *Conference on language modeling* (2025).
240. Allen-Zhu, Z. & Li, Y. Physics of language models: Part 3.1, knowledge storage and extraction. *arXiv preprint arXiv:2309.14316* (2023).
241. Nasr, M., Carlini, N., Hayase, J., Jagielski, M., Cooper, A. F., Ippolito, D., Choquette-Choo, C. A., Wallace, E., Tramèr, F. & Lee, K. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035* (2023).
242. Geva, M., Schuster, R., Berant, J. & Levy, O. *Transformer feed-forward layers are key-value memories* in *Conference on Empirical Methods in Natural Language Processing* (2021).
243. Meng, K., Bau, D., Andonian, A. & Belinkov, Y. Locating and editing factual associations in GPT. *Advances in Neural Information Processing Systems* (2022).
244. Geva, M., Bastings, J., Filippova, K. & Globerson, A. Dissecting recall of factual associations in auto-regressive language models. *Conference on Empirical Methods in Natural Language Processing* (2023).

245. Nanda, N., Rajamanoharan, S., Kramár, J. & Shah, R. *Fact finding: Attempting to reverse-engineer factual recall on the neuron level* in *AI alignment forum, 2023* (2023).

246. Lin, S., Hilton, J. & Evans, O. TruthfulQA: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958* (2021).

247. Schaeffer, R., Miranda, B. & Koyejo, S. Are emergent abilities of large language models a mirage? *Advances in Neural Information Processing Systems* **36** (2024).

248. Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., *et al.* Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022).

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

250. Nichani, E., Lee, J. D. & Bietti, A. *Understanding factual recall in Transformers via associative memories* in *International Conference on Learning Representations* (2025).

251. Gu, X., Lyu, K., Li, J. & Zhang, J. Data mixing can induce phase transitions in knowledge acquisition. *arXiv preprint arXiv:2505.18091* (2025).

252. Ou, Y., Yao, Y., Zhang, N., Jin, H., Sun, J., Deng, S., Li, Z. & Chen, H. *How do LLMs acquire new knowledge? a knowledge circuits perspective on continual pre-training* in *Findings of the Association for Computational Linguistics* (2025).

253. Charton, F. & Kempe, J. Emergent properties with repeated examples. *arXiv preprint arXiv:2410.07041* (2024).

254. Park, C. F., Lubana, E. S., Pres, I. & Tanaka, H. *Competition dynamics shape algorithmic phases of in-context learning* in *International Conference on Learning Representations* (2025).

255. Bengio, Y., Louradour, J., Collobert, R. & Weston, J. *Curriculum learning* in *International Conference on Machine Learning* (2009).

256. McCloskey, M. & Cohen, N. J. in (Psychology of Learning and Motivation, 1989).

257. Ovadia, O., Brief, M., Mishaeli, M. & Elisha, O. Fine-tuning or retrieval? comparing knowledge injection in LLMs. *arXiv preprint arXiv:2312.05934* (2023).

258. Jain, A., Maleki, A. & Saade, N. *To fine-tune or not to fine-tune* 2024.
259. Gekhman, Z., Yona, G., Aharoni, R., Eyal, M., Feder, A., Reichart, R. & Herzig, J. *Does fine-tuning LLMs on new knowledge encourage hallucinations?* in *Proceedings of the Annual Meeting of the Association for Computional Linguistic* (2024).
260. Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., *et al.* In-context learning and induction heads. *Transformer Circuits Thread* (2022).
261. Garg, S., Tsipras, D., Liang, P. S. & Valiant, G. What can transformers learn in-context? a case study of simple function classes. *Advances in neural information processing systems* **35** (2022).
262. Reddy, G. *The mechanistic basis of data dependence and abrupt learning in an in-context classification task* in *International Conference on Learning Representations* (2024).
263. Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., *et al.* Emergent abilities of large language models. *Transactions on machine learning research* (2022).
264. Baranes, A. & Oudeyer, P.-Y. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems* **61** (2013).
265. Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., *et al.* Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808* (2021).
266. Zucchet, N., d'Angelo, F., Lampinen, A. & Chan, S. The emergence of sparse attention: impact of data distribution and benefits of repetition. *Advances in Neural Information Processing Systems* (2025).
267. Bengio, Y., Ducharme, R., Vincent, P. & Jauvin, C. A neural probabilistic language model. *Journal of Machine Learning Research* **3** (2003).
268. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., *et al.* Language models are unsupervised multitask learners. *OpenAI blog* **1** (2019).
269. 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).

270. Ganguli, D., Hernandez, D., Lovitt, L., Askell, A., Bai, Y., Chen, A., Conerly, T., Dassarma, N., Drain, D., Elhage, N., *et al. Predictability and surprise in large generative models* in *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency* (2022), 1747.

271. Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., *et al.* Beyond the imitation game: quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615* (2022).

272. Anderson, P. W. More is different: Broken symmetry and the nature of the hierarchical structure of science. *Science* **177** (1972).

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

274. Nanda, N., Chan, L., Lieberum, T., Smith, J. & Steinhardt, J. Progress measures for grokking via mechanistic interpretability. *International Conference on Learning Representations* (2023).

275. Singh, A. K., Moskovitz, T., Hill, F., Chan, S. C. & Saxe, A. M. What needs to go right for an induction head? a mechanistic study of in-context learning circuits and their formation. *International Conference on Machine Learning* (2024).

276. Snell, C., Wallace, E., Klein, D. & Levine, S. *Predicting emergent capabilities by finetuning* in *Conference on Language Modelling* (2024).

277. Zucchet, N., Bornschein, J., Chan, S., Lampinen, A., Pascanu, R. & De, S. How do language models learn facts? Dynamics, curricula and hallucinations. *Conference on Language Modeling* (2025).

278. Anwar, U., Saparov, A., Rando, J., Paleka, D., Turpin, M., Hase, P., Lubana, E. S., Jenner, E., Casper, S., Sourbut, O., *et al.* Foundational challenges in assuring alignment and safety of large language models. *arXiv preprint arXiv:2404.09932* (2024).

279. Du, Z., Zeng, A., Dong, Y. & Tang, J. Understanding emergent abilities of language models from the loss perspective. *Advances in Neural Information Processing Systems* (2024).

280. Zhao, R., Qin, T., Alvarez-Melis, D., Kakade, S. & Saphra, N. Distributional scaling laws for emergent capabilities. *arXiv preprint arXiv:2502.17356* (2025).

281. Barak, B. *Emergent abilities and grokking: fundamental, mirage, or both?* 2023.

282. Marion, P., Berthier, R., Biau, G. & Boyer, C. *Attention layers provably solve single-location regression* in *International Conference on Learning Representations* (2024).

283. Lee, K., Ippolito, D., Nystrom, A., Zhang, C., Eck, D., Callison-Burch, C. & Carlini, N. *Deduplicating training data makes language models better* in *Annual Meeting of the Association for Computational Linguistics* (2021).

284. Hernandez, D., Brown, T., Conerly, T., DasSarma, N., Drain, D., El-Showk, S., Elhage, N., Hatfield-Dodds, Z., Henighan, T., Hume, T., *et al.* Scaling laws and interpretability of learning from repeated data. *arXiv preprint arXiv:2205.10487* (2022).

285. Bietti, A., Cabannes, V., Bouchacourt, D., Jegou, H. & Bottou, L. Birth of a transformer: a memory viewpoint. *Advances in Neural Information Processing Systems* (2023).

286. Edelman, E., Tsilivis, N., Edelman, B., Malach, E. & Goel, S. The evolution of statistical induction heads: in-context learning Markov chains. *Advances in Neural Information Processing Systems* (2024).

287. Nichani, E., Damian, A. & Lee, J. D. How transformers learn causal structure with gradient descent. *International Conference on Machine Learning* (2024).

288. D'Angelo, F., Croce, F. & Flammarion, N. *Selective induction heads: how transformers select causal structures in context* in *International Conference on Learning Representations* (2025).

289. Arora, S., Eyuboglu, S., Timalsina, A., Johnson, I., Poli, M., Zou, J., Rudra, A. & Ré, C. *Zoology: Measuring and improving recall in efficient language models* in *International Conference on Learning Representations* (2023).

290. Singh, A., Chan, S., Moskovitz, T., Grant, E., Saxe, A. & Hill, F. The transient nature of emergent in-context learning in Transformers. *Advances in Neural Information Processing Systems* (2023).

291. Varre, A. V., Vladarean, M.-L., Pillaud-Vivien, L. & Flammarion, N. On the spectral bias of two-layer linear networks. *Advances in Neural Information Processing Systems* (2023).

292. Jacot, A., Ged, F., Şimşek, B., Hongler, C. & Gabriel, F. Saddle-to-saddle dynamics in deep linear networks: small initialization training, symmetry, and sparsity. *arXiv preprint arXiv:2106.15933* (2021).

293. Abbe, E., Adsera, E. B. & Misiakiewicz, T. *The merged-staircase property: a necessary and nearly sufficient condition for SGD learning of sparse functions on two-layer neural networks* in *Conference on Learning Theory* (2022).

294. Pesme, S. & Flammarion, N. Saddle-to-saddle dynamics in diagonal linear networks. *Advances in Neural Information Processing Systems* (2023).

295. Abbe, E., Adsera, E. B. & Misiakiewicz, T. *SGD learning on neural networks: leap complexity and saddle-to-saddle dynamics* in *Conference on Learning Theory* (2023).

296. Zhang, Y., Singh, A. K., Latham, P. E. & Saxe, A. *Training dynamics of in-context learning in linear attention* in *International Conference on Machine Learning* (2025).

297. Watanabe, S. *Algebraic geometry and statistical learning theory* (Cambridge University Press, 2009).

298. Hoogland, J., Wang, G., Farrugia-Roberts, M., Carroll, L., Wei, S. & Murfet, D. The developmental landscape of in-context learning. *arXiv preprint arXiv:2402.02364* (2024).

299. Arora, S. & Goyal, A. A theory for emergence of complex skills in language models. *arXiv preprint arXiv:2307.15936* (2023).

300. Hendel, R., Geva, M. & Globerson, A. *In-context learning creates task vectors* in *Findings of the Association for Computational Linguistics* (eds Bouamor, H., Pino, J. & Bali, K.) (2023).

301. Todd, E., Li, M. L., Sharma, A. S., Mueller, A., Wallace, B. C. & Bau, D. Function vectors in large language models. *International Conference on Learning Representations* (2024).

302. Darcet, T., Oquab, M., Mairal, J. & Bojanowski, P. *Vision transformers need registers* in *International Conference on Learning Representations* (2024).

303. Settles, B. Active learning literature survey. *University of Wisconsin-Madison, Department of Computer Sciences* (2009).

304. Lake, B. M., Salakhutdinov, R. & Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science* **350** (2015).

305. Warstadt, A., Mueller, A., Choshen, L., Wilcox, E., Zhuang, C., Ciro, J., Mosquera, R., Paranjape, B., Williams, A., Linzen, T., *et al.* Findings of the BabyLM challenge: Sample-efficient pretraining on developmentally plausible corpora. *arXiv preprint arXiv:2504.08165* (2025).

306. Hindle, A., Barr, E. T., Gabel, M., Su, Z. & Devanbu, P. On the naturalness of software. *Communications of the ACM* **59** (2016).

307. Boeddeker, C., Hanebrink, P., Drude, L., Heymann, J. & Haeb-Umbach, R. On the computation of complex-valued gradients with application to statistically optimum beamforming. *arXiv preprint arXiv:1701.00392* (2017).

308. 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.

309. Heek, J., Levskaya, A., Oliver, A., Ritter, M., Rondepierre, B., Steiner, A. & Zee, M. v. *Flax: A neural network library and ecosystem for JAX* 2023.

310. Zucchet, N., Meier, R. & Schug, S. *Minimal LRU* 2023.

311. Foundation, W. *Wikimedia Downloads*

312. Amari, S.-I. Learning patterns and pattern sequences by self-organizing nets of threshold elements. *IEEE Transactions on Computers* (1972).

313. Petroni, F., Rocktaschel, T., Lewis, P., Bakhtin, A., Wu, Y., Miller, A. H. & Riedel, S. *Language models as knowledge bases?* in *Conference on Empirical Methods in Natural Language Processing* (2019).

314. Allen-Zhu, Z. & Li, Y. Physics of language models: Part 3.3, knowledge capacity scaling laws. *arXiv preprint arXiv:2404.05405* (2024).

315. McEliece, R. J., Posner, E. C., Rodemich, E. R. & Venkatesh, S. S. The capacity of the Hopfield associative memory. *IEEE transactions on Information Theory* **33** (1987).

316. Hinton, G. E. *Learning distributed representations of concepts* in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (1986).

317. McClelland, J. L. in *Developing cognitive competence: New approaches to process modeling* (1995).

318. Baldi, P. F. & Hornik, K. Learning in linear neural networks: a survey. *IEEE Transactions on neural networks* **6** (1995).
319. Nguyen, T. Understanding transformers via n-gram statistics. *Advances in Neural Information Processing Systems* (2024).
320. Chang, H., Park, J., Ye, S., Yang, S., Seo, Y., Chang, D.-S. & Seo, M. *How do large language models acquire factual knowledge during pretraining?* in *Advances in Neural Information Processing Systems* (2024).
321. Zhang, Y., Li, Y., Cui, L., Cai, D., Liu, L., Fu, T., Huang, X., Zhao, E., Zhang, Y., Chen, Y., *et al.* Siren's song in the AI ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219* (2023).
322. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., *et al.* Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* **114** (2017).
323. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. & Chintala, S. *Pytorch: an imperative style, high-performance deep learning library* in *Advances in Neural Information Processing Systems* (2019).

Part III

APPENDIX

A

VANISHING AND EXPLODING GRADIENTS ARE NOT THE END OF THE STORY

A.1 DEFINITION OF THE RECURRENT NETWORKS WE USE

In this section, we rigorously define all the architectures we use in the main text and in the appendix, as well as precisely describe how we initialize them.

A.1.1 *Linear recurrent neural network*

Let us start by introducing the linear recurrent neural network we are using. It satisfies

$$h_0 = 0 \tag{A.1}$$

$$h_{t+1} = Ah_t + Bx_{t+1} \tag{A.2}$$

$$y_t = Ch_t + Dx_t \tag{A.3}$$

with $x_t \in \mathbb{R}^{d_{\text{in}}}$, $h_t \in \mathbb{R}^n$, $y_t \in \mathbb{R}^{d_{\text{out}}}$, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times d_{\text{in}}}$, $C \in \mathbb{R}^{d_{\text{out}} \times n}$ and $D \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. We draw the entries of B, C and D from independent truncated normal distributions with fan_in scaling. We draw each entry of A from $\mathcal{N}(0, 1/\sqrt{n})$ independently and then apply the following post-processing to it: First we complex diagonalize A, which we can do almost surely. Note λ its eigenvalues. We then transform them according to

$$\lambda \leftarrow (\nu_0 + (1-\nu_0)\tanh(|\lambda|)) \exp\left(i \frac{\text{angle}(\lambda)}{\pi} \theta_0\right) \tag{A.4}$$

with ν_0 and θ_0 two scalars that we control. This transformation has several benefits: we are guaranteed that the magnitude of λ is within $[\nu_0, 1]$ (and in $[\nu_0, \nu_0 + (1-\nu_0)\tanh(1)]$ in the limit $n \to \infty$ as the eigenvalues of A stay within the unit circle in that limit), and conjugate pairs of eigenvalues remain conjugate. This last point ensures that the resulting matrix remains real without having to change the eigenvectors. When we split the connectivity matrix A into several independent blocks, we initialize each block separately with the scheme described above.

A.1.2 *Complex-valued RNN and linear recurrent unit (LRU)*

Both architectures have recurrence of the form

$$\begin{aligned} h_0 &= 0 && \text{(A.5)} \\ h_{t+1} &= \lambda \odot h_t + \gamma \odot B x_t && \text{(A.6)} \\ y_t &= \mathrm{Re}[C h_t + D x_t] && \text{(A.7)} \end{aligned}$$

with $\odot$ the element-wise product, $x_t \in \mathbb{R}^{d_\text{in}}$, $h_t \in \mathbb{C}^n$, $y_t \in \mathbb{R}^{d_\text{out}}$, $\lambda \in \mathbb{C}^n$, $B \in \mathbb{C}^{n \times d_\text{in}}$, $\gamma \in \mathbb{R}^n$, $C \in \mathbb{C}^{d_\text{out} \times n}$ and $D \in \mathbb{R}^{d_\text{out} \times d_\text{in}}$.

For the complex-valued linear RNN, we take

$$\begin{aligned} \lambda &= \lambda^\text{re} + i \lambda^\text{im} && \text{(A.8)} \\ \gamma &= 1 && \text{(A.9)} \end{aligned}$$

so that it can be considered as parametrizing the diagonalized version of the linear RNN.

For the LRU [69], we take

$$\begin{aligned} \lambda &= \exp(-\exp(\omega_\nu)) \exp(i \exp(\omega_\theta)) && \text{(A.10)} \\ \gamma &= \exp(\omega_\gamma). && \text{(A.11)} \end{aligned}$$

For both architectures, we use the LRU initialization so that λ is uniformly distributed on the ring between the circles of radii ν_0 and 1, with absolute angle restricted to be below θ_0. For the LRU, we initialize γ to $\sqrt{1 - |\lambda|^2}$.

A.1.3 *S4*

We consider a slightly modified version of S4 here:

$$\Delta = \mathrm{softplus}(\omega_\Delta) \tag{A.12}$$

$$h_{t+1} = \exp\left((\omega_A^{\mathrm{re}} + i\omega_A^{\mathrm{im}})\Delta\right) \odot h_t + \Delta \odot Bx_{t+1} \tag{A.13}$$

$$y_t = \mathrm{Re}[Ch_t + Dx_t] \tag{A.14}$$

with $x_t \in \mathbb{R}^{d_{\mathrm{in}}}$, $h_t \in \mathbb{C}^n$, $y_t \in \mathbb{R}^{d_{\mathrm{out}}}$, $\omega_A^{\mathrm{re}} + i\omega_A^{\mathrm{im}} \in \mathbb{C}^n$, $B \in \mathbb{C}^{n\times d_{\mathrm{in}}}$, $\omega_\Delta \in \mathbb{R}^n$, $C \in \mathbb{C}^{d_{\mathrm{out}}\times n}$ and $D \in \mathbb{R}^{d_{\mathrm{out}}\times d_{\mathrm{in}}}$. The main difference with the standard architecture is that we do not consider any state expansion so that there is no parameter sharing. The makes this architecture closer to the linear RNN architecture we mostly focus on in this paper, while keeping the kind of parametrization it uses. In the same spirit, we do not couple the input and the readout matrices in any way.

The initialization we use also differs from existing ones as we initialize Δ to 1 and $\exp\left(\Delta(\omega_A^{\mathrm{re}} + i\omega_A^{\mathrm{im}})\right)$ in the same way as λ in the LRU.

A.1.4 *GRU*

The GRU version we use is the following:

$$r_{t+1} = \sigma(W_{rx}x_{t+1} + W_{rh}h_t + b_r) \tag{A.15}$$

$$f_{t+1} = \sigma(W_{fx}x + W_{fh}h_t + b_f) \tag{A.16}$$

$$n_{t+1} = \tanh(W_{nx}x_{t+1} + b_{nx} + r \odot (W_{nh}h_t + b_{nh})) \tag{A.17}$$

$$h_{t+1} = f_{t+1} \odot h_t + (1 - f_{t+1}) \odot n_{t+1} \tag{A.18}$$

$$\tag{A.19}$$

with σ the sigmoid function, $h_t \in \mathbb{R}^n$, $x_t \in \mathbb{R}^{d_{\mathrm{in}}}$ and all the other matrices appropriately sized. We initialize the parameters with the Chrono initialization [184]: all parameters are initialized using standard practice (orthogonal initialization for the weights taking h as input, Lecun initialization for the rest, biases initialized at 0) except for b_f, which is initialized as

$$b_f \sim \log\left(\mathcal{U}\left(T_{\mathrm{min}}, T_{\mathrm{max}}\right)\right). \tag{A.20}$$

Here, T_{min} and T_{max} denote the minimum and maximal characteristic time scale. In the experiments of Section 3.6 we take

$$T_{\text{min}} = \frac{1}{1-\nu_0} \tag{A.21}$$

$$T_{\text{max}} = \frac{2}{1-\nu_0} \tag{A.22}$$

to enable the comparison with other architectures. Indeed, when ignoring the dependence of the forget gate f on the input x and on the hidden state h, this corresponds to having $f \in \left[\nu_0, \frac{1+\nu_0}{2}\right]$.

A.2 THEORY

This section introduces all the theoretical results we directly or indirectly mention in the main text, as well as provides a proof for them.

A.2.1 *Useful lemmas*

Most, if not all the calculations, that we will be doing in this section involves infinite sums. We state and prove two useful lemmas to simplify later calculations.

Lemma 1. *For $\alpha, \beta \in \mathbb{C}$ satisfying $|\alpha| < 1$ and $|\beta| < 1$, and $(u_n)_{n\in\mathbb{Z}}$ a bounded sequence satisfying $u_{-n} = u_n$, we have*

$$\sum_{n,m\geq 0} \alpha^n \beta^n u_{n-m} = \frac{1}{1-\alpha\beta}\left(u_0 + \sum_{\Delta\geq 1}(\alpha^\Delta + \beta^\Delta)u_\Delta\right) \tag{A.23}$$

Proof. The proof naturally comes from separating the indices n and m in three sets: one in which the two are equals, one in which n is larger and one in which m is larger. This gives

$$\sum_{n,m\geq 0} \alpha^n \beta^m u_{n-m} = \sum_{n=m} \alpha^n \beta^m u_{n-m} + \sum_{n>m} \alpha^n \beta^m u_{n-m} + \sum_{n<m} \alpha^n \beta^m u_{n-m} \tag{A.24}$$

$$= \sum_n \alpha^n \beta^n u_0 + \sum_m \alpha^m \beta^m \sum_{\Delta\geq 1} \alpha^\Delta u_\Delta + \sum_n \alpha^n \beta^n \sum_{\Delta\geq 1} \beta^\Delta u_{-\Delta} \tag{A.25}$$

$$= \sum_n \alpha^n \beta^n \left(u_0 + \sum_{\Delta\geq 1}(\alpha^\Delta + \beta^\Delta)u_\Delta\right) \tag{A.26}$$

$$= \frac{1}{1-\alpha\beta}\left(u_0 + \sum_{\Delta\geq 1}(\alpha^\Delta + \beta^\Delta)u_\Delta\right) \tag{A.27}$$

□

Lemma 2. *In the same conditions as Lemma 1, we have*

$$\sum_{n,m\geq 0} nm\alpha^{n-1}\beta^{m-1}u_{n-m} = \frac{\mathrm{d}}{\mathrm{d}\alpha}\frac{\mathrm{d}}{\mathrm{d}\beta}\left[\frac{1}{1-\alpha\beta}\left(u_0 + \sum_{\Delta\geq 1}(\alpha^\Delta + \beta^\Delta)u_\Delta\right)\right] \tag{A.28}$$

Proof. This follows from remarking that

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\frac{\mathrm{d}}{\mathrm{d}\beta}\left[\sum_{n,m\geq 0}\alpha^n\beta^m u_{n-m}\right] = \frac{\mathrm{d}}{\mathrm{d}\alpha}\left[\sum_{n,m\geq 0} m\alpha^n\beta^{m-1}u_{n-m}\right] \tag{A.29}$$

$$= \sum_{n,m\geq 0} nm\alpha^{n-1}\beta^{m-1}u_{n-m} \tag{A.30}$$

and using Lemma 1 to get the final result. □

A.2.2 *The curse of memory: signal propagation analysis*

We recall the assumptions that we stated in Section 3.3.2:

a) **Linear diagonal recurrent neural networks**. We restrict ourselves to networks satisfying $h_{t+1} = \lambda \odot h_t + x_{t+1}$ with λ, h_t and x_t complex numbers. Without loss of generality, we focus on the one dimensional setting. We additionally consider λs with absolute values smaller than 1.

b) **Infinite time horizon**. We consider infinite sequences and initialize the network dynamics at $t_0 = -\infty$.

c) **Wide-sense stationarity**. We assume the different quantities that the network receives, which includes the inputs x_t, to be wise-sense stationary (WSS). A random process X_t is said to be WSS if its auto-correlation function is independent of time, that is, for all $t \in \mathbb{R}$ and $\Delta \in \mathbb{R}$, $\mathbb{E}\left[X_{t+\Delta}\bar{X}_t\right] := R_X(\Delta)$.

A.2.2.1 *Forward pass*

Without loss of generality, we can take $t = 0$ given the wide-sense stationarity and infinite time horizon assumptions. Let us first remark that we have

$$h_0 = \sum_{n \geq 0} \lambda^n x_{-n} \tag{A.31}$$

so that

$$\begin{aligned}
\mathbb{E}[|h_0|^2] &= \sum_{n,m \geq 0} \lambda^n \bar{\lambda}^m \mathbb{E}\left[x_{-n} \bar{x}_{-m}\right] && \text{(A.32)} \\
&= \sum_{n,m \geq 0} \lambda^n \bar{\lambda}^m R_x(n-m) && \text{(A.33)} \\
&= \frac{1}{1-|\lambda|^2} \left(R_x(0) + \sum_{\Delta \geq 1} (\bar{\lambda}^\Delta + \lambda^\Delta) R_x(\Delta) \right). && \text{(A.34)}
\end{aligned}$$

We used Lemma 1 to obtain the last equality. In Section 3.3.2, we focused on the real case $\bar{\lambda} = \lambda$, so this formula becomes Equation 3.5. If we further assume that the auto-correlation of x decreases exponentially with decay rate ρ, that is $R_x(\Delta) = \rho^{|\Delta|}$, we can further simplify the last expression:

$$\begin{aligned}
\mathbb{E}[|h_0|^2] &= \frac{1}{1-|\lambda|^2} \left(1 + \sum_{\Delta \geq 1} (\bar{\lambda}^\Delta + \lambda^\Delta) \rho^\Delta \right) && \text{(A.35)} \\
&= \frac{1}{1-|\lambda|^2} \left(1 + \frac{\bar{\lambda}\rho}{1-\bar{\lambda}\rho} + \frac{\lambda\rho}{1-\lambda\rho} \right) && \text{(A.36)} \\
&= \frac{1-\rho^2|\lambda|^2}{|1-\rho\lambda|^2(1-|\lambda|^2)} && \text{(A.37)}
\end{aligned}$$

It follows that if the inputs are i.i.d. ($\rho = 0$), we have $\mathbb{E}[|h_0|^2] = (1-|\lambda|^2)^{-1}$, and if the inputs are constant equal to 1 ($\rho = 1$), we have $\mathbb{E}[|h_0|^2] = |1-\lambda|^{-2}$.

A.2.2.2 *Backward pass*

Differentiating the update $h_{t+1} = \lambda h_t + x_{t+1}$ with respect to λ gives

$$\frac{\mathrm{d}h_{t+1}}{\mathrm{d}\lambda} = \lambda \frac{\mathrm{d}h_t}{\mathrm{d}\lambda} + h_t \tag{A.38}$$

so that

$$\begin{aligned}\frac{\mathrm{d}h_0}{\mathrm{d}\lambda} &= \sum_{n\geq 0} \lambda^n h_{-n-1} &\text{(A.39)}\\ &= \sum_{n\geq 0} \lambda^n \sum_{m\geq 0} \lambda^m x_{-n-m-1} &\text{(A.40)}\\ &= \sum_{n,m\geq 0} \lambda^{n+m} x_{-(n+m+1)} &\text{(A.41)}\\ &= \sum_{n\geq 0} n\lambda^{n-1} x_{-n-1}. &\text{(A.42)}\end{aligned}$$

Note that some extra technicalities are needed to justify these equations as λ and h_t are complex valued: these formulas hold as they would in the real-valued case as h_t is an holomorphic function of λ.

We can now compute the variance of the sensitivity of the hidden state with respect to the parameters.

$$\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right|^2\right] = \sum_{n\geq 0}\sum_{m\geq 0} nm\lambda^{n-1}\bar{\lambda}^{m-1}R_x(n-m). \tag{A.43}$$

Using Lemma 2 gives

$$\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right|^2\right] = \frac{\mathrm{d}}{\mathrm{d}\alpha}\frac{\mathrm{d}}{\mathrm{d}\beta}\left[\frac{1}{1-\alpha\beta}\left(R_x(0) + \sum_{\Delta\geq 1}(\alpha^\Delta + \beta^\Delta)R_x(\Delta)\right)\right]_{\alpha=\lambda,\beta=\bar{\lambda}}. \tag{A.44}$$

Differentiating this quantity as a product gives

$$\begin{aligned}\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right|^2\right] = \Bigg[&\frac{1+\alpha\beta}{(1-\alpha\beta)^3}\left(R_x(0) + \sum_{\Delta\geq 1}(\alpha^\Delta + \beta^\Delta)R_x(\Delta)\right) + 0\\ &+\frac{\alpha}{(1-\alpha\beta)^2}\left(\sum_{\Delta\geq 1}\Delta\alpha^{\Delta-1}R_x(\Delta)\right)\\ &+\frac{\beta}{(1-\alpha\beta)^2}\left(\sum_{\Delta\geq 1}\Delta\beta^{\Delta-1}R_x(\Delta)\right)\Bigg]_{\alpha=\lambda,\beta=\bar{\lambda}},\end{aligned} \tag{A.45}$$

which then simplifies as

$$\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right|^2\right] = \frac{1+|\lambda|^2}{(1-|\lambda|)^3}\left(R_x(0) + \sum_{\Delta\geq 1}(\lambda^\Delta + \bar{\lambda}^\Delta)R_x(\Delta)\right) + \frac{1}{(1-|\lambda|^2)^2}\left(\sum_{\Delta\geq 1}\Delta(\lambda^\Delta + \bar{\lambda}^\Delta)R_x(\Delta)\right). \quad (A.46)$$

Note that Equation 3.6 in the main text is the real-valued version of that formula.

Let us now further simplify this equation when $R_x(\Delta) = \rho^{|\Delta|}$. If we use this in the differentiated quantity before differentiating it, we get

$$\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right|^2\right] = \frac{\mathrm{d}}{\mathrm{d}\alpha}\frac{\mathrm{d}}{\mathrm{d}\beta}\left[\frac{1}{1-\alpha\beta}\left(\frac{1-\rho^2\alpha\beta}{(1-\rho\alpha)(1-\rho\beta)}\right)\right]_{\alpha=\lambda,\beta=\bar{\lambda}}. \quad (A.47)$$

Calculating this quantity manually is painful. Instead, we use the following trick. Its denominator is rather easy to compute, it is equal to $(1-\alpha\beta)^3(1-\rho\alpha)^2(1-\rho\beta)^2$. We thus multiply it to the derivative of the function we want to compute in order to obtain a polynomial with unknown factors, and use polynomial regression tools to derive the resulting coefficients. Massaging the obtained expression to make it easier to compute the closed-form value of this quantity when $\rho = 0$ and $\rho = 1$, we get

$$\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right|^2\right] = \frac{1}{(1-|\lambda|^2)^3|1-\rho\lambda|^4}\Big((1-\rho)(1+|\lambda|^2) + \rho^2(1-|\lambda|^2)^3 + \rho(1-\rho)|\lambda|^2(\rho|\lambda|^2(1+|\lambda|^2) - 2\lambda - 2\bar{\lambda})\Big) \quad (A.48)$$

This is the quantity we plot on Figure 3.1.A, when λ is real-valued. When $\rho = 0$, this quantity becomes

$$\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right|^2\right] = \frac{1+|\lambda|^2}{(1-|\lambda|^2)^3}, \quad (A.49)$$

and it is equal to

$$\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right|^2\right] = \frac{1}{|1-\lambda|^4}, \quad (A.50)$$

when $\rho = 1$. Additionally, it will diverge whenever $|\lambda| \to 1$ when $\rho < 1$, and when $\lambda \to 1$ when $\rho = 1$.

Regarding the backpropagation of errors to the inputs, the analysis we did in the main text also holds for complex number given that h_t is an holomorphic function of x_t and it thus behaves as the forward pass once replacing the input distribution with the one of output errors $\partial_{h_t} L_t$.

A.2.2.3 *Extension to fully-connected networks*

We now turn to the non-diagonal case. For the sake of simplicity, we assume that recurrent matrix is complex diagonalizable and that its eigenvalues are all different. This will enable us to differentiate the eigenvalues and the eigenvectors. We consider dynamics of the form

$$h_{t+1} = Ah_t + x_{t+1} \tag{A.51}$$

As A is complex diagonalizable, there exists a complex-valued matrix P and a complex-valued vector λ such that

$$A = P\mathrm{diag}(\lambda)P^{-1} \tag{A.52}$$

$$P_{:i}^\dagger P_{:i} = 1 \ \ \forall i. \tag{A.53}$$

The linear recurrent neural network considered above is equivalent to its diagonal version

$$h_{t+1}^{\mathrm{diag}} = \lambda h_t^{\mathrm{diag}} + P^{-1}x_{t+1} \tag{A.54}$$

$$h_t = Ph_t^{\mathrm{diag}}. \tag{A.55}$$

We now differentiate h_t w.r.t. to A using the diagonal parametrization and obtain

$$\frac{\mathrm{d}h_t}{\mathrm{d}A} = \frac{\partial h_t}{\partial P}\frac{\partial P}{\partial A} + \frac{\partial h_t}{\partial h_t^{\mathrm{diag}}}\frac{\mathrm{d}h_t^{\mathrm{diag}}}{\mathrm{d}\lambda}\frac{\partial \lambda}{\partial A} + \frac{\partial h_t}{\partial h_t^{\mathrm{diag}}}\frac{\mathrm{d}h_t^{\mathrm{diag}}}{\mathrm{d}P^{-1}}\frac{\partial P^{-1}}{\partial A}. \tag{A.56}$$

$\mathrm{d}_A P$, $\mathrm{d}_A P^{-1}$ AND $\mathrm{d}_A \lambda$ CAN BE CONSIDERED CONSTANT. Intuitively, the eigenvalues and eigenvectors move smoothly as we restricted ourselves to the case in which eigenvalues are singular. If this is not the case, math becomes trickier as the eigenvectors are not uniquely defined. We can study the behavior of those quantities in more detail, following Boeddeker *et al.* [307]:

$$\frac{\partial \lambda}{\partial A_{ij}} = \operatorname{diag}\left(P^{-1}\frac{\partial A}{\partial A_{ij}}P\right) \tag{A.57}$$

$$\frac{\mathrm{d}P}{\mathrm{d}A_{ij}} = P\left(F \odot \left(P^{-1}\frac{\partial A}{\partial A_{ij}}P\right)\right) \tag{A.58}$$

The F introduced in the last equation is equal to

$$F_{ij} := \begin{cases} \frac{1}{\lambda_j - \lambda_i} & \text{if } i \neq j \\ 0 & \text{otherwise.} \end{cases} \tag{A.59}$$

Importantly, those two quantities do not grow to infinity as the absolute value of the eigenvalues goes to 1, which means that we can consider those derivatives to be independent of $|\lambda|$ for the sake of our analysis. Note that the previous argument assumes that eigenvalues do not collapse.

$\mathrm{d}_\lambda h_t^{\mathrm{diag}}$ IS THE DOMINATING TERM IN $\mathrm{d}_A h_t$. We wonder which of the three different terms that appear in $\mathrm{d}_A h_t$ (Equation A.56) will be the dominating one as $|\lambda|$ (or λ) goes to 1. In the previous paragraph, we have shown that the derivative of P^{-1}, P and λ can be considered constant for the sake of our analysis. We thus focus on the other terms.

First, we have

$$\frac{\partial h_{t,l}}{\partial P_{ij}} = h_{t,i}^{\mathrm{diag}} \mathbb{1}_{j=l} \tag{A.60}$$

so the magnitude of this quantity is roughly the one of h_t^{diag}, which corresponds to the low pass filtering of the inputs with different λ values.

Second, we know that $\partial_{h_t^{\mathrm{diag}}} h_t$ does not change in magnitude as λ changes, as P remains bounded. So, for the third term of the sum, we are left to study the behavior of $\mathrm{d}_{P^{-1}} h_t^{\mathrm{diag}}$. We can show that it evolves according to

$$\frac{\mathrm{d}h_{t+1,k}^{\mathrm{diag}}}{\mathrm{d}P_{ij}^{-1}} = \lambda_i \frac{\mathrm{d}h_{t+1,k}^{\mathrm{diag}}}{\mathrm{d}P_{ij}^{-1}} + x_{t+1,j} \ \text{ if } k = i \tag{A.61}$$

$$\frac{\mathrm{d}h_{t+1,k}^{\mathrm{diag}}}{\mathrm{d}P_{ij}^{-1}} = 0 \ \text{ otherwise.} \tag{A.62}$$

It follows that the third term in the sum also corresponds to a low pass filtering of the inputs.

Finally, we know that the second term, the one in $\mathrm{d}_\lambda h_t^{\mathrm{diag}}$ will grow faster to infinity as it corresponds to two consecutive low pass filters with the same λ values (c.f. calculation above). It will thus be the dominating term in the infinite memory limit.

A.2.2.4 *On the wide-sense stationarity assumption*

In our analysis, we make the assumption that the different quantities given to the network are wide-sense stationary, that is their statistics are invariant to a temporal shift. In practice, this assumption will likely never be satisfied, as sequences are finite and as parts of the sequence (e.g., the beginning of a text) can have different statistics than other parts (e.g., the end of a text).

It should be noted that the analysis we have done in the wide-sense stationary case can provide an upper bound on the different quantity we study. Indeed, if there exists a function U such that

$$|\mathbb{E}[x_n \bar{x}_m]| \leq U(|n-m|), \tag{A.63}$$

minor modifications to our analysis enable to upper bound the different quantities we study, replacing R_x by U.

In our experiments of Section 3.6, we plug the empirical covariance defined as

$$R_x^{\mathrm{empirical}}(\Delta) := \mathbb{E}_x\left[\frac{1}{T-|\Delta|+1}\sum_{t=0}^{T-|\Delta|} x_t x_{t+|\Delta|}\right] \tag{A.64}$$

into our analytical expressions. It comes with an approximation as it averages the autocorrelation for all positions, which are not equal when wide-sense stationarity is not met.

A.2.3 *Impact of input normalization and parametrization*

In this section, we consider a diagonal linear recurrent neural network of the form

$$h_{t+1} = \lambda(\omega)h_t + \gamma(\lambda)x_{t+1} \tag{A.65}$$

with $\gamma(\lambda)$ the input normalization factor and λ parametrized by a vector ω. Next, we study the effect of input normalization and reparametrization, first in the real-valued setting and then in the complex-valued one.

A.2.3.1 *Real case*

Let us start with the forward pass: as

$$h_t = \sum_{n \geq 0} \lambda^n \gamma(\lambda) x_{t-n}, \tag{A.66}$$

γ rescales the value the hidden state takes. To avoid any explosion behavior, we thus ideally want γ to be the inverse of value of $\mathbb{E}[(h_t)^2]$ without normalization, which we have computed in Equation A.34. The same behavior holds for the backpropagation of errors to the inputs as

$$\frac{\mathrm{d}L}{\mathrm{d}x_t} = \gamma(\lambda)\left(\lambda \frac{\mathrm{d}L}{\mathrm{d}x_{t+1}} + \frac{\partial L}{\partial h_t}\right). \tag{A.67}$$

We now move to the impact of the parametrization. To simplify the calculation, we will ignore the dependency of γ on λ when differentiating it. This can easily be done in automatic differentiation software by removing this dependency from the computational graph with $\gamma(\text{stop\_gradient}(\lambda))$. We then have

$$\frac{\mathrm{d}h_t}{\mathrm{d}\omega} = \frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\frac{\mathrm{d}\lambda}{\mathrm{d}\omega} \tag{A.68}$$

and $\mathrm{d}_\lambda h_t$ which is rescaled by γ compared to the calculation we did above. As a consequence, both the input normalization and the parametrization can help to mitigate the curse of memory.

A.2.3.2 *On the difficulty of parametrizing complex numbers*

We now extend the previous analysis to the complex case, and take a polar parametrization of λ: $\lambda(\omega) = \nu(\omega)\exp(i\theta(\omega))$. The effect of the input normalization does not change when moving to complex numbers. The role of the reparametrization is however a bit more subtle. As h_t is an holomorphic function of λ, we have $\mathrm{d}_{\bar{\lambda}} h_t = 0$ and

$$\frac{\mathrm{d}h_t}{\mathrm{d}\omega} = \frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\frac{\mathrm{d}\lambda}{\mathrm{d}\omega} = \frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\left(\frac{1}{2}\frac{\mathrm{d}\nu}{\mathrm{d}\omega}\exp(i\theta) + \frac{i}{2}\nu\frac{\mathrm{d}\theta}{\mathrm{d}\omega}\exp(i\theta)\right). \tag{A.69}$$

It follows that

$$\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\omega}\right|^2\right] = \frac{1}{4}\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right|^2\right]\left|\frac{\mathrm{d}\nu}{\mathrm{d}\omega}\exp(i\theta) + i\nu\frac{\mathrm{d}\theta}{\mathrm{d}\omega}\exp(i\theta)\right|^2 \tag{A.70}$$

$$= \frac{1}{4}\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right|^2\right]\left|\frac{\mathrm{d}\nu}{\mathrm{d}\omega} + i\nu\frac{\mathrm{d}\theta}{\mathrm{d}\omega}\right|^2 \tag{A.71}$$

$$= \frac{1}{4}\mathbb{E}\left[\left|\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right|^2\right]\left(\frac{\mathrm{d}\nu}{\mathrm{d}\omega}^2 + \nu^2\frac{\mathrm{d}\theta}{\mathrm{d}\omega}^2\right). \tag{A.72}$$

To simplify the analysis, we will further assume that $\mathbb{E}[|\mathrm{d}_\lambda h_t|^2]$ is only a function of ν. This asumption holds in the case of $\rho = 0$ and $\rho = 1$, c.f. Section A.2.2.2, but not necessarily otherwise. To ensure that that this quantity does not depend on λ, we thus want $\mathrm{d}_\omega \nu^2 E(\nu) = \Theta(1)$ and $\nu^2 \mathrm{d}_\omega \theta^2 E(\nu) = \Theta(1)$. The second means that the angle parametrization must depend on the value ν takes. Let us take the $\rho = 0$ example to get an idea of what the ideal parametrization should be. First, we have $\gamma(\lambda) = \sqrt{1-\nu^2}$ so that

$$E(\nu) = \gamma(\lambda)^2\frac{1+\nu^2}{(1-\nu^2)^3} = \frac{1+\nu^2}{(1-\nu^2)^2}. \tag{A.73}$$

We are left with the differential equation $\nu' = \Theta(1-\nu^2)$, which is for example solved with $\nu = \tanh(\omega_\nu)$. Now let us look at the parametrization of θ. If we ignore the ν^2 term for simplicity, the approximate differential equation it needs to solve is $\mathrm{d}_\omega\theta = \Theta(1-\nu^2)$, which can be solved by $\theta = \text{stop\_gradient}(1-\nu^2)\omega_\theta$. The exact detail of this calculation do not really matter as this is heavily input distribution dependent. However, the interesting part here is that the angle parameter must be rescaled by a function of ν. This makes intuitive sense when looking looking at the sharpness of the loss around optimality in Figure 3.1.C, but this also makes the loss even flatter further away from optimality. We will come back to this point in Section A.3.1.4, showing that in practice, such a parametrization complicates the learning of the θ. Learning complex numbers is thus difficulty, because of the angle.

A.3 LINEAR TEACHER-STUDENT TASK

This section is dedicated to detail the theoretical results behind our analysis of the teacher-student task, present all the details necessary to reproduce our empirical experiments, and provide additional analysis.

A.3.1 *1D setting*

A.3.1.1 *Calculation of the loss*

In this toy example, we are interested in learning a simple 1-dimensional linear recurrent neural network which follows the dynamics

$$h_{t+1} = \lambda h_t + x_{t+1} \tag{A.74}$$

to reproduce the hidden state h_t^* of a teacher with recurrent parameter λ^*. Note that we here allow all variables to be complex-valued. We take the loss to be

$$L(\lambda, \lambda^*) := \frac{1}{2T} \sum_{t=1}^{T} \mathbb{E}_x \left[|h_t - h_t^*|^2 \right] \tag{A.75}$$

We assume x to be drawn from a wide-sense stationary distribution so that we can focus on studying the behavior of one $L_t(\lambda, \lambda^*) := \frac{1}{2}\mathbb{E}_x \left[|h_t - h_t^*|^2 \right]$ to understand the behavior of the full loss L, in the limit of infinitely long sequences ($T \to \infty$). Moreover, to further simplify the calculations, we assume that x is real-valued and that $R_x(\Delta) = \rho^{|\Delta|}$.

Let us now proceed with the calculation:

$$L_t(\lambda, \lambda^*) := \frac{1}{2}\mathbb{E}_x \left[h_t\bar{h}_t + h_t^*\bar{h}_t^* - h_t\bar{h}_t^* - \bar{h}_t h_t^* \right]. \tag{A.76}$$

We have shown in Section A.2.2 that in the limit of $t \to \infty$,

$$\mathbb{E}_x \left[h_t\bar{h}_t \right] = \frac{1}{1 - \lambda\bar{\lambda}} \left(1 + \frac{\rho\lambda}{1 - \rho\lambda} + \frac{\rho\bar{\lambda}}{1 - \rho\bar{\lambda}} \right) \tag{A.77}$$

Similar derivations hold for the other three terms in the loss. Grouping them gives the exact value of the loss. We omit the formula as it is not particularly insightful. In the case of constant inputs ($\rho = 1$), we have

$$L_t(\lambda, \lambda^*) = \frac{1}{2} \left| \frac{1}{1-\lambda} - \frac{1}{1-\lambda^*} \right|^2. \tag{A.78}$$

In the case of i.i.d. inputs ($\rho = 0$), we have

$$L_t(\lambda, \lambda^*) = \frac{1}{2} \left(\frac{1}{1-|\lambda|^2} + \frac{1}{1-|\lambda^*|^2} - \mathrm{Re}\left[\frac{2}{1-\bar{\lambda}\lambda^*} \right] \right). \tag{A.79}$$

This is the loss we plot on Figure 3.1.B and C.

A.3.1.2 *Optimal normalization and reparametrization with uncorrelated inputs*

Having a simple closed-form solution for the value the loss takes gives us the possibility to investigate in more detail what an optimal normalization and parametrization should be. We focus on the case $\rho = 0$.

For $\rho = 0$, the optimal normalization is $\gamma(\lambda) = \sqrt{1-|\lambda|^2}$. Given that we now add an input normalization to the student, we must also add it to the teacher for the student to be able to fit it. The loss becomes

$$\begin{aligned} L_t &= \frac{1}{2} \left(\frac{\gamma(\lambda)}{1-|\lambda|^2} + \frac{\gamma(\lambda^*)}{1-|\lambda^*|^2} - \mathrm{Re}\left[\frac{2\gamma(\lambda)\gamma(\lambda^*)}{1-\bar{\lambda}\lambda^*} \right] \right) && \text{(A.80)} \\ &= 1 - \mathrm{Re}\left[\frac{\gamma(\lambda)\gamma(\lambda^*)}{1-\bar{\lambda}\lambda^*} \right]. && \text{(A.81)} \end{aligned}$$

Next, we parametrize λ as $\lambda = \nu(\omega_\nu)\exp(i\theta(\omega_\theta))$ and seek to find a parametrization such that, at optimality, $\mathbb{E}[(\mathrm{d}_{\omega_\nu} h_t)^2] = 1$ and $\mathbb{E}[(\mathrm{d}_{\omega_\theta} h_t)^2] = 1$. Given that the student perfectly fit the teacher-generated data at optimality and that the loss we use is the mean-squared error, this corresponds to having $\mathrm{d}^2_{\omega_\nu} L_t = 1$ and $\mathrm{d}^2_{\omega_\theta} L_t = 1$.

DERIVING THE OPTIMAL ν PARAMETRIZATION. We now compute the Hessian of the loss w.r.t. ω_ν. First, we can simplify our calculations by restricting ourselves to the case $\theta = \theta^*$. The loss becomes

$$L_t = 1 - \frac{\gamma(\nu)\gamma(\nu^*)}{1-\nu\nu^*}. \tag{A.82}$$

Differentiating this function a first time, we obtain

$$\frac{\mathrm{d}L_t}{\mathrm{d}\nu} = -\frac{\gamma(\nu^*)\gamma'(\nu)}{1-\nu\nu^*} - \frac{\gamma(\nu^*)\nu^*\gamma(\nu)}{(1-\nu\nu^*)^2}. \tag{A.83}$$

Differentiating it a second time gives

$$\frac{\mathrm{d}^2L_t}{\mathrm{d}\nu^2} = -\frac{\gamma(\nu^*)\gamma''(\nu)}{1-\nu\nu^*} - \frac{2\gamma(\nu^*)\nu^*\gamma'(\nu)}{(1-\nu\nu^*)^2} - \frac{2\gamma(\nu^*)\nu^{*2}\gamma(\nu)}{(1-\nu\nu^*)^3}. \tag{A.84}$$

Leveraging the fact that

$$\gamma'(\nu) = \frac{-\nu}{\gamma(\nu)} \text{ and } \gamma''(\nu) = \frac{-\gamma(\nu)^2-\nu^2}{\gamma(\nu)^3}, \tag{A.85}$$

we finally get, when $\nu = \nu^*$,

$$\frac{\mathrm{d}^2L_t}{\mathrm{d}\nu^2} = \frac{1}{(1-\nu^2)^2}. \tag{A.86}$$

Given that we are at optimality, we have

$$\frac{\mathrm{d}^2L_t}{\mathrm{d}\omega_\nu^2} = \mathbb{E}\left[\frac{\mathrm{d}h_t}{\mathrm{d}\omega_\nu}\frac{\mathrm{d}^2L_t}{\mathrm{d}h_t^2}\frac{\mathrm{d}h_t}{\mathrm{d}\omega_\nu}\right] = \nu'(\omega_\nu)^2\mathbb{E}\left[\frac{\mathrm{d}h_t}{\mathrm{d}\nu}\frac{\mathrm{d}^2L_t}{\mathrm{d}h_t^2}\frac{\mathrm{d}h_t}{\mathrm{d}\nu}\right] = \nu'(\omega_\nu)^2\frac{\mathrm{d}^2L_t}{\mathrm{d}\nu^2}. \tag{A.87}$$

To keep that quantity constant, we thus have to solve the differential equation

$$\nu'(\omega_\nu) = (1-\nu^2), \tag{A.88}$$

which gives $\nu(\omega_\nu) = \tanh(\omega_\nu)$.

DERIVING THE OPTIMAL θ PARAMETRIZATION. We now move to the parametrization of θ. We have

$$L_t = 1 - \mathrm{Re}\left[\frac{\gamma(\nu)\gamma(\nu^*)}{1-\bar{\lambda}\lambda^*}\right] = 1 - \frac{\gamma(\nu)\gamma(\nu^*)(1-\nu\nu^*\cos(\theta-\theta^*))}{|1-\bar{\lambda}\lambda^*|^2}. \tag{A.89}$$

For notational convenience, we denote

$$\alpha(\theta-\theta^*) := |1-\bar{\lambda}\lambda^*|^2 = (1-\nu\nu^*\cos(\theta-\theta^*))^2 + \nu^2\nu^{*2}\sin(\theta-\theta^*)^2. \tag{A.90}$$

We have

$$\frac{\mathrm{d}L_t}{\mathrm{d}\theta} = \gamma(\nu)\gamma(\nu^*)\left(-\frac{\nu\nu^*\sin(\theta-\theta^*)}{\alpha(\theta-\theta^*)} + \frac{(1-\nu\nu^*\cos(\theta-\theta^*))\alpha'(\theta-\theta^*)}{\alpha(\theta-\theta^*)^2}\right) \tag{A.91}$$

and

$$\frac{\mathrm{d}^2 L_t}{\mathrm{d}\theta^2} = \gamma(\nu)\gamma(\nu^*)\left(-\frac{\nu\nu^*\cos(\theta-\theta^*)}{\alpha(\theta-\theta^*)} + 2\frac{\nu\nu^*\sin(\theta-\theta^*)\alpha'(\theta-\theta^*)}{\alpha(\theta-\theta^*)^2}\right. \\ \left. + \frac{(1-\nu\nu^*\cos(\theta-\theta^*))\alpha''(\theta-\theta^*)}{\alpha(\theta-\theta^*)^2} - 2\frac{(1-\nu\nu^*\cos(\theta-\theta^*))\alpha'(\theta-\theta^*)^2}{\alpha(\theta-\theta^*)^3}\right) \tag{A.92}$$

At optimality ($\theta = \theta^*$ and $\nu = \nu^*$), we have $\alpha(0) = (1-\nu^2)^2$, $\alpha'(0) = 0$ and $\alpha''(0) = 2\nu^2$, so that

$$\frac{\mathrm{d}^2 L_t}{\mathrm{d}\theta^2} = \frac{\nu^2(1+\nu^2)}{(1-\nu^2)^2}. \tag{A.93}$$

The optimal parametrization thus has to satisfy

$$\theta'(\omega_\theta) = \frac{1-\nu^2}{\nu\sqrt{1+\nu^2}}, \tag{A.94}$$

that is

$$\theta(\omega_\theta) = \omega_\theta\frac{1-\nu^2}{\nu\sqrt{1+\nu^2}} \tag{A.95}$$

There are two things we can remark:

- First, the parametrization that we derived for the general case in Section A.2.3.2, which additionally ignored the dependence of γ on λ, is relatively accurate. The only difference is the apparition of the extra ν term, which becomes insignificant in the long memory limit $\nu \to 1$.

- Second, the optimal θ parametrization has to be a function of ν, and thus ω_ν, so the differential equation ν needs to satisfy changes. Yet, this considerably simplifies the calculation and there is no simple solution to that problem. One could still argue that the initial choice we made, that is to use a polar parametrization, is the issue. It could be, but most practical models end up using that choice so highlighting the limitations of this choice has important practical consequences.

In the rest of this section, we ignore the dependency of θ on ν, and consider the optimal parametrization in this setting to be

$$\nu(\omega_\nu^{\mathrm{opt}}) = \tanh(\omega_\nu^{\mathrm{opt}}) \tag{A.96}$$

$$\theta(\omega_\theta^{\mathrm{opt}}) = \omega_\theta^{\mathrm{opt}}\frac{1-\nu^2}{\nu\sqrt{1+\nu^2}}. \tag{A.97}$$

A.3.1.3 *Visualization of the effect of input normalization and reparametrization*

We now visualize the effect of input normalization and reparametrization on the loss landscape. We focus on two such reparametrizations:

- the one used in the LRU [69, 71] with $\gamma(\lambda) = \sqrt{1-|\lambda|^2}$, $\nu = \exp(-\exp(\omega_\nu^{\mathrm{exp}}))$ and $\theta = \exp(\omega_\theta^{\mathrm{exp}})$.
- the optimal one we derived in the previous Section (c.f. Equations A.96 and A.97), which is tailored to this specific setting.

A.3.1.4 *Learning the angle is difficult in practice: an example*

We use this one-dimensional teacher-student setting to test whether having a parametrization that avoids exploding behaviors at optimality, such as the one we derived in Section A.3.1.2, facilitates learning. Figure A.1 already hints towards the fact the basin of attraction of the global minima is either extremely narrow or that their number decreases as longer memories are considered, making learning more tedious. Figure A.2 confirms it. In this figure, we plot the learning dynamics obtained using the Adam optimizer with a learning rate of 10^{-3} for 50k steps, starting from $\lambda_0 = 0.99\exp(i\pi/4)$. We consider three different parametrizations of the angle:

$$\theta(\omega_\theta^{\mathrm{polar}}) = \omega_\theta^{\mathrm{polar}} \tag{A.98}$$

$$\theta(\omega_\theta^{\mathrm{exp}}) = \log(\omega_\theta^{\mathrm{exp}}) \tag{A.99}$$

$$\theta(\omega_\theta^{\mathrm{opt}}) = \frac{(1-\nu^2)}{\nu\sqrt{1+\nu^2}}\omega_\theta. \tag{A.100}$$

The first one does not reparametrize the angle, the second one is the one used in the LRU and the third one is the optimal one we derived above. We use $\nu = \tanh(\omega_\nu)$ to parametrize the magnitude in the three cases. We set λ^* to $\lambda^* = 0.99\exp(i\pi/100)$. The θ landscape when ν is correct therefore corresponds to the ones plotted in the last two columns of Figure A.1. This example shows that efforts to reduce the sharpness of the loss at optimality, as done in the last parametrization, inevitably make the loss flatter elsewhere and optimization impossible.

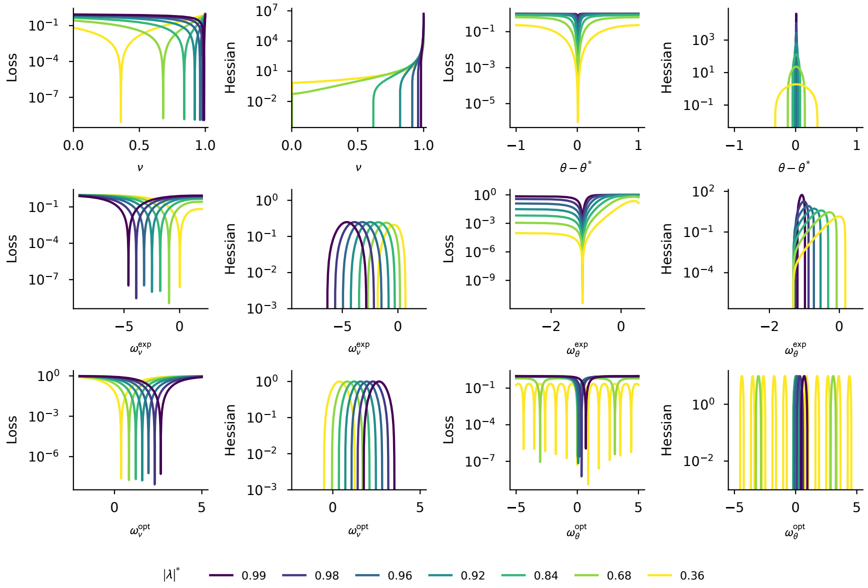


Figure A.1: Visualization of the loss landscape with input normalization, in the teacher and the student, for different parametrizations. The teacher satisfies $\lambda^* = |\lambda^*| \exp(i\pi/100)$, for different $|\lambda^*|$ values. The first two columns correspond to students with correct angle $\theta = \theta^*$ but wrong absolute value ν and the last two columns to students with correct absolute value $\nu = |\lambda^*|$ but wrong angle. When we fix one variable, we ignore how it affects the loss for the Hessian caclulation. Each line corresponds to a different parametrization: the first line uses a polar parametrization ($\lambda = \nu \exp(i\theta)$), the second line uses the double exponential parametrization used in the LRU (exp) and the third one is the optimal parametrization for that task (tanh). Overall, both reparametrizations enable to control the explosion of the Hessian. However, the size of basins of attraction around optimality, or their number, shrinks as $|\lambda^*|$ goes to 1 for the angle, but not for the absolute value, highlighting how difficult learning the angle can be.

A.3.2 *Structure of the Hessian at optimality*

In Section 3.5, we argue that the Hessian at optimality is an important object to understand the learning dynamics in the linear teacher-student task we consider. We here provide some theoretical analysis of its structure

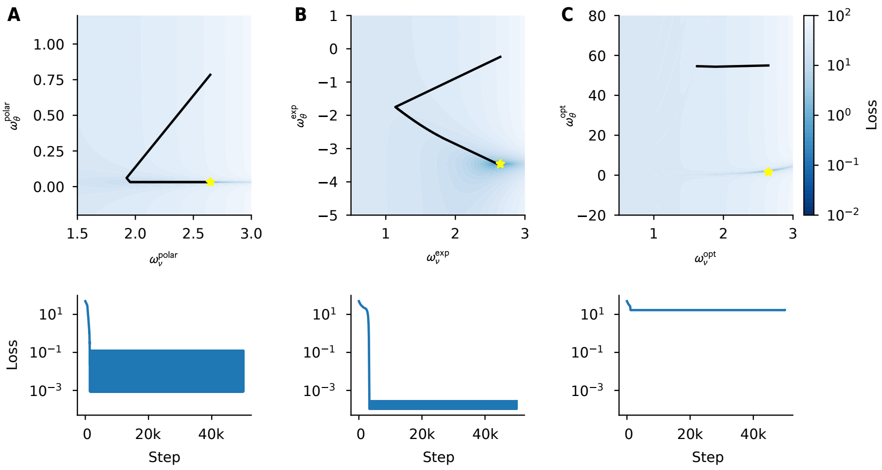


FIGURE A.2: LEARNING THE ANGLE IS DIFFICULT, EVEN IN A SIMPLE ONE-DIMENSIONAL TASK. The target λ value is equal to $\lambda^* = 0.99 \exp(i\pi/100)$ and is plotted in yellow. The black lines correspond to the Adam learning dynamics. **A.** When the angle is not reparametrized ($\theta = \omega_\theta$), the loss landscape is extremely sharp in the ω_θ direction, but Adam compensates for it. **B.** When the angle is parametrized exponentially ($\theta = \exp(\omega_\theta)$), the loss landscape becomes smoother. However, this only hold when the considered angles are small enough, as the exponential parametrization does not bring extra granularity elsewhere. **C.** When reparametrizing the angle to reduce the gradient explosion as $|\lambda| \to 1$, the loss becomes extremely tricky to navigate. The parameters are first attracted to a nearby valley, which is flat on the ω_θ direction and only indirectly connected to the global minimum. Such a reparametrization thus hinders optimization far away from optimality. See Section A.3.1.4 for more detail.

in the complex diagonal setting, that is we consider a recurrent network of the form

$$h_{t+1} = \lambda \odot h_t + b x_t \tag{A.101}$$

$$y_t = \mathrm{Re}[c^\top h_t] + d x_t. \tag{A.102}$$

with λ, b and c complex vectors of size n, with n the number of hidden neurons, and d a scalar. We additionally take the loss to be the mean-square error, which is also the one we use in our numerical experiments. Note that, as in our theoretical analysis of Section 3.3, we consider infinitely long sequences and wide-sense stationary inputs.

Recall that the Hessian of the loss is equal to

$$\frac{\mathrm{d}^2 L}{\mathrm{d}\theta^2} = \sum_t \mathbb{E}_x \left[\frac{\mathrm{d}h_t}{\mathrm{d}\theta} \frac{\partial^2 L_t}{\partial h_t^2} \frac{\mathrm{d}h_t}{\mathrm{d}\theta}^\top + \frac{\partial L_t}{\partial h_t} \frac{\mathrm{d}^2 h_t}{\mathrm{d}\theta^2} \right]. \tag{A.103}$$

At optimality, only the first term remains, as $\partial_{h_t} L_t$ is 0 for all data points. Given that we have shown earlier, e.g. in Section A.2.2, that the most sensitive parameters to learn are the recurrent ones λ, we focus on the Hessian with respect to these parameters in the following.

A.3.2.1 *Hessian for complex-valued variables*

Before delving into more specific calculations, we make a few remarks on how to deal the Hessian when having complex-valued parameters. We will mostly leverage the fact that the loss L is real-valued.

Before that, we recall a few facts about Wirtinger derivatives:

- For $f(z)$ a complex-valued function of z, the Wirtinger derivatives are defined as:

$$\frac{\mathrm{d}f}{\mathrm{d}z} = \frac{1}{2}\left(\frac{\mathrm{d}\mathrm{Re}[f]}{\mathrm{d}\mathrm{Re}[z]} - i\frac{\mathrm{d}\mathrm{Im}[f]}{\mathrm{d}\mathrm{Im}[z]} \right) \tag{A.104}$$

$$\frac{\mathrm{d}f}{\mathrm{d}\bar{z}} = \frac{1}{2}\left(\frac{\mathrm{d}\mathrm{Re}[f]}{\mathrm{d}\mathrm{Re}[z]} + i\frac{\mathrm{d}\mathrm{Im}[f]}{\mathrm{d}\mathrm{Im}[z]} \right). \tag{A.105}$$

- We have

$$\frac{\mathrm{d}f}{\mathrm{d}z} = \overline{\frac{\mathrm{d}f}{\mathrm{d}\bar{z}}}. \tag{A.106}$$

- Leveraging the fact that L is real-valued so that $\bar{L} = L$, we have

$$\frac{\mathrm{d}^2 L}{\mathrm{d}\lambda^2} = \frac{\mathrm{d}}{\mathrm{d}\lambda}\left[\frac{\mathrm{d}L}{\mathrm{d}\lambda}^\top \right] \tag{A.107}$$

$$= \frac{\mathrm{d}}{\mathrm{d}\lambda}\left[\overline{\frac{\mathrm{d}L}{\mathrm{d}\bar{\lambda}}^\top} \right] \tag{A.108}$$

$$= \overline{\frac{\mathrm{d}^2 L}{\mathrm{d}\bar{\lambda}^2}} \tag{A.109}$$

and, similarly, $\mathrm{d}_\lambda \mathrm{d}_{\bar{\lambda}} L = \overline{\mathrm{d}_{\bar{\lambda}} \mathrm{d}_\lambda L}$. Second derivatives are symmetric, so we additionally have $\mathrm{d}_\lambda \mathrm{d}_{\bar{\lambda}} L = \mathrm{d}_{\bar{\lambda}} \mathrm{d}_\lambda L^\top$, which means that the complex Hessian is a Hermitian matrix.

Taken all together, this shows that the full complex Hessian, which contains all cross derivatives, has a similar structure to the real case.

A.3.2.2 *Hessian with respect to the recurrent eigenvalues*

In this section, we compute the full complex Hessian with respect to the recurrent eigenvalue λ and defer the analysis of reparametrization to the next section.

First, let us remark that

$$\frac{\mathrm{d}L_t}{\mathrm{d}\lambda} = \frac{\partial L_t}{\partial y_t} c^\top \frac{\mathrm{d}h_t}{\mathrm{d}\lambda} \tag{A.110}$$

$$\tag{A.111}$$

so that

$$\frac{\mathrm{d}^2 L_t}{\mathrm{d}\lambda^2} = \frac{\mathrm{d}}{\mathrm{d}\lambda}\left[\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}^\top c \frac{\partial L_t}{\partial y_t}^\top\right] \tag{A.112}$$

$$= \frac{\mathrm{d}^2 h_t}{\mathrm{d}\lambda^2} c \frac{\partial L_t}{\partial y_t}^\top + \frac{\mathrm{d}h_t}{\mathrm{d}\lambda}^\top c \frac{\partial^2 L_t}{\partial y_t^2} c^\top \frac{\mathrm{d}h_t}{\mathrm{d}\lambda} \tag{A.113}$$

We assumed that we are at optimality so that the network perfectly fits the target trajectories and $\partial_{y_t} L_t = 0$. Additionally, L_t is the mean-squared error loss so that $\partial^2_{y_t} L_t = \mathrm{Id}$. It follows that

$$\left(\frac{\mathrm{d}^2 L_t}{\mathrm{d}\lambda^2}\right)_{ij} = \left(\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}^\top c c^\top \frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right)_{ij} \tag{A.114}$$

$$= \left(c^\top \frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right)_i \left(c^\top \frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\right)_j \tag{A.115}$$

$$= c_i \frac{\mathrm{d}h_{t,i}}{\mathrm{d}\lambda_i} c_j \frac{\mathrm{d}h_{t,j}}{\mathrm{d}\lambda_j}. \tag{A.116}$$

In the last equation, we made use of the fact that the parameter λ_i only affects the hidden state $h_{t,i}$ and not the others, so $\mathrm{d}_{\lambda_j} h_{t,i} = 0$ if $i \neq j$.

The previous calculation applied to one sequence, we now take the expectation over the data:

$$\frac{\mathrm{d}^2 L}{\mathrm{d}\lambda^2} = (cc^\top) \odot \mathbb{E}_{x,y}\left[\lim_{t\to\infty} \frac{\mathrm{d}h_t}{\mathrm{d}\lambda}\frac{\mathrm{d}h_t}{\mathrm{d}\lambda}^\top\right] \tag{A.117}$$

Note that we introduced a slight abuse of notation in the previous equation as $\mathrm{d}_\lambda h_t$ is in general a matrix. However, given that the hidden neurons are independent here due to the diagonal connectivity, it is effectively a vector, and we treat it that way. Let us now compute the expectation, using similar calculation techniques to the one we used in Section A.2.2:

$$\mathbb{E}_{x,y}\left[\lim_{t\to\infty} \frac{\mathrm{d}h_{t,i}}{\mathrm{d}\lambda_i}\frac{\mathrm{d}h_{t,j}}{\mathrm{d}\lambda_j}\right] = \mathbb{E}\left[\sum_{n,m\geq 0} n\lambda_i^{n-1} b_i x_{-n} m\lambda_j^{m-1} b_j x_{-m}\right] \tag{A.118}$$

$$= b_i b_j \mathbb{E}\left[\sum_{n,m\geq 0} n\lambda_i^{n-1} x_{-n} m\lambda_j^{m-1} x_{-m}\right] \tag{A.119}$$

$$= b_i b_j \sum_{n,m\geq 0} nm\lambda_i^{n-1}\lambda_j^{m-1} R_x(n-m) \tag{A.120}$$

We can now remark that this quantity is very similar to the one we have encountered in Section A.2.2, up to the presence of $b_i b_j$, and can be simplified using Lemma 2. For conciseness, we note $S(\lambda_i, \lambda_j)$ the right-hand side of the last equation without the $b_i b_j$ factor. Putting this result back in the Hessian, we get

$$\frac{\mathrm{d}^2 L}{\mathrm{d}\lambda_i \mathrm{d}\lambda_j} = b_i b_j c_i c_j S(\lambda_i, \lambda_j) \tag{A.121}$$

To gain further intuition of the behavior of this quantity, we take $R_x(\Delta) = \rho^{|\Delta|}$, ρ being a real number. A similar calculation to the one we did in Section A.2.2 gives

$$S(\lambda_i, \lambda_j) = \frac{1}{(1-\lambda_i\lambda_j)^3(1-\rho\lambda_i)^2(1-\rho\lambda_j)^2}\Big((1-\rho)(1+\lambda_i\lambda_j) + \rho^2(1-\lambda_i\lambda_j)^3 + \rho(1-\rho)\lambda_i\lambda_j(\rho\lambda_i\lambda_j(1+\lambda_i\lambda_j) - 2\lambda_i - 2\lambda_j)\Big). \tag{A.122}$$

Given the complexity of this formula, we visualize the magnitude of $S(\lambda_i, \lambda_j)$ on Figure A.3. Interestingly, we observe this quantity is large when λ_i and λ_j are conjugate to each other and inputs are uncorrelated. However, as elements in the input sequence get more correlated ($\rho \to 1$), this effect disappears and $|S|$ increases as one of the two eigenvalue gets closer to 1 in the complex plane. In both cases, the effect gets amplified as the magnitude of the eigenvalue increases.

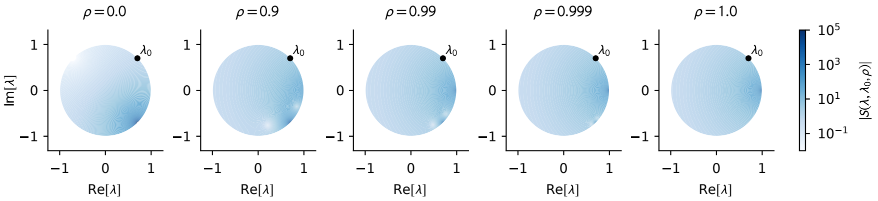


FIGURE A.3: VISUALIZATION OF $\lambda \mapsto |S(\lambda, \lambda_0)|$ FOR $\lambda_0 = 0.99 \exp(i\pi/4)$. This term measures how "similar" eigenvalues are in the Hessian. When $\rho = 0$, eigenvalues are mostly "similar" when they are conjugate to each other. As ρ increases, this effect decreases and eigenvalues become more "similar" if one of them gets close to 1.

We also need to compute $\mathrm{d}_{\bar{\lambda}}\mathrm{d}_{\lambda} L$ to get the full complex Hessian. Similarly to the previous calculation, we first have

$$\frac{\mathrm{d}L_t}{\mathrm{d}\bar{\lambda}} = \frac{\overline{\mathrm{d}\bar{L}_t}}{\mathrm{d}\lambda} = \frac{\overline{\mathrm{d}L_t}}{\mathrm{d}\lambda} = \frac{\partial L_t}{\partial y_t} \bar{c}^\top \frac{\overline{\mathrm{d}h_t}}{\mathrm{d}\lambda}. \tag{A.123}$$

It follows that

$$\begin{aligned} \frac{\mathrm{d}^2 L}{\mathrm{d}\lambda_i \mathrm{d}\bar{\lambda}_j} &= \mathbb{E}\left[\frac{\overline{\mathrm{d}h_{t,j}}}{\mathrm{d}\lambda_j} \bar{c}_j c_i \frac{\mathrm{d}h_{t,i}}{\mathrm{d}\lambda_i} \right] && \text{(A.124)} \\ &= c_i \bar{c}_j b_i \bar{b}_j S(\lambda_i, \bar{\lambda}_j). && \text{(A.125)} \end{aligned}$$

Using the symmetry with the complex Hessian matrix, we now have all its components.

A.3.2.3 *Hessian for different parametrizations*

So far, we have computed the complex Hessian, which is not of direct use as we end up optimizing real numbers in practice. Here, we study the impact of different parametrizations of λ on the Hessian. Given that this parametrization only affects λ and not the other parameters in the network and that we only consider the Hessian at optimality here, computing the Hessian of those parameters reduces to left and right multiplying the

Hessian by derivatives of λ and $\bar{\lambda}$ with respect to these parameters. For future reference, we introduce

$$H_{ij}^{\lambda} := \begin{pmatrix} \frac{\mathrm{d}^2 L}{\mathrm{d}\lambda_i \mathrm{d}\lambda_j} & \frac{\mathrm{d}^2 L}{\mathrm{d}\lambda_i \mathrm{d}\bar{\lambda}_j} \\ \frac{\mathrm{d}^2 L}{\mathrm{d}\bar{\lambda}_i \mathrm{d}\lambda_j} & \frac{\mathrm{d}^2 L}{\mathrm{d}\bar{\lambda}_i \mathrm{d}\bar{\lambda}_j} \end{pmatrix} = \begin{pmatrix} A_{ij} & B_{ij} \\ \bar{B}_{ij} & \bar{A}_{ij}. \end{pmatrix} \tag{A.126}$$

with $A_{ij} := b_i b_j c_i c_j S(\lambda_i, \lambda_j)$ and $B_{ij} = b_i \bar{b}_j c_i \bar{c}_j S(\lambda_i, \bar{\lambda}_j)$.

REAL-IMAGINARY PARAMETRIZATION: $\lambda = \omega_{\mathrm{re}} + \omega_{\mathrm{im}}$. We aim at computing the matrix

$$H_{ij}^{\mathrm{RI}} := \begin{pmatrix} \frac{\mathrm{d}^2 L}{\mathrm{d}\omega_{\mathrm{re},i} \mathrm{d}\omega_{\mathrm{re},j}} & \frac{\mathrm{d}^2 L}{\mathrm{d}\omega_{\mathrm{re},i} \mathrm{d}\omega_{\mathrm{im},j}} \\ \frac{\mathrm{d}^2 L}{\mathrm{d}\omega_{\mathrm{im},i} \mathrm{d}\omega_{\mathrm{re},j}} & \frac{\mathrm{d}^2 L}{\mathrm{d}\omega_{\mathrm{im},i} \mathrm{d}\omega_{\mathrm{im},j}} \end{pmatrix}, \tag{A.127}$$

which is the building block to compute the full Hessian. First, let us remark that $\mathrm{d}_{\omega_{\mathrm{re},i}} \lambda_i = 1/2$, $\mathrm{d}_{\omega_{\mathrm{re},i}} \bar{\lambda}_i = 1/2$, $\mathrm{d}_{\omega_{\mathrm{im},i}} \lambda_i = i/2$ and $\mathrm{d}_{\omega_{\mathrm{im},i}} \bar{\lambda}_i = -i/2$. It follows that

$$\frac{\mathrm{d}^2 L}{\mathrm{d}\omega_{\mathrm{re},i} \mathrm{d}\omega_{\mathrm{re},j}} = (\mathrm{d}_{\omega_{\mathrm{re},j}} \lambda_j \;\; \mathrm{d}_{\omega_{\mathrm{re},j}} \bar{\lambda}_j) H_{ij}^{\lambda} (\mathrm{d}_{\omega_{\mathrm{re},i}} \lambda_i \;\; \mathrm{d}_{\omega_{\mathrm{re},i}} \bar{\lambda}_i)^{\top} \tag{A.128}$$

$$= \frac{1}{4} (1 \;\; 1) H_{ij}^{\lambda} (1 \;\; 1)^{\top} \tag{A.129}$$

$$= \frac{1}{2} \left(\mathrm{Re}[A_{ij}] + \mathrm{Re}[B_{ij}] \right). \tag{A.130}$$

Once again we emphasize that the first line only holds as we are at optimality. Similar calculations give the rest of the elements of H_{ij}^{RI}:

$$H_{ij}^{\mathrm{RI}} := \frac{1}{2} \begin{pmatrix} \mathrm{Re}[A_{ij} + B_{ij}] & \mathrm{Im}[-A_{ij} + B_{ij}] \\ \mathrm{Im}[-A_{ij} - B_{ij}] & \mathrm{Re}[-A_{ij} + B_{ij}]. \end{pmatrix}. \tag{A.131}$$

Given the intuition we gained on the structure of S previously, and the fact that $A_{ij} \propto S(\lambda_i, \lambda_j)$ and $B_{ij} \propto S(\lambda_i, \bar{\lambda}_j)$, we know that this block will have large components if the two corresponding eigenvalues are conjugate of each other or aligned to each other, or if one of them is close to 1.

One other quantity that we can calculate is the trace of the Hessian H^{RI}, which is equal to the sum of its eigenvalues. Note that this does not correspond to the eigenvalues of the full Hessian matrix, as it additionally contains entries for other parameters. Yet it already provides some idea of how large the Hessian will be, as the value of this trace appears in the value of the full trace. We have

$$\begin{aligned}
\mathrm{Tr}[H^{\mathrm{RI}}] &= \sum_i \frac{1}{2}\left(\mathrm{Re}[A_{ii} + B_{ii}] + \mathrm{Re}[-A_{ii} + B_{ii}]\right) && \text{(A.132)}\\
&= \sum_i \mathrm{Re}[B_{ii}] && \text{(A.133)}\\
&= \sum_i |b_i|^2 |c_i|^2 S(\lambda_i, \bar{\lambda}_i) && \text{(A.134)}
\end{aligned}$$

where we used that $S(\lambda_i, \bar{\lambda}_i)$ is real-valued in the last line. As a side note, this formula partly justifies why studying the expected squared magnitude of $\mathrm{d}_\lambda h_t$ in Section 3.3 makes general sense, as

$$\mathbb{E}\left[\left|\frac{\mathrm{d}h_{t,i}}{\mathrm{d}\theta}\right|^2\right] = |b_i|^2 S(\lambda_i, \bar{\lambda}_i). \tag{A.135}$$

MAGNITUDE-ANGLE PARAMETRIZATION: $\lambda = \nu(\omega_\nu)\exp(i\theta(\omega_\theta))$. The calculations for this parametrization are similar to the previous one, with the following differences:

$$\begin{aligned}
\frac{\mathrm{d}\lambda}{\mathrm{d}\omega_\nu} &= \frac{\nu'(\omega_\nu)\exp(i\theta(\omega_\theta))}{2} && \text{(A.136)}\\
\frac{\mathrm{d}\bar{\lambda}}{\mathrm{d}\omega_\nu} &= \frac{\nu'(\omega_\nu)\exp(-i\theta(\omega_\theta))}{2} && \text{(A.137)}\\
\frac{\mathrm{d}\lambda}{\mathrm{d}\omega_\theta} &= \frac{i\nu(\omega_\nu)\theta'(\omega_\theta)\exp(i\theta(\omega_\theta))}{2} && \text{(A.138)}\\
\frac{\mathrm{d}\bar{\lambda}}{\mathrm{d}\omega_\theta} &= -\frac{i\nu(\omega_\nu)\theta'(\omega_\theta)\exp(-i\theta(\omega_\theta))}{2}. && \text{(A.139)}
\end{aligned}$$

After some calculations we obtain

$$\frac{\mathrm{d}^2 L}{\mathrm{d}\omega_{\nu,i}\,\mathrm{d}\omega_{\nu,j}} = \frac{\nu'(\omega_{\nu,i})\nu'(\omega_{\nu,j})}{2} \cdot \mathrm{Re}[\mathrm{e}^{i(\theta(\omega_{\theta,i})+\theta(\omega_{\theta,j})} A_{ij} + \mathrm{e}^{i(\theta(\omega_{\theta,i})-\theta(\omega_{\theta,j}))} B_{ij}] \tag{A.140}$$

$$\frac{\mathrm{d}^2 L}{\mathrm{d}\omega_{\nu,i}\,\mathrm{d}\omega_{\theta,j}} = \frac{\nu'(\omega_{\nu,i})\nu(\omega_{\nu,j})\theta'(\omega_{\theta,j})}{2} \cdot \mathrm{Im}[-\mathrm{e}^{i(\theta(\omega_{\theta,i})+\theta(\omega_{\theta,j})} A_{ij} + \mathrm{e}^{i(\theta(\omega_{\theta,i})-\theta(\omega_{\theta,j}))} B_{ij}] \tag{A.141}$$

$$\frac{\mathrm{d}^2 L}{\mathrm{d}\omega_{\theta,i}\,\mathrm{d}\omega_{\nu,j}} = \frac{\nu(\omega_{\nu,i})\theta'(\omega_{\theta,i})\nu'(\omega_{\nu,j})}{2} \cdot \mathrm{Im}[-\mathrm{e}^{i(\theta(\omega_{\theta,i})-\theta(\omega_{\theta,j})} A_{ij} + \mathrm{e}^{i(\theta(\omega_{\theta,i})-\theta(\omega_{\theta,j}))} B_{ij}] \tag{A.142}$$

$$\frac{\mathrm{d}^2 L}{\mathrm{d}\omega_{\theta,i}\,\mathrm{d}\omega_{\theta,j}} = \frac{\nu(\omega_{\theta,i})\theta'(\omega_{\theta,i})\nu(\omega_{\theta,j})\theta'(\omega_{\theta,j})}{2} \cdot \mathrm{Re}[-\mathrm{e}^{i(\theta(\omega_{\theta,i})+\theta(\omega_{\theta,j})} A_{ij} + \mathrm{e}^{i(\theta(\omega_{\theta,i})-\theta(\omega_{\theta,j}))} B_{ij}] \tag{A.143}$$

A.3.3 *Experimental details*

We use the linear RNN architecture defined in Appendix A.1.1 as teacher and implement our experiments in JAX [308], using the default Flax [309] implementation of RNNs and the LRU implementation of Zucchet, Meier & Schug [310]. Code is available here.

We initialize RNNs in the same way we initialized the teacher, and initialize the eigenvalues of the LRU and other complex-valued networks with magnitude in $[\nu_0, 1]$ and angle within $[-\theta_0, \theta_0]$.

Given that we are interested in the optimization properties of the different architectures, we only report training losses and do not perform any cross validation.

Here are additional details related to the different figures:

- **Figure 3.3**: see Tables A.1 and A.2.
- **Figure 3.4 and A.4**: for panels A and B, we use $\nu_0 = 0.99$ and draw A in a slightly different manner to the one described above (we directly draw the eigenvalues and eigenvectors so that we have two pairs of complex

eigenvalues). We use automatic differentiation to compute the Hessian. For panels C and D, we use the same setup as described in Table A.2, but keep the learning rate constant over the course of learning. We report the effective learning rate at the end of learning.

- **Figure A.5**: for panels A, B and C, we draw the magnitude and angle of 10 λ values independently, uniformly in $[\nu_0, \frac{1+\nu_0}{2}]$ and $[-\theta_0, \theta_0]$. Importantly, this means that there are no conjugate pairs, which leads to more diagonal Hessian matrices at optimality than in Figure 3.4. For panel D, see Table A.3.
- **Figure A.6**: same setup as for Figure 3.3.

As a rule of thumb, each LRU (or complex-valued diagonal network) experiment takes 3 minutes on a consumer-scale GPU (NVIDIA GeForce RTX 3070) and each RNN experiment takes 10 minutes on a CPU. The scans behind the results reported in the different figures require on the order of few hundreds run each. Including our preliminary explorations, the results we report in this section required 30 days of compute, one third of it on GPUs and two thirds on CPUs.

A.3.4 *Additional analyses*

A.3.4.1 *Structure of the loss landscape for LRUs and S4*

In the main text, we only provide an analysis of the loss landscape for the fully connected linear recurrent neural network and its complex-valued diagonal counterpart. We here complete this result by performing the same analysis for the LRU and S4. Given that S4 involves some form of parameter sharing between the magnitude and the angle of the recurrence complex eigenvalues through Δ, which is required to avoid the explosion of the angle gradients, we are particularly interested in observing whether it fully mitigates or not the gradient explosion effect. The results of Figure A.4.B and D do not reveal such benefits: the loss landscape does not have high curvature on the ω_A^{im} direction, but it is moved in the ω_Δ direction. We haven't investigated whether qualitative changes arise when changing the input distribution.

	RNN	LRU
Batch size	128	
Sequence length	300	
Hidden neurons (teacher)	10	
Input / output dimension	1	
ν_0	$\{0.32, 0.68, 0.84, 0.92, 0.96, 0.98, 0.99\}$	
θ_0	π	
Hidden neurons (student)	64	
log learning rate	$[-5, -4.5, -4, -3.5, -3, -2.5]$	$[-2.5, -2, -1.5, -1, -0.5]$
Optimizer (schedule)	Adam (cosine)	
Initialization	$[\nu_0$ teacher, $\nu_0 = 0]$	ν_0 teacher
Number iterations	10k	
Seeds	10	

Table A.1: Experimental details for Figure 3.3.A. We use $[\cdots]$ to denote hyperparameters that were scanned over with grid search and $\{\cdots\}$ to denote the variables of interest for the figure. We chose the learning rates for the two architectures on preliminary scans and verified that non of the extreme learning rates were optimal in the final scan. For the RNN, we found that initializing with $\nu_0 = 0$ gave better results than initializing with the same distribution the teacher has, so we included this choice in the scan.

	RNN / Block diag. RNN	Complex diag. RNN / LRU
Batch size	128	
Sequence length	300	
Hidden neurons (teacher)	10	
Input / output dimension	1	
ν	0.99	
θ_0	π	
Hidden neurons (student)	64 / 64 and 128	64
log learning rate	$[-5, -4.5, -4, -3.5, -3, -2.5]$	$[-2.5, -2, -1.5, -1, -0.5]$
Optimizer (schedule)	Adam (cosine)	
Initialization	$[\nu_0$ teacher, $\nu_0 = 0]$	ν_0 teacher
Number iterations	10k	
Seeds	10	

Table A.2: Experimental details for Figure 3.3.B. We use $[\cdots]$ to denote hyperparameters that were scanned over with grid search and $\{\cdots\}$ to denote the variables of interest for the figure. We chose the learning rates for the two architecture types on preliminary scans and verified that non of the extreme learning rates were optimal in the final scan. For the RNN, we found that initializing with $\nu_0 = 0$ gave better results than initializing with the same distribution the teacher has, so we included this choice in the scan. For the RNNs, we used 64 neurons for the "RNN" entry, 64 for the "block diagonal" one, and 128 for the "more neurons" one.

	RNN	Complex diag. RNN / LRU
Batch size	128	
Sequence length	300	
Hidden neurons (teacher)	10	
Input / output dimension	1	
ν_0	0.99	
$\log(\theta_0/\pi)$	$\{-2, -1.5, -1, -0.5, 0\}$	
Hidden neurons (student)	64	
log learning rate	$[-4.5, -4, -3.5, -3]$	$[-3.5, -3, \cdots, -0.5, 0]$
Optimizer (schedule)	Adam (cosine)	
Initialization	$[\nu_0$ teacher, $\nu_0 = 0]$ + θ_0 teacher	ν_0 teacher + θ_0 teacher
Number iterations	10k	
Seeds	10	

Table A.3: Experimental details for Figure A.5. We use $[\cdots]$ to denote hyperparameters that were scanned over with grid search and $\{\cdots\}$ to denote the variables of interest for the figure. We chose the learning rates for the two architectures on preliminary scans and verified that non of the extreme learning rates were optimal in the final scan. For the RNN, we found that initializing with $\nu_0 = 0$ gave better results than initializing with the same distribution the teacher has, so we included this choice in the scan.

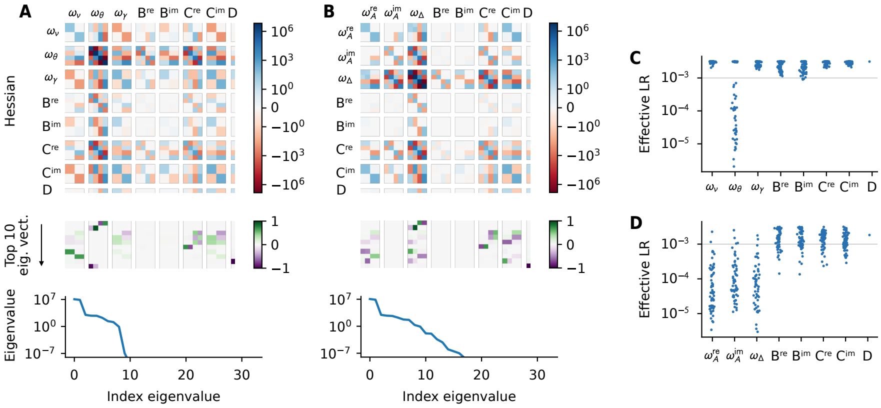


Figure A.4: Equivalent of Figure 3.4 for the LRU (A, C) and S4 (B, D). In the LRU, the exponential parametrization of the magnitude $\nu = \exp(-\exp(\omega_\nu))$ efficiently mitigates the Hessian explosion but not the one of the angle $\theta = \exp(\omega_\theta)$, consistently with the theoretical and empirical evidence we have accumulated so far. In B, Δ is set to 0.01 in the S4 architecture. Setting it to 1 (not plotted here) leads to an Hessian at optimality that has large entries on all recurrent parameters ω_A^{re}, ω_A^{im} and ω_Δ, similarly to the behavior we observed for the complex diagonal RNN studied in the main text. For panel D, we initialized Δ at 1 given that this is the initialization that yielded best performance. We note that we didn't find qualitative changes in this plot when changing the initialization scheme of S4.

A.3.4.2 *Concentrating eigenvalue distributions*

The goal of this experiment is to better understand how the concentration of eigenvalues λ affect the learning dynamics. For fully connected RNNs, there is no reason to expect a major change in behavior. However, it is different for diagonal RNNs. The theoretical analysis we have done in Section A.3.2 provides us with the following insights. When the elements in the input sequence are uncorrelated, as it is the case here, the entries in the Hessian corresponding to two different eigenvalues increase if they are aligned or conjugate to each other, and if their magnitude is large. We therefore expect that, as the interval on which the angle of the teacher's eigenvalues shrinks ($\theta_0 \to 0$), those eigenvalues will be more likely to be "similar" to each other. This results in large non-diagonal terms, as we confirm in Figure A.5.A, B and C. The LRU suffers less from this problem thanks to its reparametrization, which reduces the overall magnitude of Hessian entries related to the magnitude, and partly the one of angle parameters (when it is a small positive number). As a consequence, the performance between these two architectures increases as $\theta_0 \to 0$, as seen on Figure A.5.D.

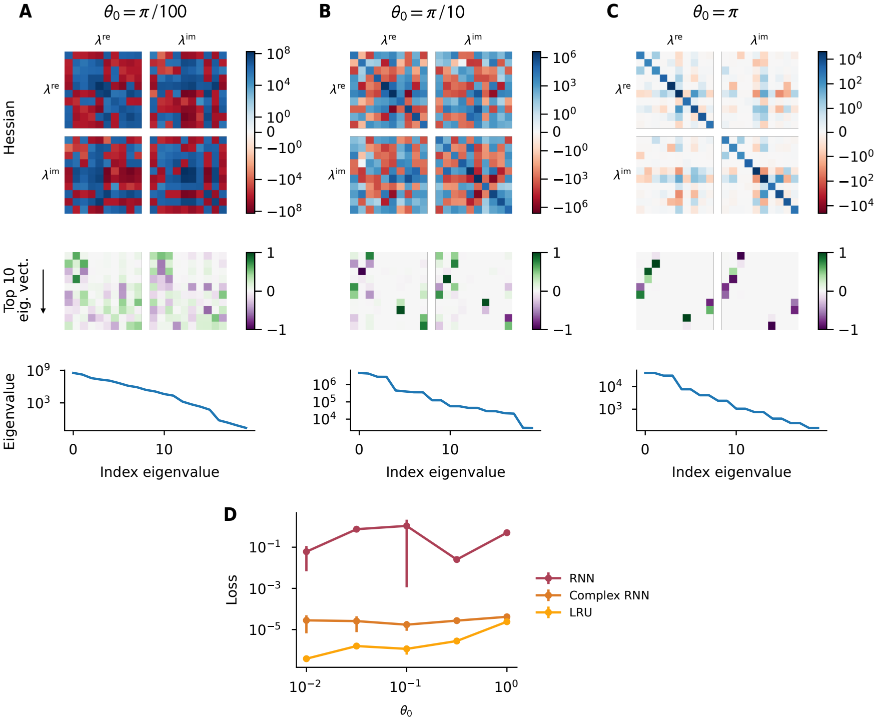


Figure A.5: Concentrating eigenvalues make the Hessian less diagonal ($\theta_0 \to 0$) and consequently increases the gap between the LRU and the complex-valued diagonal RNN. **A, B, C.** Hessian of the loss with respect to the λ parameters in the complex-valued diagonal RNN. The Hessian is computed through the theoretical formula of Equation A.131; computing it numerically marginally affects the results. Consistently with the intuition we developed in Section A.3.2, concentrating the eigenvalues affect the structure of the loss landscape. It makes the Hessian at optimality less diagonal and Adam cannot efficiently compensate it. The LRU does not suffer as much from this problem, and the gap between the two architecture widens as $\theta_0 \to 0$.

A.3.4.3 *Impact of the number of heads in fully connected linear recurrent networks*

In Figure 3.3, we have shown that constraining the connectivity matrix to be block diagonal with blocks of size 2×2 lead to a critical boost in performance. Further analysis revealed that this arises as the Hessian becomes more diagonal and Adam can thus better compensate for gradients explosion. Here, we study this behavior in more detail by interpolating between the fully connected case and the block diagonal one. This can be achieved by increasing the number of independent heads from 1 (fully connected case) to 32 (2×2 block-diagonal connectivity matrix, as we have 64 hidden neurons). In particular, we are interested in understanding how big heads can be while keeping this performance boost. We plot the final performance, as well as the evolution of the effective learning rate for the A matrix over the course of learning on Figure A.6. We find that slightly bigger heads, until 4×4 (which corresponds to 16 heads), yield similar benefits. Additionally, the learning rate analysis reveals that the adaptive learning rates of Adam can more selectively compensate for potential gradient explosion cases as the number of heads increases, allowing for bigger learning rates overall and better performance.

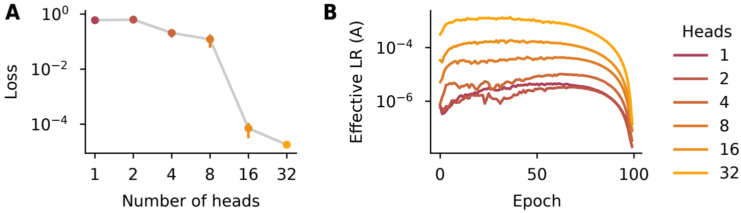


FIGURE A.6: Evolution of the performance (**A**) and effective learning rates for the A connectivity matrix (**B**) of a linear recurrent neural network as we vary the number of heads, keeping the overall number of hidden neurons fixed. It should be noted that increasing the number of heads decrease the total number of parameters, as the matrix A gets sparser.

A.4 SIGNAL PROPAGATION IN RANDOMLY INITIALIZED DEEP RECURRENT NEURAL NETWORKS

A.4.1 *Experimental setup*

We detail the experimental setup used in Section 3.6. We select the first 512 tokens from 1024 random sequences in the Wikipedia dataset [311] and pass them through the BERT [22] tokenizer and embedding layer. This yields a dataset of 1024 examples with length 512 and feature dimension 724. Figure A.7 shows the autocorrelation function of these inputs, revealing that the i.i.d. assumption serves ($\rho = 0$) as a good first order approximation. This validates the relevance of our toy experiments for studying signal propagation in more realistic settings. To refine this approximation, we can include a high correlation term (ρ close to 1).

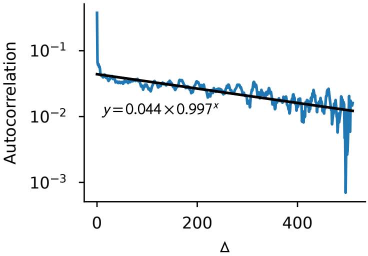


FIGURE A.7: THE EMPIRICAL AUTOCORRELATION FUNCTION (AVERAGED OVER FEATURE DIMENSIONS) OF THE BERT EMBEDDINGS USED IN SECTION 3.6 CAN BE APPROXIMATED AS A SUM OF TWO EXPONENTIALLY DECAYING FUNCTIONS. The blue line represents the autocorrelation function $R_x^{\text{empirical}}(\Delta)$ of the BERT embeddings of the Wikipedia dataset. As a first approximation, it is equal to $R_x^{\text{empirical}}(\Delta) \approx 0.376\delta_{\Delta=0}$. For a more refined approximation, we perform a linear regression of the log autocorrelation against Δ, shown by the black line. This yields the following approximation: $R_x^{\text{empirical}}(\Delta) \approx 0.332\delta_{\Delta=0} + 0.044 \times 0.997^{\Delta}$.

We examine realistic networks comprising 4 blocks with the following structure: a recurrent layer, a non-linearity, a gated linear unit [204, GLU] and a skip connection. By default, we omit normalization layers, but when included, as in Figure 3.5.C, we place one normalization layer before the recurrent layer and another one before the GLU. All the layers involved contain 256 neurons. We also incorporate a linear encoder at the beginning of the network and a linear decoder at its end.

The loss that we use is a next-token mean-squared error, defined as

$$L_t = \frac{1}{2} \| \hat{x}_t(x_{1:t-1}) - x_t \|^2 \tag{A.144}$$

where $\hat{x}_t(x_{1:t-1})$ represents the prediction of the network. Figure 3.5 reports the average squared value of the hidden state or the gradient. This average is computed over sequences, but also over all neurons / parameters and over all time steps. We compute gradients using batches of size 8.

In Figure 3.5 we vary ν_0, which controls the magnitude of the eigenvalues of the recurrent Jacobian. Specifically, we sample those magnitudes in the interval $[\nu_0, (1+\nu_0)/2]$. For the complex-valued diagonal RNN and the LRU, we apply the LRU initialization. For the LSTM, we use the Chrono initialization proposed by Tallec & Ollivier [184]: it initializes the forget gate biases such that, when the input x and the hidden state h are equal to 0, the time constant associated to f is uniformly sampled from $[\frac{1}{1-\nu_0}, \frac{2}{1-\nu_0}]$ and the input gate i is equal to $1 - f$.

While Figure 3.5.B presents aggregated gradients over layers, Figure A.8 offers a layer-wise version of this analysis. It reveals that the layer index does not significantly impact gradient magnitude. Surpisingly, given that the hidden states of the complex RNN gets larger with depth (c.f. Figure 3.5.A), this result might seem unexpected for cRNNs. We can attribute this to the backpropagated error signals also being amplified during the backward pass, as discussed in Section 3.3.2. In the first layers, hidden states are small and errors are large, while in the in last layers, errors are small and hidden states are large. Consequently, the gradient magnitude remains relatively constant accross layers. For the GRU, the gradient magnitude reported in Figure 3.5 for the non-GRU parameters included the linear encoder and decoder. As the encoder gradients are the dominating ones, this explains why the gradient magnitude for the non-GRU parameters is smaller in the layer-wise analysis.

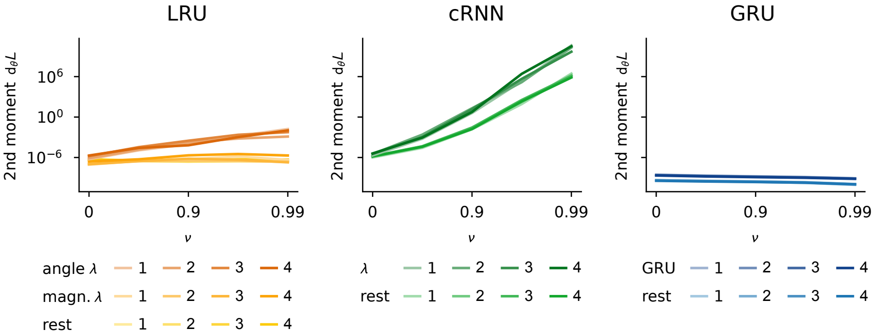


Figure A.8: Gradient magnitudes are independent of the layer. This figure presents similar plots to Figure 3.5.B, except that parameters from different layers are no longer aggregated together. Instead, each parameter group of each layer has its own line. The indices in the legend correspond to layer indices.

A.4.2 *Can gated RNNs be considered diagonal?*

In Section 3.4.2, we argued that the diagonal linear setting we focused our theory on can be a decent proxy for more general gated RNNs, whose λ values can depend on both the inputs x and the hidden state h. Here, we assess whether this holds true at initialization. To that end, we study how the Jacobian $\frac{\mathrm{d}h_{t+\Delta}}{\mathrm{d}h_t}$ of a GRU evolves as Δ grows. For the linear diagonal regime to be a good proxy, two necessary conditions must be met: First, all the non-diagonal terms should be negligible compared to the diagonal ones. Second, the diagonal terms should decay similarly as if their λ values were independent of x and h.

In Figure A.9, we report the evolution of all the diagonal terms of this Jacobian and a random subset of the non-diagonal ones. Our findigns indicate that the non-diagonal terms are much smaller than the diagonal ones under the standard initialization, supporting the first necessary condition. Additionally, input and hidden state-dependent gates do not qualitatively change the decay of the diagonal terms, particularly when the network is initialized with λ values close to 1 (which correspond to large time constants). Furthermore, we find that increasing the strength of all the hidden state to gate connections (W_{hn}, W_{hr}, W_{hz}) breaks the diagonal-like behavior and eventually leads to exploding Jacobians. However, this only occurs at values that are much larger (3 times more in this case) than the default initialization of these weights (orthogonal initialization).

In conclusion, those results confirm that the theoretical setting we have considered in this paper is a good proxy for studying signal propagation in realistic recurrent networks. While we have focused our analysis on GRUs, we expect these results to hold for other architectures such as LSTMs, Mamba, or Hawk. For the last two, given that the different gates only depend on the input, we expect the matching with our theory to be even stronger (c.f. Figure A.9.C $\sigma_h = 0$, which captures this regime).

A.4.3 *Does our theory apply to gated RNNs?*

Having established that gated RNNs behave similarly to the linear diagonal RNNs considered in our theoretical investigation, a question arises: How well can our theory describe signal propagation in gated RNNs on realistic data? To address this, we study a simplified version of the GRU network (similar to the one studied by Chen, Pennington & Schoenholz [198]), which incorporates a realistic gating mechanism:

$$f_{t+1} = \sigma(W_{fx}x_{t+1} + W_{fh}h_t + b_f) \tag{A.145}$$

$$h_{t+1} = f_{t+1} \odot h_t + (1 - f_{t+1}) \odot x_{t+1}. \tag{A.146}$$

As in the rest of this section, we provide BERT embeddings of sentences from the Wikipedia dataset as inputs.

To apply our theory to this architecture, we must address two main challenges:

1. The gate f_{t+1} depends on both h_{t+1} and x_t, making it non-constant. Based on our empirical results from the previous section, we reasonably approximate $f_{t+1} \approx \sigma(b_f) =: \lambda$, ignoring this dependency.

2. Our theoretical derivations have overlooked cases where the recurrence strength λ normalizes the input x_t. When considered, we detached the normalization factor from the computational graph (as in

Section A.2.3.2). However, we can extend our calculations from Section A.2.2 to accommodate this setting:

$$f(\alpha, \beta) := \frac{(1-\alpha)(1-\beta)}{1-\alpha\beta} \left(R_x(0) + \sum_{\Delta \geq 1} (\alpha^\Delta + \beta^\Delta) R_x(\Delta) \right) \tag{A.147}$$

$$\mathbb{E}[h_t^2] = f(\lambda, \lambda) \tag{A.148}$$

$$\mathbb{E}\left[\left(\frac{\partial h_t}{\partial \lambda} \right)^2 \right] = \left. \frac{\partial^2 f(\alpha, \beta)}{\partial \alpha \partial \beta} \right|_{\alpha=\lambda, \beta=\lambda} . \tag{A.149}$$

Note that we obtain $\mathbb{E}[(\frac{\partial^2 h_t}{\partial {b_f}^2})^2]$ by multiplying $\mathbb{E}[(\frac{\partial h_t}{\partial \lambda})^2]$ by $\sigma'(b_f)^2$.

With these adjustments in place, we can now empirically test our theoretical predictions. For simplicity, we approximate the auto-correlation function R_x (blue line in Figure A.7) as $R_x(\Delta) \approx 0.332 \delta_{\Delta=0} + 0.044 \times 0.997^\Delta$. Figure A.10 presents our results, which reveal:

- An almost perfect match between theory and practice for constant gates, confirming that our sample size is large enough.
- A very precise, though not perfect, match for context-dependent gates.
- The variance of h_t and $\frac{\partial h_t}{\partial b_f}$ shows minimal dependence on λ, indicating that the magnitude of error signals received by b_f, W_{fh}, and W_{fx} are largely independent of the time constants encoded in the network.

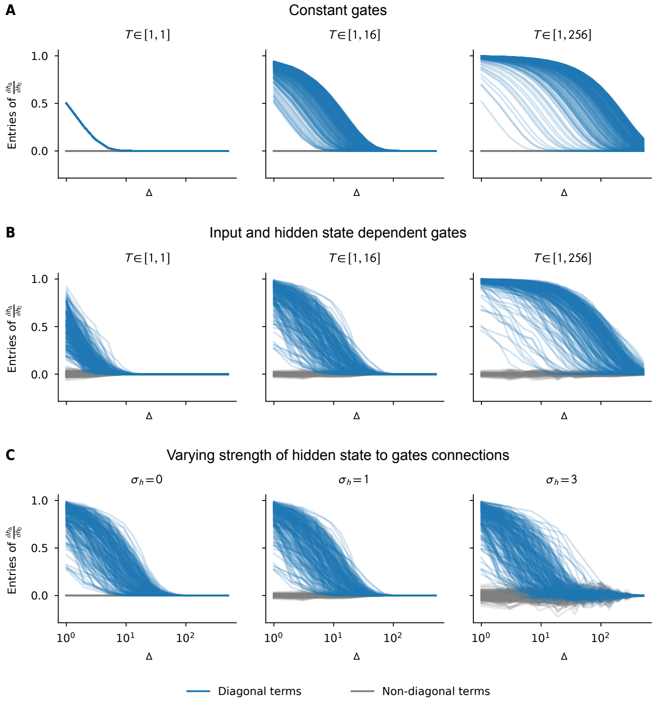


FIGURE A.9: GRUS BEHAVE LIKE DIAGONAL LINEAR NETWORKS. This figure illustrates the evolution of the recurrent Jacobian $\frac{\mathrm{d}h_\Delta}{\mathrm{d}h_0}$ of a GRU, when provided with the BERT embeddings of a sentence extracted from the Wikipedia dataset. **A.** In the first row, we take the forget gates to be independent of x and h and we sample them with the Chrono initialization [184] for different intervals. The resulting network is linear and diagonal, similar to what we have studied in the theory. These plots therefore serve as visual reference for comparison with the realistic case. **B.** The second row shows the same plots as the previous row, except that the gates are now dependent on the inputs x and on the hidden states h. As mentioned in A.1.4, we initialize all the linear layers taking x as input with LeCun initialization and the layers taking h as input with orthogonal initialization. The recurrent Jacobian evolves similarly than in the constant case, particularly on slowly decaying dimensions (large T values). **C.** In the row, we aim to break the diagonality of the model by increasing the strength σ_h of the hidden state to gate connections, for $T \in [1, 16]$. The case $\sigma_h = 0$ corresponds to gates that only depend on x, similar to architectures like Mamba or Hawk. The plot with $\sigma_h = 1$ is the same as the middle one in B. For $\sigma_h = 3$ and higher, the diagonality progressively breaks and the recurrent Jacobian eventually explodes. We note that these plots were obtained from a single example. Yet, we have found the behavior we report here to be typical of the general behavior.

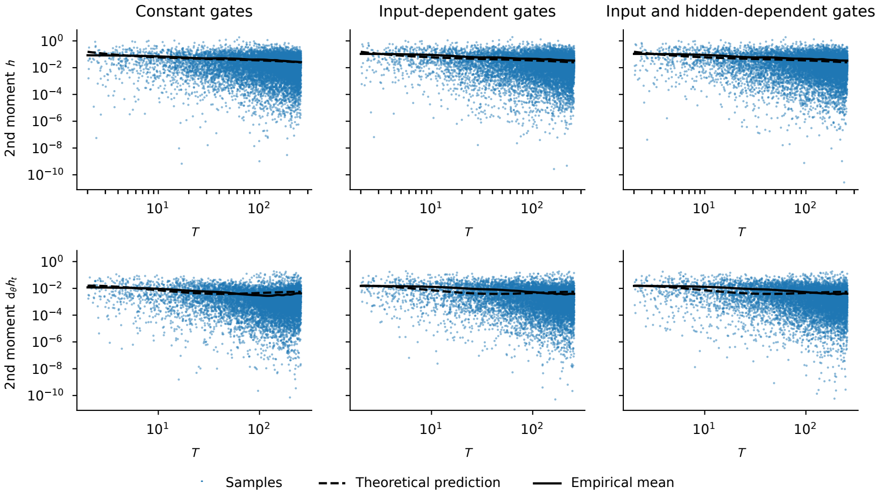


Figure A.10: The theory developed for linear diagonal recurrent networks captures signal propagation within gated recurrent neural networks. The different samples were obtained as follows: 100 different randomly initialized networks are given a different input sequence of length 512. The biases of the forget gates b_f are initialized with Chrono initialization for $T \in [1, 256]$. For each of these models / sequences, we measure $h_{512,i}^2$ and $\mathrm{d}_{b_{f,i}} h_{512,i}^2$ (i being the index of one of the 256 hidden neurons). We report this measurement as a function of the time constant T encoded by the neuron ($\lambda = T/1+T$). The empirical mean is obtained with a kernel regression with the Gaussian kernel $\mathcal{K}(a,b) := \exp(-(a-b)^2/100)$. The theoretical prediction comes from the approach described in Section A.4.3.

B

ONLINE LEARNING OF LONG-RANGE DEPENDENCIES

B.1 NETWORK ARCHITECTURE AND ALGORITHM

B.1.1 *The linear recurrent unit (LRU)*

In Section 4.2.2, we used a simplified version of the LRU for brevity. Here, we review the more detailed formulation of Orvieto *et al.* [69]. The LRU is defined as follows:

$$h_{t+1} = \lambda \odot h_t + \gamma \odot B x_t \tag{B.1}$$

$$y_t = \mathrm{Re}[C h_t] + D x_t. \tag{B.2}$$

The only difference with Equation 4.1 is the inclusion of the normalization factor $\gamma \in \mathbb{R}^N$. Its purpose is to ensure that all units maintain a comparable magnitude. Without normalization, hidden states whose λ is close to 1 can blow up. We initialize γ with

$$\gamma = \sqrt{1 - |\lambda|^2}, \tag{B.3}$$

and later update it with our learning algorithm.

We use an exponential parametrization for λ:

$$\lambda := \exp(-\exp(\nu^{\log}) + i \exp(\theta^{\log})). \tag{B.4}$$

This parametrization ensures that the norm of λ, equal to $\exp(-\exp(\nu^{\log}))$, remains below 1, guaranteeing stability of the network dynamics. This feature provides an advantage to the LRU compared to the linear RNN, as observed in Section 4.4.2. By representing θ as $\exp(\theta^{\log})$, we achieve finer tuning of θ around 0. In the following, we focus on computing gradients with respect to λ. However, in our simulations, we derive the update for $\nu^{\log}$ and $\theta^{\log}$ from these gradients through the chain rule and apply them to update the corresponding parameters. Similarly, we optimize the logarithm of γ.

B.1.2 *Complete derivation of our algorithm*

We provide a more detailed derivation of our online learning rule for a single layer of LRUs, as described in Section 4.3.1. Recall that we define s_t to be the non-zero terms of the sensitivity $\mathrm{d}_\theta h_t$ of the state h_t with respect to the parameters θ, with $\theta = \{\lambda, \gamma, B\}$ the parameters of the LRU impacting the recurrent dynamics. We show that the sensitivities evolve according to

$$s_{t+1}^{\lambda} = \lambda \odot s_t^{\lambda} + h_t \tag{B.5}$$

$$s_{t+1}^{\gamma} = \lambda \odot s_t^{\gamma} + Bx_{t+1} \tag{B.6}$$

$$s_{t+1}^{B} = \mathrm{diag}(\lambda) s_t^{B} + \gamma x_{t+1}^{\top} \tag{B.7}$$

and that gradient-following parameter updates can be calculated through

$$\Delta\lambda \propto \sum_{t=1}^{T} \delta_t \odot s_t^{\lambda} \tag{B.8}$$

$$\Delta\gamma \propto \sum_{t=1}^{T} \mathrm{Re}[\delta_t \odot s_t^{\gamma}] \tag{B.9}$$

$$\Delta B \propto \sum_{t=1}^{T} \mathrm{diag}(\delta_t) s_t^{B}, \tag{B.10}$$

with $\delta_t = \mathrm{d}_{h_t} L_t$. These updates are the ones used in our simulations. We now derive the updates per coordinate, and the vectorized form can be straightforwardly obtained from there.

Update for λ. We have

$$s_{t+1,i}^{\lambda} = \frac{\mathrm{d}h_{t+1,i}}{\mathrm{d}\lambda_i} = \frac{\mathrm{d}}{\mathrm{d}\lambda_i}\left[\lambda_i h_{t,i}\right] + 0 = \lambda_i s_{t,i}^{\lambda} + h_{t,i}. \tag{B.11}$$

The first equality uses the definition of s_{t+1}^{λ}, the second the recurrent update of h_t as in Equation B.1 and the fact that $\gamma \odot Bx_{t+1}$ does not depend on λ, and the third is the complex product rule. The gradient of the loss with respect to λ_i is then equal to

$$\frac{\mathrm{d}L}{\mathrm{d}\lambda_i} = \sum_{t=1}^{T} \sum_{j} \frac{\mathrm{d}L_t}{\mathrm{d}h_{t,j}} \frac{\mathrm{d}h_{t,j}}{\mathrm{d}\lambda_i} \tag{B.12}$$

$$= \sum_{t=1}^{T} \frac{\mathrm{d}L_t}{\mathrm{d}h_{t,i}} \frac{\mathrm{d}h_{t,i}}{\mathrm{d}\lambda_i} \tag{B.13}$$

$$= \sum_{t=1}^{T} \delta_{t,i} s_{t,i}^{\lambda}. \tag{B.14}$$

In the first line, we applied forward-mode complex differentiation (combined with h_t being a holomorphic function of λ) as in Equation 4.5. In the second, we used that h_i is independent of λ_j for $i \neq j$.

Update for γ. The derivation for the recurrent update of s^{γ} being very similar to the one for λ, we omit it. The fact that γ is a real variable does not change anything in this derivation. However, this makes the gradient computation slightly different. One can act as if γ was a complex variable $\gamma_i^{\mathbb{C}}$ and get

$$\frac{\mathrm{d}L}{\mathrm{d}\gamma_i^{\mathbb{C}}} = \sum_{t=1}^{T} \delta_{t,i} s_{t,i}^{\gamma}. \tag{B.15}$$

Now, looking at the real part of the previous equation, we have

$$\frac{\mathrm{d}L}{\mathrm{d}\gamma_i} = 2 \sum_{t=1}^{T} \mathrm{Re}[\delta_{t,i} s_{t,i}^{\gamma}]. \tag{B.16}$$

Note that the factor 2 will also indirectly appear in the update of λ, although it is not yet there in the gradient, as we have a real derivative here, while it is a complex one above.

Update for B. We have

$$s_{t+1,ji}^{B} = \frac{\mathrm{d}h_{t+1,j}}{\mathrm{d}B_{ji}} = \frac{\mathrm{d}}{\mathrm{d}B_{ji}} \left[\lambda_j h_{t,j} + \gamma_j B_{ji} x_{t+1,i}\right] = \lambda_j \frac{\mathrm{d}h_{t,i}}{\mathrm{d}B_{ji}} + \gamma_j x_{t+1,i} \tag{B.17}$$

and

$$\frac{\mathrm{d}L}{\mathrm{d}B_{ji}} = \sum_{t=1}^{T} \delta_{t,j} s_{t,ji}^{B}. \tag{B.18}$$

Note that the update for s^B is more general than the one of Equation 4.6 in the main text in which $\gamma = 1$.

B.2 EXPERIMENTAL DETAILS

We base our implementation on the S5 [68] code base[1]. All networks are trained with the AdamW optimizer, with a linear learning rate warm-up, followed by a one-cycle cosine learning rate decay. Following common practice, we do not apply weight decay for λ, γ, and use a smaller learning rate for those parameters (global learning rate times a learning rate factor). By default, we initialize λ uniformly at random in the complex disk and θ uniformly at random in $[0, 2\pi]$.

B.2.1 *Copy task experimental details and hyperparameters*

We report the hyperparameters we used and scanned over for the copy task in Tables B.1 and B.2. In Section 4.4.1, we take the default configuration reported in the LRU paper [69] for backpropagation-through-time and apply it to the different methods we consider. We use 25 epochs, with a linear warmup of 5 epochs. We tuned the learning rate for each method independently in the comparison of Figure 4.3.E and F and in the one of Table 4.1. For the 1-layer GRU architecture, following Menick *et al.* [158], we add an extra readout layer: a 1-hidden layer MLP with hidden dimension 1072 processes the hidden state to generate the output.

B.2.2 *Experimental details and hyperparameters for LRA experiments*

For all experiments, we first ran a manual coarse-grained hyperparameter tuning to identify the most important parameters, and then ran the grid search described in Table B.3. The final hyperparameters were each evaluated on 3 fresh seeds for the results reported in Table 4.2. The training time for our online learning rule on a single Nvidia RTX3090 GPU for sCIFAR, IMDB and ListOps was respectively 36, 10 and 40 hours.

For the comparison with linear RNNs, we kept the number N of hidden neurons fixed compared to the one used the LRU (c.f. Table B.3), but changed the model size H to 294 in order to obtain the same number of

---

[1] `https://github.com/lindermanlab/S5`

Hyperparameter	FIG.4.3.A/B	FIG.4.3.C/D	FIG.4.3.E/F
Learning rule	Ours	Ours	{Ours, Spat., Trunc., BP}
Pattern length	20	20	20
Padding	7	7	7
Training samples	20000	20000	20000
Number of layers	$\{1, 2, 3, 4\}$	4	4
Recurrent state size N	64	64	64
Model size H	128	128	128
$\lvert\lambda_{\min}\rvert$	0	$\{0, 0.5, 0.75, 0.9\}$	0
Epochs	25	25	25
Warmup	0	0	0
Batch-size	50	50	50
Base learning rate	10^{-3}	2×10^{-3}	$2^{[0,1,2,3]}10^{-3}$
Learning rate factor	0.5	0.5	0.5
Dropout probability	0.1	0.1	0.1
Weight-decay	0	0	0

TABLE B.1: Hyperparameter configurations for Section 4.4.1. We use $[\cdots]$ to denote hyperparameters that were scanned over with grid search and $\{\cdots\}$ to denote the variables of interest for the figure.

Layer	LRU	Linear RNN	GRU	GRU
Number layers	4	4	1	4
Pattern length	20	20	20	20
Padding	7	7	7	7
Training samples	20000	20000	20000	20000
GLU	Yes	Yes	No	Yes
N	64	64	134	91
H	128	146	134	91
Extra readout	No	No	1072	No
Number parameters	268,430	267,956	269,488	270,011
Batch-size	20	20	20	20
Base learning rate	$2^{[0,\cdots,5]}10^{-3}$	$2^{[0,\cdots,5]}10^{-3}$	$2^{[0,\cdots,5]}10^{-3}$	$2^{[0,\cdots,5]}10^{-3}$
Learning rate factor	1	1	1	1
Dropout probability	0.1	0.1	0.1	0.1
Weight-decay	0	0	0	0

Table B.2: Hyperparameter configurations for Section 4.4.2. We use $[\cdots]$ to denote hyperparameters that were scanned over.

Hyperparameter	CIFAR	IMDB	ListOps
Number of layers	4	4	4
Recurrent state size N	128	128	128
Model size H	256	256	256
$\|\lambda\|_{\text{min}}$	0.9	$[0.0, 0.9]$	$[0.0, 0.9]$
$\|\lambda\|_{\text{max}}$	0.999	1	1
Batch-size	100	32	32
Base learning rate	$[0.001, 0.004]$	$[0.001, 0.003]$	$[0.001, 0.003]$
Learning rate factor	0.5	0.5	0.5
Dropout probability	0.1	0	0.1
Weight-decay	$[0.1, 0.5]$	0.05	0.05
Epochs	180	40	35
Warmup	18	4	0

Table B.3: Hyperparameter configurations for sCIFAR, IMDB and ListOps experiments. We use $[\cdots]$ to denote hyperparameters that were scanned over with grid search. By default, the $\{\lambda, \gamma\}$ parameters have no weight decay and have a slower learning rate (c.f. learning rate factor).

parameters (our Linear RNN has 1,068,086 parameters vs. 1,063,178 for the LRU). We used the same hyperparameter optimization scheme.

C

HOW DO LANGUAGE MODELS LEARN FACTS?

C.1 RELATED WORK

C.1.1 *Associative memories and factual knowledge of neural networks*

Associative memories have been extensively studied in neuroscience. The foundational experimental work on conditioning by [61] sparked extensive theoretical research into the computational mechanism underlying associative memories. This led to breakthrough developments like Hebb's rule [2] and Hopfield networks [62, 312], which later proved instrumental in the emergence of deep learning.

In recent years, as language models have grown increasingly powerful, research has begun examining them through the lens of associative memories. Our work directly builds on this line of research. [313] first proposed that masked language models like BERT [22], could encode relational knowledge within their weights. This analysis was subsequently extended to autoregressive Transformer-based language models: [242] demonstrated that feed-forward layers act as associative key-value memories, [243] showed how this stored knowledge could be located and edited, [244, 245] revealed the functional mechanisms underlying memory recall, and [240] identified conditions under which these general mechanisms are developed. Beyond these mechanistic insights, associative memories have provided a framework for evaluating the knowledge capacity of language models. This includes empirical scaling laws [314] and theoretical analyses [250] which follow from a long history of capacity analyses of Hopfield networks and related models [e.g., 315].

While the above research primarily examines parametric memories stored in network weights, recent years have also seen growing interest in the in-context associative recall capabilities of sequential modeling networks.

Notable examples include the induction head in Transformers [260] and comparative studies showing that the primary difference between attention-based and attention-free language models can be attributed to their different in-context associative recall capabilities [289].

C.1.2 *Learning dynamics of neural networks*

The study of neural network learning dynamics has deep roots in early connectionist research [e.g., 316–318]. It has been particularly influential in the early days of deep learning. For example, the seminal work of [59] provided fundamental insights into the role of depth in neural network training.

The recent demonstration of in-context learning capabilities in large-scale models [23] has sparked renewed interest in understanding the underlying dynamics. A significant contribution to this understanding came from [260], who established a causal link between the development of induction heads and a phase transition that enables in-context learning. This mechanistic understanding has been complemented by research on the influence of training data distribution on learning trajectories and implemented algorithms. Notably, [273] and [290] revealed how different data distributions guide networks through distinct algorithmic solutions, either facilitating in-context learning or not. The generality of these findings was later confirmed by [254], who reproduced them using a simplified training distribution.

Parallel to these developments, [319] showed that Transformer-based language models' predictions can be effectively approximated by $N-$gram statistics, with the optimal N increasing throughout training. Our work observes similar dynamics during the transition from bigram to trigram predictions during the plateau phase. However, a key distinction lies in the level of abstraction: while [319]'s $N-$gram associations occur directly at the token level, our work observes these sequential dependencies at a higher level of abstraction.

Finally, and most related to ours, is the study of factual knowledge acquisition in large language models from Chang *et al.* [320]. Their key finding is that factual knowledge acquisition occurs through accumulating small probability increases when the model encounters the same knowledge, followed by gradual forgetting when not being exposed to it. These results are consistent with ours, although we find forgetting to be more pronounced, likely because of larger training distribution shifts. Additionally, while their findings were obtained by slightly altering the training distribution of a large language model, our work goes one step further by studying in depth the impact of training distributions on learning speed.

C.2 EXPERIMENTAL SETUP

C.2.1 *Rationale behind our design choices*

Our goal is to identify a synthetic setting, as simple as possible, where a model exhibits knowledge of its training data (that is a flexible usage of it, cf. Section 5.2.1, not mere memorization) using mechanisms similar to large language models. This allows precise control over the data distribution and increases the likelihood that our findings generalize to large language models.

We build upon the synthetic biography dataset and analysis of [240], as well as the mechanistic interpretability studies of [244] and [245]. [244] and [245] find that pretrained large language models encode individual-specific information in their residual stream as soon as the name of the individual is encountered. [240] observe such a behavior when training models on biographies whose order is permuted every new occurrence, along with having multiple biographies for "celebrities" (see their Q-probing analysis). We hypothesize that textual variation within the biography distribution is key to achieving this behavior. Therefore, our setup ensures unique biographies by introducing 25 distinct templates per attribute type (see next section) and permuting their order. Importantly, biographies are resampled for each new sequence. Figure C.1 demonstrates the critical importance of random ordering and the need for some textual diversity within individual biographies. While we have not ablated the total number of templates per attribute type, we believe that the overall textual diversity of the distribution is necessary for the learning of robust features. In our experiments, the individual's name is repeated in every sentence to facilitate knowledge development, though we believe later occurrences could be replaced with pronouns without significantly affecting results, as [240] found the benefits of full name repetition primarily in the non-permuted and not so diverse regime.

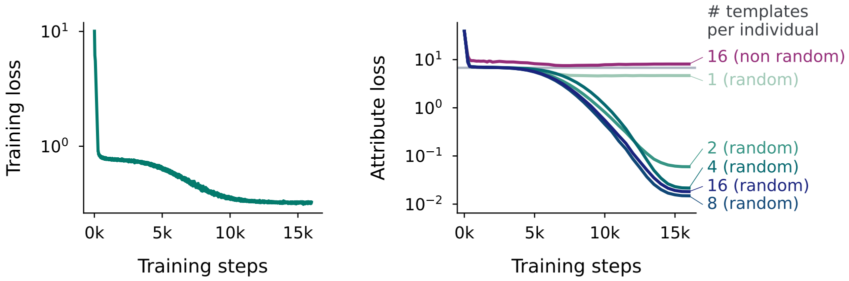


Figure C.1: (left) Training loss corresponding to the left and middle panels of Figure 5.2. (right) Ablation study demonstrating the importance of permuting the presentation order of attributes and the size of the template pool used for generating biographies. Random permutations are crucial, and some textual diversity is needed.

C.2.2 *Details about the biography generation process*

Our data generation process largely follows [240], with a few important differences regarding how templates are generated and manipulated.

Prior to training, we generate a population of individuals, each with a full name (first, middle, and last) and six attributes: birth place, birth date, university, major, company, and current location. These are generated as follows:

- Full name. First and middle names are sampled from a list of 900, and last names from a list of 1,000, resulting in 810 million potential unique names. We ensure that each individual has a unique full name. These names are among the most used worldwide.
- Birth place and current location. Sampled from a list of the 800 largest cities worldwide. Unlike [240], we do not correlate current location with the company.
- University. Sampled from the 200 largest universities worldwide.
- Major. Sampled from a list of 100 majors.
- Company. Sampled from the 500 companies with largest valuations.

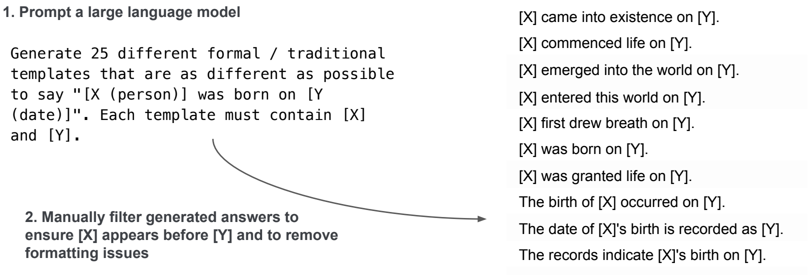
- **Birth date.** Year, month, and day are sampled independently (between 1900-2100, January-December, and 1-31, respectively). This allows unrealistic dates (e.g., February 31st), but should not significantly impact tokenization.

These values are chosen to reasonably approximate the token distribution a large language model might encounter during pre-training.

For the sake of our analysis, the template generation requires extra care, as we want all the information needed to predict the attribute value and as we want to evaluate the model on sentences it has never seen. Such considerations were not needed in Allen-Zhu & Li [240], so we had to adapt their setup. In our dataset, templates are generated prior to training with the assistance of a large language model, using the prompting scheme in Figure C.2. This yields 25 distinct templates per attribute type. As discussed in Section 5.2.2, we (manually) ensure that the individual's name and information identifying the attribute type precede the attribute value, allowing a model with perfect knowledge to achieve zero loss. For each individual, we pick 20 templates for training and keep 5 for evaluation, ensuring the model encounters novel template-individual combinations during evaluation, thus measuring knowledge rather than memorization. Additionally, we introduce special tokens for tags (like name or birth date) in the templates and replace these tags by the desired content only after tokenization. This way, we can make sure that, after this manipulation, the tokens coming from names or attribute values directly appear in the token sequence (e.g. "1990." will be tokenized as "[1990][.]" and not "[1990.]" as it would usually be). Such a precaution facilitates analysis as we can be sure that each of the name or attribute value tokens only contain this information, and not some unrelated semantic information about the sentence.

Biography generation, for both training and evaluation, is summarized in Figure 5.1 and involves two steps:

1. **Individual sampling.** By default, we sample individuals uniformly at random from the population. This distribution is modified in some experiments (Section 5.4.1) and can be time-step dependent (Section 5.2.1 and Appendix C.6.5). Here are the detail of these different distributions:
    - **Inverse power law / Zipf distribution.** The i-th individual is sampled with probability proportional to $i^{-\alpha}$ with α an hyperparameter.



1. Prompt a large language model

```
Generate 25 different formal / traditional
templates that are as different as possible
to say "[X (person)] was born on [Y
(date)]". Each template must contain [X]
and [Y].
```

2. Manually filter generated answers to ensure [X] appears before [Y] and to remove formatting issues

[X] came into existence on [Y].
[X] commenced life on [Y].
[X] emerged into the world on [Y].
[X] entered this world on [Y].
[X] first drew breath on [Y].
[X] was born on [Y].
[X] was granted life on [Y].
The birth of [X] occurred on [Y].
The date of [X]'s birth is recorded as [Y].
The records indicate [X]'s birth on [Y].

Figure C.2: Illustration of the template creation process. At the end of it, we have 25 different templates per attribute type.

Recall that we get the uniform distribution when $\alpha = 0$ and, when $\alpha = 1$, we get the Zipf distribution.

- “Celebrities” distribution. A subset of the individuals of a size `n_celebrities` is oversampled is sampled `weight_celebrities` times more frequently larger than individuals outside the group. Here, `n_celebrities` and `weight_celebrities` are the two hyperparameters we vary.

- “Warm-up” distribution. Training begins on a subset of `indiv_warmup` individuals for `epochs_warmup` epochs. An epoch is here defined as the number of training steps required to see as many biographies as there are individuals in the entire population (number of individuals divided by batch size steps). Once the warm-up is over, we train on the full population. In both phases, individuals within the relevant group are uniformly sampled.

- Sequential distribution. The population is divided into `n_groups` groups and we go through these groups in increasing order `n_repeats` times. Individuals within each group are uniformly sampled.

2. Biography sampling. For a given individual, we uniformly sample templates from the appropriate pool (training or evaluation) for each attribute. We then randomize the template order and concatenate them to form a single-sequence biography.

C.2.3 *Architecture, optimization and metrics*

In almost all our experiments, we use the 44M-parameters Transformer architecture of [248]. It has 8 layers, uses a residual stream of dimension 512, with one hidden layer multi-layer perceptrons with 2048 hidden neurons. It has 8 heads and each head has keys and values of dimension 64. The only deviation to this architecture is the model size ablation of Figure C.3 (lower right), in which we use a 163M-parameters (12 layers, 16 heads, dimension 896) and a 400M-parameters (12 layers, 12 heads, dimension 1536) architecture. The default hyperparameter configuration is detailed in Table C.1. For most experiments, we perform a sweep over different learning rate values, selecting the best performing one based on final training loss.

The experiments in Figure 5.2 use 5 different seeds. However, given the low variance of the results, we preferred to use compute to explore more diverse hyperparameter configurations rather than performing our analysis on more seeds and ended up using a single seed for the rest of the experiments.

Throughout this study, we use two main metrics. The first one is the attribute loss, which is the sum of cross-entropy losses measured on all attribute value tokens, which is then averaged by the number of attribute values in the sequence (always 6) and by the batch size. Importantly, we don't average by the number of tokens within each attribute value, so that the comparison to the no-knowledge baseline is easier. The second metric we use is the attribute accuracy, which is the accuracy the model gets when correctly predicting all consecutive attribute value tokens. For example, if the model always predicts correctly all attribute value tokens except the first one, it will get an accuracy of 0. This way, this metric is rather conservative, and that we are sure that for the model to be correct it has to get all the tokens requiring recall abilities (as opposed to completion abilities) right. The no-knowledge baseline is computed as the average entropy of all attribute values from all types, that is the average logarithm of the number of possible attribute values.

C.3 ADDITIONAL ANALYSIS OF THE LEARNING DYNAMICS (SECTION 5.3.1)

C.3.1 *The three phases are robust to sensible hyperparameter choices*

In Figure C.3, we verify whether the three phases we identify are robust to sensible parameter changes. We find that they are robust to changes in learning rates, total number of individuals, weight decay values, batch size, model size, and to changing the sequence mixing architecture to a recurrent one (we use the Hawk model for that [71]).

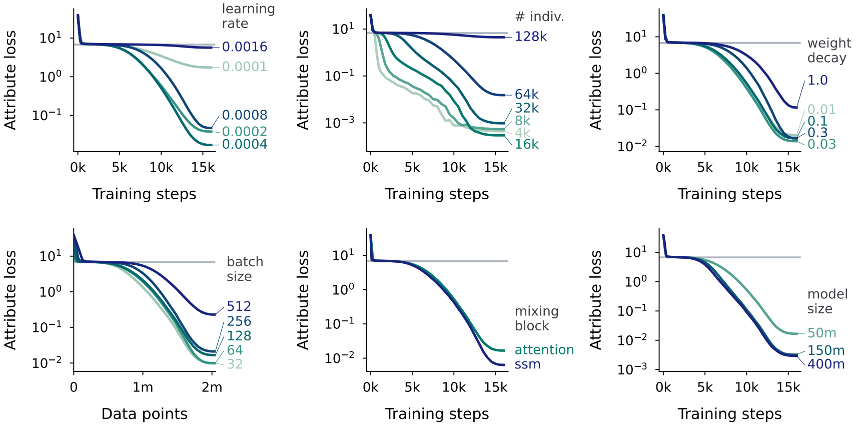


Figure C.3: Different hyperparameter configurations lead to qualitatively similar learning dynamics.

C.4 DETAILS OF THE MECHANISTIC STUDY AND ADDITIONAL ANALYSES (SECTION 5.3.2)

C.4.1 *Implementation of the attention patching experiment*

Our attention patching experiment consists of training a reference model, then restarting training with the same initial parameters but providing the model with the attention patterns the reference model produced when seeing the same samples. Two hyperparameters control this: `reference_patching` (the number of training steps for which the reference model was trained) and `start_patching` (the step at which patching begins; before this, the modified model uses its own attention). There are two key configurations: when `start_patching` and `reference_patching` are equal, this effectively freezes the model's attention at a given point; when `start_patching` is 0, patching occurs throughout training, as presented in the main text.

We implement attention patching through a twin architecture. We initialize the modified model with the initial parameters and the reference model with the desired parameters. Both models process the input sequence layer-wise. At each layer, the reference model generates attention scores and output. The attention scores are sent to the corresponding layer in the modified model (and the output to the next layer), with gradients stopped on the attention scores to prevent training the reference model. The modified layer uses the provided attention pattern instead of its own if patching has begun. While generating all attention scores from the reference model beforehand is possible and potentially simpler code-wise, our layer-wise implementation is more memory-efficient, particularly for long sequences, and many layers and heads. That said, given that memory is not an issue in our experiments given the small size of the models, it would probably have been good enough. We always re-initialize the optimizer state to default values when starting patching. While we found some significant loss increase just after the beginning of patching with this strategy, we also found them to be smaller than without re-initializing them.

The experimental setup that we have described here is richer than the one presented in the main text as we can now vary the beginning of patching. We provide the results of this more extensive analysis in Figure C.4 for completeness. Those results confirm our conclusions from the main text.

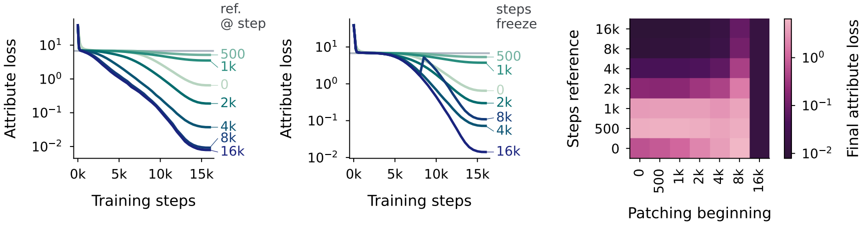


Figure C.4: Extended analysis of the attention patching experiment. (right) Patching starts at the beginning of learning (`start_patching` = 0). It is the same figure than Figure 5.3 (middle). (middle) In this experiment, the attention pattern are frozen, that is `start_patching` = `reference_patching`. (right) Final attribute loss when independently varying the number of steps at which we start patching and the number of steps the reference model was traning.

C.4.2 *Details of the attention pattern analysis*

In our analysis, the model's attention patterns during learning, we examine which tokens the network attends to when processing or predicting specific tokens. This approach is motivated by prior work (discussed in Section 5.3.2 and visualized in Figure C.5) that identified specific attention-based circuits for factual recall tasks, each with distinct signatures observable through attention patterns. We focus on the following circuits and their signatures:

- Name tokens grouping circuit. This circuit, present in the model's first layer, groups the embeddings of name tokens to represent the individual's full name. This occurs when processing the last name token, leading to high attention on other name tokens in the first layer. Focusing on the last name token is crucial to differentiate this mechanism from others like name completion. The *name* → *name* line in the right panel of Figure 5.3 represents the average attention from the last name token to all other name tokens in the first layer, averaged over name occurrences (typically 6, corresponding to the number of attributes), heads, and 512 input sequences. Figure C.6 (left) shows this quantity for all layers.

- Extraction circuit. The final attention layer selects and relays relevant information based on the requested attribute type. We expect high attention to name tokens (but not text tokens) when predicting the first attribute value token, once this circuit is established. Again, focusing on

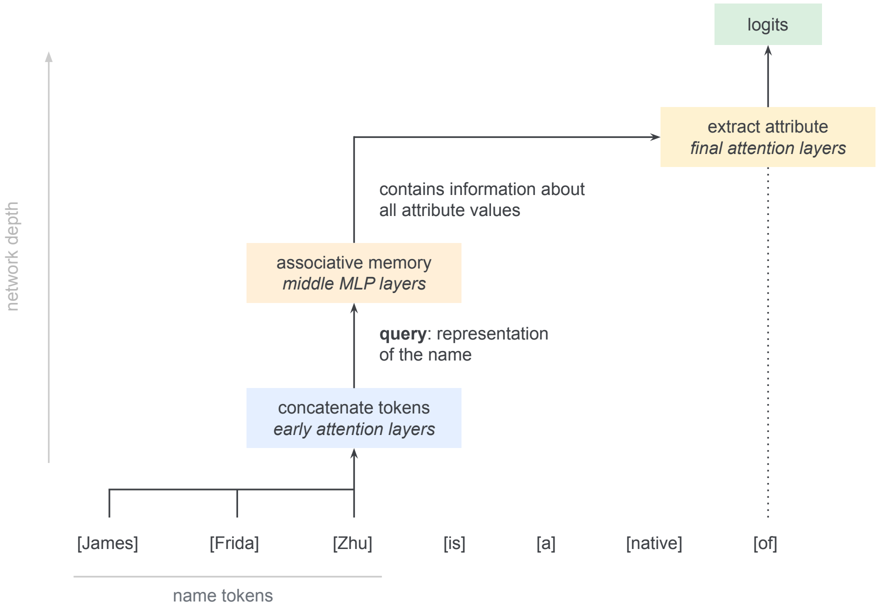


FIGURE C.5: High-level description of how Transformer-based language models solve associative recall tasks. Figure adapted from [245]. This model motivates our fine-grained attention pattern analysis (described in Section C.4.2) and is closely related, albeit slightly simplified, to the one of [244].

> the first token is crucial to isolate this from completion mechanisms. The *attribute* → *text* and *attribute* → *name* lines in Figure 5.3 are generated accordingly. Figure C.6 (middle left and middle right) shows the layer-wise breakdown of these quantities.

We do not include the attribute type propagation circuit highlighted in [244] in this analysis to keep it simple. That said, the relatively high attention to text tokens in the first attention layers when predicting attribute values is consistent with that circuit.

As a separate observation, Figure C.6 (right) reports the evolution of the sharpness of attention patterns in each layer, defined as the normalized entropy of the attention distribution (divided by the logarithm of the number of attendable tokens). Initially uniform, attention patterns become

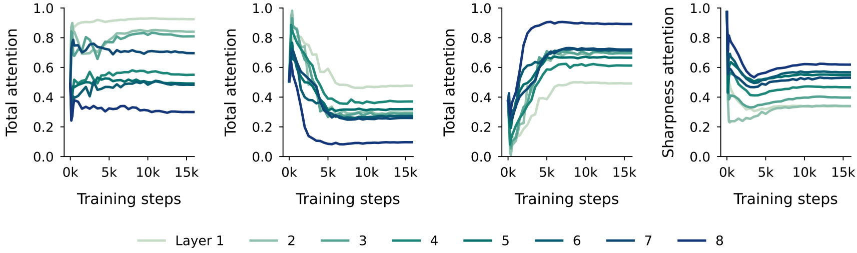


FIGURE C.6: Layer-wise analysis of the attention patterns of the model over the course of learning. (left) Attention given to the name tokens when seeing the last name tokens. (middle left) Attention given to the template tokens when predicting the first attribute value token. (middle right) Attention given to the name tokens when predicting the first attribute value token. (right) Sharpness of the attention probability distribution, defined as the entropy of the distribution divided by its mask value ($\log t$ with t the index of the current token).

progressively sharper, with a slight increase in sharpness starting around the end of the plateau.

We conclude this section by discussing the limitations of this analysis. It ignores the impact of the values (e.g., attention patterns do not matter when values are equal to 0), is purely correlational, and relies on simplified mental models of the network's internal workings. Nevertheless, it enables refining the conclusions drawn from our attention patching intervention.

C.5 ADDITIONAL ANALYSIS FOR THE IMPACT OF DATA DISTRIBUTION PROPERTIES (SECTION 5.4)

C.5.1 *Learning curves for different data distributions*

We here provide some examples of learning curves when modifying the training distribution. We focus on the 8k training steps, 64k individuals regime as it is one in which we observe large differences between distributions.

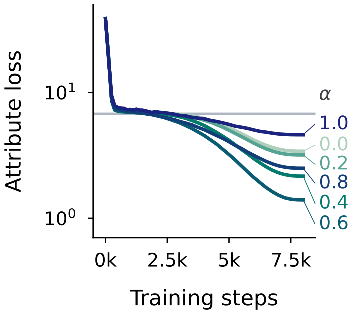


Figure C.7: Learning curves for the inverse power law distribution, obtained for 8k training steps and 64k individuals.

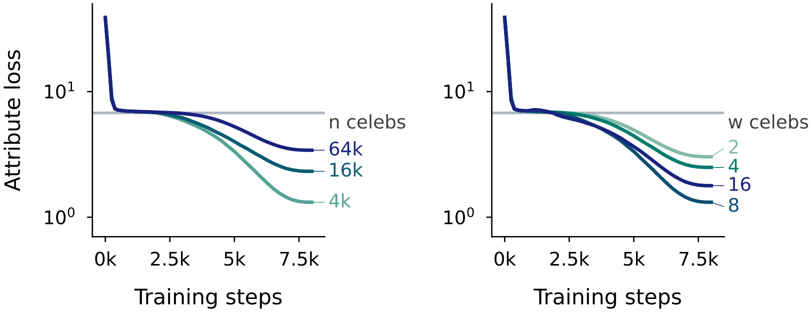


Figure C.8: Learning curves for the celebrities distribution, obtained for 8k training steps and 64k individuals. In the left plot, the weight for celebrities is set to 8 and in the right plot the number of celebrities is set to 4k.

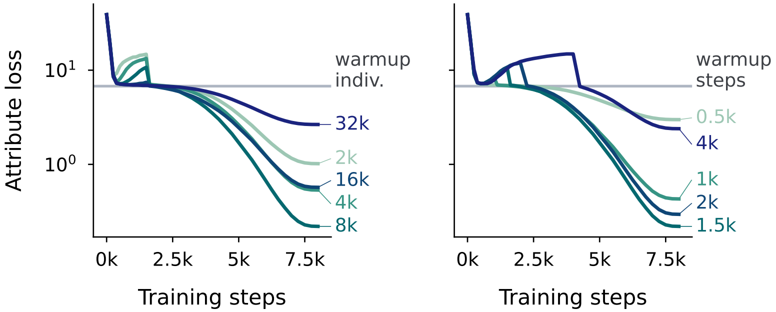


FIGURE C.9: Learning curves for the warm-up distribution, obtained for 8k training steps and 64k individuals. In the left plot, the number of warm-up steps is set to 1.5k and in the right plot the number of warm-up individuals is set to 8k.

C.5.2 *Extensive comparison of the performance of different data distributions*

In Figure C.10, we provide a detailed version of Figure 5.4 (right). We vary the number of training steps and the number of individuals, and report both attribute accuracy and attribute loss. Additionally, we include the celebrities' distribution in the comparison. The results stay qualitatively the same as what we reported in the main text.

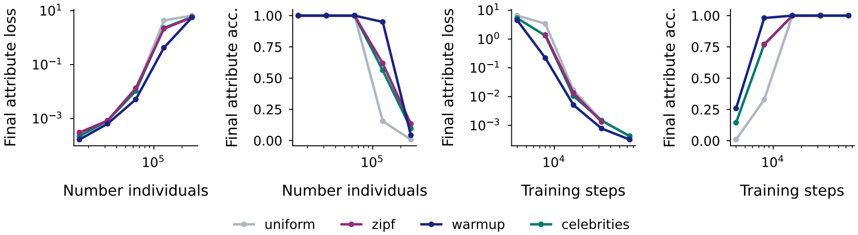


FIGURE C.10: Extensive comparison of the final performance of the model when trained on different classes of data distribution. For each class, we pick the best data distribution hyperparameter specifying the class. (left and middle left) We vary the total number of individuals, (middle right and right) we vary the number of training steps.

It should be noted that we have not tried to optimize the warm-up strategy more than through the hyperparameter grid search reported in Section C.7.4

as it is not the main focus of this paper, and it is possible that even better results could be achieved. For example, we only allow for changing distributions every epoch, with an epoch being defined as seeing all individuals in the training distribution. This implies that for a high number of individuals, tuning is rather coarse. Another potential improvement is to gradually increase the distribution size.

In Figure C.11 and C.12, we plot the final performance of the model for different hyperparameter choices. In the low training step regime (8k steps plot in Figure C.11), we find the optimal hyperparameter choice to be within our grid search. However, in the large number of individuals scenario, it seems outside our range (cf. 128k individuals plot of Figure C.12) and in particular suggests a long warmup phase with a high number of individuals. This leads us to wonder whether the benefits we observe from the warming-up could be achieved by just training on fewer individuals. To that extent, we run an additional run with a uniform distribution on the individual, with a population size of 83k individuals. The model reaches an attribute accuracy of 97.72%, which would become 63.26% when evaluating on 128k individuals. On the other side, the model trained with warm-ups reaches 94.92% accuracy. This refinement of our results strengthens our confidence in the benefits of warm-ups, particularly as we have not optimized the strategy beyond a relatively small hyperparameter grid search.

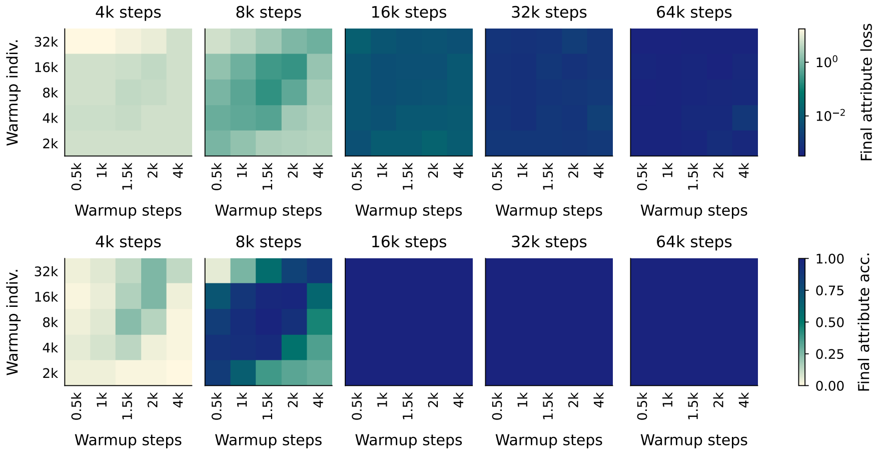


Figure C.11: Visualization of how final performance evolves as the hyperparameters of the warm-up distribution are changed. The total number of individuals is fixed to 64k and these plots correspond to Figure C.10 (middle right and right).

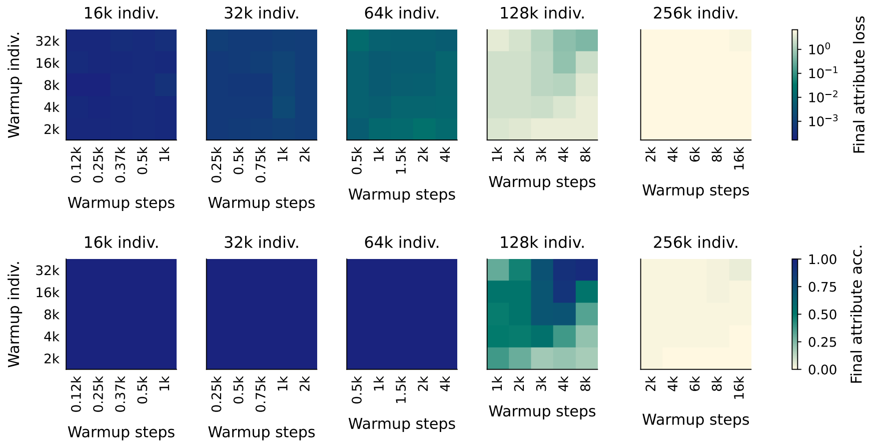


Figure C.12: Visualization of how final performance evolve as the hyperparameters of the warm-up distribution are changed. The total number of steps is fixed to 16k and these plots correspond to Figure C.10 (left and middle left).

C.6 DETAILS OF THE FINE-TUNING ANALYSIS AND ADDITIONAL EXPERIMENTS (SECTION 5.5)

C.6.1 *Experimental details*

In our fine-tuning experiments, we generate `n_individuals_finetune` new individuals and use their biographies for fine-tuning. By default, we use fine-tune a model trained for 16k steps on 64k individuals, with individuals sampled uniformly (i.e., the default parameters in Table C.1). In the experiments with replay, we use sampling according to the "celebrities" distribution described in Section C.2.2, using a weight of `weight_replay_finetune` for the 64k pre-training individuals, and a weight of 1 for the fine-tuning individuals. We use a constant learning rate of $3 \cdot 10^{-5}$ for fine-tuning, the rest of the optimizer staying as it is during pre-training.

C.6.2 *Hallucinations*

As briefly mentioned in Section 5.5, our framework offers a simple way to monitor fact-conflicting hallucinations (as defined in [321]) during training. These hallucinations are characterized by overconfidence in predicting facts absent (or rare) from the training data. To assess this, we evaluate the model on held-out individuals not present in the training distribution. A hallucinating model would predict incorrect attribute values with high confidence. For these held-out individuals, the attribute loss must exceed the no knowledge baseline, with higher values indicating overconfidence (i.e., hallucinations). Figure C.13 shows that hallucinations appear shortly after the plateau phase. However, the model remains less confident in these hallucinations than in its grounded predictions for seen individuals (see middle right and right panels). The higher probability mass on the most likely prediction and lower entropy of the predictive distribution for seen individuals suggest that hallucinations can be detected to some extent. We found these observations to be robust to changes in the input distribution. For example, progressively increasing the population size did not significantly affect these metrics.

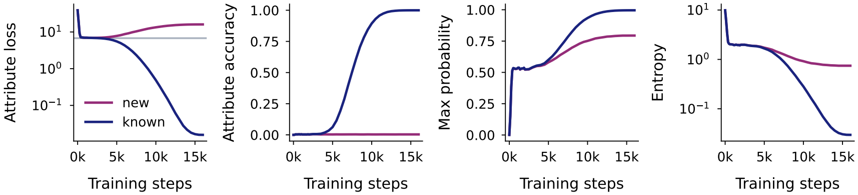


FIGURE C.13: The model starts hallucinating during training. The blue line corresponds to the model performance on seen individuals (as elsewhere in the paper) and the purple one to its performance on 16k held-out individuals. (left) attribute loss, (middle left) knowledge accuracy, (middle right) average probability of the most likely predicted token (for attribute values), and (right) average entropy of the predictive distribution (for attribute values). Overall, the model is less confident in its hallucinations than in grounded predictions.

This experiment is only a preliminary exploration of hallucinations within our setup. Future work could, for example, investigate how the confidence gap evolves when increasing the number of individuals. For instance, if we consider the network's associative memory to be a linear key-value database, increasing the number of individuals while holding the number of hidden neurons constant would bring key representations (individual names) closer, making it harder to distinguish unseen keys, thereby reducing the confidence gap and increasing hallucinations. Understanding which training individuals indirectly inform predictions for unseen individuals and what is the underlying similarity metric is another intriguing direction. Finally, on the more practical side, this framework could serve as a test-bed for methods aiming at detecting or mitigating hallucinations, as successful general-purpose methods should also be performing well in this simplified setting. We leave these investigations to future work.

C.6.3 *Additional analysis for fine-tuning*

In the main text, we have presented results for two sets of fine-tuning experiments, one with replay and one without replay. In Figure 5.5, we have reported fine-tuning dynamics in a fine-tuning loss vs. pre-training loss reference frame. We here visualize the evolution of the model performance as a function of time. Importantly, these plots make it easier to remark that the drop in model performance on pre-training data occurs very early during fine-tuning (on the order of a hundred steps), and that the increase in performance on fine-tuning data is happening on a longer time-scale (see Figure C.14). Additionally, the accuracy plots show that the increase in loss to values close to no-knowledge baseline does not necessarily mean that all pre-training knowledge is erased as some knowledge (i.e. non-zero accuracy) about the pre-training data remains.

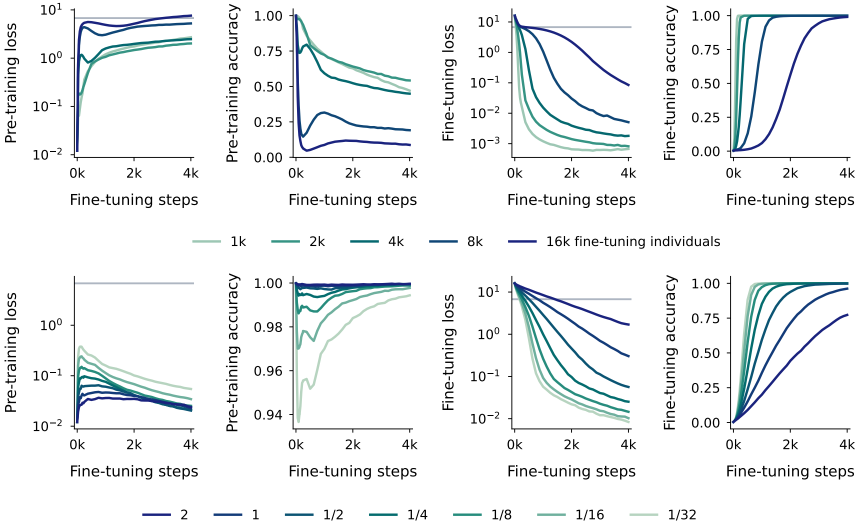


Figure C.14: Evolution of the performance of the model on the pre-training distribution (left and middle left panels) and on the fine-tuning distribution (middle right and right panels), as fine-tuning progresses. In the first row, there is no replay of the pre-training data during fine-tuning and we vary the number of fine-tuning individuals. In the second row, obtained for 4k fine-tuning individuals, we introduce some replay whose weight we vary (the weight corresponds to how much bigger the probability of sampling a pre-training individual is compared to one in the fine-tuning set). The data presented here is the same as the one in Figure 5.5 (middle and right panels), but here plotted as a function of time and including accuracy.

In Figure C.15, we provide the evolution of (some of) the attention scores for one of the fine-tuning experiments with no replay (4k individuals), focusing on the metrics that we have introduced in Section C.4.2 as they are most relevant to our analysis. We find attention scores to be particularly stable, which enables us to conclude that the performance drop during fine-tuning is not strongly linked to changes in attention patterns.

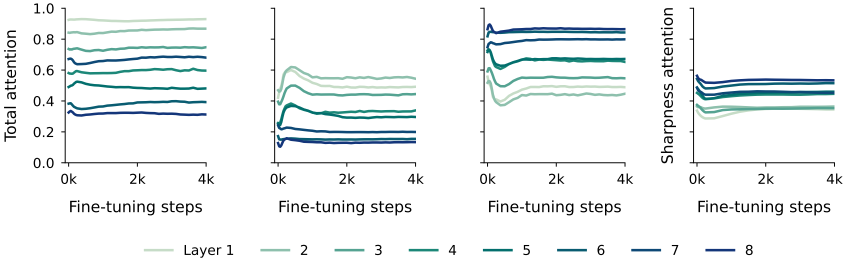


Figure C.15: The attention patterns remain remarkably relatively during fine-tuning. We here report the following metrics: (left) attention to name tokens when seeing the last name token, (middle left) attention to general text tokens when predicting the first attribute value token, (middle right) attention to the last name token when predicting the first attribute value token, (right) (normalized) entropy of the probability distribution defined by attention scores. The attention patterns analyzed here were obtained on the pre-training distribution; performing the same analysis on the fine-tuning distribution (not reported here) barely changes observed behaviors. See Section C.4.2 for the rationale behind the choice of metrics.

We complement our analysis with an experiment in which we fine-tune the model on rare, but seen during pre-training, individuals. We do so using our celebrities distribution, setting the total number of individuals to 128k, `n_celebrities` to 8k and `weight_celebrities` to 8. After training on 16k training steps, the model confidently and correctly predicts the attribute value of the celebrities but gets around 50% accuracy on non-celebrities, which are the majority of the population. Overall, we find that fine-tuning on existing individuals does not lead to fast performance drop fine-tuning on new individuals has (Figure C.16). We are able to add a significant amount of knowledge in the model's weights, as the accuracy on the 120k non-celebrities goes from 50% to close to 70%. These results supporting that fine-tuning on existing individuals is net-positive are confirmed by our alternating training distribution experiment of Figure C.22.

C.6.4 *Reproducing fine-tuning behavior on a toy associative memory problem*

In this section, we investigate whether the fine-tuning behavior observed in our language model experiments can be attributed to changes in their feed-forward associative memories. To this end, we train a single-hidden-

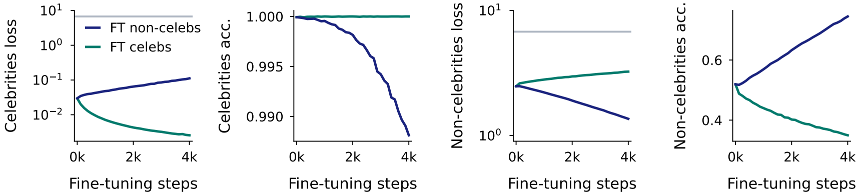


Figure C.16: Evolution of the model's performance when fine-tuned on rare (FT on non-celebrities line) or frequent (FT on celebrities line) individuals from the pre-training distribution. See Section C.6.3 for more detail.

layer multi-layer perceptron on a synthetic heterogeneous associative recall task, analogous to retrieving attribute values (e.g., university, major) given an individual's name.

Our network has 256 hidden neurons with ReLU activation. Both keys (names) and values (attributes) are represented by 64-dimensional random embeddings drawn from a normal distribution and then projected to the L2 unit sphere. The model learns to map keys to one of 30 possible values, effectively performing 30-way classification. It is learned with the cross-entropy loss and the AdamW optimizer with a cosine learning rate schedule (initial learning rate 0.03, 16k steps, batch size 128, weight decay 0.01).

We pre-train the model on 8192 key-value pairs and then fine-tune it with a constant learning rate of 0.001 for 4000 steps. Fine-tuning incorporates a variable number of new key-value pairs and optionally includes replay of pre-training data. The replay weight controls the relative sampling probability of pre-training versus fine-tuning examples (e.g., a weight of 2 with 2 pre-training and 1 fine-tuning examples results in sampling probabilities of 2/5, 2/5, and 1/5).

Mirroring Section 5.5, we conduct two experiments:

1. Fine-tuning without replay, while varying the number of new samples (left panel of Figure C.17, top row of Figure C.18).
2. Fine-tuning on 128 new samples and varying replay weight (right panel of Figure C.17, bottom row of Figure C.18).

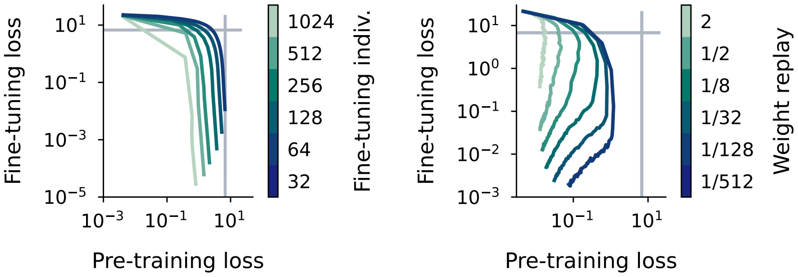


FIGURE C.17: Equivalent of Figure 5.5 for the toy associative memory model. In the right plot, we set the number of fine-tuning examples to 128. See Section C.6.4 for experimental details.

We also experimented with fine-tuning on old key-value pairs, as in Figure C.16, and once again observed qualitatively similar behaviors. Overall, these results provide evidence for the hypothesis that mostly feed-forward associative memories are altered during the incorporation of new knowledge through fine-tuning.

C.6.5 *Experiments with regular changes in training distribution*

To complement our fine-tuning analysis, we consider the setup in which the training distribution is regularly changed, either to a new group of individuals every change (this corresponds to varying `number_groups` and fixing `number_repeats` to 1) or alternating between two groups of individuals (this corresponds to varying `number_repeats` and fixing `number_groups` to 2). While this kind of experiment is further away from the current training paradigms of large language models and is more akin to a continual learning setup (e.g., [322]), it enables us to monitor the plasticity of the network, that is how fast it learns new knowledge and forgets about existing ones, in a dynamic way. Overall, we find that our fine-tuning findings are robust across different stages of learning and these new results enable us to sometimes refine our conclusions.

We observe a few interesting patterns. First, it becomes increasingly easier to learn a new distribution over time (Figure C.19), which is likely induced by the improvement of the attention pattern quality over the course of

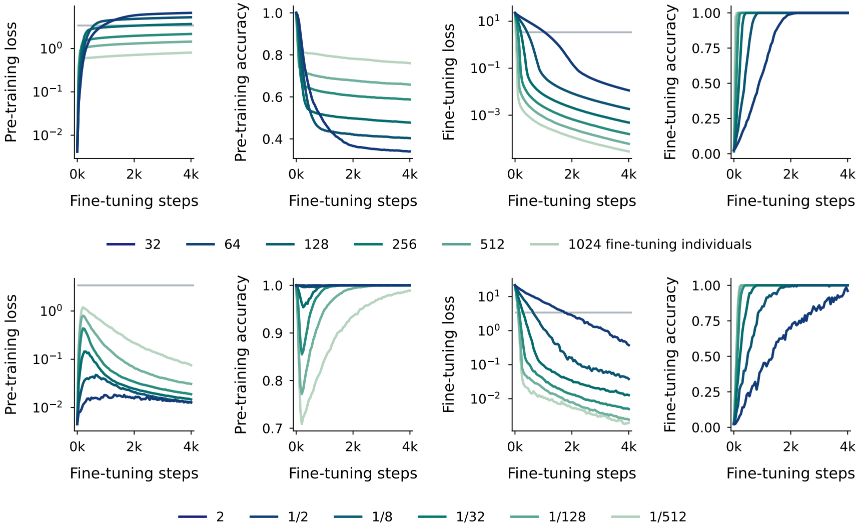


FIGURE C.18: Equivalent of Figure C.14 for the toy associative memory model. See Section C.6.4 for experimental details.

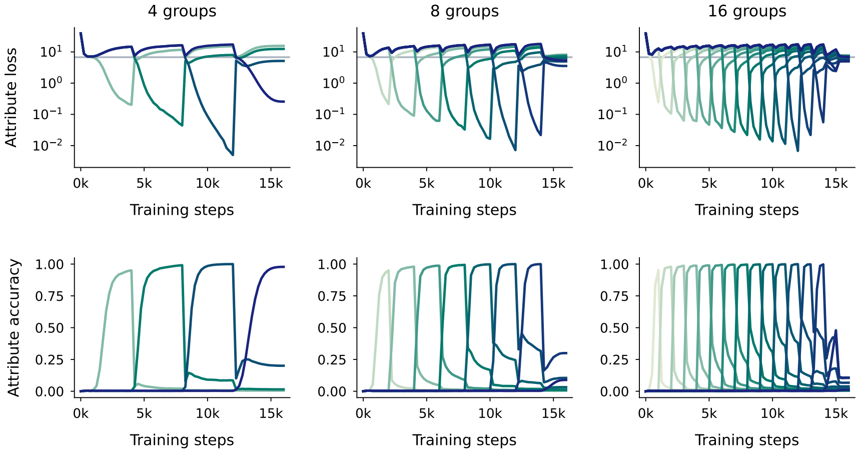


FIGURE C.19: Evolution of the model performance on different groups of individuals when it is trained sequentially on these groups. The color indicates the group of people the model is evaluated on: the darker the color, the later the network was trained on this group. For all plots the total number of individuals considered is equal to 64k. See Section C.6.5 for more detail.

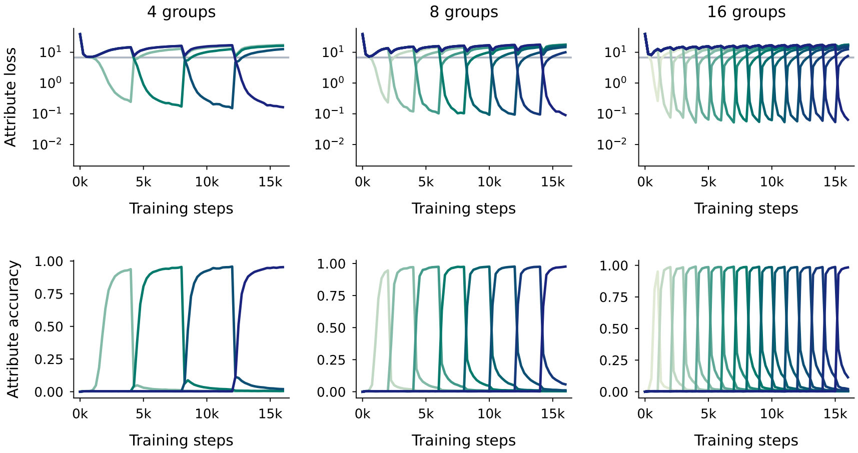


Figure C.20: Same as Figure C.19, but with a constant learning rate scheduler.

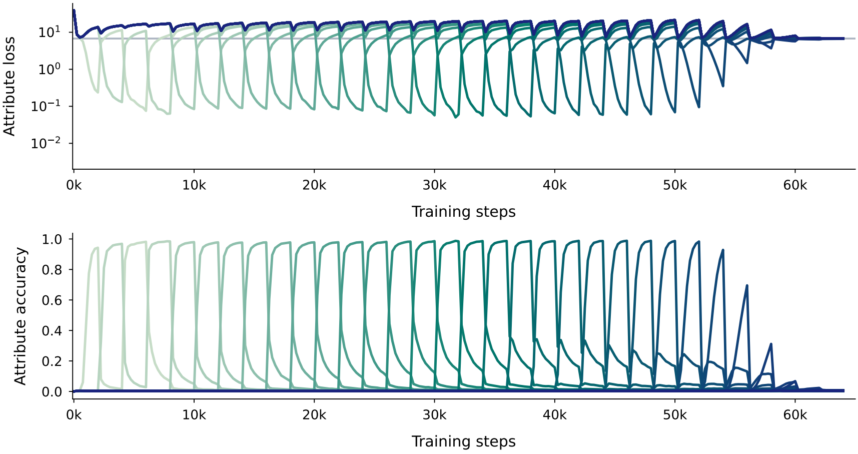


Figure C.21: Similar to the middle panel of Figure C.19 (8 groups), this time when training for longer. The total number of individuals is adapted so that the size of each group remains the same as before. All other hyperparameters remain the same.

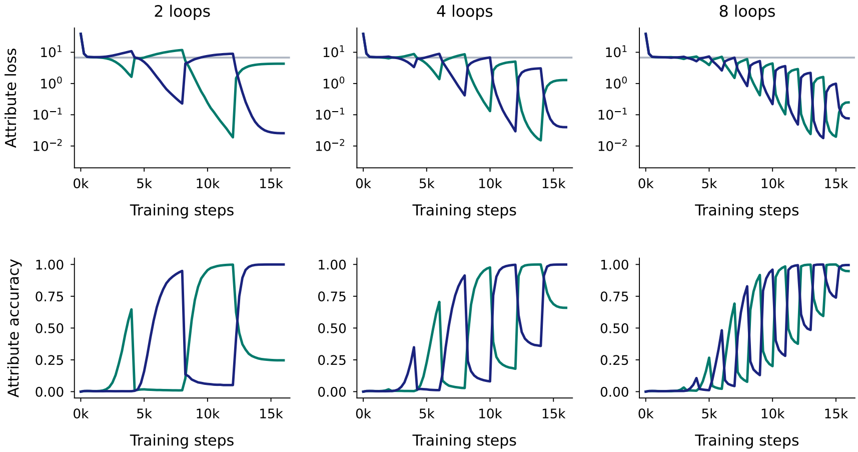


Figure C.22: Evolution of the model performance on different groups of individuals when it is trained alternating between two groups. The color indicates the group of people the model is evaluated on: green is for the first group, blue for the second one. For all plots the total number of individuals considered is equal to 64k. See Section C.6.5 for more detail.

learning (Figure C.23), until the cosine learning rate scheduler eventually damps down weight changes. At the same time, forgetting, measured in log loss differences, becomes more important and occurs much faster than learning, confirming the results of Figure 5.5. However, changing distributions multiple times during early learning dynamics when the learning rate is relatively high reduces this plasticity increase, as it partially impairs the future ability of the network to learn new groups (c.f. performance drop from Figure C.19 middle to Figure C.21, or Figure C.20). For small learning rates (i.e. at the end of training), performance on old data sometimes gets improved after some initial performance drop, without it being replayed (Figure C.23), an intriguing phenomenon that we have partially observed during fine-tuning (Figure C.14, top row). Additionally, for very small learning rates, learning eventually becomes harder and forgetting is not catastrophic, leading to the model remembering more about the second to last group than the last one (see e.g., the 8 groups plot in Figure C.19). Still, too many distribution changes to unknown distributions when learning rates are small ultimately remove any knowledge from the network (e.g. Figure C.21).

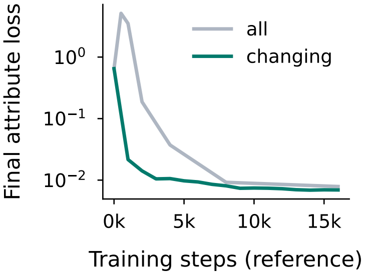


Figure C.23: Final attribute loss at the end of an attention patching experiment in which the model of Figure C.19 (8 groups) serves as reference (green line), as a function of the number of steps the reference model was trained. For comparison, we include the one we obtained when training on all individuals at once (grey line), as in Figure C.4. We use this metric as a measure of how good the attention patterns of a given model (the reference model) at a given time are for learning and solving the task at hand. We do not observe an initial bump in performance when changing individual distribution. There are two main reasons for that: First, attention patterns get formed early on due to training occurring on less individuals. Second, the granularity that we use (2k steps) would not allow us to see such a bump here.

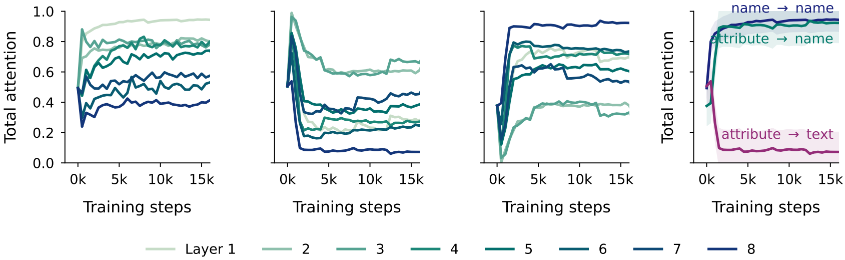


Figure C.24: Evolution of the attention patterns for the experiment reported in the middle panel of Figure C.19 (one loop over 8 different groups of individuals). In the first three panels, we report (left) the average attention to name tokens when seeing the last name token, (middle left) attention to general text tokens when predicting the first attribute value token, (middle right) attention to the last name token when predicting the first attribute value token. In the right panel, we select the last layer line from the attribute to name and attribute to text plots, and the first layer line from the name to name line. See Section C.4.2 for the rationale behind the choice of metrics. The attention patterns are more unstable than in the constant distribution scenario (Figure C.6). Yet they remain relatively stable, particularly for the one highlighted in the right panel towards the end of learning, in light of the frequent distribution changes and the relatively high learning rate used for most of the training.

C.7 HYPERPARAMETER CONFIGURATIONS

C.7.1 *Default configuration*

Name	Value	Description
Model		
`parameters`	44M	Number of parameters.
`n_layers`	8	Number of layers.
`n_heads`	8	Number of attention heads per attention layer.
`d_model`	512	Model dimension (residual stream).
`d_hidden`	2048	Hidden dimension of the multi-layer perception.
`key_size`	64	Dimension of the key and values.
`sequence_mixer`	Attention	Which sequence mixing block we are using.
Training		
`training_steps`	16k	Number of training steps.
`batch_size`	128	Batch size.
`lr_scheduler`	cosine	Learning rate scheduler, default is cosine scheduler (no warm-up, final learning rate: 10^{-7}).
`lr`	$4 \cdot 10^{-4}$	Maximum learning rate.
`weight_decay`	0.1	Weight decay.
`optimizer`	AdamW	Optimizer (momentum parameters for the AdamW optimizer are $\beta_1 = 0.9$, $\beta_2 = 0.95$).
Data		
`sequence_length`	512	Sequence length.
`n_individuals`	64k	Number of different individuals in the population.
`indiv_dist_train`	uniform	Distribution from which we sample individuals whenever we generate a new biography (at train time).
`indiv_dist_eval`	uniform	Same as previous line, but for evaluation.
`shuffle_templates`	True	Whether to shuffle templates.
`n_templates`	25	Number of templates per attribute type.
`n_templates_train`	20	Number of templates used in training.
`n_sequences_eval`	16k	Number of sequences for evaluation.
Miscellaneous		
`n_seeds`	1	Number of seeds per hyperparameter configuration.
`tokenizer`	SentencePiece	Tokenizer, as in [248].
`vocab_size`	32k	Vocabulary size of the tokenizer.
`checkpoints`	50	One checkpoint every 125 steps until 2k steps, and 32 checkpoints uniformly spaced over the entire learning trajectory.
`accelerator`	Google TPUv3	Hardware accelerator.

Table C.1: Default hyperparameters used in our experiments.

C.7.2 *Hyperparameters for Section 5.3.1*

C.7.3 *Hyperparameters for Section 5.3.2*

C.7.4 *Hyperparameters for Section 5.4*

Inverse power law distribution (Figure 5.4, C.7 and C.10)	
`n_individuals`	[4k, 8k, 16k, 32k, 64k, 128k, 256k]
`training_steps`	[8k, 16k, 32k]
`indiv_dist_train`	Inverse power law
- `alpha`	[0, 0.2, 0.4, 0.6, 0.8, 1]
Compute time: 1584 hours (train) and 521 hours (eval).	

Celebrities distribution (vary number of individuals) (Figure C.8 and C.10)	
`n_individuals`	[4k, 8k, 16k, 32k, 64k, 128k, 256k]
`indiv_dist_train`	Celebrities
- `n_celebrities`	[4k, 16k, 64k]
- `weight_celebrities`	[2, 4, 8, 16]
Compute time: 1128 hours (train) and 768 hours (eval).	

Celebrities distribution (vary number of steps) (Figure C.8 and C.10)	
`training_steps`	[4k, 8k, 16k, 32k, 64k, 128k, 256k]
`indiv_dist_train`	Celebrities
- `n_celebrities`	[4k, 16k, 64k]
- `weight_celebrities`	[2, 4, 8, 16]
Compute time: 1512 hours (train) and 541 hours (eval).	

Warm-up distribution (vary number of individuals) (Figure 5.4, C.9, C.10 and C.12)	
`n_individuals`	[4k, 8k, 16k, 32k, 64k, 128k, 256k]
`indiv_dist_train`	Warm-up
- `indiv_warmup`	[2k, 4k, 8k, 16k, 32k]
- `epochs_warmup`	[1, 2, 4, 8]
Compute time: 1944 hours (train) and 1152 hours (eval).	

C.7.5 *Hyperparameters for Section 5.5*

HALLUCINATIONS (Figure 5.5 and C.13)	
`lr`	$4 \cdot 10^{-4}$
Compute time: 3 hours (train) and 5 hours (eval).	

FINE-TUNING WITHOUT REPLAY (Figure 5.5, C.14 and C.15)	
`n_individuals_finetune`	$[1k, 2k, 4k, 8k, 16k]$
`lr_finetune`	$3 \cdot 10^{-5}$
Compute time: 17 hours (train) and 95 hours (eval).	

FINE-TUNING WITH REPLAY (Figure 5.5 and C.14)	
`n_individuals_finetune`	4k
`weight_replay_finetune`	$[2, 1, 1/2, 1/4, 1/8, 1/16, 1/32]$
`lr_finetune`	$3 \cdot 10^{-5}$
Compute time: 26 hours (train) and 174 hours (eval).	

FINE-TUNING ON CELEBRITIES (Figure C.16)	
`n_individuals`	128k
`training_steps`	16k
`indiv_dist_train`	Celebrities
- `n_celebrities`	8k
- `weight_celebrities`	8
`lr_finetune`	$3 \cdot 10^{-5}$
Compute time: 6 hours (train) and 48 hours (eval).	

Main experiment (Figure 5.2)

`lr`	$[10^{-4}, 2 \cdot 10^{-4}, 4 \cdot 10^{-4}, 8 \cdot 10^{-4}, 1.6 \cdot 10^{-3}]$
`n_seeds`	5

Compute time: 78 hours (train) and 154 hours (eval).

Ablation number of individuals (Figures 5.2 and C.3)

`lr`	$[10^{-4}, 2 \cdot 10^{-4}, 4 \cdot 10^{-4}, 8 \cdot 10^{-4}, 1.6 \cdot 10^{-3}]$
`n_individuals`	[4k, 8k, 16k, 32k, 64k, 128k, 256k]
`n_seeds`	5

Compute time: 600 hours (train) and 214 hours (eval).

Ablation weight decay (Figure C.3)

`lr`	$[10^{-4}, 2 \cdot 10^{-4}, 4 \cdot 10^{-4}, 8 \cdot 10^{-4}, 1.6 \cdot 10^{-3}]$
`weight_decay`	[0.01, 0.03, 0.1, 0.3, 1]

Compute time: 86 hours (train) and 29 hours (eval).

Ablation batch size (Figure C.3)

`lr`	$[10^{-4}, 2 \cdot 10^{-4}, 4 \cdot 10^{-4}, 8 \cdot 10^{-4}, 1.6 \cdot 10^{-3}]$
`training_steps`	[4k, 8k, 16k, 32k, 64k]
`batch_size`	[32, 64, 128, 256, 512]

Compute time: 685 hours (train) and 148 hours (eval).

Ablation model size (Figure C.3)

`lr`	$[10^{-4}, 2 \cdot 10^{-4}, 4 \cdot 10^{-4}, 8 \cdot 10^{-4}, 1.6 \cdot 10^{-3}]$
`parameters`	[50m, 150m, 400m]

Compute time: 30 hours (train) and 89 hours (eval).

Ablation sequence mixer (Figure C.3)

`lr`	$[10^{-4}, 2 \cdot 10^{-4}, 4 \cdot 10^{-4}, 8 \cdot 10^{-4}, 1.6 \cdot 10^{-3}]$
`sequence_mixer`	[Attention, RG-LRU [71]]

Compute time: 32 hours (train) and 15 hours (eval).

Ablation template variety (Figure C.1)

`lr`	$[10^{-4}, 2 \cdot 10^{-4}, 4 \cdot 10^{-4}, 8 \cdot 10^{-4}, 1.6 \cdot 10^{-3}]$
`shuffle_templates`	[True, False]
`templates_train`	[1, 2, 4, 8, 16]

Compute time: 163 hours (train) and 53 hours (eval).

Attention patching experiment (Figures 5.3 and C.4)

`start_patching`	[0k, 0.5k, 1k, 2k, 4k, 8k, 16k]
`reference_patching`	[0k, 0.5k, 1k, 2k, 4k, 8k, 16k]

Compute time: 73 hours (train) and 160 hours (eval).

Attention pattern analysis (Figures 5.3 and C.6)

`lr`	$4 \cdot 10^{-4}$

Run taken from results of Figure 5.2.

Warm-up distribution (vary number of steps) (Figure 5.4, C.9, C.10 and C.11)

`training_steps`	[4k, 8k, 16k, 32k, 64k, 128k, 256k]
`indiv_dist_train`	Warm-up
- `indiv_warmup`	[2k, 4k, 8k, 16k, 32k]
- `epochs_warmup`	[1, 2, 4, 8]

Compute time: 2808 hours (train) and 1008 hours (eval).

Sequential learning with new groups (Figure C.19, C.20 (and C.21))

`n_individuals`	64k (256k)
`training_steps`	16k (64k)
`lr_scheduler`	[cosine, constant] (cosine)
`indiv_dist_train`	Sequential
- `n_groups`	[4, 8, 16] (32)
- `n_repeats`	1

Compute time: 26 (+13) hours (train) and 78 (+432) hours (eval).

Sequential learning with alternating groups (Figure C.22)

`indiv_dist_train`	Sequential
- `n_groups`	2
- `n_repeats`	[2, 4, 8]

Compute time: 10 hours (train) and 39 hours (eval).

D

THE EMERGENCE OF SPARSE ATTENTION

D.1 DETAILS OF THE THEORETICAL RESULTS

D.1.1 *Overview of the assumptions*

Throughout our theoretical analysis, we introduce a few assumptions to simplify it. We summarize them here:

Asm. 1 **Gradient flow dynamics.** To study learning dynamics, we focus on the gradient flow of the expected loss. Compared to the standard deep learning regime, this removes stochastic noise induced by sampling, uses gradient descent as optimizer instead of optimizer with adaptive learning rates such as Adam, and requires an infinitely small learning rate. Due to the last point, phenomena such as the edge of stability [201] are out of the picture. Yet, this is a very standard assumption (e.g. [59]) and will enable us to use tools from dynamical systems to understand how the behavior of the model changes over the course of learning.

Asm. 2 **Uniform attention at initialization.** At initialization, the attention patterns of Transformers generally display remarkable uniformity at initialization. We take advantage of this observation in our model by initializing a as a vector of zeros. This enables us to reduce the dynamics of the attention part of the model to a single scalar.

Asm. 3 W^* **has norm 1 columns.** We assume that the columns of W^* all have norm 1. While this is not strictly necessary for performing our analysis, it simplifies the formulas that we obtain and helps to keep the notation concise. Additionally, this is the expected behavior when drawing the entire of W^* i.i.d. from a zero-mean normal distribution with variance $\frac{1}{d}$ when d goes to infinity.

Asm. 4 **Initialize the weights W at 0.** When drawing the entries of W i.i.d. and independently of those of W^*, W is almost surely orthogonal to W^* as $d \to \infty$. The components of W not aligned with W^* exhibit a fast decay towards 0, so this assumption corresponds to having already completed this process. With such an assumption, we can reduce the dynamics of the weights to a single scalar, w, which will be the projection of W on a normalized W^*.

Asm. 5 **Large sequences and large dimension.** Finally, we assume that we are in the large T limit and B is negligible in front of T. This makes sense in our context as we are interested in modeling the learning of sparse attention patterns. We additionally assume that d is large, which is a reasonable assumption given that we are interested in modeling the high-dimensional data neural networks have to process. These assumptions will enable us to ignore some negligible terms and keep our calculations more concise.

D.1.2 *Derivation of the expected loss and its gradients*

Recall that the loss is given by

$$L = \frac{1}{2}\mathbb{E}_x\left[\|y - y^*\|^2\right].$$

with

$$y = W\sum_{t=1}^{T} \mathrm{softmax}(a)_t \, x_t$$

and

$$y^* = W^* x_T.$$

For notational clarity, we define attention patterns as $\alpha_t := \mathrm{softmax}(a)_t$. Without loss of generality, we assume that the tokens repeated in tokens appear at positions $\{T - B + 1, \cdots, T\}$ so that the last B tokens are always the same, and that the repeated input $\tilde{x}$ is the first vector of the canonical basis[1] of $\mathbb{R}^d$, that is $\tilde{x} = [1, 0, \cdots 0]^\top$. For the sake of conciseness, we denote by

$$\Sigma_t := \mathbb{E}_x \left[x_t x_t^\top \right]$$

the input covariance of each token. Note that for $t \leq T - B$, it is equal to $\Sigma_t = \frac{1}{d}\mathrm{Id}$.

Using the insights mentioned above and the properties of the data distribution, the loss reduces to:

$$\begin{aligned}
L &= \frac{1}{2}\mathbb{E}_x \left[\|y - y^*\|^2 \right] \\
&= \frac{1}{2}\mathbb{E}_x \left[\left\| \sum_t \alpha_t W x_t - W^* x_T \right\|^2 \right] \\
&= \frac{1}{2}\mathbb{E}_x \left[\sum_{t \leq T-B} \alpha_t^2 \|W x_t\|^2 + \left\| \sum_{t > T-B} \alpha_t W x_t - W^* x_T \right\|^2 \right] \\
&= \frac{1}{2}\mathbb{E}_x \left[\sum_{t \leq T-B} \alpha_t^2 \|W x_t\|^2 + \left\| \sum_{t > T-B} \alpha_t W x_T - W^* x_T \right\|^2 \right] \\
&= \frac{1}{2} \left[\sum_{t \leq T-B} \alpha_t^2 \mathrm{tr}\left(W \Sigma_t W^\top \right) \right. \\
&\qquad \left. + \,\mathrm{tr}\left(\left(\sum_{t > T-B} \alpha_t W - W^* \right) \Sigma_T \left(\sum_{t > T-B} \alpha_t W - W^* \right)^\top \right) \right] \\
&= \frac{1}{2} \left[\sum_{t \leq T-B} \frac{\alpha_t^2}{d} \|W\|_F^2 \right. \\
&\qquad \left. + \,\mathrm{tr}\left(\left(\sum_{t > T-B} \alpha_t W - W^* \right) \Sigma_T \left(\sum_{t > T-B} \alpha_t W - W^* \right)^\top \right) \right].
\end{aligned}$$

In the third line, we used the fact that the first $T - B$ tokens are independent of each other and independent of the last B ones, in the fourth line that the last B tokens are all equal to x_T and in the fifth line the equality $\mathbb{E}_x[\|Ax\|] = \mathrm{tr}(A\mathbb{E}[xx^\top]A^\top)$.

---

[1] This amounts to a right multiplication of the weights W by an orthogonal matrix.

Leveraging this formula, we get the following gradients for the loss:

$$\nabla_W L = \sum_{t>T-B} \alpha_t \left(\sum_{t>T-B} \alpha_t W - W^* \right) \Sigma_T + \sum_{t \leq T-B} \frac{\alpha_t^2}{d} W \tag{D.1}$$

$$\nabla_{\alpha_t} L = \begin{cases} \operatorname{tr}\left(W \Sigma_T \left(\sum_{t>T-B} \alpha_t W - W^* \right)^\top \right) & \text{if } t > T-B \\ \frac{\alpha_t}{d} \|W\|^2 \ \text{otherwise.} \end{cases} \tag{D.2}$$

Note that we have not calculated the gradients with respect to a here, but only with respect to α.

D.1.3 *Analysis of the dynamics without cross-sample repetition*

In this section, we study the dynamics without cross-sample repetition, that is, with $p = 0$.

D.1.3.1 *Reducing the learning dynamics to two dimensions*

In general, the parameters evolve in a high-dimensional space. However, with a few reasonable assumptions, it is possible to show that they evolve in a 2-dimensional subspace:

- **Attention is initialized uniformly.** The first assumption we make is that attention is perfectly uniform (Assumption 2 from Section D.1.1), that is $\alpha_t = \frac{1}{T}$ or $a_t = 0$. Given that the gradients of these parameters are the same for $t \leq T-B$ and $t > T-B$, cf. the calculation in the previous section, all attention values will have two possible values. If we additionally leverage the fact that $\sum_t \alpha_t = 1$, everything is captured by a single scalar Δa, that is defined as $a_T - a_1$. Indeed

$$\begin{aligned} \alpha_t &= \frac{\exp(a_t)}{\sum_{t'} \exp(a_{t'})} \\ &= \frac{1}{(T-B)\exp(a_1 - a_t) + B\exp(a_T - a_t)} \end{aligned}$$

so that, for $t > T - B$,

$$\alpha_t = \frac{1}{(T-B)\exp(-\Delta a) + B} \tag{D.3}$$

and for $t \leq T - B$,

$$\alpha_t = \frac{(1 - B\alpha_T)}{(T-B)}. \tag{D.4}$$

- **Weights are initialized at** 0**.** We assume that $W = 0$ at initialization, following Assumption 4 of Section D.1.1. As $\Sigma_T = \frac{1}{d}\mathrm{Id}$, we have

  $$\nabla_W L = \sum_{t > T-B} \frac{\alpha_t}{d} \left(\sum_{t > T-B} \alpha_t W - W^* \right) + \sum_{t \leq T-B} \frac{\alpha_t^2}{d} W.$$

  This implies that whenever W is aligned with W^*, which is the case when $W = 0$, it remains aligned for the rest of learning. Yet, we are not entirely done as the precise parametrization is important to ensure that the dynamics match. Such a property is achieved when $W = wW^* / \|W^*\|_F$ as, under the gradient flow dynamics on W, we have

  $$\begin{aligned} w &= \frac{\langle W^*, W\rangle_F}{\|W^*\|_F} \\ \dot{w} &= \frac{\langle W^*, \dot{W}\rangle}{\|W^*\|_F} = -\frac{\langle W^*, \nabla_W L\rangle_F}{\|W^*\|_F} \end{aligned}$$

  and under the gradient flow dynamics directly on w, we get

  $$\dot{w} = -\nabla_w L = -\left\langle \frac{\mathrm{d}W}{\mathrm{d}w}, \nabla_W L \right\rangle_F = -\frac{\langle W^*, \nabla_W L\rangle_F}{\|W^*\|_F}.$$

  In the calculations above, we used $\langle \cdot, \cdot \rangle_F$ to denote the element-wise dot product (i.e., $\langle A, B\rangle_F = \sum_{ij} A_{ij} B_{ij}$). The corresponding norm is the Froebenius norm $\|\cdot\|_F$.

- **Each column of the target weights** W^* **has norm 1.** It follows that the Froebenius norm of W^* satisfies $\|W^*\|_F^2 = d$. This is Assumption 3 from Section D.1.1.

Under these assumptions, studying the gradient flow dynamics on the original loss therefore reduce to study the gradient flow dynamics on the simplified loss

$$L = \frac{1}{2}\left[\sum_{t\leq T-B} \frac{(1-B\alpha)^2}{d(T-B)^2}\frac{w^2}{\|W^*\|_F^2}\|W^*\|_F^2 + \frac{1}{d}\left(\frac{B\alpha w}{\|W^*\|}-1\right)^2 \operatorname{tr}\left(W^*W^{*\top}\right)\right] \tag{D.5}$$

$$= \frac{1}{2}\left(\frac{(1-B\alpha)^2 w^2}{d(T-B)} + \frac{\left(B\alpha w - \sqrt{d}\right)^2}{d}\right) \tag{D.6}$$

with the attention given to token T being equal to

$$\alpha = \frac{1}{(T-B)\exp(-\Delta a) + B}.$$

Note that we have dropped the T subscript for notational conciseness. We plot this reduced loss landscape in Figure D.1, as well as how the parameters evolve under the gradient flow dynamics.

As $\mathrm{d}_{\Delta\alpha}\alpha = \alpha(1-B\alpha)$, the gradient of this loss with respect to w and Δa are

$$\nabla_w L = \frac{(1-B\alpha)^2 w}{d(T-B)} + \frac{B\alpha(B\alpha w - \sqrt{d})}{d} \tag{D.7}$$

$$\nabla_{\Delta a} L = \alpha(1-B\alpha)\left(-\frac{B(1-B\alpha)w^2}{d(T-B)} + \frac{Bw(B\alpha w - \sqrt{d})}{d}\right). \tag{D.8}$$

The entries of the Hessian are

$$\begin{aligned}
\frac{\mathrm{d}^2 L}{\mathrm{d}w^2} &= \frac{(1-B\alpha)^2}{d(T-B)} + \frac{B^2\alpha^2}{d} \\
\frac{\mathrm{d}^2 L}{\mathrm{d}w\,\mathrm{d}\Delta a} &= \alpha(1-B\alpha)\left(-\frac{2B(1-B\alpha)w}{d(T-B)} + \frac{B(2B\alpha w - \sqrt{d})}{d}\right) \\
\frac{\mathrm{d}^2 L}{\mathrm{d}\Delta a^2} &= \frac{1-2B\alpha}{\alpha(1-B\alpha)}\nabla_{\Delta a} L + \alpha(1-B\alpha)\left(\frac{B^2 w^2}{d(T-B)} + \frac{B^2 w^2}{d}\right).
\end{aligned}$$

D.1.3.2 *Approximate dynamics around initialization*

Despite having reduced the dynamics to two dimensions, it is still too complex to be analyzed mathematically. In order to gain insight into the early

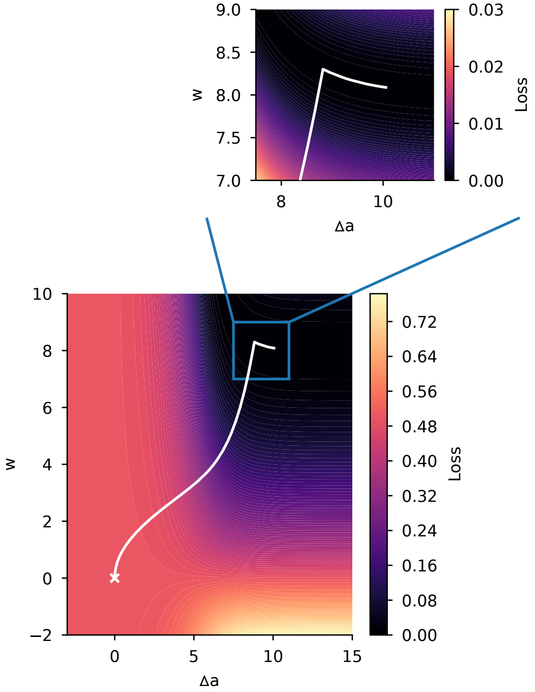


Figure D.1: Loss landscape for the reduced model ($T = 256$, $d = 64$, without any repetition). The white line corresponds to the gradient flow on this loss, initialized in $(0, 0)$ (white cross).

behavior of the dynamics, we linearize the dynamics around initialization and compute the time it will require to leave the initial loss plateau.

LINEARIZED DYNAMICS AT INITIALIZATION. Linearizing the dynamics at initialization corresponds to considering the gradient flow on the quadratic approximation to the loss:

$$\begin{aligned} L(w, \Delta a) = L(0,0) + \nabla L(0,0)^\top \begin{pmatrix} w \\ \Delta a \end{pmatrix} \\ + \frac{1}{2} \begin{pmatrix} w \\ \Delta a \end{pmatrix}^\top H(0,0) \begin{pmatrix} w \\ \Delta a \end{pmatrix} + o(\|w\|^2 + \|\Delta a\|^2), \end{aligned}$$

with $\nabla L(0,0)$ the gradient of the loss and $H(0,0)$ the Hessian of the loss at initialization. Plugging in the values the different variable take at initialization gives

$$\begin{aligned} \nabla_w L &= -\frac{B}{\sqrt{d}T} \\ \nabla_{\Delta a} L &= 0 \end{aligned}$$

and

$$\begin{aligned} \frac{\mathrm{d}^2 L}{\mathrm{d}w^2} &= \left(\frac{T-B}{dT^2} + \frac{B^2}{dT^2} \right) \\ \frac{\mathrm{d}^2 L}{\mathrm{d}w \, \mathrm{d}\Delta a} &= -\frac{T-B}{T^2} \frac{B}{\sqrt{d}} \\ \frac{\mathrm{d}^2 L}{\mathrm{d}\Delta a^2} &= 0. \end{aligned}$$

Under Assumption 5 from D.1.1, $T \to \infty$, B stays negligible compared to T, and d is large enough so that $\sqrt{d}$ is negligible in front of d, so that $\frac{T-1}{T^2}$ becomes $\frac{1}{T}$, $\mathrm{d}_w^2 L$ becomes $\frac{1}{dT}$ and it is therefore negligible in front of the cross-term second-order derivative. The gradient flow dynamics around initialization becomes

$$\begin{pmatrix} \dot{w} \\ \Delta a \end{pmatrix} = \begin{pmatrix} \frac{B}{\sqrt{d}T} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & \frac{B}{\sqrt{d}T} \\ \frac{B}{\sqrt{d}T} & 0 \end{pmatrix} \begin{pmatrix} w \\ \Delta a \end{pmatrix} + o(\|w\| + \|\Delta a\|). \tag{D.9}$$

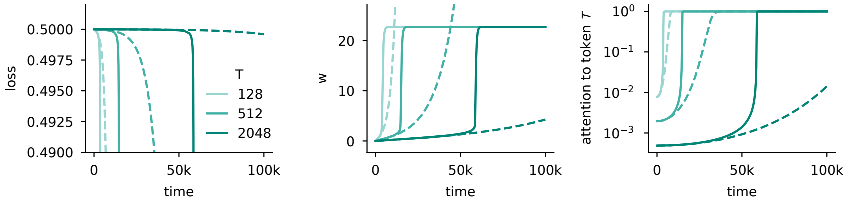


FIGURE D.2: COMPARISON OF THE EARLY DYNAMICS (PLAIN LINES) WITH THEIR CORRESPONDING LINEAR APPROXIMATION AROUND LINEAR CONDITIONS (DASHED LINES). d is here fixed to 512.

Figure D.2 provides a visualization of how well these approximate dynamics match the original one.

The Hessian matrix has a positive eigenvalue $\frac{B}{\sqrt{d}T}$ and one negative one $-\frac{B}{\sqrt{d}T}$. This implies that the initial parameters are in the vicinity of an unstable fixed point and that the negative eigenvalue of the Hessian will dictate how fast we are escaping these initial conditions. Its eigenvalues are

$$\lambda_\pm = \pm \frac{B}{\sqrt{d}T}$$

with eigenvectors

$$v_\pm = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ \pm 1 \end{pmatrix}.$$

Let P be the change of basis matrix defined by these eigenvectors. This matrix P transforms the original coordinates $(w, \Delta a)$ to the eigenbasis coordinates, (z_+, z_-). The dynamics in the new basis is

$$\dot{\begin{pmatrix} z_+ \\ z_- \end{pmatrix}} = \frac{B}{\sqrt{2d}T} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \frac{B}{\sqrt{d}T} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} z_+ \\ z_- \end{pmatrix}.$$

This yields

$$\begin{aligned} z_+ &= \frac{1}{\sqrt{2}} \left(\exp(\lambda t) - 1\right) \\ z_- &= \frac{1}{\sqrt{2}} \left(1 - \exp(-\lambda t)\right) \end{aligned}$$

and

$$
\begin{aligned}
w &= \sinh(\lambda t) \\
\Delta a &= \cosh(\lambda t) - 1
\end{aligned}
$$

INITIAL EVOLUTION OF THE LOSS. We can now derive a close-form equation for the temporal evolution of the quadratic approximation of the loss: as

$$L(w, \Delta a) = 0.5 - \lambda w - \lambda w \Delta a + o(\|w\|^2 + \|\Delta a\|^2),$$

we get that the loss is initially approximately equal to

$$
\begin{aligned}
L &\approx \frac{1}{2} - \lambda \sinh(\lambda t) - \lambda \sinh(\lambda t)(\cosh(\lambda t) - 1) \\
&= \frac{1}{2} - \lambda \sinh(\lambda t) \cosh(\lambda t) \\
&= \frac{1}{2}\left(1 - \lambda \sinh(2\lambda t)\right)
\end{aligned}
$$

using the identity $\sinh(x)\cosh(x) = \frac{1}{2}\sinh(2x)$.

TIME NEEDED TO ESCAPE THE INITIALIZATION PLATEAU. We can now compute the time T_ε it will take for the (approximate) loss to be equal to $(1 - \varepsilon)$ its initial value. From this definition, we get that T_ε satisfies

$$\frac{1}{2}(1 - \lambda \sinh(2\lambda t)) = \frac{1 - \varepsilon}{2},$$

that is

$$T_\varepsilon = \frac{\sqrt{d}T}{2B} \operatorname{arcsinh}\left(\frac{\varepsilon\sqrt{d}T}{B}\right).$$

Given that we are in the regime of large T and d, we finally get

$$T_\varepsilon = \frac{\sqrt{d}T}{2B} \log\left(\frac{2\varepsilon\sqrt{d}T}{B}\right)$$

by using the fact that $\operatorname{arcsinh}(x) \sim \frac{1}{2}\log(x)$ as $x \to \infty$.

D.1.3.3 *Phase transition and late phase dynamics*

Once we escape initial conditions and attention to the relevant token(s) starts increasing, the impact of the other tokens on the loss starts becoming negligible, the loss approximately becomes

$$L \approx \frac{1}{2d}\left(B\alpha w - \sqrt{d}\right)^2. \tag{D.10}$$

This loss features multiplicative interactions akin to a one-hidden-layer neural network, and therefore it comes as no surprise that the loss exhibits similar sharp phase transitions as the ones observed when learning these networks, cf. Saxe, McClelland & Ganguli [59] for an in depth analysis of these dynamics (one needs to linearize the mapping $\Delta a \mapsto \alpha$ to get the exact mapping to this set of results).

The learning dynamics exhibits an interesting behavior after the phase transition: the projection of w on W^* starts decreasing, as seen on Figure D.1 for example. In the following, we will argue that w is close to equilibrium in that phase and that learning consists mainly in slowly pushing Δa to infinity and that the corresponding equilibria decrease. We will show this by investigating the structure of the Hessian when w is at equilibrium.

First, let us compute the value that w takes at equilibrium, which requires solving the equation $\nabla_w L = 0$. Using Equation D.7, it gives

$$\left(\frac{(1-B\alpha)^2}{d(T-B)} + \frac{(B\alpha)^2}{d}\right) w = \frac{B\alpha}{\sqrt{d}},$$

that is

$$w^\infty(\alpha) := \left(\frac{(1-B\alpha)^2}{d(T-B)} + \frac{(B\alpha)^2}{d}\right)^{-1} \frac{B\alpha}{\sqrt{d}}.$$

We plot w^∞ as a function of α in Figure D.3.left.

Let us now consider the case in which $B\alpha$ converges to 1, that is $B\alpha = 1 - \varepsilon$ with $\varepsilon \to 0$. This gives

$$w^\infty = \frac{B\alpha d}{(B\alpha)^2\sqrt{d}} + o(\varepsilon^2) = \frac{\sqrt{d}}{1-\varepsilon} + o(\varepsilon^2).$$

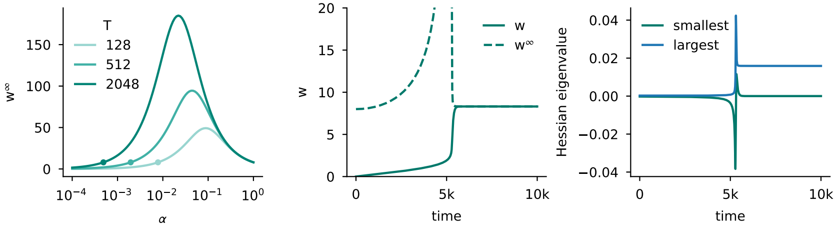


FIGURE D.3: (left) w^∞ as a function of α for different T values ($d = 64$, $B = 1$). (middle) Evolution of w on the gradient flow dynamics ($T = 512$, $d = 64$, $B = 1$) and comparison with its instantaneous equilibrium value w^*. (right) Evolution of the smallest and largest eigenvalue of the Hessian of the loss along the gradient flow dynamics.

We can then plug this value into the different components of the Hessian using the second derivatives we calculated for the simplified loss:

$$\begin{aligned}
\frac{\mathrm{d}^2 L}{\mathrm{d}w^2} &= \frac{1}{d} - \frac{2\varepsilon}{d} + o(\varepsilon^2) \\
\frac{\mathrm{d}^2 L}{\mathrm{d}w\,\mathrm{d}\Delta a} &= \frac{B\varepsilon}{\sqrt{d}} + o(\varepsilon^2) \\
\frac{\mathrm{d}^2 L}{\mathrm{d}\Delta a^2} &= \left(\frac{B + B^2}{T - B} + B^2\right)\varepsilon + o(\varepsilon^2).
\end{aligned}$$

The Hessian is thus equal to

$$H(w^\infty, \Delta a) = \begin{pmatrix} \frac{1-2\varepsilon}{d} & \frac{B\varepsilon}{\sqrt{d}} \\ \frac{B\varepsilon}{\sqrt{d}} & \left(\frac{B+B^2}{T-B} + B^2\right)\varepsilon \end{pmatrix} + o(\varepsilon^2)$$

when w^∞ is at optimality and $B\alpha$ close to 1 (i.e. $\Delta a \to \infty$). As ε goes to 0, the loss becomes increasingly flat in the Δa dimension, highlighting that Δa will be the bottleneck in terms of learning and w will always be at its equilibrium values w^∞. We confirm this in simulation in the middle panel of Figure D.3.

It is now worth comparing the loss landscape structure in this late phase compared to the one early on during learning. In the early stages, the eigenvalues are small ($B/\sqrt{d}T$) and w and Δ both contribute to them. At the end of learning, the loss is much sharper in the w direction ($1/d$) than in the Δa direction dimension (ε), because of the softmax within the attention.

From this analysis of the two extremes of the learning dynamics, in discrete time, w would be the bottleneck in terms of the learning rate, due to the final sharpness of the loss. A complete picture is hard to obtain analytically because calculations become more involved in the middle of learning, so we resort to simulations and plot the results in Figure D.3.right. We find that the behavior we described analytically holds for reasonably long in the early and late dynamics. In the middle of the dynamics, the geometry of the loss landscape changes drastically and the loss is the sharpest at this time (and thus the maximum learning rate we can take in discrete time will be dependent on geometry of the loss around the phase transition).

D.1.3.4 *Empirical validation*

In these sections, our aim is to verify whether our theory still captures the behavior of the model when the assumptions we made are not met. In particular, we will relax Assumption 3 (W^* has unit norm columns), 4 (W initialized to 0), 5 (long sequences) and partially Assumption 1 (gradient flow dynamics). Indeed, we sample W and W^* from $\mathcal{N}(0, 1/d)$, take T to be 256 (thus finite), d to be 256 and consider stochastic gradient descent dynamics with batch size 32 and learning rate 1. We report the comparison of the evolution of the loss, the attention α to the relevant to the T-th token, and the projection w of the weights W on W^* in Figure D.4 for both the reduced dynamics (Theory line, as in Equations D.7 and D.8) and for simulations (with the parameters described above). Overall, we find that the simplified dynamics is able to depict the ones of the actual model quite accurately.

D.1.4 *Analysis of the dynamics with cross-sample repetition*

When considering cross-sample repetition, Σ_T will have a larger magnitude in the direction of the token $\tilde{x}$ that appears more frequently during learning. As a result, weights learn faster in that direction and having a single scalar to capture the behavior of the entire weight matrix is no longer possible. In the following, we show how to reduce it to two scalars using similar techniques as before, as well as proceed to the analysis.

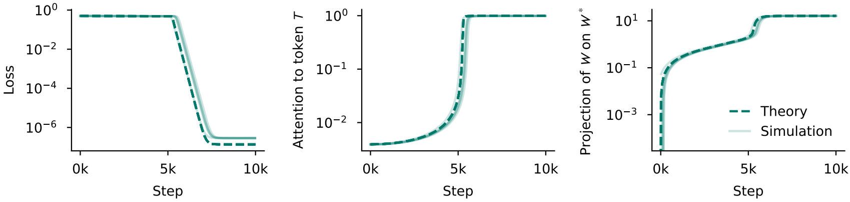


Figure D.4: The gradient flow dynamics on the simplified loss of Equation D.6 match the stochastic gradient descent dynamics of the original objective. We report the dynamics of 5 different seeds for the simulation lines. Changing the seed modified the target mapping, the sampling process, as well as the model initialization. Additional details are provided in Section D.1.3.4.

D.1.4.1 *Reducing the learning dynamics to three dimensions*

Cross-sample repetition does not affect attention directly, so we can still capture there entire behavior of attention with

$$\alpha = \frac{1}{(T-1)\exp(-\Delta a) + 1}.$$

Note that we do not consider any in-context repetition here for simplicity, but all our results can be extended to this case. Let us now look at the weights. Without loss of generality, we can assume $\tilde{x} = [1, 0, \cdots, 0]^\top$, which yields

$$\Sigma_T = \mathrm{diag}\left(p + \frac{1-p}{d}, \frac{1-p}{d}, \cdots, \frac{1-p}{d}\right).$$

The gradient of the loss with respect to the i-th column $W_{:i}$ of the weight matrix therefore becomes

$$\nabla_{W_{:i}} L = \sum_{t>T-B} \alpha_t \left(p\delta_{i=1} + \frac{1-p}{d}\right)\left(\sum_{t>T-B} \alpha_t W_{:i} - W^*_{:i}\right) + \sum_{t \leq T-B} \frac{\alpha_t^2}{d} W_{:i}.$$

Importantly, when initialized at 0, each column of W will evolve on the line spanned by the corresponding column of W^*. However, they will not move at the same speed as the first column has different dynamics than the other columns. We therefore introduce two scalars $\tilde{w}$ and w such that $W_{:1} = \tilde{w} W^*_{:1}$ and $W_{:i} = \frac{w}{\sqrt{d-1}} W^*_{:i}$. As for the analysis in the previous section, the $\sqrt{d-1}$

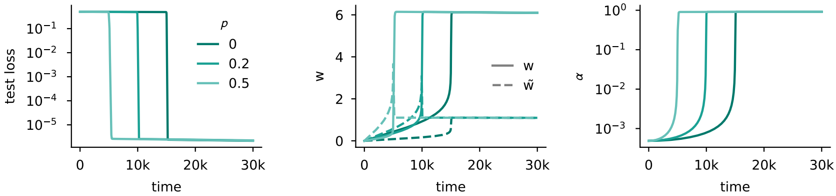


FIGURE D.5: CROSS-SAMPLE REPETITION SPEEDS UP EMERGENCE. This is because the repeated component $\tilde{w}$ (dashed lines in the middle plot) learns faster, which leads to an earlier increase of attention α to the relevant token (right) and faster learning overall (left). The results reported here were obtained by simulating the gradient flow on the loss of Equation D.11 for $T = 512$ and $d = 64$.

scaling factor is here to ensure that the learning speed coincide. To keep the notation as concise as possible, we introduce $\tilde{\sigma} := p + \frac{1-p}{d}$ and $\sigma := \frac{1-p}{d}$.

Under these assumptions, the loss simplifies to

$$L = \frac{1}{2}\left(\frac{(1-\alpha)^2(w^2+\tilde{w}^2)}{d(T-1)} + \tilde{\sigma}\left(\alpha\tilde{w}-1\right)^2 + \sigma\left(\alpha w - \sqrt{d-1}\right)^2\right). \quad \text{(D.11)}$$

The gradients flow dynamics on this loss are reported on Figure D.5. The gradients are

$$\nabla_{\tilde{w}} L = \frac{(1-\alpha)^2\tilde{w}}{d(T-1)} + \alpha\tilde{\sigma}(\alpha\tilde{w}-1) \quad \text{(D.12)}$$

$$\nabla_{w} L = \frac{(1-\alpha)^2 w}{d(T-1)} + \frac{\alpha(1-p)(\alpha w - \sqrt{d-1})}{d} \quad \text{(D.13)}$$

$$\nabla_{\Delta a} L = \alpha(1-\alpha)\left(-\frac{(1-\alpha)(w^2+\tilde{w}^2)}{d(T-1)} + \tilde{w}\tilde{\sigma}(\alpha\tilde{w}-1)\right. \quad \text{(D.14)}$$

$$\left. + w\sigma(\alpha w - \sqrt{d-1})\right). \quad \text{(D.15)}$$

The entries of the Hessian are

$$\begin{aligned}
\frac{\mathrm{d}^2 L}{\mathrm{d}\tilde{w}^2} &= \frac{(1-\alpha)^2}{d(T-1)} + \alpha^2 \tilde{\sigma} \\
\frac{\mathrm{d}^2 L}{\mathrm{d}w^2} &= \frac{(1-\alpha)^2}{d(T-1)} + \alpha^2 \sigma \\
\frac{\mathrm{d}^2 L}{\mathrm{d}w\,\mathrm{d}\tilde{w}} &= 0 \\
\frac{\mathrm{d}^2 L}{\mathrm{d}\tilde{w}\,\mathrm{d}\Delta a} &= \alpha(1-\alpha)\left(-\frac{(1-\alpha)\tilde{w}}{d(T-1)} + (2\alpha\tilde{w}-1)\,\tilde{\sigma}\right) \\
\frac{\mathrm{d}^2 L}{\mathrm{d}w\,\mathrm{d}\Delta a} &= \alpha(1-\alpha)\left(-\frac{(1-\alpha)w}{d(T-1)} + \left(2\alpha w - \sqrt{d-1}\right)\sigma\right) \\
\frac{\mathrm{d}^2 L}{\mathrm{d}\Delta a^2} &= \frac{1-2\alpha}{\alpha(1-\alpha)} \nabla_{\Delta a} L + \alpha(1-\alpha)\left(\frac{(w^2+\tilde{w}^2)}{d(T-1)} + (\tilde{\sigma}\tilde{w}^2 + \sigma w^2)\right)
\end{aligned}$$

D.1.4.2 *Approximate dynamics around initialization*

Around initialization we get

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} \tilde{w} \\ w \\ \Delta a \end{pmatrix} = \begin{pmatrix} \frac{\tilde{\sigma}}{T} \\ \frac{\sigma\sqrt{d-1}}{T} \\ 0 \end{pmatrix} + H \begin{pmatrix} \tilde{w} \\ w \\ \Delta a \end{pmatrix}$$

with

$$H := \begin{pmatrix} -\frac{T-1}{dT^2} - \frac{\tilde{\sigma}}{T^2} & 0 & \frac{\tilde{\sigma}(T-1)}{T^2} \\ 0 & -\frac{T-1}{dT^2} - \frac{\sigma}{T^2} & \frac{\sigma(T-1)\sqrt{d-1}}{T^2} \\ \frac{\tilde{\sigma}(T-1)}{T^2} & \frac{\sigma(T-1)\sqrt{d-1}}{T^2} & 0 \end{pmatrix}.$$

Under Assumption 5, this dynamics becomes

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} \tilde{w} \\ w \\ \Delta a \end{pmatrix} = \begin{pmatrix} \frac{p}{T} \\ \frac{1-p}{\sqrt{d}T} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & \frac{p}{T} \\ 0 & 0 & \frac{1-p}{\sqrt{d}T} \\ \frac{p}{T} & \frac{1-p}{\sqrt{d}T} & 0 \end{pmatrix} \begin{pmatrix} \tilde{w} \\ w \\ \Delta a \end{pmatrix}.$$

The characteristic polynomial of the Hessian is

$$X^3 - X\left(\frac{(1-p)^2}{dT^2} + \frac{p^2}{T^2}\right)$$

which means that the eigenvalues of the Hessian are 0 and

$$\pm \frac{1}{\sqrt{d}T} \sqrt{p^2 d + (1-p)^2},$$

which provides a scaling factor for the exit time of

$$\frac{\sqrt{d}T}{\sqrt{p^2 d + (1-p)^2}}.$$

As a sanity check, we can remark that when there is no cross-sample repetition, we recover the same scaling factor as in our previous analysis. As p increases, it progressively removes the effect of the dimension d in the escape time.

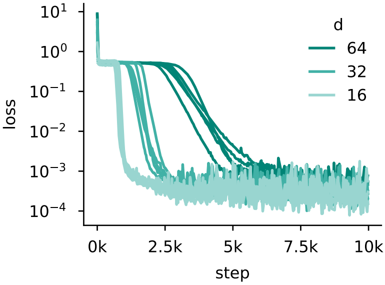


FIGURE D.6: THE LEARNING DYNAMICS OF TRANSFORMERS EXHIBIT SHARP PHASE TRANSITIONS IN THE SINGLE-LOCATION TASK. Here, $T = 128$ and the architecture and training details are the one described in Appendix D.4.3.

D.2 DETAILS AND ADDITIONAL ANALYSIS FOR THE LINEAR REGRESSION TASK

D.2.1 *Implementation of the task*

For our experiments, we implement a slightly different version of the task than the one we introduced in the main text and that we used for our theoretical analysis. The difference lies in where the relevant token is and whether there exists a feature to indicate it:

- **Random relevant token position**. In the simple version of the task, the relevant token was always presented at position T for convenience, as the output of the toy attention model we consider does not depend on the specific position. For experiments with more realistic Transformer models, we make this position random to avoid skip connections playing a specific role, and we resample it for every new sequence. The only exception to this is the task specifics ablation of Figure D.9, in which we sometimes fixed it throughout the entire training run (the relevant position being still randomly sampled at the beginning of training).
- **Relevant token feature**. As the position of the relevant token is resampled for every new sequence, we need to provide information to the network about which token is relevant for the task. To this end, we append an extra feature to each input token that is 1 whenever the token

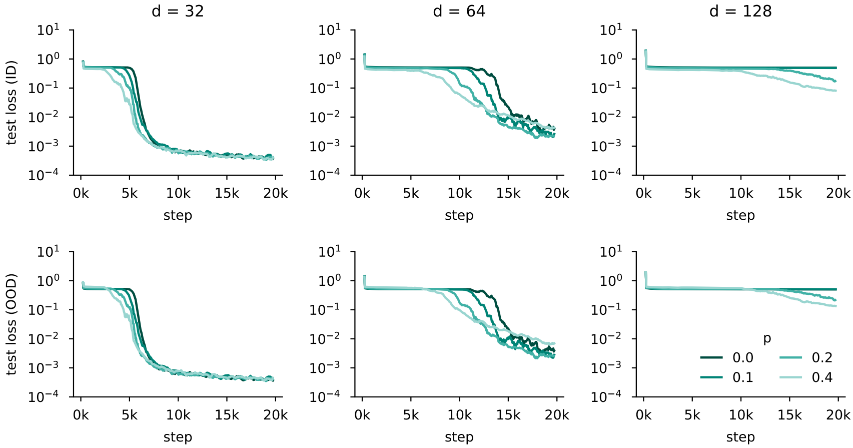


Figure D.7: Effects of cross-sample repetition on the learning dynamics of a Transformer in the linear regression task. The top line corresponds to the test loss on the same data distribution than the one on which the network is trained, that is with repetition. The bottom line is data without any repetition. The results are obtained for $T = 256$. See Appendix D.4.3 for the rest of the training details.

is relevant. As in the previous point, the only experiment in which this feature is not added to the input is the ablation of Figure D.9.

D.2.2 *Examples of learning curves*

Figures D.6 and D.7 provide examples of the learning dynamics of Transformers on the variation of the single-location linear regression task described in the previous section.

D.2.3 *Additional ablations on the impact of task variation, model size and optimizer on plateau length*

In the main text, we claim that the architecture, the details of the task, as well as the optimizer change the dependence of the plateau length on the different data parameters (here we study d and T). These changes collectively explain why the theoretical exponents of the power laws we obtained in our simplified regime do not directly translate to practice. This section provides the ablation underlying this claim; we detail them by increasing importance.

- **Architecture.** The toy model we consider has a single head and a single layer whereas the Transformer architecture we consider has 2 layers and 4 heads. In Figure D.8, we ablate the number of layers and number of heads. We find that they have some importance, by slowing the learning process (note that we did not tune the learning rate to each architecture size). However, they do not significantly alter the dependency in d and T captured through the power law exponents.

- **Task specifics.** In the more realistic version of the task, the Transformer is provided with an extra input feature which indicates whether the token is relevant for the task, and the relevant token position is random. However, the toy model does not have access to this feature (it cannot use it) and the position is fixed. In Figure D.9, we ablate these different choices and find that randomizing the position significantly increase the dependence in T and, to a smaller extent, the dependence on d. Interestingly, the power laws are almost the same for fixed position when the relevant feature is given to the model or not. We argue that this may be because the relevant feature provides a weaker statistical signal, likely because of the initial embedding layer from dimension d to 256, than the positional encoding. Learning to use would therefore be slower, and the network eventually ignores this feature.

- **Optimizer.** Our theoretical were obtained by analyzing the gradient flow dynamics. While gradient flow has tight links with gradient descent, and thus the emergence time under the gradient flow is approximately proportional to the number of steps before emergence (assuming small enough learning rates), this does not hold for Adam. To test how much Adam changes emergence time, we ablate the choice of the optimizer in Figure D.10. We find that Adam to be significantly faster than SGD (up to almost two order of magnitudes for the hard versions of the task), both in absolute terms and in sensitivity to the difficulty of the task.

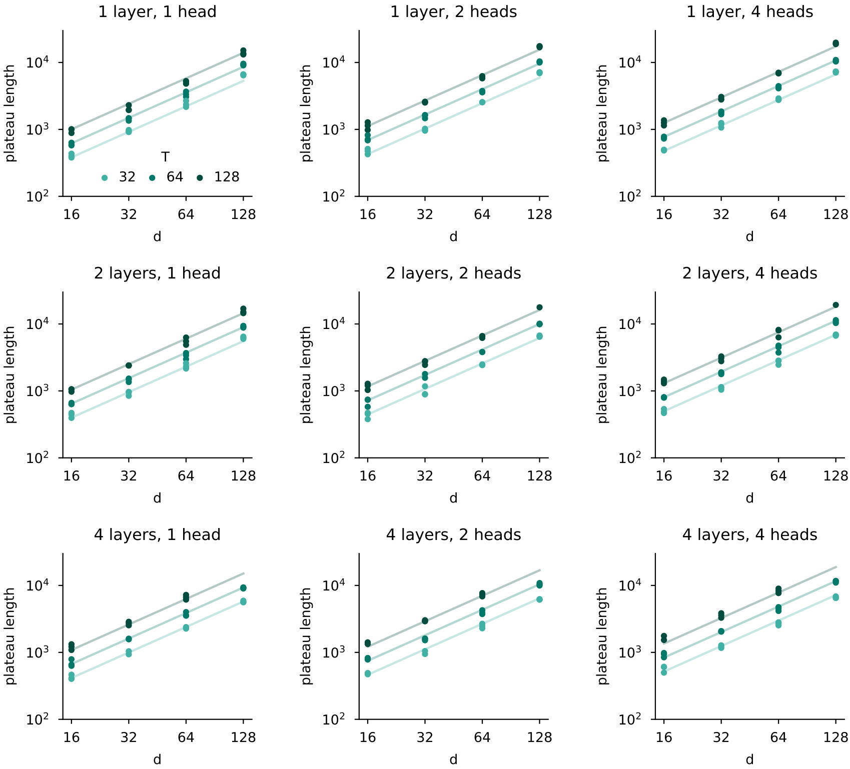


Figure D.8: Changing the width of the network (by increasing the number of heads) increases time spent on the initial plateau more significantly than increasing the depth of the network. We plot the evolution of the plateau length, for Transformers with different number of layers and number of heads. We obtain the following scaling law: $T_{\text{plateau}} = 1.03\, d^{1.27}\, T^{0.70}\, N_{\text{layers}}^{0.06}\, N_{\text{heads}}^{0.16}$ ($R^2 = 0.995$), which corresponds to the lines in the plots above.

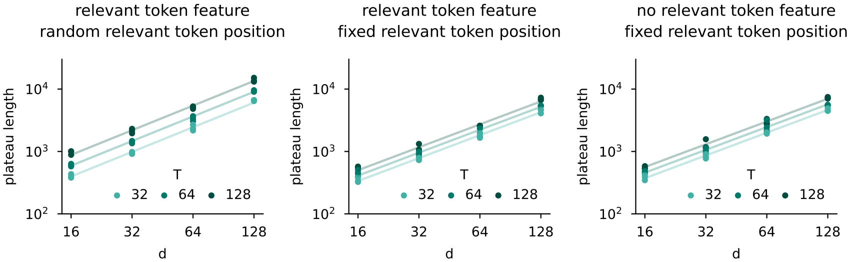


Figure D.9: Emergence is faster when positional information is available than when the network has to use semantic information. In this set of experiments, we vary the nature of the task. The network can be given a "relevant token feature" indicating whether the token is relevant for the task is given (that is semantic information about the nature of the token). The position of the relevant token for the task can be either fixed or random. If it is random, the network must use semantic information to solve the task. If not, it (may) use positional information. In particular, this means that the network cannot solve the task when it has no access to the relevant token feature and the positions are random; this is why we do not report results in this configuration here. The different scaling laws plotted are: (*left*) $T_{\text{plateau}} = 1.43\, d^{1.31}\, T^{0.58}$, $R^2 = 0.998$ (relevant feature, random position). (*middle*) $T_{\text{plateau}} = 4.20\, d^{1.22}\, T^{0.29}$, $R^2 = 0.996$ (relevant feature, fixed position). (*right*) $T_{\text{plateau}} = 4.61\, d^{1.21}\, T^{0.30}$, $R^2 = 0.999$ (no relevant feature, fixed position).

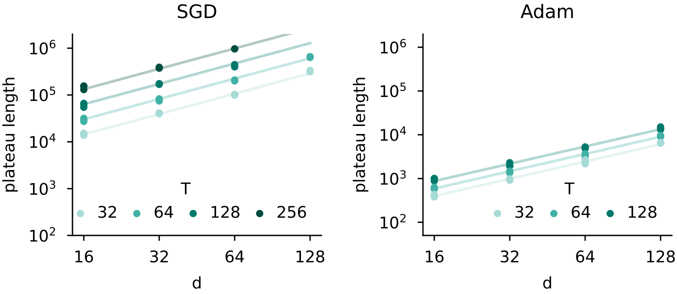


FIGURE D.10: SWITCHING SGD FOR ADAM LEADS TO AN IMPORTANT ACCELERATION OF LEARNING. Here, we train a single head single layer transformer with (left) SGD with a learning rate of $5 \cdot 10^{-3}$ and (right) Adam with a learning rate of 10^{-4} (these learning rates are manually tuned). Changing the optimizer has a huge effect on emergence time, both in terms of absolute time but also of scaling as the problem gets harder: $T_{\text{plateau}} = 6.42\, d^{1.44}\, T^{1.07}$ ($R^2 = 0.997$) for SGD and $T_{\text{plateau}} = 1.45\, d^{1.31}\, T^{0.57}$ for Adam ($R^2 = 0.998$).

D.3 DETAILS AND ADDITIONAL EXPERIMENTS FOR THE IN-CONTEXT ASSOCIATIVE RECALL TASK

D.3.1 *Implementation of the task*

Recall that each sequence consists of N_{pairs} key-value token pairs (5 such pairs in the example below) followed by a query token (`Z` below), as shown in the following example:

`Y I   A X   U R   Z Y   C A   Z ?`

The number of possible tokens is the vocabulary size N_{tokens} and each of these tokens has a corresponding one-hot encoding. Both keys and values are sampled from the same pool of tokens.

The generative process behind the task is as follows:

- **Set up query distribution.** We define a query distribution p_{query} which corresponds to uniformly sampling one of the 2-repeated tokens with probability p and uniformly sampling all the tokens with probability $1 - p$ (recall that p is the cross-sample repetition probability). Said otherwise, this probability distribution has mass $\frac{p}{2} + \frac{1-p}{N_{\text{tokens}}}$ on the repeated tokens and mass $\frac{1-p}{N_{\text{tokens}}}$ on the rest of the tokens.
- **Sample the in-context mapping.** For each new sequence, we sample the mapping between keys and queries. This mapping is injective, meaning that two different keys will always be associated with different values. In practice, we sample a random permutation for this mapping.
- **Sample the query**. For each new sequence, we sample the query following the probability distribution p_{query} defined when creating the task. The target output for the task, will be the value indicated by the in-context mapping.
- **Fill the context.** For each new sequence, we finally fill the context as follows. We first decide whether we put the query (and its corresponding value) in the N_{pairs} possible positions by sampling a Bernoulli variable with success probability $\frac{B}{N}$ (on expectation the query thus appears B times in the context). We then fill the rest of the context by sampling

keys from the $N_{\text{tokens}} - 1$ remaining tokens without replacement, and appending their corresponding values directly after.

D.3.2 *Learning dynamics and additional analysis*

We here provide two additional results: the learning dynamics on the training data, which show that repetition accelerates emergence in similar ways to what is predicted in our theory (Figure D.11) and the evolution of the attention scores over the course of learning, which confirms that emergence occurs jointly with attention to relevant tokens getting sparser (Figure D.12). We can make additional comments regarding the last point: it is interesting to see that all heads in the same layer have similar attention to relevant tokens, and that the copying mechanism (to group together the key and the value) only occurs in layer 3 and the selection mechanism only in layer 4.

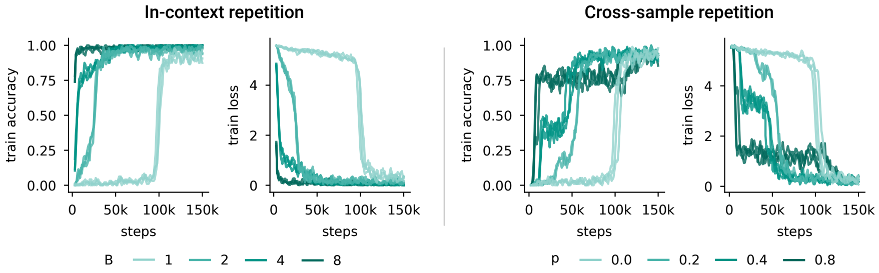


FIGURE D.11: REPETITION ACCELERATES EMERGENCE IN THE ASSOCIATIVE RECALL TASK AS PER THE THEORY. This plot is the training counterpart of the plots of Figure 6.6. As we measure the performance on the training data, there is no overfitting and repetition only speeds up emergence.

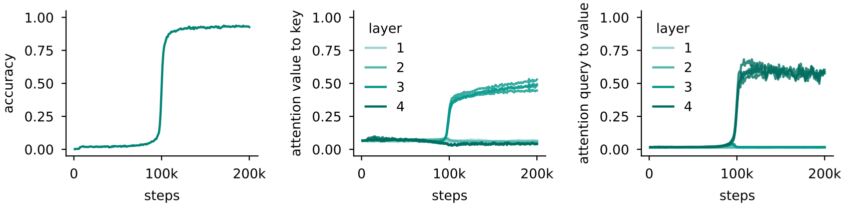


Figure D.12: The emergence of associative recall co-occurs with an increase of attention to relevant tokens. (left) Evolution of the performance of the model over time, as in Figure 6.5 left ($N_{\text{pairs}} = 32$ and $N_{\text{tokens}} = 256$). There is no repetition in this experiment. (middle) Evolution of the attention from the relevant value token to the relevant key token (the first layer of a two-layers manually constructed attention head would have this attention value equal to 1). (right) Evolution of the attention from the query token to the relevant value token (the second layer of the induction head would get this value equal to 1).

D.4 METHODS

D.4.1 *Measurement of the plateau length*

To quantify the emergence time of sparse attention, we measure the duration of the initial plateau in the learning dynamics. We define the plateau length as the number of training steps required for the model to reach a specific threshold performance from initialization. This performance threshold depends on the task we consider:

- For the linear regression task with the toy model, we define the plateau length as the time required for the test loss to decrease to 0.2 of its initial value when following the gradient flow dynamics, that is 0.1. The test loss is the loss without any form of repetition for the vanilla dynamics (Figure 6.2.a) or for the dynamics with cross-sample repetition (Figure 6.3 right), and with the same B value as during training for in-context repetition. Note that we use a test loss with repetition for the latter as the toy model cannot generalize to the no repetition case.
- For the linear regression task learned with a Transformer, we use the same threshold as above, except for the cross-sample repetition where we used a threshold of 0.4. This threshold being different partially explains why the lighter lines in Figure 6.4 middle and right do not map (the other reason being that we additionally tuned the learning rate for the right plot). The reason behind this change comes from the shape of the learning curves: after emergence, the learning speed can be drastically different for different p values (see Figure D.7, and hence we need it to be closer to 0.5, the initial loss value, to accurately capture the emergence time.
- For the in-context associative recall task, we pick an accuracy threshold, here 5%, as the random guess loss varies as a function of the vocabulary size.

It is important to note that in the first case, the plateau length corresponds to a physical time, whereas for the last two cases it corresponds to a number of optimization steps.

D.4.2 *Fitting power laws on plateau length*

To quantify the relationship between plateau length and various factors (sequence length T, input dimension d, in-context repetition B, cross-sample repetition probability p), we fit power laws to our empirical measurements using least squares regression on log-transformed data. We also fit power laws on the number of heads and the number of layers in D.8 according to similar principles to what is detailed below.

For the general form of our scaling laws, we assume:

$$T_{\text{plateau}} = C\, d^{\alpha} \left(\frac{T}{B}\right)^{\beta} f(p,d)^{\gamma} \tag{D.16}$$

where C is a constant, α, β, and γ are the scaling exponents, and $f(p,d) = \sqrt{p^2 d + (1-p)^2}$ is a function of the repetition probability (which is rooted in our theoretical analysis of Appendix D.1.4. Note that when considering the associative recall task, the vocabulary size is replacing d and the number of pairs is replacing T.

We perform the fitting by linearizing this relationship through a logarithmic transformation:

$$\log T_{\text{plateau}} = \log C + \alpha \log d + \beta \log \left(\frac{T}{B}\right) + \gamma \log f(p,d) \tag{D.17}$$

and performing a linear regression to find the missing coefficients. We report the score (R^2) of the linear fit as a measure of fit quality for all scaling laws. Most reported scaling laws had $R^2 > 0.98$, indicating excellent fit to the empirical data. For the cross-sample repetition results of Figure 6.4 right, where we could not fit accurately any power law on the obtained results.

More precisely, the exact parameters we fit for each plot are:

- Figure 6.2.c: C, α and β (B is fixed to 1).
- Figure 6.3 left: C, α and β (T is fixed to 4096).
- Figure 6.3 right: C, α, β and γ, such that $2\alpha = \beta = \gamma$ (T is fixed to 4096).
- Figure 6.4 left and middle: C, α and β. The same power law is fitted on the data of the two plots.

- Figure 6.5 right: C, α and β (B is fixed to 1).

D.4.3 *Architectural and training details*

We report the default hyperparameters we use in our experiments in Table D.1 and report the deviations we make to this setup below:

- In Figure 6.4 right, we additionally tune the learning rate (in $\{10^{-4}, 3 \cdot 10^{-4}\}$) based on the shortest average plateau length as we found cross-sample repetition to significantly change the learning rate the Transformers we considered need.
- In Figure 6.5 right, we additionally tune the learning rate (in $\{10^{-5}, 3 \cdot 10^{-5}, 10^{-4}\}$) as large vocabulary sizes and number of pairs need smaller learning rates.
- In Figure D.8, we ablate the number of layers and the number of heads in the Transformer architecture.
- In Figure D.10, we ablate the choice of Adam as optimizer and replace it by stochastic gradient descent. Note that we here consider a single layer and a single head.

It should be additionally noted that we adapt the number of iterations depending on the difficulty of the task and roughly aim to end training significantly after emergence. However, for the most challenging versions of the task (e.g., large T values in Figure 6.4), the phase transition sometimes occurs after the maximum number of training steps we attribute to the task (in that case 50k steps).

D.4.4 *Reproducibility*

Our theoretical simulations were run in JAX [308] and our Transformer experiments in PyTorch [323]. All our experiments were run on Nvidia RTX 3090 GPUs. The typical training run takes one hour, although training on shorter sequences can be significantly shorter. The code is pub-

Parameter Type	**Theory**	**Linear regression**	**Associative recall**
Architecture Parameters			
Model	Toy model	Transformer	Transformer
Layers	1	2	4
Number of heads	N/A	4	4
Embedding dimension	N/A	256	256
QK dimension	N/A	64	64
MLP factor	N/A	4	4
Positional encoding	N/A	sinusoidal	sinusoidal
Layer normalization	No	No	No
Skip connections	No	Yes	Yes
MLP between layers	No	Yes	Yes
Training Parameters			
Optimizer	GD	Adam	Adam
Learning rate	1.0	10^{-4}	10^{-4}
Scheduler	None	None	None
Batch size	Full	32	32
Seeds	N/A	5	3

TABLE D.1: Default hyperparameters for the different types of experiments.

licly available at the following url: `https://github.com/NicolasZucchet/The-emergence-of-sparse-attention`.

CURRICULUM VITAE

PERSONAL DATA

Name	Nicolas Zucchet
Date of Birth	January 16, 1997
Place of Birth	Clamart, France
Citizen of	France

EDUCATION

09/2021 – 09/2025	ETH Zürich, Zürich, Switzerland *Doctoral Studies*
09/2019 – 08/2021	ETH Zürich Zürich, Switzerland *Computer science MSc*
09/2016 – 08/2020	École polytechnique Palaiseau, France *Ingénieur polytechnicien (MSc)*
09/2014 – 08/2016	Lycée Hoche Versailles, France *Classe préparatoire aux grandes écoles (MPI*)*

EMPLOYMENT

09/2024 – 02/2025	Google DeepMind London, United Kingdom *Student researcher*
05/2021 – 08/2021	University of Bern Bern, Switzerland *Visiting student*
05/2019 – 08/2019	Prophesee Paris, France *Research intern*
06/2018 – 08/2018	Amadeus Sophia-Antipolis, France *Software developer intern*

PUBLICATIONS

1. Zucchet, N., Feng, Q., Laborieux, A., Zenke, F., Senn, W. & Sacramento, J. Teaching signal synchronization in deep neural networks with prospective neurons. *arXiv preprint arXiv:2511.14917* (2025).
2. Zucchet, N., d'Angelo, F., Lampinen, A. & Chan, S. The emergence of sparse attention: impact of data distribution and benefits of repetition. *Advances in Neural Information Processing Systems* (2025).
3. Zucchet, N., Bornschein, J., Chan, S., Lampinen, A., Pascanu, R. & De, S. How language models learn facts? Dynamics, curricula and hallucinations. *Conference on language modeling* (2025).
4. Zucchet, N. & Orvieto, A. Recurrent neural networks: vanishing and exploding gradients are not the end of the story. *Advances in Neural Information Processing Systems* (2024).
5. Zucchet, N., Kobayashi, S., Akram, Y., von Oswald, J., Larcher, M., Steger, A. & Sacramento, J. Gated recurrent neural networks discover attention. *arXiv preprint arXiv:2309.01775* (2023).
6. Von Oswald, J., Schlegel, M., Meulemans, A., Kobayashi, S., Niklasson, E., Zucchet, N., Scherrer, N., Miller, N., Sandler, M., Agüera y Arcas, B., Vladymyrov, M., Pascanu, R. & Sacramento, J. Uncovering mesa-optimization algorithms in transformers. *arXiv preprint arXiv:2309.05858* (2023).
7. Zucchet, N., Meier, R., Schug, S., Mujika, A. & Sacramento, J. *Online learning of long-range dependencies* in *Advances in Neural Information Processing Systems* (2023).
8. 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* (2022).
9. 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* in *Annual Workshop on Optimization for Machine Learning* (2022).

10. Zucchet, N., Schug, S., von Oswald, J., Zhao, D. & Sacramento, J. *A contrastive rule for meta-learning* in *Advances in Neural Information Processing Systems* (2022).
11. Zucchet, N. & Sacramento, J. Beyond backpropagation: bilevel optimization through implicit differentiation and equilibrium propagation. *Neural Computation* **34** (2022).
12. 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).