Nested Learning: The Illusion of Deep Learning Architecture

Ali Behrouz , Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni

Google Research

Abstract

Over the last decades, developing more powerful neural architectures and simultaneously designing optimization algorithms to effectively train them have been the core of research efforts to enhance the capability of machine learning models. Despite the recent progresses, particularly in developing Language Models (LMs), there are fundamental challenges and unanswered questions about how such models can *continually learn/memorize, self-improve, and find effective solutions*. In this paper, we present a new learning paradigm, called Nested Learning (NL), that coherently represents a machine learning model with a set of nested, multi-level, and/or parallel optimization problems, each of which with its own "*context flow*". Through the lenses of NL, existing deep learning methods learns from data through *compressing* their own context flow, and *in-context learning* naturally emerges in large models. NL suggests a philosophy to design more expressive learning algorithms with more "*levels*", resulting in higher-order in-context learning and potentially unlocking effective continual learning capabilities. In addition to its neuro-scientific motivation, we advocate for NL by presenting three core contributions: (1) Expressive Optimizers: We show that known gradient-based optimizers, such as Adam, SGD with Momentum, etc., are in fact associative memory modules that aim to compress the gradients' information (by gradient descent). Building on this insight, we present other "more expressive" optimizers with deep memory and/or more powerful learning rules; (2) Self-Modifying Learning Module: Taking advantage of NL's insights on learning algorithms, we present a sequence model that learns how to modify itself by learning its own update algorithm; and (3) Continuum Memory System: We present a new formulation for memory system that generalizes the traditional viewpoint of "long-term/short-term memory". Combining our self-modifying sequence model with the continuum memory system, we present a continual learning module, called Hope, showing promising results in language modeling, knowledge incorporation, and few-shot generalization tasks, continual learning, and long-context reasoning tasks.

> "We cannot solve our problems with the same thinking we used when we created them!"
>
> — Attributed to Albert Einstein

1 Introduction

For decades, AI research has focused on designing machine learning algorithms that learn from data (Pitts 1943; McCulloch et al. 1948; McCulloch 1949; Samuel 1959) or experience (Sutton et al. 1998; Connell et al. 1999; Silver et al. 2025); often by optimizing an objective $\mathcal{L}(\boldsymbol{\theta})$ over parameters $\boldsymbol{\theta} \in \Theta$ with gradient-based methods. While traditional machine learning techniques required careful engineering and domain expertise to design feature extractors, limiting their ability to directly process and learn from natural data (LeCun et al. 2015), deep representation learning offered a fully automated alternative to discover the representations needed for the task. Thereafter, deep learning has been an inseparable part of the large-scale computational models with seminal success in chemistry and biology (Jumper et al. 2021), games (Silver et al. 2016, 2018), computer vision (Krizhevsky et al. 2012; Dosovitskiy et al. 2021), and multimodal and natural language understanding (Achiam et al. 2023; Liu et al. 2024a; Comanici et al. 2025).

Stacking of multiple layers, as it is done in deep learning models, provides the models with better expressive power in representing complex features, and more internal computations (e.g., #FLOPS) (Montúfar et al. 2014; Poole et al. 2016; Hestness et al. 2017), all of which are critical and desirable characteristics for static tasks that require in-distribution predictions over an a-priori fixed set. This deep design, however, is not a universal solution to all the challenges and cannot help the expressive power of the models in multiple aspects, for example: (i) The computational depth of deep models might not change with more layers (Merrill et al. 2022; Sanford et al. 2024), leaving their ability to implement complex algorithms untouched compared to traditional shallow approaches (Merrill et al. 2024); (ii) The capacity of some class of parameters might show marginal improvement with increasing the depth/width of the model (Kaplan et al. 2020); (iii) The training process might converge to a suboptimal solution, mainly due to the suboptimal choice of the optimizer

§Correspondence to: {alibehrouz, razaviyayn, mirrokni}@google.com and peilin.zhong@columbia.edu.

§A version of this work is published at Neural Information Processing Systems (NeurIPS) 2025.

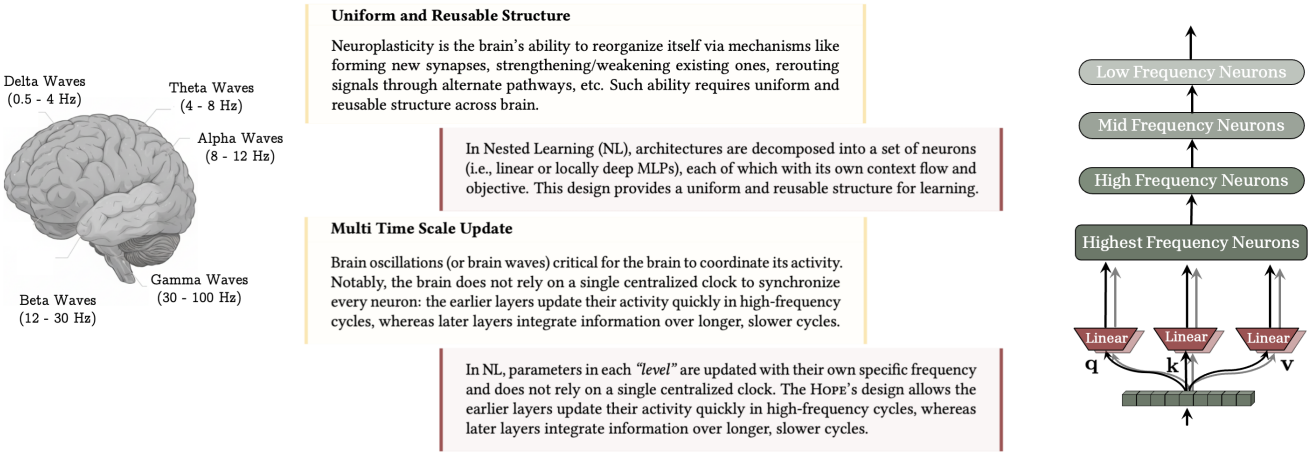

Figure 1: The uniform and reusable structure as well as multi time scale update in the brain are the key components to unlock the continual learning in humans. Nested Learning (NL) allows for multi time-scale update for each component of the brain, while showing that well-known architectures such as Transformers are in fact linear layers with different frequency updates.

or its hyperparameters; and (iv) The model's ability to fast adapt to a new task, continually learn, and/or generalize to out-of-distribution data might not change with stacking more layers and requires more careful designs.

The core part of the efforts to overcome the above challenges and to enhance the capability of deep learning models concentrate on: (1) developing more expressive class of parameters (i.e., neural architectures) (Fukushima 1980; Schmidhuber et al. 1997; Krizhevsky et al. 2012; Vaswani et al. 2017; Behrouz et al. 2025c); (2) introducing objectives that can better model the tasks (Rumelhart et al. 1986; Kingma et al. 2014b; Hjelm et al. 2019; Goodfellow et al. 2020; Alshammari et al. 2025); (3) designing more efficient/effective optimization algorithms to find better solutions or with more resilience to forgetting (Kingma et al. 2014a; Gupta et al. 2018; Farajtabar et al. 2020; Jordan et al. 2024); and (4) scaling the model size to enhance its expressivity, when the "right" choice of architecture, objective, and optimization algorithms are made (Brown et al. 2020; Kaplan et al. 2020; Hoffmann et al. 2022). Collectively, these advancements and new findings on scaling patterns of deep models have established the foundations upon which Large Language Models (LLMs) have been built.

The development of LLMs marks a pivotal milestone in deep learning research: a paradigm shift from task-specific models to more general-purpose systems with various emergent capabilities as a result of scaling the "right" architectures (Brown et al. 2020; Schaeffer et al. 2023). Despite all their success and remarkable capabilities in diverse sets of tasks (Nijkamp et al. 2023; Wang et al. 2023; Comanici et al. 2025), LLMs are largely static after their initial deployment phase, meaning that they successfully perform tasks learned during pre- or post-training, but are unable to continually acquire new capabilities beyond their immediate context. The only adaptable component of LLMs is their *in-context learning* ability–a (known to be emergent) characteristic of LLMs that enables fast adaption to the context and so perform zero- or few-shot tasks (Brown et al. 2020). Beyond in-context learning, recent efforts to overcome the static nature of LLMs either are computationally expensive, require external components, lack generalization, and/or might suffer from catastrophic forgetting (Akyürek et al. 2024a; Eyuboglu et al. 2025; yu et al. 2025), which has led researchers to question if there is a need to revisit how to design machine learning models and if a new learning paradigm beyond stacking of layers is required to unleash the capabilities of LLMs in continual setups.

Current Models only Experience the Immediate Present. As an analogy and to better illustrate the static nature of LLMs, we use the example of anterograde amnesia–a neurological condition where a person cannot form new long-term memories after the onset of the disorder, while existing memories remain intact (Scoville et al. 1957). This condition limits the person's knowledge and experiences to a short window of present and long past–before the onset of the disorder–which results in continuously experiencing the immediate present as if it were always new. The memory processing system of current LLMs suffer from a similar pattern. Their knowledge is limited to either, the immediate context that fits into their context window, or the knowledge in MLPs that stores long-past, before the onset of *"end of pre-training."* This analogy, has motivated us to take inspiration from neurophysiology literature and how brain consolidate its short-term memories.

1.1 Human Brain Perspective and Neurophysiological Motivation

Human brain is highly efficient and effective when it comes to continual learning, which is often attributed to neuroplasticity—the brain's remarkable capacity to change itself in response to new experiences, memories, learning, and even damage (Pascual-Leone et al. 2005; Johnston 2009). Recent studies support that the formation of Long-term memory involves at least two distinct but complementary consolidation processes (Frey et al. 1997; Goto et al. 2021; Yang et al. 2024): (1) A rapid "online" consolidation (also known as synaptic consolidation) phase occurs immediately or soon after learning, even during wakefulness. This is when new and initially fragile memory traces are stabilized and begin transferring from short-term to long-term storage; (2) An "offline" consolidation (also known as systems consolidation) process repeats the replay of the recently encoded patterns—during sharp-wave ripples (SWRs) in the hippocampus, coordinated with cortical sleep spindles and slow oscillations—strengthens and reorganizes the memory and supports transfer to cortical sites (Foster et al. 2006; Ji et al. 2007; Peyrache et al. 2009).

Coming back to the analogy of anterograde amnesia, evidence indicates that the condition can impact both stages, but especially the online consolidation phase, mainly due to the fact that hippocampus is the gateway for encoding new declarative memories, and so its damage means new information never will be stored in long-term memory. As mentioned above, the design of LLMs, and more specifically Transformer-based backbones, suffers from a similar condition after the pre-training phase. That is, the information provided in the context, never impacts the long-term memory parameters (e.g., feedforward layers), and so the model is not capable of acquiring new knowledge or skill, unless the information is still stored in the short-term memory (e.g., in-context or attention). To this end, although the second stage is equally, or even more, crucial for the consolidation of memories, and its absence can damage the process and might cause loss of memory (Drummond et al. 2000; Yoo et al. 2007), in this work, we focus on the first stage: memory consolidation as an online process. As discussed earlier, the memory processing, its online consolidation, and so continual learning ability of humans are known to be highly relied on the neuroplasticity as well as neural oscillations (Bliss et al. 1993; Buzsaki et al. 2004; Klinzing et al. 2019).

Multi Time scale Processing System. Brain oscillations (also known as brainwaves)–a rhythmic fluctuations in brain activity–is not mere byproducts of brain function but is increasingly understood to play a crucial role in various cognitive functions such as attention, memory, and decision-making, and to be a core mechanism for organizing neural computation, coordinating communication between brain regions, and gating the synaptic plasticity that underlies learning and memory (Fell et al. 2011; Cavanagh et al. 2014; Fries 2015). These brainwaves are the results of the brain coordinating its computations in different timescales and frequency updates, where each frequency determines how often groups of brain neurons become active and share updated information. More specifically, such neural oscillations are typically categorized into distinct frequencies, each of which has been associated with different cognitive functions and, critically, different timescales of information processing: ranging from (1) fast Gamma waves (frequency of 30-150 Hz) that are mainly associated with sensory information to (2) Beta waves (frequency of 13 - 30 Hz) that are mainly associated with active thinking (Buzsaki et al. 2004; Buschman et al. 2007; Lundqvist et al. 2016), and (3) slow Delta and Theta waves (frequency of 0.5 - 8 Hz), mainly responsible for memory consolidation and learning (Marshall et al. 2006; Diekelmann et al. 2010; Ngo et al. 2013; Staresina et al. 2015; Heusser et al. 2016; Daume et al. 2024).

In deep learning models, however, the weights of the architectures are fixed at test time and also it is common in pre-training to use the same update rate for all the blocks/layers in the model. Later, in Section 6, however, we show that in-context learning provides an extreme case of this design and in fact, Transformer architectures are based on two extreme frequencies of update: i.e., ∞ and 0 for attention and MLP blocks, respectively.

Brain's Uniform and Reusable Structure. As discussed earlier, neuroplasticity is the brain's remarkable capability to change itself in response to new memories, knowledge, and even damage (Pascual-Leone et al. 2005; Johnston 2009). This characteristic suggests a uniform architecture where neural elements are not rigidly dedicated to one function but are instead reusable, capable of being flexibly redeployed to support different cognitive needs. One real-world example of neural reusability is hemispherectomy–the surgical removal or disabling of one cerebral hemisphere, usually to alleviate severe epilepsy. Amazingly, if this surgery is done in childhood, patients can lead largely normal lives into adulthood with high functioning cognition and intact neural network organization that contains all the same core brain networks present in a typical two-hemisphere brain (networks for language, vision, etc.). This extraordinary outcome provides real-life proof of the brain's uniform architecture. That is, even half a brain can reallocate resources and reorganize so that the person can function extremely well. Such cases, along with documented instances of individuals living relatively normally with missing pieces of cortex, highlight the brain's uniform and reusable structure.

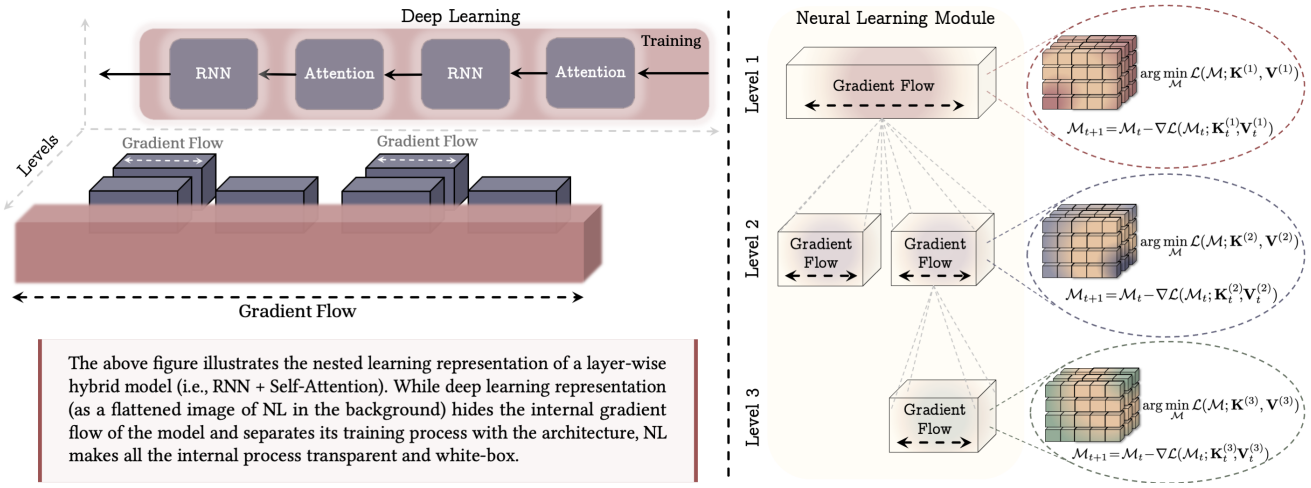

Figure 2: Nested Learning Paradigm that represent a machine learning model and its training procedure as a set of nested optimization problems. (**Left**) An example of Hybrid architecture. While deep learning perspective, as the flattened image of NL, does not provide insight about the depth of computation in the blocks, NL transparently represent all the inner gradient flows. (**Right**) A Neural Learning Module: A computational model that learns how to compress its own context flow. For example, the first level corresponds to the model's most outer-loop training, often refer to as "*pre-training*" step.

Furthermore, this interpretation of brain's uniform and reusable structure demonstrate that memory in human brain is not an isolated system in some specific areas, and mainly is distributed across brain. That is, contrary to the traditional models of memory that often implied that different types of memory reside in distinct brain structures (e.g. short-term memory in frontal cortex vs. long-term memory in the hippocampus and cortex), modern research advocate for distributed neural circuits memory processing across multiple regions (Christophel et al. 2017; Kitamura et al. 2017; Roy et al. 2022).

The modern deep learning architectures in recent years, however, at least on the surface, seem to be heterogeneous and are based on a combination of a subset of self-attention variants (Vaswani et al. 2017), modern recurrent neural networks (Katharopoulos et al. 2020; Schlag et al. 2021; Behrouz et al. 2025c; Peng et al. 2025b), canon layers (Allen-Zhu 2025), global convolutions (Hasani et al. 2023; Poli et al. 2023), and MLP blocks (Shazeer 2020). This raises the question of whether we need a new uniform architecture, or if our beliefs about the heterogeneity of current models need to be revisited.

1.2 Contributions and Roadmap

In this paper, we aim to present a unifying learning paradigm that not only provides new insights about existing algorithms, methods, and architectures, but it also reveals a new dimension to stacking layers in deep learning with enhancement of the computational depth, and continual learning ability of models. After discussing preliminary concepts and backgrounds in §2, we present:

Nested Learning Paradigm (§3). To answer the questions raised above and to provide new insights on overcoming the design challenges in continual learning, architecture design, and computational depth of modern deep learning models, we present Nested Learning (NL)–a learning paradigm that allows each component of the machine learning model to have its own internal gradient flow on its own context in multiple levels, representing a model and its learning process (i.e., optimization) as an inter-connected system of nested, multi-level, and/or parallel optimization problems. We argue that the optimization process and the learning algorithms/architectures are fundamentally the same concepts but are in different levels of a system with different context (i.e., gradient vs. tokens). Furthermore, they are two inter-connected components where the learning algorithm/architecture generates the context for optimizers (i.e., the gradients), advocating for designing architecture-specific optimizers. We discuss different ways of knowledge transfer between levels, resulting in unifying and generalizing concepts like meta-learning, in-context learning, recurrent neural networks, hypernetworks, etc.

Optimizers and Architectures as Learning Module (§4, §5). Building on the NL's viewpoint, we argue that training *a deep neural network* with backpropagation process and gradient descent is a compression and an optimization problem that aims to train an associative memory to map layers' inputs to their corresponding local error in the prediction. Accordingly, we argue that pre-training is a form of in-context learning, where the context is the entire pre-training data and the layers are compressing the context into their parameters. We demonstrate that such arguments are also valid for other popular gradient-based optimizers and they are associative memories that aim to compress the gradients into their parameters. From NL's terminology, gradient-based optimizers such as gradient descent with momentum, Adam (Kingma et al. 2014a), and AdaGrad (Duchi et al. 2011) can be decomposed into a two-level nested optimization problems, each of which is optimized with a simple gradient descent. In particular, this viewpoint makes it apparent that for compressing the gradients, in theory, Adam is the optimal associative memory with respect to the element-wise L_2 regression objective.

We revisit previous findings on representing architectures as associative memories (Behrouz et al. 2025b) and decompose their optimization process into a set of nested optimization problems, all of which optimized with gradient descent. Building on the above findings–i.e., popular gradient-based optimizers and modern architectures are both a set of nested and/or parallel optimization problems–we argue that the combination of these two–i.e., training an architecture with a specific optimizer–can also be represented as a set of nested and/or parallel optimization problem. Therefore, a neural learning module (a joint system of architecture and its training/optimization process) is a uniform model, in which all elements are linear or deep MLPs, while they are optimizing their own internal objective in different levels with different frequencies.

Building upon the associative memory perspective of optimizers, we design a set of new learning updates (optimization steps) with more expressive memory structure or memory management in compressing the gradients. In particular, we argue that the choice of optimizer depends on the context of the optimization. A powerful optimizer for compressing the gradients might not be the best choice for compressing the tokens. To this end, we present a new variant of gradient descent, called Delta Gradient Descent (DGD), that its update not only depends on the current input, but also the state of the weight of the neural network, resulting in capturing the dependencies of data samples without i.i.d. assumption.

Main Takeaways and Revisiting Common Terms: Continual and In-context Learning, Pre-Training, and Learning (§6). We discuss the main takeaways of NL about principal concepts and revisit some common terms: (1) We argue that continual learning can be viewed as a learning problem on a sequences of incoming contexts or episodes where different levels are responsible for compressing their own in-context knowledge and transfer it to higher levels. Based on this, we advocate for designing models and pipelines that do not rely on test/training phase, and rather continually manage their knowledge and memory; (2) In-context learning is the characteristic of "having multiple nested levels". Accordingly, Transformers in-context learning comes from being a non-parametric solution to a certain regression objective on tokens, while modern recurrent models uses parametric learning processes in their lower levels; (3) We further revisit other terminologies such as learning/memorization, hybrid architectures, looped architectures, and learned optimizers.

Continuum Memory System, Self-Referential Titans, and Hope (§7, §8). We generalize the traditional viewpoint of "long-term/short-term memory" (LSM) by presenting the Continuum Memory Systems (CMSs) and see memory as a distributed inter-connected system with a spectrum of frequency updates. In this design, higher-frequency neurons are responsible for fast adaption but store memories/knowledge for a short period of time, while lower frequency neurons are responsible for more persistent knowledge. Comparing to LSM, we show that this multi-frequency design results in a loop process for memory of the model, meaning that knowledge can partially be recovered when it is forgotten. While we mainly design this memory system as a replacement of MLP blocks in Transformers, we take advantage of this intuition to design Multi-scale Momentum Muon (M3) optimizer–an optimization algorithm with multiple momentum terms–further supporting the importance of CMSs design in different contexts.

Evaluations (§9). To support the effectiveness of our proofs-of-concept as well as the importance of nested learning design, we perform experimental evaluation on (1) Continual learning and in-context learning tasks including (i) learning new language, (ii) class incremental learning, and (iii) question/answering on a new corpus; (2) Long context understanding tasks, including needle-in-a-haystack (Hsieh et al. 2024) and BABILong (Kuratov et al. 2024) benchmarks; (3) Language modeling and common-sense reasoning tasks; (4) In-context recall and memorization tasks; (5) Language recognition tasks; and (6) comparing different optimizers, including our M3 optimizer. Our results indicate the effectiveness of NL viewpoint in designing models with continual learning ability, having multiple levels of computations, and self-referential process.

2 Preliminaries

In this section we discuss the notations and review the background concepts.

Notations. We let $x \in \mathbb{R}^{N \times d_{\text{in}}}$ be the input, $\mathcal{M}_t$ represent the state of memory/model $\mathcal{M}$ at time t, $\mathbf{K}$ be the keys, $\mathbf{V}$ be the values, and $\mathbf{Q}$ be the query matrices. We use bold lowercase letters with subscript t to refer to the vector corresponds to the input t (i.e., $\boldsymbol{k}_t, \boldsymbol{v}_t$, and $\boldsymbol{q}_t$). We further refer to the distribution of any random variable $\mathcal{T}$ as $p(\mathcal{T})$. Throughout the paper, we use simple MLPs with $\mathcal{L}_{\mathcal{M}} \geq 1$ layers and residual connection as the architecture of the memory module $\mathcal{M}(\cdot)$. When it is needed, we parameterized the memory module with $\boldsymbol{\theta}_{\mathcal{M}} \supseteq \{W_1, W_2, \ldots, W_{\mathcal{L}_{\mathcal{M}}}\}$, which at least includes the parameters of linear layers in the MLP. We use superscript with parenthesis to refer to parameters in different *levels* of nested learning (different update frequency): i.e., either by using level index $W^{(\ell)}$ or its corresponding frequency $W^{(f_\ell)}$.

Gradient Descent. Gradient descent is one of the most widely used optimization algorithms for high-dimensional problems, and its variants are standard tools for training large models. Given an objective $\mathcal{L}(\cdot;\cdot)$, the stochastic gradient descent (SGD) with step size $\eta_t > 0$ (a.k.a. learning rate) updates parameters by:

$$W_{t+1} \;=\; W_t - \eta_t \nabla_{W_t} \mathcal{L}(W_t; \boldsymbol{x}_t), \tag{1}$$

for data sample $\boldsymbol{x}_t$ from training set. The gradient descent formulation admits several equivalent characterizations that are useful in analysis. One of those equivalent formulations is steepest-descent in the Euclidean metric, where one step of gradient descent is equivalent to:

$$W_{t+1} \;=\; \arg\min_{W} \left\{ \langle \nabla_W \mathcal{L}(W_t; \boldsymbol{x}_t),\, W \rangle \;+\; \tfrac{1}{2\eta_t} \|W - W_t\|_2^2 \right\}, \tag{2}$$

which is minimizing a first-order Taylor approximation regularized by a quadratic proximal term. The GD step above is precisely a proximal update on the linearization of $\mathcal{L}(\cdot;\cdot)$ at W_t, revealing an implicit bias toward small moves in L_2-distance. Accumulating steps (with a constant learning rate η) yields the *follow-the-regularized-leader* (FTRL) form

$$W_{t+1} \;=\; \arg\min_{W} \left\{ \left\langle \sum_{s=1}^{t} \nabla \mathcal{L}(W_s; \boldsymbol{x}_s),\, W \right\rangle \;+\; \frac{1}{2\eta} \|W - W_1\|_2^2 \right\}, \tag{3}$$

whose solution is $W_{t+1} = W_1 - \eta \sum_{s=1}^{t} \nabla \mathcal{L}(W_s; \boldsymbol{x}_s)$. These two formulations are used in this paper interchangeably. However, our discussions and formulations are generally valid for many other optimization algorithms as well.

Meta Learning. Designing an effective machine learning model often requires making decisions about its architecture parameterized by $\boldsymbol{\theta} \in \Theta$, objective $\mathcal{L}(\theta)$, and an optimizer, aiming to iteratively optimize the objective. Meta learning paradigm (or learning to learn) (Schmidhuber et al. 1996; Finn et al. 2017; Akyürek et al. 2022) aim to automate a part of such decisions by modeling it as a *two-level* optimization procedure, in which the outer model aims to learn to set parameters for the inner procedure to maximize the performance across a set of tasks. That is, given an objective parameterized by a parameter Φ: i.e., $\ell(\boldsymbol{\theta}, \mathcal{D}; \Phi)$, one can formalize the outer loop process as optimizing parameter Φ over a set of tasks:

$$\Phi^* = \arg\min_{\Phi} \; \mathbb{E}_{\mathcal{T}_i \sim p(\mathcal{T})} \left[\ell(\theta, \mathcal{T}_i; \Phi) \right], \tag{4}$$

where $p(\mathcal{T})$ is the distribution of tasks. While initial studies on meta-learning used supervised settings for the outer loop (Schmidhuber et al. 1996), recently, a more flexible family of methods that use an unsupervised process for the outer loop has gained popularity (Finn et al. 2017; Brown et al. 2020; Akyürek et al. 2022; Chen et al. 2022; Qu et al. 2025). In addition to the growing interest to use meta learning methods for a diverse set of downstream tasks (Finn et al. 2017; Munkhdalai et al. 2019; Chen et al. 2022; Qu et al. 2025), in recent years, it also has shown popularity as a paradigm to design powerful sequence models (Sun et al. 2024; Behrouz et al. 2025c).

Fast Weight Programmers (FWPs). Fast weight programmers (more recently refer to as linear Transformers) (Hinton et al. 1987; Schmidhuber 1992; Ba et al. 2016; Schlag et al. 2021) are recurrent neural networks whose memory (or hidden state) is *matrix-valued*: a time-varying fast-weight matrix $\mathcal{M}_t \in \mathbb{R}^{d_{\text{out}} \times d_{\text{key}}}$ that serves as a short-term memory. A separate

"programmer" (slow net) maps each input $\boldsymbol{x}_t \in \mathbb{R}^{d_{\text{in}}}$ to query, key, and value vectors and updates the fast weights online. A basic (Hebbian/outer-product) FWP—often called *vanilla* FWP—update its parameters with:

$$\mathcal{M}_t = \alpha_t \mathcal{M}_{t-1} + \boldsymbol{v}_t \phi(\boldsymbol{k}_t)^\top, \tag{5}$$

and retrieve from memory with $y_t = \mathcal{M}_t \, \phi(q_t)$, where $\phi(\cdot)$ is an element-wise feature map (often applied to both keys and queries). Unlike traditional RNNs or early variants of modern RNNs (Schmidhuber et al. 1997; Sun et al. 2023; Botev et al. 2024) with vector states, the *matrix* $\mathcal{M}_t$ is the recurrent state; it is *written* by rank-one update and *read* by a matrix–vector multiplication, providing a compact, learnable key–value memory with constant state size across time.

In-context Learning. The concept of "in-context learning" initially defined by Brown et al. (2020) as the ability of a language model to leverage knowledge acquired during pre-training in order to infer and perform a new task solely based on its context (e.g., few examples, or natural language instructions). This broad and general definition, which simply is applicable for any language model with any architectural backbones and/or objective, later was formalized in a way that only described in-context learning for Transformer architectures trained with next token prediction objectives. Accordingly, despite extensive research on the algorithms/problems that a transformer-based model can learn in-context (Akyürek et al. 2022, 2024b; Zhang et al. 2024a; Dherin et al. 2025), the in-context learning as its general form is relatively underexplored. Throughout this paper, we use the most general definition of "in-context learning" and refer to it as the ability of a model to adapt itself to and learn from a given context. Our NL formulation connects ICL with the concept of associative memory, offering a unified explanation for the ICL capabilities of models regardless of their architectural backbone and/or objectives.

3 Nested Learning

This section discusses the motivations, formal definitions, and general high-level implications of Nested Learning (NL). We start with a formulation of associative memory and then by using step-by-step examples, we build the intuition behind architecture decomposition and its connection to modeling a neural network as an integrated system of optimization problems. We aim to first show how existing methods and concepts in deep learning fall under the NL paradigm and then we present new formulations that go beyond traditional methods and/or provide insights on how to improve existing algorithms and designs.

3.1 Associative Memory

Associative memory—the ability to form and retrieve connections between events—is a fundamental mental process and is an inseparable component of human learning (Terry 2017). Often in the literature, the concept of memorization and learning are used interchangeably; in neuropsychology literature, however, these two are clearly distinguished. More specifically, following neuropsychology literature (Okano et al. 2000), we build our terminology based on the following definition of memory and learning:

> **Learning vs. Memorization:**
>
> *Memory is a neural update caused by an input, and learning is the process for acquiring effective and useful memory.*

In this work, our goal is to first show that all the elements of a computational sequence model, including optimizers and neural networks, are *associative memory systems* that compress their own *context flow*. Broadly speaking, associative memory is an operator that maps a set of keys to a set of values. We follow the general definition of associative memory by Behrouz et al. (2025b):

Definition 1 (Associative Memory)**.** *Given a set of keys $\mathcal{K} \subseteq \mathbb{R}^{d_k}$ and values $\mathcal{V} \subseteq \mathbb{R}^{d_v}$, associative memory is an operator $\mathcal{M}(\cdot)$ that maps the set of keys $\mathcal{K}$ to values $\mathcal{V}$. To learn such mapping from the data, an objective $\tilde{\mathcal{L}}(\cdot;\cdot)$ measures the quality of the mapping and $\mathcal{M}$ can be computed by:*

$$\mathcal{M}^* = \arg\min_{\mathcal{M}} \quad \tilde{\mathcal{L}}(\mathcal{M}(\mathcal{K}); \mathcal{V}). \tag{6}$$

While the operator itself is a memory and the mapping acts as a memorization process (i.e., memorizing the connections of events in the context), acquiring such effective operator based on the data, is a learning process. Notice that, here, keys and values can be any arbitrary event that memory aims to map them and are not limited to tokens. Later, we will discuss that given a context flow, keys and values might be tokens, gradients, sub-sequences, etc. Furthermore, while the term of associative memory is more common in neuroscience and neuropsychology literature, the above formulation is also closely related to data compression and low-dimensional representation. That is, one can interpret the optimization process in Equation 6 as the training process of a network $\mathcal{M}(.)$ that aims to compress the mappings into its parameters, representing them in a lower dimensional space.

In sequence modeling, where keys and values are input tokens (e.g., tokenized text), the choice of objective and the optimization process for solving Equation 6 can result in distinct sequence modeling architectures (see Liu et al. 2024b and Behrouz et al. 2025b) such as global/local softmax attention (Vaswani et al. 2017), or other modern recurrent models (Katharopoulos et al. 2020; Sun et al. 2023; Behrouz et al. 2025c). This simple formulation of sequence models provides us with better understanding of their internal process and also a tool to simply compare their modeling power based on their objective and optimization process. In the following, using step-by-step examples, we discuss how this formulation can be applied to all components of a neural architecture (including its optimization process in pre-training) and in fact, how a model is an integrated system of multi-level, nested, potentially parallel memories, each of which with its own context flow.

A Simple Example of MLP Training. We start with a simple example, in which we aim to train a 1-layer MLP (parameterized with W) for task $\mathcal{T}$ and on dataset $\mathcal{D}_{\text{train}} = \{x_1, \ldots, \boldsymbol{x}_{|\mathcal{D}_{\text{train}}|}\}$ by optimizing the objective $\mathcal{L}(\cdot;\cdot)$ with gradient descent. In this case, the training process objective is to solve the following optimization problem:

$$W^* = \arg\min_{W} \; \mathcal{L}(W; \mathcal{D}_{\text{train}}), \tag{7}$$

whose optimization by (stochastic/online) gradient descent results in a weight update rule:

$$W_{t+1} = W_t - \eta_{t+1} \underbrace{\nabla_W \mathcal{L}(W_t; \boldsymbol{x}_{t+1})}_{\text{Surprise}} = W_t - \eta_{t+1} \underbrace{\nabla_{y_{t+1}} \mathcal{L}(W_t; \boldsymbol{x}_{t+1})}_{\text{Surprise in the Output}} \otimes \boldsymbol{x}_{t+1}, \qquad \text{where } \boldsymbol{x}_{t+1} \sim \mathcal{D}_{\text{train}}, \tag{8}$$

where $y_{t+1} = W x_{t+1}$ is the output of the model for input x_{t+1} and we used the simplifying notation: $\nabla_{y_{t+1}} \mathcal{L}(W_t; \boldsymbol{x}_{t+1}) := \frac{\partial \mathcal{L}}{\partial y}|_{y=W x_{t+1}}$. In this case, $\nabla_W \mathcal{L}(W_t; \boldsymbol{x}_{t+1})$ is the surprise metric that shows how much the current input is different from previously observed data. Similarly, $\nabla_{y_{t+1}} \mathcal{L}(W_t; \boldsymbol{x}_{t+1})$ is the surprised metric for the output (or more accurately *local surprise signal in representation space* that quantifies the mismatch between the current output and the structure the objective $\mathcal{L}(\cdot;\cdot)$ enforces)–measuring how much surprising the model's prediction is for this input. Given this formulation, one can let the surprise value of output be $u_{t+1} = \nabla_{y_{t+1}} \mathcal{L}(W_t; \boldsymbol{x}_{t+1})$ and reformulate the backpropagation process as the solution to an optimization problem on finding an associative memory that maps input data points $\mathcal{D}_{\text{train}} = \{x_t\}_{t=1}^{|\mathcal{D}_{\text{train}}|}$ to their corresponding $u_{t+1} = \nabla_{y_{t+1}} \mathcal{L}(W_t; \boldsymbol{x}_{t+1})$. That is, we let $\mathcal{M}(\cdot) = W_t \cdot$ parametrizes the memory, and use dot-product similarity to measure the quality of W_t's mapping between x_{t+1} and $\nabla_{y_{t+1}} \mathcal{L}(W_t; \boldsymbol{x}_{t+1})$:

$$W_{t+1} = \arg\min_{W} \; \langle W \boldsymbol{x}_{t+1}, u_{t+1} \rangle + \frac{1}{2\eta_{t+1}} \|W - W_t\|_2^2 = \arg\min_{W} \; \langle W \boldsymbol{x}_t, \nabla_{y_{t+1}} \mathcal{L}(W_t; \boldsymbol{x}_{t+1}) \rangle + \frac{1}{2\eta_{t+1}} \|W - W_t\|_2^2. \tag{9}$$

Therefore, this formulation translates the training phase of the model as a process of acquiring effective memory that maps data samples to their Local Surprise Signal (LSS) in representation space–measuring how surprising its corresponding output is. This gradient can be viewed as an error in the prediction (with gradient being zero when the loss is minimized). Later in Section 4, we discuss the backpropagation process as an associative memory in more details, but as a preliminary takeaway from this simple example:

> **Training a Linear Layer with Backpropagation as a Surprise-based Memory:**
>
> *A linear layer trained with backpropagation learns from data by memorizing how surprising their predicted outputs are; i.e., backpropagation can be viewed as an associative memory that maps each data sample to the error of its corresponding prediction.*

Accordingly, in this example, our model has *a single gradient flow* over the data samples, which is only active over dataset $\mathcal{D}_{\text{train}} = \{x_1, \ldots, \boldsymbol{x}_{|\mathcal{D}_{\text{train}}|}\}$ and *will be frozen for any other data samples afterwards (i.e., inference or test time).*

In the above example, we can replace the gradient descent algorithm with its momentum-based variant, resulting in the update rule of:

$$W_{t+1} = W_t - \boldsymbol{m}_{t+1}, \tag{10}$$

$$\boldsymbol{m}_{t+1} = \boldsymbol{m}_t + \eta_{t+1} \nabla_W \mathcal{L}(W_t; \boldsymbol{x}_{t+1}) = \boldsymbol{m}_t + \eta_{t+1} \nabla_{y_{t+1}} \mathcal{L}(W_t; \boldsymbol{x}_{t+1}) \otimes \boldsymbol{x}_{t+1}. \tag{11}$$

In Equation 11, given the previous state of Equation 10 (at time t), the value of $\nabla_W \mathcal{L}(W_t; \boldsymbol{x}_{t+1})$ or similarly $\nabla_{y_{t+1}} \mathcal{L}(W_t; \boldsymbol{x}_{t+1})$ does not depend on the output of recurrence in Equation 11 and so can be pre-computed beforehand: Leting $u_{t+1} = \nabla_W \mathcal{L}(W_t; \boldsymbol{x}_{t+1})$, Equation 11 can be reformulated as:

$$W_{t+1} = W_t - \boldsymbol{m}_{t+1}, \tag{12}$$

$$\boldsymbol{m}_{t+1} = \arg\min_{\boldsymbol{m}} \; -\langle \boldsymbol{m}, \nabla_{W_t} \mathcal{L}(W_t; \boldsymbol{x}_{t+1}) \rangle + \frac{1}{2\eta_{t+1}} \|\boldsymbol{m} - \boldsymbol{m}_t\|_2^2 = \arg\min_{\boldsymbol{m}} \; -\langle \boldsymbol{m}\, \boldsymbol{x}_{t+1}, \nabla_{y_{t+1}} \mathcal{L}(W_t; \boldsymbol{x}_{t+1}) \rangle + \frac{1}{2\eta_{t+1}} \|\boldsymbol{m} - \boldsymbol{m}_t\|_2^2. \tag{13}$$

Given this formulation, one can interpret the momentum term as either: (1) a value-less associative memory that compress the gradients into its parameters, or (2) an associative memory that learns how to map data points to their corresponding LSS-value. Interestingly, this formulation reveals that gradient descent with momentum can be viewed as a two-level optimization procedure, where the memory is optimized by simple gradient descent algorithm[1].

Concluding the above examples, we observed that the training process of a 1-layer MLP with: (1) Gradient descent is a *1-level* associative memory that learns how to map data points to their corresponding LSS-value; and (2) Gradient descent with momentum is a *2-level* associative memory (or optimization process) that the inner-level learns to store gradient values into its parameters, and then the outer-level updates the slow weight (i.e., W_t) with the value of the inner-level memory. While these are the most simple examples with respect to both architecture and optimizer algorithms, one might ask if similar conclusion can be made in more complex setups.

An Example of Architectural Decomposition. In the next example, we replace the MLP module in our previous example with a linear attention (Katharopoulos et al. 2020). That is, we aim to train a 1-layer linear attention for task $\mathcal{T}$ and on a sequence of $\mathcal{D}_{\text{train}} = \{x_1, \ldots, \boldsymbol{x}_{|\mathcal{D}_{\text{train}}|}\}$ by optimizing the objective $\mathcal{L}$ with gradient descent. Recalling the unnormalized linear attention formulation:

$$\boldsymbol{k}_t = W_{\boldsymbol{k}} \boldsymbol{x}_t, \qquad \boldsymbol{v}_t = W_{\boldsymbol{v}} \boldsymbol{x}_t, \qquad \boldsymbol{q}_t = W_{\boldsymbol{q}} \boldsymbol{x}_t, \tag{14}$$

$$\mathcal{M}_t = \mathcal{M}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\top, \tag{15}$$

$$y_t = \mathcal{M}_t \boldsymbol{q}_t \; . \tag{16}$$

As discussed in earlier studies (Liu et al. 2024b; Behrouz et al. 2025b), the recurrence in Equation 15 can be reformulated as the optimization process of a matrix-valued associative memory $\mathcal{M}_t(\cdot)$ to compress the mappings of keys and values into its parameters. Specifically, in Definition 1, if we let $\tilde{\mathcal{L}}(\mathcal{M}_{t-1}; \boldsymbol{k}_t, \boldsymbol{v}_t) := -\langle \mathcal{M}_{t-1} \boldsymbol{k}_t, \boldsymbol{v}_t \rangle$ and aim to optimize the memory with gradient descent, the memory update rule is: (Note that $\nabla \tilde{\mathcal{L}}(\mathcal{M}_{t-1}; \boldsymbol{k}_t, \boldsymbol{v}_t) = -\boldsymbol{v}_t \boldsymbol{k}_t^\top$ and we let learning rate $\eta_t = 1$)

$$\mathcal{M}_{t+1} = \arg\min_{\mathcal{M}} \; -\langle \mathcal{M} \boldsymbol{k}_{t+1}, \boldsymbol{v}_{t+1} \rangle + \frac{1}{2} \|\mathcal{M} - \mathcal{M}_t\|_2^2 \tag{17}$$

$$\Rightarrow \; \mathcal{M}_{t+1} = \mathcal{M}_t - \nabla \tilde{\mathcal{L}}(\mathcal{M}_t; \boldsymbol{k}_{t+1}, \boldsymbol{v}_{t+1}) = \mathcal{M}_t + \boldsymbol{v}_{t+1} \boldsymbol{k}_{t+1}^\top, \tag{18}$$

which is equivalent to the update rule of an unnormalized linear attention in Equation 15. Also, as we observed in the first example, training a linear layer with gradient descent can be viewed as a 1-level optimization of an associative memory (Equation 8) and so the general training/updating process of projection layers (i.e., $W_{\boldsymbol{k}}$, $W_{\boldsymbol{v}}$, and $W_{\boldsymbol{q}}$) is itself an optimization process of associative memory. Therefore, training a linear attention with gradient descent can be seen as a two-level optimization process, where the outer-loop (also known as training process) optimizes the projection layers with gradient descent, while the inner-loop optimizes the inner memory of $\mathcal{M}_t$ with gradient descent.

In the examples we discussed so far, we have two associative memories, each of which has their own optimization process and gradient flow. That is, in the optimization of the outer-level parameters of $W_{\boldsymbol{k}}, W_{\boldsymbol{v}}$, and $W_{\boldsymbol{q}}$ there is no gradient

[1] We use the term two- or multi-level to describe the optimization *procedure*. This differs from classical multi-level optimization, where the optimization *problems* are arranged hierarchically.

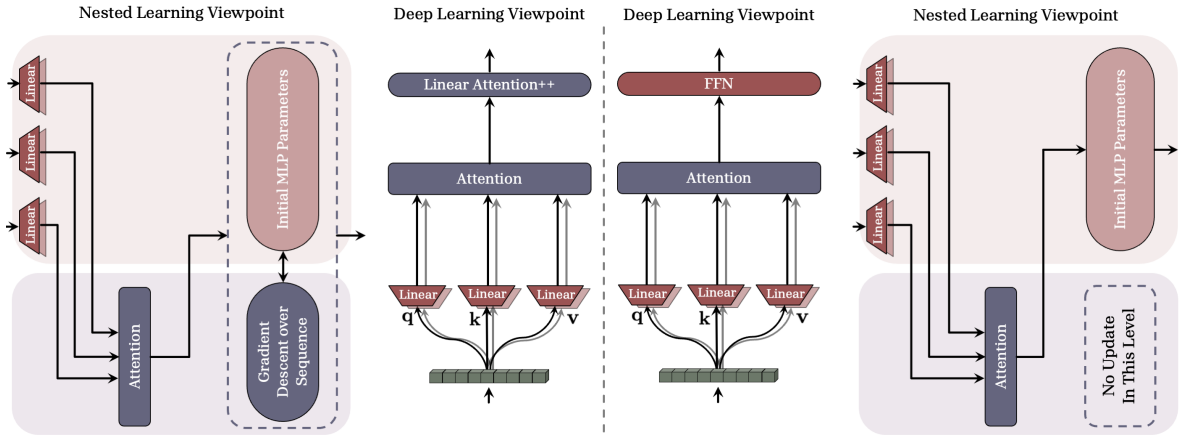

Figure 3: An example of comparing a FFN (e.g., MLP) with linear attention in a Transformer-based backbone, optimizing with gradient descent. The red components are blocks in the first level (with frequency 1), while blue components are blocks in the second level (frequency L). Linear attention with learnable initial memory state (referred to as Linear Attention++) is the same as an MLP layer but with in-context learning ability and adaptation to the input sequence.

with respect to parameter $\mathcal{M}(\cdot)$ and so there is no backpropagation through it. Similarly, in the inner-level, there is no backpropagation through projection layers and they are considered frozen. Furthermore, it is notable that in this example, the above formulation is also closely connected to FWPs perspective of linear attentions (Schlag et al. 2021), where projections are considered slow weights, and memory update in Equation 15 is the fast weight update rule.

Architectural Decomposition with More Levels. In both examples above, we discussed how they can be viewed as a 2-level optimization process (coinciding with their FWPs interpretations). In practice, however, we may need to use more powerful optimization process, and/or more powerful recurrent update rules for memory. As a simple example, assume we use gradient descent with momentum to train a linear attention model. As we saw above, the linear attention component can be decomposed into two nested optimization processes. Similarly, the model here can be represented as a 2-level optimization problem, where (1) the inner level optimizes the memory to compress the context using gradient descent (Equation 17), and (2) the outer level optimizes the projection layers with gradient descent with momentum. Interestingly, we saw that "gradient descent with momentum" algorithm itself can be viewed as a 2-level optimization process where the momentum term itself is an associative memory that compress the past gradients into its parameters.

3.2 Nested Optimization Processes

In the previous section, we provided examples to demonstrate how one can decompose a machine learning model into a set of nested or multi-level optimization procedures. Next, we first present a formalization of nested learning problems and then define Neural Learning Module–an integrated computational system that learns from data.

In previous sections, we decomposed the model into a set of optimization process. However, it is still unclear if we can define a hierarchy (or order) over these processes, and uniquely represent the model in this format. Inspired by the hierarchy of brain waves that indicates the information processing frequency rate of each part (discussed in Section 1), we use the update rate of each optimization process to order the components in multiple levels. To this end, we let the one update step over one data point to be the unit of time, and define the update frequency rate of each component as:

Definition 2 (Update Frequency)**.** *For any component of A, which can be a parametric component (e.g., learnable weights or momentum term in gradient descent with momentum) or a non-parametric component (e.g., attention block), we define its frequency, denoted by f_A, as its number of updates per unit of time.*

Given the above update frequency, we can order the components of a machine learning algorithm based on operator $(\cdot \succ \cdot)$. We say A is faster than B and denote $A \succ B$ if: **(1)** $f_A > f_B$, or **(2)** $f_A = f_B$ but the computation of the B's state at time t requires the computation of A's state at time t. In this definition, when $A \not\succ B$ and $B \not\succ A$, we let $A \overset{f}{=} B$, which indicates that A and B has the same frequency update, but their computation is independent of each other (Later, we provide an

example of this cases in AdamW optimizer). Based on the above operator, we sort the components into an ordered set of "*levels*", where (1) components in the same level have the same frequency update, and (2) the higher the level is, the lower its frequency. Given the above formulations of levels and update frequency, we next formally define nested learning:

Definition 3 (Nested System)**.** *A (ordered) nested system is a system with K (ordered) levels such that each level k, $1 \leq k \leq K$, consists of a set of optimization problems $\{(\mathcal{L}_i^{(k)}, C_i^{(k)}, \Theta_i^{(k)})\}_{i=1}^{N_k}$, where $\mathcal{L}_i(\cdot;\cdot)$ is the optimization objective in the i-th problem, C_i is its context (the data that is optimized on), Θ_i is the feasible set of its parameters, and each parameter is optimized using gradient descent:*

$$\theta_{i_{t+1}}^{(k)} = \arg\min_{\Phi_i^{(k)}} \langle \Phi_i^{(k)} x_{t+1}, -\nabla \mathcal{L}_i^{(k)}(\theta_{i_t}^{(k)}; x_{t+1}) \rangle + \frac{1}{2\eta_{i_{t+1}}^{(k)}} \|\Phi_i^{(k)} - \theta_{i_t}^{(k)}\|_2^2 \quad \text{where } x_{t+1} \sim C_i^{(k)}, \quad \text{and} \quad \Phi_i{}^{(k)} \in \Theta_i^{(k)}. \tag{19}$$

Notice that each optimization process has its own gradient flow, and therefore sometimes we refer to them as a box of gradient flow corresponding to an optimization problem. Through this paper, we further generalize our definition of nested systems, and allow finding non-parametric solutions for some boxes (i.e., optimization problems).

The above definition provides a general flexible definition for a nested system that does not specify if there is any dependence among different boxes (i.e., a box can determines the context or the parameter space of another box). In the next sections, we discuss how knowledge/information can be transferred between different levels or boxes. Throughout this paper, we focus on the Nested Systems of Associative Memories (NSAM) that is a nested system, in which each optimization process is an associative-memory. More formally,

Definition 4 (Nested System of Associative Memories)**.** *A nested system of associative memory (NSAM) is a system with K (ordered) levels such that each level k, $1 \leq k \leq K$, consists of a set of optimization problems $\{(\mathcal{L}_i^{(k)}, C_i^{(k)}, \Theta_i^{(k)})\}_{i=1}^{N_k}$, where $C_i = \{(k_j^{(i)}, v_j^{(i)})\}_{j=1}^{L_i}$ is a set of key-value pairs, $\mathcal{L}_i(\cdot;\cdot,\cdot)$ measures the quality of memory learned mappings in the i-th problem, Θ_i is the set of feasible memory parameters with each parameter optimized using gradient descent:*

$$\theta_{i_{t+1}}^{(k)} = \arg\min_{\Phi_i^{(k)}} \langle \Phi_i^{(k)} k_{t+1}^{(i)}, -\nabla \mathcal{L}_i^{(k)}(\theta_{i_t}^{(k)}; k_{t+1}^{(i)}, v_{t+1}^{(i)}) \rangle + \frac{1}{2\eta_{i_{t+1}}^{(k)}} \|\Phi_i^{(k)} - \theta_{i_t}^{(k)}\|_2^2, \tag{20}$$

where $(k_{t+1}^{(i)}, v_{t+1}^{(i)}) \sim C_i^{(k)}$ and $\Phi_i{}^{(k)} \in \Theta_i^{(k)}$.

Given a query q, for each associative memory $\mathcal{M}_i^{(k)}$, we use $\mathcal{M}_i^{(k)}(q)$ to refer to the forward pass process (i.e., retrieval process) of the memory. While our formulation of NSAMcan simply be defined by any optimization process beyond gradient descent, and initially, it may seem that the strict condition of *optimization by gradient descent* can limit the modeling power of the definition, through this paper, we show that modern architectures alongside certain well-known optimization algorithms can be viewed as instances of NSAM. We then build on top of this intuition and discuss how to go further with stacking multiple levels and design models with enhanced continual learning capabilities.

The new dimension of stacking multiple levels is an important characteristic of nested learning, where the depth of computation can be enhanced with increasing the number of levels. Depending on the design, context, and the type of knowledge transfer, this depth of computation can itself be viewed as different concepts such as: higher-order in-context learning ability, latent computation (e.g., Loop Transformers), multiple memory systems, and more expressive optimizers. Later, we discuss all such implications, but next, we use a simple example that connects MLP layers in Transformer architectures with linear or deep memory blocks.

Example of an MLP Layer vs. Linear Attention. Let us compare two models: (1) a Transformer architecture, and (2) the same backbone but with replacing the MLP block with a linear attention mechanism (sharing the keys and values from previous layer), in which the initial state of its memory is meta-learned (similar to Behrouz et al. (2025c) or Sun et al. (2024)). We refer to the second variant as Adaptive Transformer or AdaTransformer. Both models are optimized on Next Token Prediction (NTP) objective with gradient descent. As discussed in the initial examples, and also illustrated in Figure 3, both models have two levels, and for the sake of clarity, we use red (resp. blue) to highlight computations/weight in the first level (resp. second level). More formally, let $X = \{x_i\}_{i=1}^{T}$ be an input sequence of tokens, the output of both blocks are computed as (For the sake of simplicity, we assume $\texttt{MLP}(\cdot) = \cdot\, W_{\texttt{MLP}}$, and remove normalizations):

$$
\begin{aligned}
&\boldsymbol{k}_t = \boldsymbol{x}_t W_{\boldsymbol{k}}, \qquad \boldsymbol{v}_t = \boldsymbol{x}_t W_{\boldsymbol{v}}, \qquad \boldsymbol{q}_t = \boldsymbol{x}_t W_{\boldsymbol{q}}, \\
&\boldsymbol{y}_{\texttt{attn}} = \texttt{Attn}\left(\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}\right), \\
&\boldsymbol{y}_{\texttt{block}} = \texttt{MLP}\left(\boldsymbol{y}_{\texttt{attn}}\right) = \boldsymbol{y}_{\texttt{attn}} W_{\texttt{MLP}}, \quad \text{(Transformer Block)}
\end{aligned}
$$

$$
\begin{aligned}
&\boldsymbol{k}_t = \boldsymbol{x}_t W_{\boldsymbol{k}}, \qquad \boldsymbol{v}_t = \boldsymbol{x}_t W_{\boldsymbol{v}}, \qquad \boldsymbol{q}_t = \boldsymbol{x}_t W_{\boldsymbol{q}}, \\
&\boldsymbol{y}_{\texttt{attn}} = \texttt{Attn}\left(\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}\right), \\
&\boldsymbol{y}_{\texttt{block}} = \boldsymbol{y}_{\texttt{attn}} W_{\texttt{LinAttn}}. \qquad \text{(AdaTransformer Block)}
\end{aligned}
$$

The formulation of both blocks seem to be very similar and the only difference comes from the level of $W_{\texttt{MLP}}$ and $W_{\texttt{LinAttn}}$ weights. That is, while $W_{\texttt{MLP}}$ is in the first level and so is persistent with respect to the context, $W_{\texttt{LinAttn}}$ is adaptive and is updated in-context by ($\mathcal{M}(\cdot)$ is parametrized by $W_{\texttt{LinAttn}}$):

$$
\mathcal{M}_t = \mathcal{M}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\top. \tag{21}
$$

In earlier variants of linear attention, the initial state of $\mathcal{M}(\cdot)$ or equivalently $W_{\texttt{LinAttn}}$ is considered as zero matrix, $\mathcal{M}_0 = \mathbf{0}$. Similar to more advanced design choices (Sun et al. 2024; Behrouz et al. 2025c), however, this initial state can be meta-learned to adapt fast to a context. In this setting, the initial state of $\mathcal{M}_0 = W_{\texttt{LinAttn}_{\texttt{init}}}$ is optimized in the first level with NTP objective, while given the context, $W_{\texttt{LinAttn}}$ is optimized in the second level as an associative memory with dot-product objective (Behrouz et al. 2025b).

The above example is also valid when using more advanced and deep MLP blocks in Transformer architecture (such as SwiGLU (Shazeer 2020)) and compared it with its recurrent memory counterpart (Behrouz et al. 2025a). Furthermore, this simple example implies that the current perspective on hybrid architectures as the combination of expressive softmax attention and efficient recurrent models is somewhat misleading and it follows the conventional Transformer backbone design but with additional in-context learning capabilities for MLP blocks. We discuss this further in Section 6 and Section 7. As a takeaway of the discussion in this subsection about the concept of nested systems and nested learning:

> **Stacking Levels in Nested Learning:**
>
> ***Nested learning*** *allows computational models that are composed of multiple (multi-layer) levels to learn from and process data with different levels of abstraction and frequencies of update.*

As discussed earlier, it is common in the literature to separate architectures from their optimization processes and to treat them as independent design choices, with the aim of combining algorithms that achieve the greatest expressive power in each aspect. In practice, however, a Transformer architecture (Vaswani et al. 2017) that is optimized with stochastic gradient descent can learn a very different solution than the same architecture when Adam optimizer is used (Kingma et al. 2014a). Accordingly, when interacting with such machine learning algorithms, we observe that despite the similarity in the architecture axes, the overall trained models show different predictions or generates different outputs. From the NL's viewpoint, however, a machine learning algorithm is represented as an interconnected system of optimization problems and model's actions, predictions, and generation of output depends on this system as a whole, not necessarily each of its sub-components. To this end, we define the term of neural learning module to refer to this representation of a model, where architecture and optimization process jointly determine the model and its outputs. While such joint representation might not seem significant in the current machine learning pipelines, where there is a training and then test phases, it becomes more important in the continual setup we advocate for, where there is no training/test phases (see more discussions in Section 8).

Neural Learning Modules are Inter-connected Systems. Based on the definition of neural learning module, one important question is how the architecture and optimization process are interconnected systems and how they can affect each other. Recall the general formulation for training a neural network: Given a task $\mathcal{T}$, its corresponding data distribution $p(\mathcal{T})$, a model $f(\cdot;\cdot)$ parameterized by $\Phi_{\mathcal{T}}$, and an objective $\mathcal{L}(\cdot;\cdot)$ we aim to learn parameters $\Phi^*_{\mathcal{T}}$ such that:

$$
\Phi^*_{\mathcal{T}} = \arg\min_{\Phi} \; \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim p(\mathcal{T})} \left[\mathcal{L}\left(\Phi; \boldsymbol{x}, \boldsymbol{y}\right)\right]. \tag{22}
$$

In practice, we optimize the above problem based on a given dataset $\mathcal{D}_{\text{train}}$ and an optimization algorithm such as stochastic gradient descent:

$$
\Phi_{t+1} = \Phi_t - \eta_{t+1} \nabla_{\Phi_t} \mathcal{L}(\Phi_t; \boldsymbol{x}_{t+1}, \boldsymbol{y}_{t+1}), \qquad \text{where } (\boldsymbol{x}_{t+1}, \boldsymbol{y}_{t+1}) \sim \mathcal{D}_{\text{train}}. \tag{23}
$$

One interpretation for the optimization process of model $f(\cdot;\cdot)$ in Equation 23 is to see the model as the data generator for the optimization process in Equation 23. That is, as discussed in the first example in Section 3.1, and later we will show in Section 4, the optimization process is an associative memory that aims to compress the patterns between the training data and its gradients (or surprise) and so the dataset for internally training such memory (i.e., the gradients of the model) is generated by the model. Therefore, the type of the model can result in generating dataset (i.e., gradients) with different patterns and distributions over time. The effect of the optimization process and this data generation also fed back at the model itself, where the next state of the parameters in the model are determined by the optimization algorithms. As we will discuss in Section 4, looking at optimizers as associative memories on the gradients of the model implies that each optimizer has some special traits such as better memory management, higher compression, etc. Therefore, the choice of such algorithms requires understanding the generated gradients and also changes of the model in the parameter space.

3.3 Knowledge Transfer Between Levels

So far, we mainly focused on the concept of nested learning and how optimization problems are located in different levels. It is, however, still unclear that how nested optimization problems (in different levels) can affect each other, or in general how they can contribute to the output of the system, and so be interconnected. In this section, we discuss several potential knowledge transfer methods between components in different levels. For the sake of clarity, we discuss the knowledge transfer between two levels and blocks, i.e., $\mathcal{B}^{(0)} = (\mathcal{L}^{(0)}, \mathcal{C}^{(0)}, \boldsymbol{\Theta}^{(0)})$ and $\mathcal{B}^{(1)} = (\mathcal{L}^{(1)}, \mathcal{C}^{(1)}, \boldsymbol{\Theta}^{(1)})$ with corresponding memories $\mathcal{M}^{(0)}(\cdot)$ and $\mathcal{M}^{(1)}(\cdot)$, respectively:

Direct Connection of Levels (Parametric). The first type of knowledge transfer is to directly incorporate the weights in different levels or blocks. To this end, the forward pass or the retrieval process from the lower-frequency (i.e., higher-level) memory system is also conditioned on the parameters of the higher-frequency (i.e., lower-level) memory:

$$\mathcal{M}^{(0)}(\cdot) := \mathcal{M}^{(0)}(\cdot\,;\, \boldsymbol{\Theta}^{(1)}). \tag{24}$$

In a more specific formulation, and as a special variant of the above formulation, we can condition the output of $\mathcal{M}^{(0)}(\cdot)$ based on the output (or forward pass) of the higher-frequency memory:

$$\mathcal{M}^{(0)}(\cdot) := \mathcal{M}^{(0)}(\cdot\,;\, \mathcal{M}^{(1)}(\cdot)), \tag{25}$$

where we slightly abused notation by hiding the dependence on the second argument. As an example of this type of knowledge transfer, in linear Transformer (or FWP) (Katharopoulos et al. 2020; Schlag et al. 2021), where the initial memory state is zero, the stored knowledge of a lower level (fast weight) directly affect the output of the model in another level. That is, one can re-write the forward pass (memory retrieval) as:

$$\boldsymbol{y}_t = \mathcal{M}_t \boldsymbol{q}_t = \overbrace{\mathcal{M}_t \underbrace{\left[\boldsymbol{x}_t W_q\right]}}^{\substack{\text{Forward pass of the} \\ \text{higher-frequency memory}}} \qquad \substack{\text{Forward pass of the} \\ \text{lower-frequency memory}} \tag{26}$$

Direct Connection of Levels (Non-Parametric). Another form of direct connection between levels is a non-parametric variant of the above formulation, where the block $\mathcal{B}^{(1)}$ is optimized by finding non-parametric solution. Therefore, the forward pass of the low-frequency memory is conditioned on the context and output of higher-frequency memory:

$$\mathcal{M}^{(0)}(\cdot) := \mathcal{M}^{(0)}(\cdot\,;\, \mathcal{C}^{(1)}), \qquad \text{or similarly,} \qquad \mathcal{M}^{(0)}(\cdot) := \mathcal{M}^{(0)}(\cdot\,;\, \mathcal{M}^{(1)}(\cdot;\mathcal{C}^{(1)})). \tag{27}$$

As an example of this variant, one can refer to Transformers and softmax attention module (Vaswani et al. 2017). There is an important characteristic for both above variants: there is no backpropagation through any state of the blocks in two different levels and knowledge transfers through direct conditioning the output of one level on the other's output/parameters. Therefore, in this process the state of each block is treated as hyperparameter for the other.

Knowledge Transfer via Backpropagation. Another form of knowledge transfer is through backpropagation, where there is a gradient flow between blocks in different levels. The forward pass of this design is the same as the forward

pass discussed above. The backward pass, however, is the main difference, where in the above two cases the state of each associative memory is considered as hyperparameters of the other but here both states are optimized in the same gradient flow. Therefore, for a simple case of two blocks in two levels, we have:

$$\mathcal{M}^{(0)}(\cdot) := \mathcal{M}^{(0)}(\cdot\,;\,\mathcal{M}^{(1)}(\cdot)) \qquad \text{(Forward Pass)}$$

$$\begin{cases} \boldsymbol{\Theta}^{(1)}_{t+1} = \boldsymbol{\Theta}^{(1)}_{t} - \eta^{(1)}_{t+1}\, \boldsymbol{\delta}_1\, \hat{\boldsymbol{x}}^{\top}_{t+1}, \\ \boldsymbol{\Theta}^{(0)}_{t+1} = \boldsymbol{\Theta}^{(0)}_{t} - \eta^{(0)}_{t+1}\, \boldsymbol{\delta}_0\, \boldsymbol{x}^{\top}_{t+1}, \end{cases} \qquad \text{(Backward Pass)}$$

where $\boldsymbol{\delta}^{(0)}_{t+1} = \boldsymbol{J}_{\phi^{(0)}}(\boldsymbol{x}_{t+1})^{\top}\left(\boldsymbol{\Theta}^{(1)}_{t+1}\boldsymbol{\delta}^{(1)}_{t+1}\right)$, $\hat{\boldsymbol{x}}_{t+1} = \phi^{(0)}(\mathcal{M}^{(0)}(\boldsymbol{x}_t))$, $\phi^{(0)}(\cdot)$ is non-linearity, and $\boldsymbol{J}_{\phi^{(0)}}(\cdot)$ is the Jacobian. In this design, two blocks are in the same flow of gradient, but are updated based on different frequencies. We provide an example this design in Section 7, when we discuss continuum memory systems.

Knowledge Transfer via Initialization. Model Agnostic Meta-Learning (MAML) (Finn et al. 2017), is one of the most popular form of meta-learning (or learning to learn) that aims to learn a global initial point for a model so it could learn fast a new task. From the nested learning perspective, there are two nested optimization processes, in which the inner problem iterates over its own context and based on its internal objective, the higher-level problem measures its own learned weight as the initial point of the inner problem. More formally, we have

$$\boldsymbol{\Theta}^{(1)}_{0} = \arg\min_{\Phi}\ \mathbb{E}_{C\sim C^{(0)}}\left[\boldsymbol{\ell}(\mathcal{M}^{(1)}(\cdot;\Phi), C)\right], \tag{28}$$

where the higher level block learns the best initial value over all possible contexts the lower level problem might have. As discussed earlier, any MAML-based learning model is an instance of this case, but as a more specific example, we refer to the example of (MLP Layer vs. Linear Attention), which is discussed in Section 3.2 and Figure 3.

Connections with Generation. One of the most common form of knowledge transfer is through generating weights or context. That is, one lower-frequency (resp. higher-frequency) block generates the weight of a higher-frequency (resp. lower-frequency) block. More formally,

$$\begin{cases} \boldsymbol{\Theta}^{(1)} = \boldsymbol{g}\left(\mathcal{M}^{(0)}; \mathcal{L}^{(0)}, C^{(0)}, \boldsymbol{\Theta}^{(0)}\right), \text{ or} \\ \boldsymbol{\Theta}^{(0)} = \boldsymbol{g}\left(\mathcal{M}^{(1)}; \mathcal{L}^{(1)}, C^{(1)}, \boldsymbol{\Theta}^{(1)}\right), \end{cases} \qquad \text{(Weight Generation)}$$

$$\begin{cases} C^{(1)} = \boldsymbol{g}\left(\mathcal{M}^{(0)}; \mathcal{L}^{(0)}, C^{(0)}, \boldsymbol{\Theta}^{(0)}\right), \text{ or} \\ C^{(0)} = \boldsymbol{g}\left(\mathcal{M}^{(1)}; \mathcal{L}^{(1)}, C^{(1)}, \boldsymbol{\Theta}^{(1)}\right). \end{cases} \qquad \text{(Context Generation)}$$

There are two important examples of the above form for knowledge transfer: (1) Hypernetworks: where the weights of a targeted neural network is generated by another (generator) network. (2) Optimization process: where the architecture generates the input for the optimizer. That is, the context (or input data) of an optimizer is the gradients that are generated by the architecture. For more discussion on this topic, see Section 4. Note that this example is not necessarily about "*learned optimizers*" and it is valid for the commonly used optimization process and algorithms such as gradient descent, Adam (Kingma et al. 2014a), AdaGrad (Duchi et al. 2011), etc.

A Note on Designing Neural Learning Modules. In the above, we discussed only some examples of possible knowledge transfer methods and also the potential connections of different levels. The formulation of NL and neural learning module, however, is general and so is not limited to the above specific set of methods. Accordingly, to design a neural learning module from a nested learning perspective, there are two important steps and design choices:

> **Designing Neural Learning Modules:**
>
> *There are two high-level design choices in developing a neural learning module: (1) The design of optimization problems and their frequency (i.e., designing components in* NSAM*); (2) The design of knowledge transfer between levels.*

It is notable that with different choices of knowledge transfer, some learning paradigms can be seen as a part of a neural learning model: E.g., (1) Meta learning, when two blocks in two levels transfer their knowledge with one level meta-learns the other; more specifically, (2) Model Agnostic Meta Learning (MAML) (Finn et al. 2017), when knowledge transfer is though learning the initialization; (3) Hypernetworks, when one higher-frequency block generates the weights for the

other lower-frequency block; (4) Learned optimizers, when the knowledge transfer is through data generation (i.e., one high-frequency block generates the gradient for the other lower-frequency block).

4 Optimizers as Learning Modules

In this section, we start with viewing backpropagation process and optimizing a neural network from the associative memory and data compression perspective. Then, we discuss how variants such as momentum-based optimizers are instances of nested associative memory systems. Finally, we discuss alternative methods leading to deep optimizers with higher expressive power from the associative memory perspective.

4.1 Backpropagation as an Associative Memory

Updating the weights of a neural network through backpropagation (Linnainmaa 1970; Rumelhart et al. 1986) has been the critical component of training large-scale deep neural networks. Intuitively, in this optimization process, first, the error of the model's output with respect to target is calculated, and then each layer is updated based on its contribution to this error. This section aims to explain this process through the lens of associative memory and discuss how it fits within the nested learning paradigm. For the sake of clarity and simplicity, we assume a deep MLP model, but all the derived formulations in the following can simply be adapted to other architectures as well. Given an MLP with L layers parameterized with $\{W_\ell \cdot + \boldsymbol{b}_\ell\}_{\ell=1}^{L}$, the required gradients in backpropagation are computed as:

$$\frac{\partial \mathcal{L}}{\partial W_\ell} = \boldsymbol{\delta}_\ell \, \hat{\boldsymbol{x}}_{\ell-1}^\top, \qquad \text{and} \qquad \boldsymbol{\delta}_\ell = \underbrace{\boldsymbol{J}_{\phi_\ell}\left(\boldsymbol{z}_\ell\right)^\top \left(W_{\ell+1}^\top \boldsymbol{\delta}_{\ell+1}\right)}_{\text{local output surprise for layer } \ell}, \tag{29}$$

where $\boldsymbol{z}_\ell = W_\ell \, \hat{\boldsymbol{x}}_{\ell-1} + \boldsymbol{b}_\ell$ is pre-activation, and so $\hat{\boldsymbol{x}}_\ell = \phi_\ell\left(\boldsymbol{z}_\ell\right)$ is the output of ℓ-th layer, $\phi_\ell(\cdot)$ is its non-linearity, and $\boldsymbol{J}_{\phi_\ell}(\cdot)$ is the Jacobian. Therefore, the update of the ℓ-th layer with gradient descent is computed as:

$$W_{\ell_{t+1}} = W_{\ell_t} - \eta_{\ell_{t+1}} \, \boldsymbol{\delta}_\ell \, \hat{\boldsymbol{x}}_{\ell-1}^\top . \tag{30}$$

Here, $\hat{\boldsymbol{x}}_{\ell-1}$ is the input of the layer and $\boldsymbol{\delta}_\ell$ measures the local error signal for layer ℓ or equivalently is a metric that measures the surprise of layer ℓ's output given its input. Similar to our example in Section 3.1, we can write Equation 30 as:

$$W_{\ell_{t+1}} = \arg\min_{W} \langle W \hat{\boldsymbol{x}}_{\ell-1}, \boldsymbol{\delta}_\ell \rangle + \frac{1}{2\eta_{\ell_{t+1}}} \| W - W_{\ell_t} \|_F^2, \tag{31}$$

which is an associative memory module that aim to map the input of each layer $\hat{\boldsymbol{x}}_{\ell-1}$ to its local error signal, $\boldsymbol{\delta}_\ell$ (see Definition 1). That is, the above formulation implies that *the training process of a neural network with gradient descent and backpropagation can be viewed as a compression process*, in which each layer stores the mappings between its input and the corresponding local error signal. Later in §4.5, we discuss how this viewpoint helps with designing more expressive learning rules for backpropagation.

> **Training a Deep Neural Network with Backpropagation as a Surprise-based Memory:**
>
> *A neural network trained with backpropagation learns from data by memorizing how surprising their predicted outputs are; i.e., backpropagation is an associative memory that maps each data point to the error in its corresponding prediction.*

Backpropagation ≠ Linear Attention. A common misinterpretation for Equation 30 is to assume $\boldsymbol{\delta}_\ell$ is a pre-computed term and so backpropagation (at least on a linear layer) recovers Hebbian-rule, resulting in the equivalency of the optimization process and performing linear attention on gradients. Our formulation, however, shows that the update rule in backpropagation is a self-referential process (Schmidhuber 1993), where the values of the associative memory is generated by itself, making it a more complex associative memory than a simple linear attention on gradients (see Section 4.5).

4.2 Momentum-based Optimizers as Associative Memories

Momentum-based optimizers are the major components of modern machine learning models' training (Duchi et al. 2011; Kingma et al. 2014a; Jordan et al. 2024). To explain momentum-based optimizers as associative memories, let us start from

a simple gradient descent algorithm:

$$W_{t+1} = W_t - \eta_t \nabla_{W_t} \mathcal{L}(W_t; \boldsymbol{x}_{t+1}), \tag{32}$$

which updates the current state of the weights based on the momentary gradient (surprise). This update rule does not incorporate the previous tokens and also the loss landscape that have been traversed so far, resulting in slower (or less robust) convergence in many scenarios. To fix this, momentum-based gradient descent methods incorporate an Exponential Moving Averages (EMAs) of past gradients:

$$\begin{aligned} W_{\ell_{t+1}} &= W_{\ell_t} + \boldsymbol{m}_{\ell_{t+1}} \\ \boldsymbol{m}_{\ell_{t+1}} &= \alpha_{\ell,t+1} \boldsymbol{m}_{\ell_t} - \eta_{\ell,t+1} \nabla_{W_{\ell_t}} \mathcal{L}\left(W_{\ell_t}; \boldsymbol{x}_{t+1}\right) = \alpha_{\ell,t+1} \boldsymbol{m}_{\ell_t} - \eta_{\ell,t+1}\, \boldsymbol{\delta}_\ell\, \hat{\boldsymbol{x}}_{\ell-1}^\top, \end{aligned} \tag{33}$$

where matrix (or vector) $\boldsymbol{m}_t$ is the momentum at state t and α_t and η_t are (adaptive) learning and momentum rates, respectively, and $\boldsymbol{\delta}_\ell$ and $\hat{\boldsymbol{x}}_{\ell-1}$ are defined the same as in Equation 29. Similar to Equation 31 and one of the examples in Section 3.1, assuming $\alpha_{t+1} = 1$, the momentum term can be viewed as the result of optimizing the following objective with gradient descent:

$$\min_{\boldsymbol{m}} \ \langle \boldsymbol{m}\, \hat{\boldsymbol{x}}_{\ell-1}, \boldsymbol{\delta}_\ell \rangle. \tag{34}$$

The case of $\alpha_{t+1} \neq 1$ is equivalent to GD on the above minimization plus an ℓ_2-regularization on the momentum term. Thus, momentum can indeed be viewed as an associative memory module that learns how to compress the past gradients of the objective into its parameters. Contrary to Equation 31, which was a simple 1-level associative memory and the update was directly applied to the memory, here the state of the momentum determines the update for the weights. In other words, it is a 2-level optimization procedure, in which the inner-loop learns the momentum and the outer-loop uses the state of the momentum to update the weights.

From this perspective, we can generalize the definition of momentum from EMAs to any arbitrary associative memory module that aims to compress the past gradients or maps the input of each token to its corresponding local error. This generalized momentum can be expressed as:

$$W_{\ell_{t+1}} = W_{\ell_t} + \boldsymbol{m}_{\ell_{t+1}}, \tag{35}$$

$$\tag{36}$$

where $\boldsymbol{m}_\ell$ is the solution of the following associative memory, optimized by gradient descent:

$$\min_{\boldsymbol{m}} \ \tilde{\mathcal{L}}\left(\boldsymbol{m};\ \hat{\boldsymbol{x}}_{\ell-1}, -\boldsymbol{\delta}_\ell\right). \tag{37}$$

Here, the objective $\tilde{\mathcal{L}}(\cdot)$ is different from the original objective of the problem at hand, and $\tilde{\mathcal{L}}(\cdot)$ is the objective that defines the momentum and measures the quality of its mappings. In fact, the momentum term in this formulation aims to adapt in-context (recall that the context of the momentum is the gradients) to the local error rates based on the input of the layer. Most popular optimizers are formulated as element-wise update rule (for computational efficiency reasons) and so in Appendix B, we first explore the element-wise associative memory formulation of momentum and connect it to popular optimizers such as Adam (Kingma et al. 2014a). Showing that Adam can be viewed as the optimal associative memory to the L_2-regression objective that aims to predict the variance of gradients, we discuss other similar algorithms such as RMSProp (Hinton et al. 2012), SignSGD and its momentum-based variants (Bernstein et al. 2018), NAdam (Dozat 2016), AMSGrad (Reddi et al. 2016), RAdam (Liu et al. 2020), and Lion (Chen et al. 2023) are also instances of an associative memory that aims to compress the gradients. We then go beyond element-wise formulation and show that AdaGrad (Duchi et al. 2011) is also an associative memory module. Due to the connection of AdaGrad with optimizers such as Shampoo (Gupta et al. 2018) and Soap (Vyas et al. 2025)–i.e., as the approximation of the preconditioning term–we then conclude that all these optimizers can be re-formulated as associative memory. Next, we discuss another class of optimizers based on preconditioning and reformulate them from NL's perspective in more details:

Preconditioning and Approximation of Hessian. Another class of algorithms is preconditioning algorithms where the idea is to approximate Hessian inverse to mimic the behavior of Newton's algorithm. Formally, gradient descent with preconditioning is defined as:

$$W_{\ell_{t+1}} = W_{\ell_t} - \eta_{t+1}\, \boldsymbol{P}_{t+1}^{-1}\, \boldsymbol{g}_{\ell_{t+1}}, \tag{38}$$

where *preconditioner* $\boldsymbol{P}_{t+1}$ is often a positive-definite matrix. A critical interpretation of preconditioner is their role in performing gradient descent in a transformed coordinate system, which can be viewed as a mapping from gradients to that system of interest. Accordingly, we reformulate and interpret the preconditioner in Equation 38 as an associative memory that maps the set of gradients (or a function of gradients denoted as $\boldsymbol{g}$) to the system of our choice, denoted as $\hat{\boldsymbol{g}}$:

$$W_{\ell_{t+1}} = W_{\ell_t} - \eta_{t+1}\, \boldsymbol{P}_{t+1}^{-1}\big(\boldsymbol{g}_{\ell_{t+1}}\big)\,, \tag{39}$$

where internally (in a nested level), $\boldsymbol{P}_{t+1}$ learns how to perform this mapping using an objective:

$$\min_{\boldsymbol{P}} \quad \tilde{\mathcal{L}}\left(\boldsymbol{P}\left(\hat{\boldsymbol{g}}\right); \boldsymbol{g}\right). \tag{40}$$

Given this viewpoint, the main question is about finding the best coordinate system that can empower the compression process. The most simple variant is an identity mapping, where we preserve the metric system and use $\boldsymbol{P}$ to map $\boldsymbol{g}$ (i.e., gradients in this case) to itself, resulting in preconditioning terms in Adam (Kingma et al. 2014a) and AdaGrad (Duchi et al. 2011), as discussed in Appendix B. These results, along with the representation of Adam and its variants as associative memories, show that not only momentum-based optimizers are associative memories, but they also can be decomposed into a set of nested learning problems, each of which optimized with gradient descent. In a more general form, however, one can use more nested levels and optimize the inner problems in Equation 40 with gradient descent, resulting in:

$$\boldsymbol{P}_{t+1} = \boldsymbol{P}_{t+1} - \zeta_{t+1} \nabla_{\boldsymbol{P}_t} \tilde{\mathcal{L}}\left(\boldsymbol{P}_t; \boldsymbol{g}_{t+1}, \hat{\boldsymbol{g}}_{t+1}\right). \tag{41}$$

In the NL framework, to design an effective preconditioning, one needs to find the right choice of $\hat{\boldsymbol{g}}$ and $\tilde{\mathcal{L}}$. This viewpoint can also lead to other classes of algorithms with gradient/momentum orthogonalization: e.g., Muon and its variants (Jordan et al. 2024; Cesista 2025; Keigwin et al. 2025). Recalling Muon optimizer (Jordan et al. 2024):

$$\begin{aligned} W_{\ell_{t+1}} &= W_{\ell_t} + \texttt{NewtonSchulz}_k\left(\boldsymbol{m}_{\ell_{t+1}}\right) \\ \boldsymbol{m}_{\ell_{t+1}} &= \alpha_{\ell,t+1} \boldsymbol{m}_{\ell_t} - \eta_{\ell,t+1} \nabla_{W_{\ell_t}} \mathcal{L}\left(W_{\ell_t}; \boldsymbol{x}_{t+1}\right), \end{aligned} \tag{42}$$

where $\texttt{NewtonSchulz}_k(\cdot)$ performs k steps of Newton-Schulz orthogonalization process. From the above discussion about the general formulation of preconditioning, one can see $\texttt{NewtonSchulz}_k(\cdot)$ operator as a mapping from gradients of momentum term to a proper metric system. The choice of proper coordinate system in Muon is to orthogonalize the gradients and so we aim to find a mapping $\boldsymbol{P}(\cdot)$ by minimizing a loss function $\min_{\boldsymbol{P}} \tilde{\mathcal{L}}(\boldsymbol{P}; \boldsymbol{O}, \boldsymbol{m})$ where objective $\tilde{\mathcal{L}}(\cdot;\cdot,\cdot)$ measures the quality of mapping from $\boldsymbol{O}$ to either $\boldsymbol{m}$ or $\boldsymbol{g}$ by $\boldsymbol{P}(\cdot)$. A critical challenge in this process is that the parameter $\boldsymbol{O}$ itself is not given and so the mapping requires learning both the mapping and the proper orthogonal space. A simple formulation measuring orthogonalization, can be achieved by defining the objective as:

$$\tilde{\mathcal{L}}(\boldsymbol{P}(\boldsymbol{g}); \boldsymbol{g}) = \left\|\boldsymbol{P}(\boldsymbol{g})^\top \boldsymbol{P}(\boldsymbol{g}) - \boldsymbol{I}\right\|_F^2, \tag{43}$$

where $\boldsymbol{P}(\boldsymbol{g})$ is the orthogonal space that we aim to directly learn from gradients. This objective ensures that the gradients (or momentum) and their mapping are relatively close while the mapping is to an orthogonal space. Optimizing the above objective to find $\boldsymbol{O} = \boldsymbol{P}(\boldsymbol{g})$ with one step of gradient descent results in:

$$\boldsymbol{O}_{i+1} = \boldsymbol{O}_i - \zeta_{i+1} \nabla_{\boldsymbol{O}_i} \tilde{\mathcal{L}}\left(\boldsymbol{O}i; \boldsymbol{g}_t\right) = \boldsymbol{O}_i - \zeta_{i+1}\left(\boldsymbol{O}_i - \boldsymbol{g}_t + 2\boldsymbol{O}_i\left(\boldsymbol{O}_i^\top \boldsymbol{O}_i - \boldsymbol{I}\right)\right), \tag{44}$$

which recovers the 3-degree polynomial (initial value $\boldsymbol{O}_0 = \boldsymbol{g}_t$). In a summary, the higher-frequency level learns the orthogonal mapping and then the lower-frequency process use the learned mapping to optimize the weights. Later, in Section 4.4, we discuss a more general viewpoint that considers $\texttt{NewtonSchulz}_k(\cdot)$ as a polynomial mapping to enhance the capacity of the memory.

4.3 Long Context in Optimizers: An Example of Continual Learning with Orthogonal Tasks

When removing the boundary between train and test time, and moving towards models that can continually learn for a long period of time, the role of (online) optimizers becomes more prominent: mainly due to the need for finding effective "solutions" rather than converging faster. Finding an effective solution, however, requires global understanding of the objective to avoid "local minima" as well as moving toward directions that might cause (catastrophic) forgetting of long

past learned tasks. From associative memory perspective and as discussed earlier, the momentum term is expected to be a memory of past gradients, helping the optimization process to have a more global view of the loss landscape. The current design of momentum, however, acts as a simple low-pass filter that smoothifies the gradient updates and thus has limited capacity with only incorporating information from recent past.

To better illustrate this limitation, let $\beta > 0$ be the momentum's decay term, and so the contribution of i-th gradient before to the current state of the momentum can be calculated as $\beta^i (1-\beta)$. Considering the cumulative sum of gradients' contributions to the current state of the momentum term (i.e., $\boldsymbol{S}_t = \sum_{i=0}^{t} \beta^i (1-\beta)$) and the commonly used value of $\beta = 0.9$ in optimization setups, the last 6 gradients (resp. 43 gradients) are responsible for at least 50% (resp. 99%) of the cumulative contribution, i.e., $\boldsymbol{S}_t$. This indicates that gradients and generally the global information beyond only past 43 steps contribute less than 1%, limiting the understanding of the objective landscape and the ability to find effective solutions. This simple example shows that the current design is limited even in incorporating the long past information, let alone its ability to properly retrieve information needed for the current state of the momentum.

Coming back to the setup of continual learning and considering one of its simple variants with orthogonal tasks. We let $\{(\mathcal{T}_i, \mathcal{D}_i, \mathcal{L}_i)\}_{i=1}^{n}$ be the set of tasks, their corresponding data, and their objectives such that for task $\mathcal{T}_i$:

$$\mathcal{L}_i(W) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}_i}\left[(W^\top \boldsymbol{x} - \boldsymbol{y})^2\right], \tag{45}$$

and gradients live in orthogonal directions of $\{\boldsymbol{u}_i\}_{i=1}^{n}$. Given the task $\mathcal{T}_t$ (for large enough $t > 1$), when optimizing the problem with gradient descent with momentum, and after many steps on task t, the gradients now point along $\boldsymbol{u}_t$. Accordingly, the momentum term is gradually shifted and now is approximately in $\boldsymbol{u}_t$ direction. This can lead to gradual forgetting about past gradients, which in turn may cause catastrophic forgetting in the model. That is, the optimization process can move the weights in a direction that damages the performance on previous tasks, mainly because the optimizer has no memory of the old gradient subspace that it should avoid. This failure is not about the capacity of the model but the memory management of the optimization process, failing at finding an effective solution.

> **Long Context Understanding in Optimizers**
>
> *Moving from static models to neural learning modules that can continually learn from data/experience, the optimization process itself can benefit from long-term compression/understanding of gradient subspace to find effective solutions over diverse sets of tasks and for a long period of time..*

Motivated by this observation, we next discuss more expressive variants of momentum that are capable of better memory management and higher memory capacity:

4.4 More Expressive Designs for Momentum as an Associative Memory

So far we discussed that (1) momentum term can be viewed as an associative memory that aims to compress the (past) gradients into its parameters; and (2) for developing models that can continually learn for a long period of time and on diverse sets of tasks, the optimization process need proper information about long past and the global properties of loss landscape. Next, we discuss how Nested Learning and associative memory viewpoint could result in designing optimizers with diverse memory management/structure:

Extension: More Expressive Association. As discussed earlier, vanilla momentum term can be viewed as a value-less associative memory. To allow more expressive associative memory and following the original definition of associative memory (i.e., mapping keys to values), we let value parameter $\boldsymbol{v}_i = \boldsymbol{P}_i$ and so the momentum aims to minimize:

$$\min_{\boldsymbol{m}} \ \langle \boldsymbol{m}\, \nabla\mathcal{L}(W_i; \boldsymbol{x}_i)^\top, \boldsymbol{P}_i \rangle, \tag{46}$$

or equivalently, minimizes $\langle \boldsymbol{m}, \boldsymbol{P}_i \nabla\mathcal{L}(W_i; \boldsymbol{x}_i) \rangle$. Using gradient descent results in the update rule of:

$$\begin{aligned} W_{i+1} &= W_i + \mathbf{m}_{i+1} \\ \mathbf{m}_{i+1} &= \alpha_{i+1}\mathbf{m}_i - \eta_t \boldsymbol{P}_i \nabla\mathcal{L}\left(W_i; \boldsymbol{x}_i\right). \end{aligned} \tag{47}$$

This formulation is equivalent to preconditioning the momentum GD. Notice that preconditioning means that the momentum term is an associative memory that learns how to compress the mappings between $\boldsymbol{P}_i$ and the gradient term $\nabla\mathcal{L}(W_i; \boldsymbol{x}_i)$.

While any reasonable choice (e.g., random features) of preconditioning can improve the expressivity of the initial version of GD with momentum per se is a value-less memory (i.e., mapping all gradients to a single value), the above perspective gives more intuition about what/why preconditioning can be useful. That is, the momentum acts as a memory that aims to map gradients to their corresponding values, and so a function of gradients (e.g., information about Hessian) can provide the memory with a more meaningful mappings. Note that our discussion on preconditioning in Equation 39 is different from here and indicates that one can learn the preconditioner as a proper mapping, while the above formulation shows when using preconditioning, momentum term (as a memory for gradients) aims to map them to their mapping function.

Extension: More Expressive Objectives. Revisiting the formulation of momentum: for a given gradients $\nabla \mathcal{L}\left(W_i; \boldsymbol{x}_i\right)$, the momentum mapping is based on dot-product similarity as the internal objective (see Equation 47) and so its update rule is a Hebbian-rule (Hebb 2005). This update rule, however, not only has a limited capacity (Storkey 1997), but it also makes the update rule of momentum independent of its current state, limiting its ability to track/compress the the loss landscape information. A natural extension is to replace the internal objective with L_2-regression loss (for measuring the corresponding key-value mapping fitness) and minimize the loss function $\|\boldsymbol{m} \nabla \mathcal{L}(W_i; \boldsymbol{x}_i)^\top - \boldsymbol{P}_i\|_2^2$, resulting in the update rule of:

$$W_{i+1} = W_i + \mathbf{m}_{i+1}, \tag{48}$$

$$\mathbf{m}_{i+1} = \mathbf{m}_i \left(\alpha_{i+1} - \nabla \mathcal{L}\left(W_i; \boldsymbol{x}_i\right)^\top \nabla \mathcal{L}\left(W_i; \boldsymbol{x}_i\right)\right) - \eta_t \boldsymbol{P}_i \nabla \mathcal{L}\left(W_i; \boldsymbol{x}_i\right), \tag{49}$$

This update is based on delta-rule (Prados et al. 1989) and so it allows the memory (momentum) to better manage its limited capacity (i.e., $O(N)$) and better memorize the series of past gradients. For example, we may learn to forget some of the past gradients during the optimization process (similar to what happens when we move from linear attention to delta rule in associate memory). We refer to this variant of momentum term as Delta Momentum.

Extension: More Expressive Memory. Viewing momentum as a compressor or a memory that store past gradients into its elements (parameters), its capacity not only depends on its update rule (similar to the above), but it also requires more expressive structure that allows for larger capacity. The current formulation is based on a linear layer (i.e., matrix-valued) to compress the past gradient values, but this linear nature may limit the capability to only learn linear mappings of past gradients. To increase the learning capacity of this module, one can use more complex mappings such as replacing a linear matrix-valued memory for momentum with an MLP. This design allows momentum to memorize more gradients and so provides better information for the optimization process. We extend the formulation in Equation 33 as:

$$W_{i+1} = W_i + \mathbf{m}_{i+1}\left(\boldsymbol{u}_i\right), \quad \text{and} \quad \mathbf{m}_{i+1} = \alpha_{i+1}\mathbf{m}_i - \eta_t \nabla \mathcal{L}^{(2)}(\boldsymbol{m}_i; \boldsymbol{u}_i, \mathbb{1}), \tag{50}$$

where $\boldsymbol{u}_i = \nabla \mathcal{L}\left(W_i; \boldsymbol{x}_i\right)$ and $\nabla \mathcal{L}^{(2)}(\cdot)$ is the internal objective of momentum (e.g., dot product similarity $\langle \boldsymbol{m}(\boldsymbol{u}_i^\top), \mathbb{1} \rangle$). We refer to this variant as Deep Momentum Gradient Descent (DMGD). While it is clear from this example, it is worth emphasizing that the internal loss function and the model need to be designed carefully to obtain an effective momentum module.

Extension: Memory with Higher-order Feature Maps. One of the commonly used techniques to enhance the capacity of a memory is to use higher-order feature maps on the keys (Katharopoulos et al. 2020; Kacham et al. 2024). Using this technique on the momentum term, one can obtain:

$$W_{i+1} = W_i + \mathbf{m}_{i+1} \quad \text{and} \quad \mathbf{m}_{i+1} = \alpha_{i+1}\mathbf{m}_i - \eta_t \boldsymbol{P}_i \phi(\nabla \mathcal{L}\left(W_i; \boldsymbol{x}_i\right)), \tag{51}$$

where $\phi(\cdot)$ is a higher-order feature mapping (that may be learned through its internal objective).

Extension: Nonlinear Outputs. Building upon the associative memory perspective of the momentum, one common technique to enhance the representation power of memory module is to use non-linearity on top of its output (Sun et al. 2024; Behrouz et al. 2025c). That is, we re-formulate Equation 50 as:

$$W_{i+1} = W_i + \sigma\left(\mathbf{m}_{i+1}\left(\boldsymbol{u}_i\right)\right), \quad \text{and} \quad \mathbf{m}_{i+1} = \alpha_{i+1}\mathbf{m}_i - \eta_t \nabla \mathcal{L}^{(2)}(\boldsymbol{m}_i; \boldsymbol{u}_i, \mathbf{I}), \tag{52}$$

where $\sigma(\cdot)$ is an arbitrary non-linearity. As an example, we let $\sigma(\cdot) = \texttt{NewtonSchulz}(\cdot)$, where $\texttt{Newton-Schulz}(\cdot)$ is the iterative Newton-Schulz method (Higham 2008), and $\boldsymbol{m}(\cdot)$ be a linear layer; resulting in Muon (Jordan et al. 2024).

A Toy Example for Long Context in Optimizer. In Section 4.3, we discussed that in complex setups, including continual learning with orthogonal tasks, we may need more complex momentum terms, either with higher capacity or better memory management. To better illustrate the potential gains of other momentum memory designs, we use a toy example of a time-varying curvature. Since the standard momentum acts as a low-pass filter, if the landscape changes at a high frequency, then standard momentum which aims to use the weighted average of past gradients, will be under the influence of irrelevant gradient terms, delaying the convergence. As an illustrative example, consider:

$$\boldsymbol{\psi}(r,\theta) = r^2 + k \times (r - \theta + \alpha \sin(\omega r))^2, \tag{53}$$

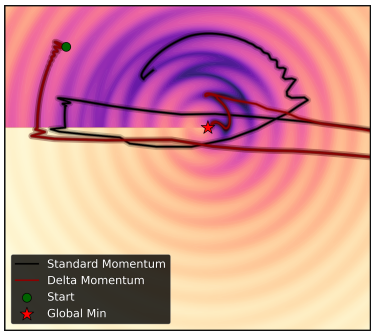

Figure 4: Optimization of function $\boldsymbol{\psi}(r,\theta)$ with standard momentum and our delta momentum.

and aims to optimize it using a standard momentum and our delta momentum. We start the optimization process from point $(r_0, \theta_0) = (-3.5, 2)$ and continue until one of the algorithms converge to the optimal solution. The result is visualized in Figure 4. The delta momentum finds the solution faster, mainly due to its gradient-dependent weight decay that helps the momentum term to decay or stop when it is needed.

4.5 Going Beyond Simple Gradient Descent and Momentum

Coming back to the discussion in Section 3.1 about the pre-training process and backpropagation being a form of associative memory, in this section, we aim to take advantage of NL's viewpoint and present a more general form for gradient descent. As observed in Section 4.1, backpropagation with gradient descent is an associative memory that aims to map the input data to the surprised caused by its predicted output $\nabla_{y_t} \mathcal{L}(W_t; \boldsymbol{x}_t)$:

$$W_{t+1} = W_t - \eta_{t+1} \nabla_W \mathcal{L}(W_t; \boldsymbol{x}_t) = W_t - \eta_{t+1} \nabla_y \mathcal{L}(W_t; \boldsymbol{x}_t) \otimes \boldsymbol{x}_t, \qquad \text{where } \boldsymbol{x}_t \sim \mathcal{D}_{\text{train}}, \tag{54}$$

which from the associative memory perspective and proximal gradient viewpoint is equivalent to:

$$W_{t+1} = \arg\min_W \ \langle W \boldsymbol{x}_t, \nabla_{y_t} \mathcal{L}(W_t; \boldsymbol{x}_t) \rangle + \frac{1}{2\eta_t} \|W - W_t\|_2^2. \tag{55}$$

This step aims at learning the negative of the gradient direction. The main drawback of the dot-product similarity as the inner objective is that its corresponding update rule and so learning algorithm treats each data sample (gradients) independent of the state, meaning that the state of the weights and so previous gradients do not affect the update term to the current state. While this design can be effective for nested problems with independent elements in their context (e.g., i.i.d. samples for training), it can be restrictive for context with highly dependent elements (e.g., tokens in a sequence). Defining $\mathbf{u}_t = -\nabla_{y_t} \mathcal{L}(W_t; \boldsymbol{x}_t)$, one can extend this process to more expressive objectives such as $\mathcal{L}_2$ regression loss:

$$W_{t+1} = \arg\min_W \ \frac{1}{2} \|W \boldsymbol{x}_t - \mathbf{u}_t\|_2^2 + \frac{1}{2\eta_t} \|W - W_t\|_2^2. \tag{56}$$

For the cases that $\boldsymbol{x}_t$ is normalized (e.g. in normalized memory systems or in neural networks with normalization layers, $\|\boldsymbol{x}_t\|_2 = \lambda$), and by defining $\eta'_t = \frac{\eta_t}{1+\eta_t}$, we can use Sherman-Morrison lemma to get (see Appendix C for the details):

$$\begin{aligned} W_{t+1} &= W_t \left(\mathbf{I} - \eta'_t \boldsymbol{x}_t \boldsymbol{x}_t^\top\right) - \eta'_t \nabla_{W_t} \mathcal{L}(W_t; \boldsymbol{x}_t) \\ &= W_t \left(\mathbf{I} - \eta'_t \boldsymbol{x}_t \boldsymbol{x}_t^\top\right) - \eta'_t \nabla_{y_t} \mathcal{L}(W_t; \boldsymbol{x}_t) \otimes \boldsymbol{x}_t, \qquad \text{where } \boldsymbol{x}_t \sim \mathcal{D}_{\text{train}}. \end{aligned} \tag{57}$$

This new algorithm, which based on Delta rule (Prados et al. 1989) we refer to as Delta Gradient Descent (DGD), updates the weights not only with respect to the current elements, but it also incorporates the previous state of weights, resulting in an adaptive decay term based on the current data sample. Next, we discuss a generalized viewpoint about the process of backpropagation with gradient descent, which later will help us to formulate Generalized Gradient Descent family of learning rules:

Training a Deep Neural Network with Backpropagation is a Self-Referential Process:

As discussed earlier in Section 4.1, one common misinterpretation for gradient descent is to view it as a form of linear recurrence (e.g., linear attention). In a conventional linear recurrence, however, keys and values are independent of the state of the memory, and so allows for parallelization of the formulation. The values in "gradient descent as associative memory" viewpoint are a function of the state of the memory, and so it is a self-referential model (Schmidhuber 1993) that controls its own learning process by generating its own values. More formally, one can reformulate the process as:

$$W_{t+1} = W_t + \eta_{t+1} \boldsymbol{v}_t \otimes \boldsymbol{x}_t, \\ \boldsymbol{v}_t = \boldsymbol{f}_{W_t}(\boldsymbol{x}_t) = -\nabla_{y_t} \mathcal{L}(W_t; \boldsymbol{x}_t), \tag{58}$$

which means that at each step $\boldsymbol{v}_t$ is generated by memory W_t and $\boldsymbol{x}_t$ as its input.

Based on the above interpretation, one can define backpropagation with gradient descent in a general form as *any* self-referential model that aims to compress training samples as keys and map them to *self-generated* values to better control its own learning process. Based on this definition, our above formulation of Appendix C, is only a simple instance that uses $\mathcal{L}_2$ regression loss; in general, however, one can define Generalized Gradient Descent (GGD) as:

Definition 5 (Generalized Gradient Descent (GGD) Learning Rule)**.** *Generalized Gradient Descent (GGD) learning rule is a self-referential associative memory that aims to compress data samples and map them to a set of self-generated keys:*

$$W_{t+1} = \arg\min_{W} \tilde{\mathcal{L}}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \texttt{Ret}\left(W, \{W_i\}_{i=t-c+1}^{t}\right), \tag{59}$$

where $\boldsymbol{u}_t$ is a self-generated value:

$$\boldsymbol{u}_t = \boldsymbol{f}_{W_t}(\boldsymbol{x}_t), \tag{60}$$

for some function $\boldsymbol{f}_{W_t}(\cdot)$ parameterized by W_t. Here, $\tilde{\mathcal{L}}(\cdot)$ measures the quality of the mapping, and $\texttt{Ret}(\cdot)$ ensures that the solution for the new instance is not far away from the current state.

Similarly, this formulation can be adapted for the momentum term, resulting in Generalized Momentum (GM). However, it is notable that the momentum itself, is a conventional associative memory and its keys and values are given, or more specifically are generated by a lower-frequency level. In Section 4.2, we explored a special case of this formulation, where $\mathcal{L}(\cdot)$ is L_2 regression loss.

A Note on Optimizers in Continual Learning Setup. As discussed above, optimizers themselves are learning modules or associative memories that aim to compress the gradients into their parameters. These parameters are not necessarily trainable in the conventional terminology but indeed momentum-based optimizers store the knowledge about the loss landscape, helping them to better update the weights. When the "end of pretraining" happens for a neural learning module, the knowledge stored about the distribution of gradients/data, which are stored in the momentum term(s) is removed from the model and so continuing the training without recovering the momentum states can affect the model's ability to learn new capabilities. When the model is in continual learning setup, the knowledge about data is stored in the conventional parameters (optimized with backpropagation), but the knowledge about how the model optimizes itself and about the objective space are optimized in the lower-frequency levels of optimization (e.g., momentum terms).

5 Existing Architectures as Neural Learning Modules

Modern sequence models such as Transformers (Vaswani et al. 2017) and recurrent models (Katharopoulos et al. 2020; Schlag et al. 2021; Sun et al. 2024; Behrouz et al. 2025c) are the backbones of recent advances in language models. Recently, the equivalency of such models with associative memories that aim to learn a mapping from keys to values from data have been studied in different settings and objectives (Liu et al. 2024b; Sun et al. 2024; Behrouz et al. 2025b; Wang et al. 2025). Particularly, we focus on the general framework of Miras (Behrouz et al. 2025b), which defines associative memory as Definition 1 and optimizes the internal objective (called "attentional bias") with a choice of optimization algorithm on an arbitrary class of functions (i.e., memory architecture). While this formulation alone indicates that the well-known architectures are instances of nested systems of associative memory (NSAM), next, we review this equivalency for some learning rules and architectures.

From now on, we assume that keys $\{\boldsymbol{k}_i\}_{i=1}^{L}$, values $\{\boldsymbol{v}_i\}_{i=1}^{L}$, and queries $\{\boldsymbol{q}_i\}_{i=1}^{L}$ are given: they often are defined as the projections of the input, i.e.,

$$\boldsymbol{k}_t = \boldsymbol{x}_t W_k, \qquad \boldsymbol{v}_t = \boldsymbol{x}_t W_v, \qquad \boldsymbol{q}_t = \boldsymbol{x}_t W_q. \tag{61}$$

In this design, since projection parameters (i.e., W_k, W_v, and W_q) are optimized in a lower frequency level, the sequence model component (e.g., self-attention) has a higher frequency and so the learning process of the associative memory happens in a lower level. Accordingly, for the sake of clarity, we only discuss the higher frequency level (i.e., the internal learning process of the associative memory).

Softmax Attention. From the associative memory viewpoint: given keys $\{\boldsymbol{k}_i\}_{i=1}^{L}$, values $\{\boldsymbol{v}_i\}_{i=1}^{L}$, and queries $\{\boldsymbol{q}_i\}_{i=1}^{L}$, `Softmax` attention block (Bahdanau 2014; Vaswani et al. 2017) can be reformulated as a non-parametric solution to the $\ell_2(\cdot)$ regression objective with Nadaraya-Watson estimators (Fan 2018; Zhang et al. 2022):

$$\mathcal{M}^* = \arg\min_{\mathcal{M}} \sum_{i=1}^{L} \mathbf{s}(\boldsymbol{k}_i, \boldsymbol{q}) \|\boldsymbol{v}_i - \mathcal{M}\|_2^2 = \sum_{i=1}^{L} \frac{\mathbf{s}(\boldsymbol{k}_i, \boldsymbol{q})}{\sum_{j=1}^{L} \mathbf{s}(\boldsymbol{k}_j, \boldsymbol{q})} \boldsymbol{v}_i, \tag{62}$$

where L is the sequence length (Sun et al. 2024). This formulation optimizes the memory $\mathcal{M}(\cdot)$ with respect to the entire context; however, one design choice can be to limit the optimization process to the past c tokens, resulting in:

$$\mathcal{M}^* = \arg\min_{\mathcal{M}} \sum_{i=t-c+1}^{t} \mathbf{s}(\boldsymbol{k}_i, \boldsymbol{q}_i) \|\boldsymbol{v}_i - \mathcal{M}\|_2^2 = \sum_{i=t-c+1}^{t} \frac{\mathbf{s}(\boldsymbol{k}_i, \boldsymbol{q})}{\sum_{j=t-c+1}^{t} \mathbf{s}(\boldsymbol{k}_j, \boldsymbol{q})} \boldsymbol{v}_i, \tag{63}$$

which is equivalent to the sliding window attention (SWA). Therefore, attention and its more expressive variants (Wang et al. 2025) also are instances of Definition 1, when instead of gradient descent or other parametric methods, we find the optimal non-parametric solution to the mapping.

RNNs with Hebbian Rule. The first generation of modern recurrent architectures (e.g., Linear attention (Katharopoulos et al. 2020), RetNet (Sun et al. 2023), RWKV (Peng et al. 2023), lightening attention (Li et al. 2025)) are based on Hebbian-like learning rules (Hebb 2005). For this class of models, the inner objective to measure the quality of mapping between keys and values is the dot-product similarity. That is, given a matrix-valued memory $\mathcal{M} \in \mathbb{R}^{d \times n}$, keys and values $\boldsymbol{k}, \boldsymbol{v} \in \mathbb{R}^d$, objective $\tilde{\mathcal{L}}(\mathcal{M}; \boldsymbol{k}_t, \boldsymbol{v}_t) := -2\langle \mathcal{M}\boldsymbol{k}_t, \boldsymbol{v}_t \rangle$, and a kernel $\phi(\cdot)$, we optimize the equivalent associative memory optimization problem (see Definition 1) with gradient descent and weight decay, resulting in:

$$\mathcal{M}_t = \alpha_t \mathcal{M}_{t-1} - \eta \underbrace{\nabla_{\mathcal{M}_{t-1}} \tilde{\mathcal{L}}\left(\mathcal{M}_{t-1}; \phi(\boldsymbol{k}_t), \boldsymbol{v}_t\right)}_{-\boldsymbol{v}_t \phi\left(\boldsymbol{k}_t^\top\right)} = \alpha_t \mathcal{M}_{t-1} + \eta_t \, \boldsymbol{v}_t \phi\left(\boldsymbol{k}_t^\top\right), \tag{64}$$

which recovers the original linear attention recurrence (Katharopoulos et al. 2020). Given different settings for α_t (i.e., either is 1, learnable, channel-wise, and/or input-dependent) and also $\phi(\cdot)$ (i.e., identity, polynomial kernels, etc.), the above recurrence recovers different variants of linear attention with Hebbian rule (Katharopoulos et al. 2020; Sun et al. 2023; Arora et al. 2024; Beck et al. 2024; Kacham et al. 2024; Peng et al. 2024). Therefore, the variants of linear attention with Hebbian rule can be reformulated as the process of an optimization problem, in which the memory aims to learn the mapping between keys and values based on dot-product similarity objective, with gradient descent.

RNNs with Delta Rule. To improve the memory management and to enhance the memory capacity of the above group, several studies suggest replacing Hebbian rule with Delta rule as the learning algorithm in recurrent neural networks (Schlag et al. 2021), resulting in models such as DeltaNet (Schlag et al. 2021), Longhorn (Liu et al. 2024b), and RWKV7 (Peng et al. 2025b). When letting $\mathcal{M} \in \mathbb{R}^{d \times n}$, delta rule is equivalent to optimizing MSE objective $\tilde{\mathcal{L}}_t = \|\mathcal{M}_t \boldsymbol{k}_t - \boldsymbol{v}_t\|_2^2$ with $\mathrm{Ret}_t(\mathcal{M}, \mathcal{M}_{t-1}) = \|\mathcal{M}_t - \mathcal{M}_{t-1}\|_F^2$ as local retention, and stochastic gradient descent as the optimizer:

$$\mathcal{M}_t = \mathcal{M}_{t-1} - \eta_t \underbrace{\nabla_{\mathcal{M}_{t-1}} \tilde{\mathcal{L}}\left(\mathcal{M}_{t-1}; \phi\left(\boldsymbol{k}_t\right), \boldsymbol{v}_t\right)}_{\left(\mathcal{M}_{t-1}\boldsymbol{k}_t - \boldsymbol{v}_t\right)\boldsymbol{k}_t^\top} = \left(\mathbf{I} - \eta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top\right) \mathcal{M}_{t-1} + \eta_t \, \boldsymbol{v}_t \boldsymbol{k}_t^\top. \tag{65}$$

Using other forms of retention gates (e.g., $\mathrm{Ret}_t(\mathcal{M}, \mathcal{M}_{t-1}) = \|\mathcal{M}_t - \alpha_t \, \mathcal{M}_{t-1}\|_F^2$), optimization algorithms with weight decay (e.g., regularizing with $\|\mathcal{M}_t\|_q^q$ for a given $q > 0$), multiple steps of gradient descent, and/or different formulations of

learnable parameters such as η_t and α_t can result in diverse variants of delta rule (Irie et al. 2021; Liu et al. 2024b; Sun et al. 2024; Behrouz et al. 2025b; Hu et al. 2025; Peng et al. 2025b; Siems et al. 2025; Wang et al. 2025). Therefore, Delta rule and its variants are all instances of an optimization problem, in which the model aims to learn a mapping between keys and values based on the L_2-regression objective.

Beyond Conventional Learning Rules: Omega, Oja's, and Non-Euclidean Learning Rules. More recently, there have been growing interests in designing architectures from the associative memory perspective (see Definition 1) and use more complex internal objectives, and/or optimization algorithms, resulting in learning algorithms beyond Delta and Hebbian rules (Irie et al. 2022a; Von Oswald et al. 2023; Behrouz et al. 2025a,b; Zhang et al. 2025). More specifically, to enhance the stability of Hebbian rule (discussed in Equation 64), Irie et al. (2022a) introduced OjaNet based on Oja's rule (Oja 1982) with the following recurrence:

$$\mathcal{M}_t = \alpha_t \mathcal{M}_{t-1} + \eta_t \, \boldsymbol{v}_t \left(\phi(\boldsymbol{k}_t)^\top - \mathcal{M}_{t-1}^\top \boldsymbol{v}_t\right). \tag{66}$$

In the associative memory formulation (as in Definition 1), this recurrence can simply be reformulated as one step of gradient descent as:

$$\mathcal{M}_t = \mathcal{M}_{t-1} - \eta_t \underbrace{\nabla_{\mathcal{M}_{t-1}} \tilde{\mathcal{L}}\left(\mathcal{M}_{t-1}; \phi\left(\boldsymbol{k}_t\right), \boldsymbol{v}_t\right)}_{\mathcal{M}_{t-1}^\top \boldsymbol{v}_t - \boldsymbol{v}_t \phi(\boldsymbol{k}_t)^\top}, \tag{67}$$

where $\tilde{\mathcal{L}}(\mathcal{M}; \boldsymbol{k}_t, \boldsymbol{v}_t) = -2\langle \mathcal{M}\boldsymbol{k}_t, \boldsymbol{v}_t\rangle + \|\mathcal{M}^\top \boldsymbol{v}_t\|_2^2$ and $\phi(\cdot)$ is a kernel (Irie et al. 2022a, 2025). Although this design enhances the Hebbian learning rule by enforcing a unit-norm constraint for the single-neuron, it has been reported to empirically underperform models based on Delta learning rule (Irie et al. 2022a). To further enhance the Delta rule through the design of more expressive objectives, recently, Behrouz et al. (2025b) suggested going beyond Euclidean spaces and use $L_p = \|\cdot\|_p^p$ norm for the internal regression objective, showing better empirical performance and robustness in long context tasks compared to Delta rule and its variants.

While the majority of learning rules are online update mechanisms–meaning that at each state, the models only need to keep the memory and the *current* (batch of) input–Omega rule (Behrouz et al. 2025a) suggest an update rule based on a set of past (batches of) inputs (or all inputs). More specifically, given a memory $\mathcal{M}$ with an arbitrary structure, keys and values $\boldsymbol{k}, \boldsymbol{v} \in \mathbb{R}^d$, an arbitrary objective $\tilde{\mathcal{L}}(\mathcal{M}; \boldsymbol{k}_t, \boldsymbol{v}_t)$, and a kernel $\phi(\cdot)$, Omega rule is defined as:

$$\mathcal{M}_t = \alpha_t \, \mathcal{M}_{t-1} - \sum_{i=t-c+1}^{t} \gamma_{t,i} \; \tilde{\mathcal{L}}(\mathcal{M}_t; \phi(\boldsymbol{k}_i), \boldsymbol{v}_i), \tag{68}$$

where $c \geq 1$ is the local window of cached inputs. Note that in the special case of $\gamma_{t,i} = 1$ and c equal to the entire context length, the optimal solution of the above design collapses into an online case, where the update rule only depends on the current state and the current input (Von Oswald et al. 2023). For further discussion with more details about representing architectures as associative memories and so an optimization problem, we refer the reader to Behrouz et al. (2025b).

A Note on Gating in Modern Sequence Models. One of the recent architectural changes in modern language models is the gating of a linear layer's output with the output of the sequence model. Despite significant improvement resulted by this method, it is still unclear that how it enhances the performance. As we discussed in Figure 3 and its corresponding example, the main difference between feedforward network and modern recurrent memory modules (e.g., linear attention (Katharopoulos et al. 2020) or deep memory modules (Behrouz et al. 2025c)) when their initial state of the memory is meta-learned, is the second level in memory modules that perform in-context learning and adapt its state with the context. From this viewpoint, when the initial value of the memory is not meta-learned, it only relies on the in-context adaption of the memory and so there is no persistent memory system that stores the knowledge of pre-training in this block. Therefore, when the initial value of memory is not meta-learned, which is common in earlier variants of linear transformers, the gating of linear attention acts as a persistent memory and the initialization of the memory module.

5.1 Revisiting the Human Brain Perspective of Nested Learning

In Section 1.1, we discussed how structure in human brain is uniform and reusable, and if we need a new architecture in deep learning, or if our beliefs about the heterogeneity of current models need to be revisited. In the previous sections,

we observed that both optimization process of neural networks as well as neural architectures can be formulated as a set of nested and/or parallel optimization problems, in which the memory structure is a feedforward layer (e.g., either Deep MLPs, linear layers, etc.) and the objective is optimized with gradient descent or Newton's methods.

From this perspective, modern architectures, are a set of artificial neurons (i.e., linear or deep feedforward networks), and each group of neurons has their own internal objective and so update mechanism. To this end, as a simple example, let us recall the AdaTransformer in Figure 3: Given $X = \{\boldsymbol{x}_i\}_{i=1}^{T}$ as the input sequence, the output of the block is computed as (For the sake of simplicity, we assume $\texttt{MLP}(\cdot) = \cdot\, W_{\texttt{MLP}}$, and remove normalizations):

$$
\begin{aligned}
&\boldsymbol{k}_t = \boldsymbol{x}_t W_k, \qquad \boldsymbol{v}_t = \boldsymbol{x}_t W_v, \qquad \boldsymbol{q}_t = \boldsymbol{x}_t W_q, \\
&\boldsymbol{y}_{\texttt{attn},t} = \texttt{Attn}\left(\boldsymbol{k}_t, \boldsymbol{v}_t, \boldsymbol{q}_t\right), \\
&\boldsymbol{y}_{\texttt{block},t} = \boldsymbol{y}_{\texttt{attn},t}\, W_{\texttt{LinAttn},t}, \qquad \text{where} \\
&W_{\texttt{LinAttn},t} = W_{\texttt{LinAttn},t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^{\top},
\end{aligned}
\tag{69}
$$

The lowest frequency level is responsible for the optimization of W_k, W_v, and W_q, all of which are feedforward networks and so are uniform. The attention itself is also the non-parametric matrix-valued solution of a regression objective, again verifying that the structure is a matrix of artificial neurons (i.e., parameters). Finally, the Linear Attention++ component is equivalent to the optimizing the dot-product similarity of the mappings over the linear class of functions. Therefore, all the parameters are matrix-valued or deep feedforward layers, meaning that the only difference in the components of an architecture is their level, objective, and/or learning update rule.

> **Modern Deep Learning Models Have Uniform and Reusable Structure**
>
> Neural learning modules consist of a set of feedforward networks, each of which are optimized in different levels and time scales. The heterogeneity we observe in deep learning architectures, however, is due to the lack of view to this new NL's axis, resulting in observing only the solution of optimization problems and so causing the illusion of deep learning architectures.

6 Takeaways and Revisiting Common Terms

In the previous sections, we discussed the concept of nested learning and how existing well-known components of neural networks such as popular optimizers and architectures fall under the NL paradigm. In this section, we discuss the takeaways, connection of different concepts, and the implications of NL perspective on common terms.

Memory and Learning. For a long period of time, in machine learning models, memory have been treated as a separate block with a *clear* distinction between its parameters and other components. Such designs often assume a short and/or long-term memory blocks, where short-term memory is responsible for the local context, while long-term memory is the storage for the persistent knowledge in models. In human brain, however, memory is considered as a distributed interconnected system without a clear known components that are independently responsible for short or long-term memory. In NL, we build upon a common terminology for memory and learning in neuropsychology literature, indicating that: Memory is a neural update caused by an input and so learning is the process of acquiring useful memory (Okano et al. 2000). From this viewpoint, any update by gradient descent (or any other optimization algorithms) in any levels of neural learning module is considered as a form of memory. Interestingly, our findings in Section 4.1 on gradient descent being (self-referential) associative memory is aligned with this terminology. Furthermore, based on this terminology, in continuum memory system, the neural updates are applied at different frequencies and so memories are stored with different time scales, resulting in more robust memory management with respect to catastrophic forgetting.

> **Memory and Learning from Nested Learning Perspective:**
>
> Memory is not an isolated system and is distributed throughout the parameters. Particularly, any update that is caused by the input is a stored memory in the neural network, and the process of effectively store, encode, and in general acquire such memories is referred to as learning process.

A General Note on Parameters of a Model. The parameters of a model are one of its critical components that shape the units for knowledge storage, internal computations, and adaptability. Over the past decades, only (a subset of) the parameters in the architecture of machine learning models have been referred to as the *learnable* entities, mainly due to the fact that they are the only components that we have been directly and intentionally optimizing over the training data. From the NL viewpoint, such parameters are placed in the lowest frequency level (the highest level) and are updated for every (batch of) samples. They are, however, are not the only parameters that contributes to the model internal computation, knowledge storage, and adaptability. As discussed earlier in Section 4.2, momentum is an example of such cases, where its parameters are updated over time (by gradient descent) and stores the knowledge about the loss landscape of the model so far. Such information are critical when the model is continually learn, mainly due to the fact that to find an effective solution, the optimizer needs to have more information about the global properties of the loss landscape. Another example of such cases is the memory (or hidden state) of recurrent neural networks. Although those parameters are not directly optimized in the lowest frequency level, they store important knowledge about the current context. With the change of the context, the compressed knowledge in these parameters are removed due to the lack of knowledge transfer between these levels.

> **Models Have More Parameters Than We Knew:**
>
> *The parameters of a neural learning module are not limited to those optimized in the pre-training level; all parameters that appear in the NL representation of the model contribute to its performance and expressivity.*

More Computations per Neuron. One common misinterpretation about the concept of NL is to restrict the model design and stacking multiple levels to CMS case. In general, stacking levels can help the model to enhance the depth of computations and also perform more internal computations per each parameter in the lowest frequency level. One example of such designs is Muon optimizer and $\texttt{NewtonSchulz}_k(\cdot)$ operation (see Equation 42 - 44), which we showed that is equivalent to an internal optimization process. In this design, per each step of momentum update, we need k-steps of internal process to learn how to map gradients to an orthogonal space.

In-Context Learning. Throughout this paper, we use the most general definition of "in-context learning" and refer to it as the ability of a model to adapt to and learn from a given context. Following the definition of NSAM, each block or level has its own context flow and so any neural update or adaptation to that context is considered as a form of in-context learning. Due to the popularity of Transformers as well as being the first model that in-context learning is studied for, the concept of in-context learning sometimes is referred to as conditioning the output on the entire context. Considering the general term of in-context learning, this formulation is only one of the instances of in-context learning, which we referred to as *non-parametric in-context learning*. In general, however, the memory in recurrent models is performing in-context learning, in which the output is conditioned on the *compressed context*. Therefore, from NL perspective, all the levels are performing in-context learning but on their own context flow with their own learning update and optimization process.

Building on this definition, in-context learning is a model's capability that is transparent from its NL representation, and per se it is not an emergent characteristic but a direct consequence of having multiple levels in the NL representation of the neural learning module. Although this might seem contradictory with previous claims about ICL being an emergent characteristic (Brown et al. 2020; Singh et al. 2023), it is notable that the good performance of the model in ICL tasks also requires a powerful low-frequency level, enabling the high frequency level to adapt fast. When the model is not well-trained, the higher-frequency level is on its own to learn from the context. This setup might result in a poor performance, mainly due to the fact that there might be not enough data in context to allow the high-frequency parameters to converge.

(Test-Time) Learning/Memorization. Recently, the concept of test time training (Sun et al. 2024; Wang et al. 2025) or test time memorization (Behrouz et al. 2025b) has gained popularity as a backbone framework to design powerful sequence models. In these frameworks, given the context, a new component/block aims to compress the context into its parameters using a learning rule and objective function. In this formulation when the context is removed the acquired in-context knowledge diminishes along with it. As also discussed in the previous part, this update mechanism and learning process is indeed an instance of *"parametric in-context learning"*:

> **Test Time Training/Memorization are Instances of In-Context Learning**
>
> *The concepts commonly referred to as test-time training and test-time memorization are in fact instances of parametric in-context learning, where the acquired in-context knowledge does not persist once the current context is removed.*

Pre-training and Test Time. From the nested learning perspective, the lowest frequency level (i.e., the highest level) corresponds to a learning phase that is often referred to as pre-training. Accordingly, pre-training is one of the levels and so has its own context flow (i.e., pre-training dataset), objective (e.g., next token prediction), and optimization process (e.g., AdamW). Accordingly, one can interpret the pre-training as one of the possible instances of in-context learning, where the context is the entire pre-training data.

> **Pre-training is In-Context Learning with Ultra-Large Context Length:**
>
> *From NL's viewpoint pre-training is only one of the possible instances of in-context learning, where the context is the entire pre-training data. The distinction of training and test time in models is the results of disconnecting the knowledge transfer process from the highest frequency level (e.g., the context of Transformers) to the low frequency levels (i.e., pre-training).*

This formulation and viewpoint is specifically important when we shift from pre-training paradigm to models that are capable of continually interact and learn from data/world (e.g., Sutton (2025)):

Continual Learning. From NL's perspective each phase of training for a model is defined as one of the low-frequency levels, which by design, we might want to stop the data processing in one level (e.g., "End of Pre-training"), or continue it without any knowledge transfer to other levels (e.g., conventional formulation of in-context learning in Transformers). Accordingly, any machine learning model, no matter if it is during its pre-training or at test time, is performing continual learning as given a data sample, it requires performing internal computations to provide the output. However, the knowledge from that learning might not last or transfer to more persistent levels, mainly due to the lack of knowledge transfer between levels.

> **No Training or Test Time in Neural Learning Modules:**
>
> *For a neural learning module, there is no boarder and clear distinction between training and test time. The model only experiences two different states: when it receives information as input, or when it is an isolated learning system.*

We also revisit some of the common terms in the architectural design in the following:

Existing Architectural Backbones and Hybrid Models. As discussed earlier in Section 5.1, from the NL's perspective all modern architectures are uniform and in fact are feedforward layers (linear or non-linear MLP blocks) that are trained based on their own context flow and optimization problem. When viewing models from deep learning perspective, we see the final solution of such optimization problem (i.e., we see attention rather than a non-parametric solution to a regression loss), which results in the illusion of having distinct and non-uniform architectures.

> **Recurrent Models are Replacing MLP Blocks:**
>
> *From NL's viewpoint, (deep or linear memory) recurrent models are MLP blocks that a new level is added to their internal computation. Accordingly, existing hybrid architectures can be seen as conventional Transformer models, when we added a new level of computation to some of the MLP blocks.*

Another important aspect of understanding the existing architectures during their so-called "pre-training" phase is to understand how they transfer their knowledge from one level to another.

> **Knowledge Transfer from In-Context Learning**
>
> *Although both modern deep and linear recurrent models are unified using associative memory perspective, there is still an important difference between the existing instances: While deep memory modules such as Titans, Atlas, Miras, and TTT take advantage of knowledge transfer from their high-frequency level to their lower-frequency level through meta-learning the initial state of memory, most linear memory recurrent models have no knowledge transfer process between their levels.*

Neural Learning Module as an Inter-Connected System. One of the critical messages in NL is the fact that neural learning modules are inter-connected systems, meaning that the design of each component can significantly affect the

design of other parts. This fact motivates follow up and future studies to better understand how one can properly design a neural learning module with all components work together in a harmony. As an example of such, the context of optimizers (i.e., gradients) is generated by the architecture component. Therefore, different architectures might show different characteristics in the generated gradient patterns and so one optimizer might not be the best option for all the architectures (Zhang et al. 2024b).

Architectures Generates the Context for Optimizers

Neural learning modules are inter-connected systems, where architecture generates the context for optimizers (i.e., gradients). Therefore, the proper memory management of gradients (i.e., optimization algorithm) relies on the choice of architectures. In future, when viewing models as a neural learning modules, we need to design architecture specific optimizers so this inter-connected system works perfectly in harmony.

Optimizers vs. Learned Optimizers. Finally, we want to emphasize that our formulation of momentum, gradient descent, and/or other gradient-based optimizers show that they are associative memory modules aiming to compress the data and gradients into their parameters. Such update and compression process is based on gradient descent and so has a very similar nature to the learning process of learned optimizers. From the NL's viewpoint, both vanilla optimizers as well as learned optimizers are instances of the same concept but with different frequency and context flow: Although the parameters of learned optimizers are located in the lowest frequency level (i.e., to train and be optimized along with other parameters in the pre-training), the parameters of vanilla optimizers are located in their own level and so has their own gradient flow.

7 Continuum Multi-Timescale Memory System

Existing architectural backbones consist of (1) a *working memory* module (e.g., attention), which is responsible to actively fuse the information across sequence length, and (2) a feed-forward layer (e.g., MLP) that fuse information across features and acts as the persistent memory or knowledge storage of pre-training phase. From the NL perspective, pre-training is the phase that the most outer level of the learning module is updated over its *limited* context flow. Accordingly, in the continual setup, such pre-training phase is also rarely updated over time, and so its corresponding knowledge storage needs to rarely be updated over time. Given this intuition, we extend the traditional view-point of long-term/short-term memory system and suggest a knowledge storage feed-forward for each level (frequency domain).

7.1 Continuum Memory System (CMS)

Given the definition of frequency (Definition 2), Continuum Memory System (CMS) is formalized as a chain of MLP blocks $\texttt{MLP}^{(f_1)}(\cdot), \ldots, \texttt{MLP}^{(f_k)}(\cdot)$, each of which associated with a chunk size of $C^{(\ell)} := \frac{\max_t C^{(i)}}{f_\ell}$ such that given input $x = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$ the output of the chain is calculated as (we disregard normalizations for the sake of clarity):

$$\boldsymbol{y}_t = \texttt{MLP}^{(f_k)}(\texttt{MLP}^{(f_{k-1})}(\cdots \texttt{MLP}^{(f_1)}(\boldsymbol{x}_t))), \tag{70}$$

where the parameters of ℓ-th MLP block, i.e., $\boldsymbol{\theta}^{(f_\ell)}$, are updated every $C^{(\ell)}$ steps:

$$\boldsymbol{\theta}^{(f_\ell)}_{i+1} = \boldsymbol{\theta}^{(f_\ell)}_{i} - \begin{cases} \sum_{t=i-C^{(\ell)}}^{i} \eta_t^{(\ell)} f(\boldsymbol{\theta}^{(f_\ell)}_t; \boldsymbol{x}_t) & \text{if } \; i \equiv 0 \pmod{C^{(\ell)}}, \\ 0 & \text{otherwise.} \end{cases} \tag{71}$$

Here $\eta_t^{(\ell)}$ are learning rates corresponds to $\boldsymbol{\theta}^{(f_\ell)}$, and $f(\cdot)$ is the error component of an arbitrary optimizer (e.g., $\nabla \mathcal{L}(\boldsymbol{\theta}^{(f_\ell)}_t; \boldsymbol{x}_t)$ in gradient descent). The conventional Transformer block (Vaswani et al. 2017) is a special instance of this formulation, where $k = 1$. It is notable that Equation 71 provides an important interpretation: parameters $\boldsymbol{\theta}^{(f_\ell)}_t$ are responsible for compressing their own context into the their parameters and so they are a representative of abstract knowledge of their context.

As discussed in Section 3.3, different levels might have different process of knowledge transfer. Accordingly, while the above formulation suggests a spectrum of memory systems in different levels and so with different frequencies, their connections can vary based on the design. In the following, we discuss some potential variants:

Nested Continuum Memory Systems. The first variant is a fully nested continuum memory system, in which the initial state of the MLP block in level $s+1$ is meta-learned in level s. This design allows for higher-order in-context learning ability, where each of the levels has its own context flow and re-initialized after the end of the context. More specifically, given an arbitrary $1 \leq s \leq k$,

$$\boldsymbol{\theta}_0^{(f_{s+1})} = \arg\min_{\Phi} \; \mathbb{E}_{\mathcal{T}\sim C^{(s)}} \left[\ell(\Theta, \mathcal{T}; \Phi) \right], \tag{72}$$

where $C^{(s)}$ is the context length of the MLP block in s-th level. Following this design, at the end of the optimization process of each block (i.e., after $\lceil C^{(s)}/C^{(s+1)} \rceil$ steps.) the value of the memory will be re-initialized to $\boldsymbol{\theta}_0^{(f_{s+1})}$. Note that the update mechanism of each block in its own level remain unchanged (i.e., Equation 71).

Sequential Continuum Memory Systems. In the second variant, the MLP blocks are located sequentially (i.e., the output of the MLP block in level s is the input for the MLP block in level $s+1$) and also the initial state of MLP blocks are all connected through backpropagation in the lowest frequency level. Given an arbitrary $1 \leq s \leq k$,

$$\boldsymbol{\theta}_0^{(f_s)} = \arg\min_{\Phi} \; \mathbb{E}_{\mathcal{T}\sim C^{(1)}} \left[\ell(\Theta, \mathcal{T}; \Phi) \right], \tag{73}$$

where $C^{(1)}$ is the context length of the MLP block in the lowest frequency level. Since the initial state of all memories are meta-learned in the lowest frequency, the most persistent knowledge of all components is the compression of the same context flow.

Independent (Head-wise) Continuum Memory Systems. In this variant, we keep the knowledge transfer process in Equation 73, but change the output computation in Equation 70. While the previous formulation designs the memory system as a sequence of blocks, and so making their input/out dependent to each other, this variant uses independent blocks with different context length and then combine them using an aggregation process:

$$\boldsymbol{y}_t = \texttt{Agg}\left(\texttt{MLP}^{(f_k)}(\boldsymbol{x}_t), \texttt{MLP}^{(f_{k-1})}(\boldsymbol{x}_t), \cdots, \texttt{MLP}^{(f_1)}(\boldsymbol{x}_t) \right). \tag{74}$$

The above $\texttt{Agg}(\cdot)$ is an arbitrary function that aggregates all the inputs to compute the output. For example, one straightforward and simple design choice is to use a learnable weighted sum of the input.

CMS Design Helps with Continual Learning. Based on the design of CMS, a fair question is to ask: Why and how CMS can help with longer context length and generally continual learning. Here, we provide a simple answer to this question: Viewing MLP blocks in CMS as the storage of model's knowledge catastrophic forgetting can happen when we update a block and as its result, the old knowledge stored in its parameters are forgotten. In CMS design, however, when updating an arbitrary block of $\texttt{MLP}^{(f_s)}(\cdot)$ for some $1 \leq s \leq k$, the potentially forgotten knowledge from $\texttt{MLP}^{(f_s)}(\cdot)$ is still stored in other components such as $\texttt{MLP}^{(f_{s'})}(\cdot)$, where $s' < s$. Also, in this case (i.e., the knowledge is already forgotten from $\texttt{MLP}^{(f_s)}(\cdot)$ but it is still in $\texttt{MLP}^{(f_{s'})}(\cdot)$ for $s' < s$) the knowledge transfer through backpropagation (for their initial state) can circle back the knowledge to $\texttt{MLP}^{(f_s)}(\cdot)$, resulting in a loop through time dimension, and so hardly forgetting important knowledge.

Is CMS Efficient Enough? A common concern when updating the parameters of a model in a continual manner is its efficiency. Therefore, a fair question is to ask if CMS causes significant computational overhead for the model. To answer this question, let us recall from Section 5 that modern recurrent neural networks are also continually updating a subset of their parameters (i.e., their memory state). These parameter updates, however, take advantage of sequence parallelization as well as updating only a small number of parameters. To this end, for CMS, we highlight two points:

- In the CMS design, at each time, updates are restricted to blocks approaching their scheduled update time (based on their frequency). As a simple example, consider a Transformers but with replacing its MLP blocks with CMS (later in Section 8, refer to this variant as Hope-Attention). Let the model have L_{layer} layers, 4 levels of MLP blocks in CMS with highest frequency of $\hat{f}$, and hidden dimension of d_{in}. On average, the update cost is for $O\left(\frac{1}{\hat{f}} \times \frac{L_{\text{layer}}}{5} \times d_{\text{in}}^2\right)$ of parameters, which consists of only a small number of parameters at each time.

- The update mechanism of Equation 71, not only helps with the enhancing the persistent memory of the model, but it also unlocks the sequence parallelization for higher frequency levels. More specifically, for input $\boldsymbol{x}_i$ when $i \not\equiv 0 \pmod{C^{(\ell)}}$ there is no sequential process inside the chunk and so all the computations for tokens correspond to different values of $i \not\equiv 0 \pmod{C^{(\ell)}}$ can be done in parallel. The details of such training algorithm is the same as the training procedure in Sun et al. (2024) and Behrouz et al. (2025c).

Therefore, in summary, CMS can be fast in practice, mainly due to the fact that it updates only small number of parameters at each time, and also its design unlocks sequence parallelization.

7.2 Continuum Memory System In Optimizers

As a proof of concept and to support the effectiveness of CMS in different context flows, in this section we present Multi-scale Momentum/Memory Muon (M3) optimizer. Particularly, we aim to use NL's associative memory viewpoint to design an optimizer that not only compress the recent gradients effectively, but it also has a capability of incorporating the information about long past gradients. In Appendix B (Equation 101) we discuss that how Adam optimizer is an instance of associative memory, in which the gradients are mapped to their variance until that point. Following the discussion about the need of long-context capability of optimizers in Section 4.3, we first replace the simple associative memory formulation of H term in Equation 102 with our CMS (independent variant, Equation 74) with a two-level memory system, which we refer to the memories as $\boldsymbol{M}^{(1)}$ and $\boldsymbol{M}^{(2)}$:

$$\begin{aligned} \boldsymbol{M}_t^{(1)} &= \boldsymbol{M}_{t-1}^{(1)} + \beta_1 \boldsymbol{g}_t, \\ \boldsymbol{M}_t^{(2)} &= \boldsymbol{M}_t^{(2)} - \beta_2 \begin{cases} \sum_{i=t-\hat{C}}^{t} \boldsymbol{g}_i & \text{if} \quad t \equiv 0 \pmod{\hat{C}} \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \tag{75}$$

Algorithm 1 Multi-scale Momentum Muon (M3)

Input: Initial weights $\boldsymbol{\Theta}_0$, objective $\mathcal{L}(\cdot)$, learning rate $\eta > 0$, Newton–Schulz steps T, momentum factor $1 > \beta_1, \beta_2, \beta_3, \alpha \geq 0$, $\epsilon > 0$, frequency f;

Initialize momentums: $\boldsymbol{M}_0^{(1)}, \boldsymbol{M}_0^{(2)} \leftarrow \boldsymbol{0}$, $\boldsymbol{V}_0 \leftarrow \boldsymbol{0}$;
for lower-frequency iteration $k = 0, 1, 2, \ldots$ **do**
  Slow Memory: $\boldsymbol{M}_t^{(2)} = \boldsymbol{M}_{t-1}^{(2)} + \beta_3 \sum_{i=(k-1)f}^{kf} \boldsymbol{g}_i$;
  $\boldsymbol{O}_t^{(2)} \leftarrow \texttt{Newton-Schulz}_T\left(\boldsymbol{M}_t^{(2)}\right)$;
  for $t = kf+1, kf+2, \ldots, (k+1)f$ **do**
    Compute Gradient: $\boldsymbol{g}_t = \nabla_{\boldsymbol{\Theta}_t} \mathcal{L}(\boldsymbol{\Theta}_t)$;
    First Momentum: $\boldsymbol{M}_t^{(1)} = \boldsymbol{M}_{t-1}^{(1)} + \beta_1 \boldsymbol{g}_t$;
    Second Momentum: $\boldsymbol{V}_t = \boldsymbol{V}_{t-1} + \beta_2 \boldsymbol{g}_t^2$;
    $\boldsymbol{O}_t^{(1)} \leftarrow \texttt{Newton-Schulz}_T\left(\boldsymbol{M}_t^{(1)}\right)$;
    $\boldsymbol{\Theta}_t \leftarrow \boldsymbol{\Theta}_{t-1} - \eta \frac{\boldsymbol{O}_t^{(1)} + \alpha\, \boldsymbol{O}_t^{(2)}}{\sqrt{\boldsymbol{V}_t + \epsilon}}$;
  end for
end for

where $\hat{C}$ is the chunk size that we update the lower-frequency momentum term. Finally, to aggregate the momentum terms (the choice of $\texttt{Agg}(\cdot)$ in Equation 74), we use a simple weighted summation with the use of parameter $\alpha > 0$ as the coefficient of $\boldsymbol{M}_t^{(2)}$. Following our discussion on the importance of $\texttt{Newton-Schulz}_T(\cdot)$ to map the gradients to a proper metric space (see Section 4.2 and Equation 43), following Muon (Jordan et al. 2024), we use $\texttt{Newton-Schulz}_T(\cdot)$ on the output of the momentum terms, before aggregating them with weighted sum. This helps the associative memory to better manage its capacity by updating its parameters in a proper direction. The pseudocode for M3 is in Figure 1. In summary, one can say that M3 is the combination of Adam (Kingma et al. 2014a), Muon (Jordan et al. 2024), and our Continuum Memory System.

Notably, this optimizer is designed as a proof-of-concept to support the design of CMS. The M3 optimizer per se, however, might suffer from computational overhead and so face challenges when scaling to larger networks (see Figure 12).

7.3 Ad-hoc Level Stacking: Initializing CMS with Pre-Trained Models

In our discussion on Figure 3, we observed that the initial state of the memory modules are optimized in lower-frequency levels and so one can interpret them as MLP blocks in the vanilla Transformer architectures (Vaswani et al. 2017). Therefore, a natural question is if we can leverage pre-trained models to initialize CMS blocks. One of the important advantages of NL is its flexibility to view and modify parameters in different levels. That is, since each level has its own context flow and optimization process, one can simply initialize the parameters in each level independently so that it helps the model to adapt faster to the levels' context flow. To this end, in this section, we suggest initializing the parameters in a level with a model's pre-trained weights. More formally, given a CMS with $\{\texttt{MLP}^{(f_i)}(\cdot)\}_{i=1}^{k}$, and a set of pre-trained MLP blocks $\{\texttt{MLP}_{\text{pre-trained}_i}(\cdot)\}_{i=1}^{k}$ we use Equation 71 to update $\{\texttt{MLP}^{(f_i)}(\cdot)\}_{i=1}^{k}$ in different levels; we, however, use the trained parameters of $\{\texttt{MLP}_{\text{pre-trained}_i}(\cdot)\}_{i=1}^{k}$ as the initial state of CMS blocks: $\texttt{MLP}_0^{(f_i)}(\cdot) = \texttt{MLP}_{\text{pre-trained}_i}(\cdot)$.

Why This Initialization Should Work? In NL, when there is a knowledge transfer process between two levels, the

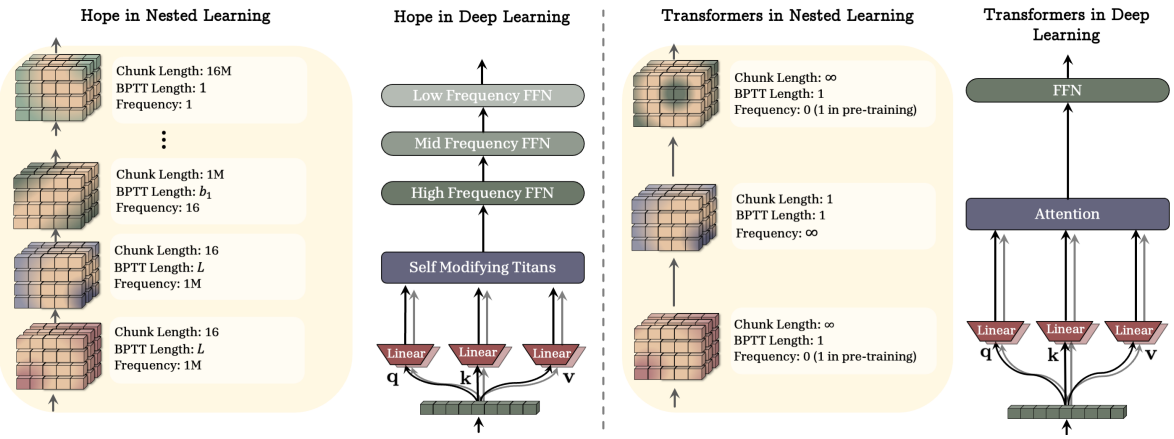

Figure 5: A comparison of Hope architectural backbone with Transformers (Normalization and potential data-dependent components are removed for the sake of clarity).

higher frequency level can take advantage of the knowledge stored in the lower frequency level and so adapt faster to its own context flow. The internal learning rate in the higher-frequency level, however, can control the capacity of the model for adaptability. That is, consider the above case, where all the blocks are initialized with pre-trained MLP blocks, setting $\eta_t^{(\ell)} \to 0$ keeps the updated memory blocks close to their initial states, resulting in directly using of pre-trained blocks, without adaption. Later in Section 9, we use this method to adapt pre-trained Transformer architectures to the Hope's setup.

8 Hope: A Self-Referential Learning Module with Continuum Memory

As we discussed earlier in Section 5.1, architectures in nested learning are uniform, i.e., a set of feedforward neural network blocks, each of which with its own context, update frequency, and internal objective. Sequence models–a common term to refer to blocks often with the highest update frequency that fuse information across tokens in the input sequence–are critical components for memory management and in-context learning ability of models. Following our earlier discussion in Section 5, modern sequence models can be seen as associative memories and so are nested optimization problems. From this perspective, global softmax attention or its more expressive higher-order variants are perfect memories (forcing to cache all past tokens) with frequency update of infinity as they are non-parametric solutions for optimizing (local) L_2-regression objective with Nadaraya-Watson estimators (Fan 2018; Zhang et al. 2022) (see Equation 62). Therefore, parametric solutions (e.g., modern RNNs) for the similar objectives and when the parameter search space are the same (i.e., matrix-valued memory) are not expected to outperform softmax attention when the model size and data scales. To this end, and to design powerful sequence models, we need to understand where Transformers are limited and how one can overcome such limitations.

From the nested learning perspective, Transformers are two-level components, where projections and MLP blocks are optimized in the first level and the second level is responsible for in-context learning with finding the non-parametric solution and so conditioning the output on the context. This design, however, has limited computational depth as also stated in recent studies on state-tracking and similar computational capabilities of models (Merrill et al. 2024; Sanford et al. 2024; Grazzi et al. 2025). Furthermore, Transformers' parameters are static throughout their context, meaning that their found solution to map tokens in the context (since it is non-parametric solution) remains the same and so they lack the ability to modify themselves (at least in-context). More specifically, the initial linear blocks, $W_{\boldsymbol{k}}$, $W_{\boldsymbol{v}}$, and $W_{\boldsymbol{q}}$, that projects input data to keys, values, and queries, are fixed after the pre-training stage (i.e., are in the first level) and so the Transformer's ability to contextualize and map tokens is bounded by the knowledge stored in these blocks. For example, given a 1-layer Transformer, the projection of each token is a function of the token itself and its position; therefore, as an example, it can miss the diverse possible encodings of words whose meaning depend on the context, rather than the word itself. Although with increasing the depth of the model this issue might fade in later layers, we should not rely on the depth to compensate the models ability as it still is a bottleneck to unleash the capability of the model in earlier layers.

To overcome the above challenge, recently, the use of short convolutions and canon layers (Allen-Zhu 2025) have became a de facto component in modern models. Despite their success in mixing local tokens, still the models are fundamentally limited to adapt to the context and capture the global information beyond the local mixing. In the next part, we discuss a fundamental solution by presenting self-referential Titans that allows all the components to perform in-context learning, and adapt and modify themselves:

8.1 Deep Self-Referential Titans

A general formulation for the associative memory-based blocks is to project the data into keys, values, and queries and learns how to map keys to values and how to retrieve from the mapping based on queries. More formally, for a parametric associative memory, let $\boldsymbol{x}_t \in \mathbb{R}^d$ for $t = 1, \ldots, L$ be the input, we have:

$$\boldsymbol{k}_t = \boldsymbol{x}_t W_{\boldsymbol{k}}, \qquad \boldsymbol{v}_t = \boldsymbol{x}_t W_{\boldsymbol{v}}, \qquad \boldsymbol{q}_t = \boldsymbol{x}_t W_{\boldsymbol{q}}, \qquad \eta_t = \boldsymbol{x}_t W_{\eta}, \qquad \alpha_t = \boldsymbol{x}_t W_{\alpha}, \tag{76}$$

$$\min_{\mathcal{M}} \; \mathcal{L}\left(\mathcal{M}; \boldsymbol{k}_t, \boldsymbol{v}_t\right), \qquad \text{with an optimization algorithm} \tag{77}$$

$$\boldsymbol{y}_t = \mathcal{M}_t \boldsymbol{q}_t \; . \tag{78}$$

For the sake of clarity, we use red (resp. blue) to highlight computations/weight in the upper level (resp. lower level). Similar to example in Figure 3, we can add a new level for each of $W_{\boldsymbol{k}}, W_{\boldsymbol{v}}, W_{\boldsymbol{q}}, W_{\eta}$, and W_{α} and allow them to be updated in-context. For the sake of efficiency, a simple version is to share the values for all the components in the nested system of associative memories:

$$\boldsymbol{k}_t = \mathcal{M}_{\boldsymbol{k},t-1}\left(\boldsymbol{x}_t\right), \qquad \boldsymbol{v}_t = \mathcal{M}_{\boldsymbol{v},t-1}\left(\boldsymbol{x}_t\right), \qquad \boldsymbol{q}_t = \mathcal{M}_{\boldsymbol{q},t-1}\left(\boldsymbol{x}_t\right), \qquad \eta_t = \mathcal{M}_{\eta,t-1}\left(\boldsymbol{x}_t\right), \qquad \alpha_t = \mathcal{M}_{\alpha,t-1}\left(\boldsymbol{x}_t\right), \tag{79}$$

$$\min_{\mathcal{M}_{\square}} \; \mathcal{L}\left(\mathcal{M}_{\square}; \square_t, \boldsymbol{v}_t\right), \qquad \text{with an optimization algorithm}, \quad \square \in \{\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}, \eta, \alpha\}, \tag{80}$$

$$\min_{\mathcal{M}_{\text{mem}}} \; \mathcal{L}\left(\mathcal{M}_{\text{mem}}; \boldsymbol{k}_t, \boldsymbol{v}_t\right), \qquad \text{with an optimization algorithm}, \tag{81}$$

$$\boldsymbol{y}_t = \mathcal{M}_{\text{mem},t}\left(\boldsymbol{q}_t\right) \; , \tag{82}$$

where the initial states of all memories, i.e., $\mathcal{M}_{\square,0}$ for any $\square \in \{\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}, \eta, \alpha, \text{memory}\}$ are meta-learned across all sequences/contexts. As discussed earlier, the meta-learning of the initial states of memories is essential for both fast-adaption, training stability, robustness to noise in the data.

This design provides a fully adaptive memory, where all the components can adapt themselves in-context. It, however, (1) still lacks self-modification, where the model in response to new data changes its own parameters or learning process (Schmidhuber 2003); (2) has suboptimal design as it shares of keys and values for all the memories. In continual learning, where the model requires consistent weight/knowledge update in response to new data, it is critical for the model to not solely rely on data, and instead learns how to modify itself when it is needed. Motivated by the above points, and inspired by the self-modifying mechanisms that generate their own values based on the context (Schmidhuber 1993, 2003; Irie et al. 2022b), we present self-modifying deep associative memory where the models generate their own values:

$$\boldsymbol{y}_t = \mathcal{M}_{\text{memory},t-1}\left(\boldsymbol{q}_t\right), \qquad \boldsymbol{k}_t = \mathcal{M}_{\boldsymbol{k},t-1}\left(\boldsymbol{x}_t\right), \qquad \boldsymbol{v}_t = \mathcal{M}_{\boldsymbol{v},t-1}\left(\boldsymbol{x}_t\right), \qquad \eta_t = \mathcal{M}_{\eta,t-1}\left(\boldsymbol{x}_t\right), \qquad \alpha_t = \mathcal{M}_{\alpha,t-1}\left(\boldsymbol{x}_t\right), \tag{83}$$

$$\hat{\boldsymbol{v}}_{\square,t} = \mathcal{M}_{\square,t-1}\left(\boldsymbol{v}_t\right), \qquad \text{(Generating its own values for each memory)} \tag{84}$$

$$\min_{\mathcal{M}_{\square}} \; \mathcal{L}\left(\mathcal{M}_{\square}; \boldsymbol{k}_t, \hat{\boldsymbol{v}}_{\square,t}\right), \qquad \text{with an optimization algorithm}, \quad \square \in \{\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}, \eta, \alpha, \text{memory}\}, \tag{85}$$

where $\boldsymbol{q}_t = \boldsymbol{x}_t W_{\boldsymbol{q}}$ is the only non-adaptive projection, η_t is the learning rate in optimization process, and α_t is the retention gate (forget gate or weight decay) in the optimization process. Note that, again, the initial states of all memories, i.e., $\mathcal{M}_{\square,0}$ for any $\square \in \{\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}, \eta, \alpha, \text{memory}\}$ are meta-learned across all sequences/contexts, and so are optimized in the higher levels (or outer-loop).

Learning the mappings for associative memory modules (see Equation 85) requires a choice of optimization algorithm as well as an objective $\mathcal{L}$ that measures the quality of mappings. A simple and common choice for objective and optimization process are L_2-regression loss, and gradient descent algorithm. As for the objective, we use L_2-regression loss, i.e., $\mathcal{L}(\mathcal{M}; \boldsymbol{k}, \boldsymbol{v}) = \|\mathcal{M}(\boldsymbol{k}) - \boldsymbol{v}\|_2^2$. As discussed earlier (see Section 4.5), the choice of optimizer highly depends on the context of optimization. For example, gradient descent from associative memory perspective is based on dot-product similarity and so the update at each step, is solely based on the input and does not incorporate the previous data samples to the update.

When performing optimization in the token space, however, we know tokens are highly correlated. Therefore, following our discussion in Section 4.5, we use our DGD with weight decay, resulting in general update rule of:

$$\boldsymbol{y}_t = \mathcal{M}_{\text{memory},t-1}\left(\boldsymbol{q}_t\right), \qquad \boldsymbol{k}_t = \mathcal{M}_{\boldsymbol{k},t-1}\left(\boldsymbol{x}_t\right), \qquad \boldsymbol{v}_t = \mathcal{M}_{\boldsymbol{v},t-1}\left(\boldsymbol{x}_t\right), \qquad \eta_t = \mathcal{M}_{\eta,t-1}\left(\boldsymbol{x}_t\right), \qquad \alpha_t = \mathcal{M}_{\alpha,t-1}\left(\boldsymbol{x}_t\right), \tag{86}$$

$$\hat{\boldsymbol{v}}_{\square,t} = \mathcal{M}_{\square,t-1}\left(\boldsymbol{v}_t\right), \qquad \text{(Generating its own values for each memory)} \tag{87}$$

$$\mathcal{M}_{\square,t} = \mathcal{M}_{\square,t-1}\left(\alpha_t \boldsymbol{I} - \eta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top\right) - \eta_t \nabla \mathcal{L}_{\mathcal{M}_{\square,t-1}}\left(\mathcal{M}_{\square,t-1}; \boldsymbol{k}_t, \hat{\boldsymbol{v}}_{\square,t}\right), \quad \square \in \{\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}, \eta, \alpha, \text{memory}\}. \tag{88}$$

Here, the architecture of the memories are arbitrary and even we are not forced to use the same architecture for all components. We use a 2-layer MLP block as the architecture of all the memories:

$$\mathcal{M}_{\square}(\cdot) = (\cdot) + W_{\square,1}\sigma(W_{\square,2}(\cdot)). \tag{89}$$

8.2 Fast and Parallelizable Training

In the above, we discussed how to design a model that can learn to generate its own latent values and so modify itself. The main challenge from the practical point of view is the efficiency of the method and if its training is parallelizable. We follow the chunk-wise training algorithm of non-linear update rules (Sun et al. 2024; Behrouz et al. 2025c) and use update frequency of $f_{\square} = \frac{L}{C_{\square}}$, where L is the context length. While there is no limitation to use different chunk-sizes, in our experiments, we use two different value of chunk sizes, one for the update of $\mathcal{M}_{\text{memory}}(\cdot)$ and the other for all the other memories in the self-referential Titans.

In more details, given an input sequence $\{\boldsymbol{x}_t\}_{t=1}^{L}$ and chunk size $1 \leq C \leq L$, we split the sequence into $\lceil \frac{L}{C} \rceil$ chunks of $\{\boldsymbol{x}_{((i-1)C+t)}\}_{t=1}^{C}$ for $i = 1, \ldots, \lceil \frac{L}{C} \rceil$, and then generate all elements in Equation 86 at the end of each chunk for the next chunk. This allows for generating all the elements for the entire chunk in parallel, before starting the computation for this chunk. Furthermore, to update the memory modules based on Equation 88, we take the gradient with respect to the last state of the previous chunk. Again, this allows for computing all the gradients for the next chunk in parallel. In more details, given this chunk-wise updating procedure, the update rule for the self-referential Titans is computed as:

$$\boldsymbol{y}_t = \mathcal{M}_{\text{memory},C\times\lceil \frac{t}{C} \rceil}\left(\boldsymbol{q}_t\right), \quad \boldsymbol{k}_t = \mathcal{M}_{\boldsymbol{k},C\times\lceil \frac{t}{C} \rceil}\left(\boldsymbol{x}_t\right), \quad \boldsymbol{v}_t = \mathcal{M}_{\boldsymbol{v},C\times\lceil \frac{t}{C} \rceil}\left(\boldsymbol{x}_t\right), \quad \eta_t = \mathcal{M}_{\eta,C\times\lceil \frac{t}{C} \rceil}\left(\boldsymbol{x}_t\right), \quad \alpha_t = \mathcal{M}_{\alpha,C\times\lceil \frac{t}{C} \rceil}\left(\boldsymbol{x}_t\right),$$

$$\hat{\boldsymbol{v}}_{\square,t} = \mathcal{M}_{\square,C\times\lceil \frac{t}{C} \rceil}\left(\boldsymbol{v}_t\right), \qquad \text{(Generating its own values for each memory)}$$

$$\mathcal{M}_{\square,t} = \mathcal{M}_{\square,t-1}\left(\alpha_t \boldsymbol{I} - \eta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top\right) - \eta_t \nabla \mathcal{L}_{\mathcal{M}_{\square,C\times\lceil \frac{t}{C} \rceil}}\left(\mathcal{M}_{\square,C\times\lceil \frac{t}{C} \rceil}; \boldsymbol{k}_t, \hat{\boldsymbol{v}}_{\square,t}\right), \qquad \square \in \{\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}, \eta, \alpha, \text{memory}\}. \tag{90}$$

Here, the architecture of the memories are arbitrary and even we are not forced to use the same architecture for all components. We use a 2-layer MLP block as the architecture of all the memories:

$$\mathcal{M}_{\square}(\cdot) = (\cdot) + W_{\square,1}\sigma(W_{\square,2}(\cdot)). \tag{91}$$

Since all the gradients as well as new keys, values, learning-rates, and weight decays can be computed in parallel before starting the processing of the current chunk, the above updates accepts the fast parallelizable dual form that is discussed by Sun et al. (2024) and Behrouz et al. (2025c). To better illustrate the above update rule for self-referential Titans, let us derive the recurrent formula for the simplest case of matrix-valued memory. We derive the recurrent form for two different objectives:

- Dot-product similarity $\mathcal{L}(\mathcal{M}; \boldsymbol{k}, \boldsymbol{v}) = -\langle \mathcal{M}\boldsymbol{k}, \boldsymbol{v} \rangle$: Given this objective and linear memory, the gradient is calculated as $\boldsymbol{v}\boldsymbol{k}^\top$, which results in update rule of:

$$\mathcal{M}_{\square,t} = \mathcal{M}_{\square,t-1}\left(\alpha_t \boldsymbol{I} - \eta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top\right) - \eta_t \hat{\boldsymbol{v}}_{\square,t} \boldsymbol{k}_t^\top, \qquad \square \in \{\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}, \eta, \alpha, \text{memory}\} \tag{92}$$

- L_2-regression loss: Given this objective and linear memory, the gradient is calculated as $(\mathcal{M}\boldsymbol{k} - \boldsymbol{v})\boldsymbol{k}^\top$, which results in update rule of:

$$\mathcal{M}_{\square,t} = \mathcal{M}_{\square,t-1}\left(\alpha_t \boldsymbol{I} - \eta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top\right) - \eta_t \left(\mathcal{M}_{\square,C\times\lceil \frac{t}{C} \rceil}\boldsymbol{k}_t - \hat{\boldsymbol{v}}_{\square,t}\right) \boldsymbol{k}_t^\top, \qquad \square \in \{\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}, \eta, \alpha, \text{memory}\}. \tag{93}$$

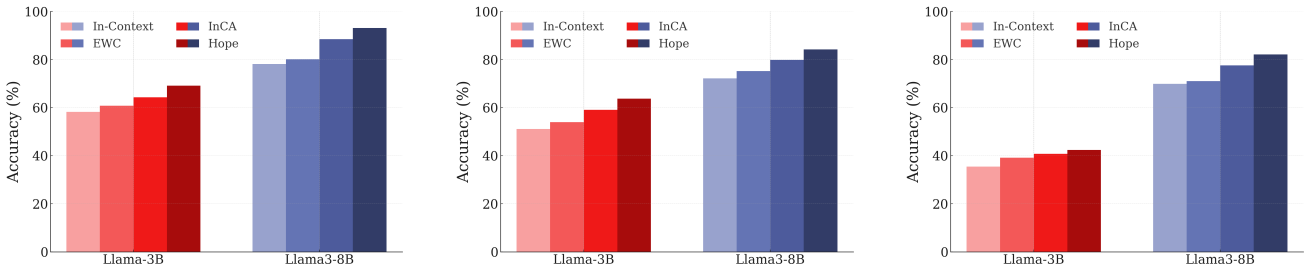

Figure 6: Class-incremental learning in the text classification domain on (Left) CLINC dataset (Larson et al. 2019), (Middle) Banking dataset (Casanueva et al. 2020), and (Right) DBpedia dataset (Auer et al. 2007). Hope-enhanced architecture achieves the best accuracy among other continual learning methods, including ICL.

8.3 Hope Neural Learning Module

In the previous sections, we first discussed Continuum Memory System (CMS) that allows for more persistent storage of memories and defines memory as a spectrum of blocks with different frequencies of update. Due to the larger capacity and constraints for scaling the parameters, often CMS requires simple learning rule but higher capacity to store more persistent knowledge. On the other hand, in the previous section, we discussed the design of a self-modifying Titans, where it can generate its own keys and so learning update to better adapt to the context. Contrary to CMS, the self-modifying Titans has a small capacity but is using a complex and expressive learning rule. Accordingly, these two systems seem to be complementary and their combination can enhance the model expressiveness from different aspects.

To this end, we present Hope architecture: A neural learning module that incorporates self-modifying Titans followed by Continuum Memory System. The Hope design is illustrated in Figure 5. Formally, let $\boldsymbol{x}_t \in \mathbb{R}^d$ for $t = 1, \ldots, L$ be the input, the Hope forward pass is defined as (we remove the normalization and convolution layers for the sake of clarity):

$$\boldsymbol{o}_t = \mathcal{M}_{\text{memory},t-1}\left(\boldsymbol{q}_t\right), \qquad \boldsymbol{k}_t = \mathcal{M}_{\boldsymbol{k},t-1}\left(\boldsymbol{x}_t\right), \qquad \boldsymbol{v}_t = \mathcal{M}_{\boldsymbol{v},t-1}\left(\boldsymbol{x}_t\right), \qquad \eta_t = \mathcal{M}_{\eta,t-1}\left(\boldsymbol{x}_t\right), \qquad \alpha_t = \mathcal{M}_{\alpha,t-1}\left(\boldsymbol{x}_t\right), \tag{94}$$

$$\hat{\boldsymbol{v}}_{\square,t} = \mathcal{M}_{\square,t-1}\left(\boldsymbol{v}_t\right), \tag{95}$$

$$\mathcal{M}_{\square,t} = \mathcal{M}_{\square,t-1}\left(\alpha_t \boldsymbol{I} - \eta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top\right) - \eta_t \nabla \mathcal{L}_{\mathcal{M}_{\square,t-1}}\left(\mathcal{M}_{\square,t-1}; \boldsymbol{k}_t, \hat{\boldsymbol{v}}_{\square,t}\right), \quad \square \in \{\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}, \eta, \alpha, \text{memory}\}. \tag{96}$$

$$\boldsymbol{y}_t = \texttt{MLP}^{(f_k)}(\texttt{MLP}^{(f_{k-1})}(\cdots \texttt{MLP}^{(f_1)}(\boldsymbol{o}_t))), \tag{97}$$

where the block's output for token t is $\boldsymbol{y}_t$. In our experiments, we also normalize $\boldsymbol{q}$ and $\boldsymbol{k}$ with L_2 normalization and also use local convolutions with window size of 4.

Hope-Attention. We also use another variant of Hope, in which we simply replace the self-modifying Titans with softmax global attention (Vaswani et al. 2017).

9 Experiments

In this section, we empirically evaluate the performance of different components we discussed throughout the paper. More specifically, (1) we first focus on the presented optimization algorithms and compare them with state-of-the-art methods; (2) We then focus on in-context and continual learning tasks and show how the nested learning paradigm and more specifically higher order in-context learning enhances the capabilities of models. We compare the continuum memory system with simple MLP layers and discuss how a pre-trained model can be adapted to be a continual learner; (3) We then focus on the language modeling and long context understanding of Hope model and compare it with Transformers and modern recurrent architectures. The details of the experiments and their setups are explained in the corresponding sections.

9.1 Hope: Continual Learning and Long Context Understanding

One of the main goals of NL is to enhance the continual learning capabilities, and so in this section, we evaluate NL and its implications such as Continuum Memory System (CMS) and Hope on multiple continual learning and long context understanding tasks. For each of the tasks, we use the best reported results on the benchmark as the baselines.

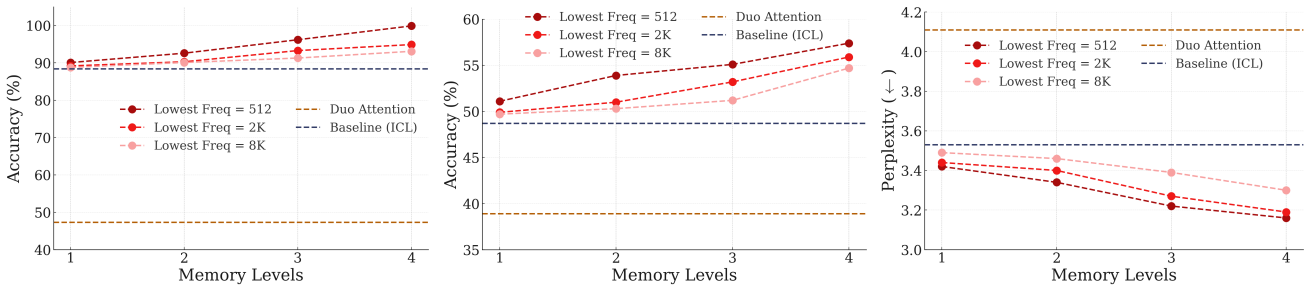

Figure 7: The effect of memory levels on the in-context learning performance of the model in (Left) MK-NIAH of RULER (Hsieh et al. 2024), (Middle) LongHealth (Adams et al. 2025), (Right) QASPER (Dasigi et al. 2021) benchmarks. Note that for QASPER benchmark (Right), the lower values indicate better performance.

Class Incremental Learning. First, we focus on class-incremental learning tasks on three datasets:

- CLINC (Larson et al. 2019): CLINC is a multi-domain intent classification benchmark designed for task-oriented dialog systems, with a special focus on detecting out-of-scope (OOS) queries. It has 150 in-scope intent classes spanning 10 broad domains (e.g. Banking, Travel, Home, Weather, Small Talk, etc.) with 23.7K total queries, of which 22.5K are in-scope and 1.2K are out-of-scope.
- Banking (Casanueva et al. 2020): Banking dataset is a single-domain intent classification benchmark focused on fine-grained customer service queries in the banking domain. In this dataset, each example is a short customer query (e.g. "How can I reset my card PIN?") that must be classified into the correct banking intent/category. There are 3083 total examples labeled with the 77 intents (heavily imbalanced classes).
- DBpedia (Auer et al. 2007): DBpedia is a text classification benchmark from Wikipedia where article abstracts are expected to be categorized into ontology topic classes. In other words, given a short Wikipedia description, the goal is to predict its high-level topic/category (such as whether the article is about a book, a film, an animal, a place, etc.). The dataset has over 340K examples labeled across those 70 second-level classes, but we sample 10K training and 1K test instances for the 70-class DBpedia task.

As the backbone of our Hope models we use Llama3-8B and Llama-3B (Dubey et al. 2024), and then employ our technique discussed in Section 7.3 to make the MLP blocks capable of adaption, placing them in different levels with different frequency updates followed by continual pre-training with 15B tokens. Following Momeni et al. (2025), we use simple in-context learning (ICL) capability of Llama-3 models (with the same process of continual pre-training with 15B tokens but without any change in the MLP blocks), Elastic Weight Consolidation (EWC) (Kirkpatrick et al. 2017), and In-context Continual Learning with an External Learner (InCA) (Momeni et al. 2025) as the baselines of our evaluation. The results are reported in Figure 6. Hope shows the best performance across all continual learning baselines, including models with external learner (i.e., InCA). Comparing Hope with ICL, the main difference comes from Hope's multiple levels of in-context learning (or equivalently, different frequency of updates for MLP blocks), indicating the effectiveness of CMS's design for enhancing continual learning capabilities. Furthermore, the superior performance of Hope, compared to InCA and EWC, indicates that the knowledge transfer between levels plays a critical role in the performance of the model.

The Effect of Levels on In-context Learning. Despite showing improvement when using CMS in the above tasks, to better understand and evaluate the effect of levels and their frequency on the in-context level ability of the model, we perform in-context question/answering and multi-key long context understanding. More specifically, we use:

- LongHealth (Adams et al. 2025): This is a benchmark for long-context clinical question answering with multiple-choice QA tasks based on extensive fictional patient records, testing an LLM's ability to extract and reason over detailed medical documents. The dataset includes 20 comprehensive patient case documents (across various diseases), each about 5.1K–6.8K words in length, and we use 200 questions sampled from patient records.
- QASPER (Dasigi et al. 2021): This benchmark is an information-seeking QA dataset centered on full-length NLP research papers. In particular, it contains around 5K QA pairs grounded in around 1.6L NLP research papers. Also, we use the full text of each paper as the context for the model.

Table 1: Needle-In-A-Haystack experiments with: (1) Single needle with three levels of difficulty: single-needle tasks—S-NIAH-1 (passkey retrieval), S-NIAH-2 (numerical needle), and S-NIAH-3 (UUID-based needle); (2) multi-query; (3) multi-key; and (4) multi-value settings of the benchmark.

	S-NIAH-1 (pass-key retrieval)			S-NIAH-2 (number in haystack)			S-NIAH-3 (uuid in haystack)		
Model	4K	8K	16K	4K	8K	16K	4K	8K	16K
Transformer	88.6	76.4	79.8	100	98.8	94.2	78.0	69.2	40.8
Hope-Attention	100	100	100	100	98.4	94.4	76.8	68.8	42.4
RWKV-7	100	100	99.6	93.8	44.8	12.6	63.8	13.2	5.8
Comba	100	100	99.4	92.6	47.2	13.4	62.4	13.8	7.4
DLA	96.4	71.2	44.0	79.6	42.6	28.2	18.2	8.8	4.0
Titans	100	100	100	99.6	84.6	75.4	74.2	42.8	21.2
Hope	100	100	100	99.2	88.4	78.2	73.2	46.2	24.8

	MK-NIAH-1 (multi-key line retrieval)			MQ-NIAH (multi-query)			MV-NIAH (multi-value)		
Model	4K	8K	16K	4K	8K	16K	4K	8K	16K
Transformer	79.4	83.0	61.4	58.9	48.0	29.8	37.5	34.1	21.5
Hope-Attention	80.2	84.8	60.8	60.4	47.8	30.6	35.2	34.4	24.8
RWKV-7	21.4	18.8	9.6	20.4	14.8	8.6	16.2	13.4	6.8
Comba	21.4	19.4	8.2	21.8	15.2	6.4	16.5	13.5	7.2
DLA	27.4	20.0	11.8	26.4	22.0	6.4	25.6	12.8	9.6
Titans	26.4	23.6	8.2	22.8	19.8	9.4	24.6	15.1	8.2
Hope	29.4	24.8	14.8	31.7	24.8	14.2	31.4	17.2	11.4

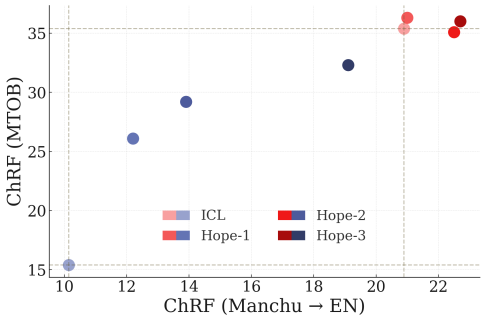

Figure 8: Continual Translation of a Novel Language (CTNL) task. Red points are the results with only one language. Blue points are the results in the continual leanring setup.

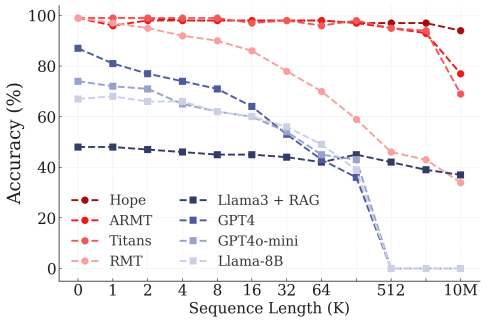

Figure 9: BABILong Benchmark. Red points are the results of fine-tuned models, and Blue points are the large models' zero-shot results.

- MK-NIAH (Hsieh et al. 2024): We use the task of multiple keys in needle-in-haystack from RULER (Hsieh et al. 2024). This setup requires models to not only locate but also extract multiple pieces of information distributed throughout a long text.

As for the baseline, we use ICL, which is the same as Hope with 1-level of memory, and also DuoAttention (Xiao et al. 2025). It is notable that methods such as Cartridges (Eyuboglu et al. 2025) have shown promising performance, even better than DuoAttention and sometimes ICL. Here, however, we exclude their comparison with Hope mainly due to the fact that Hope has higher memory usage and there are fundamental differences in their computational costs (e.g., self-studying, etc.), requiring further experiments with careful and controlled design in the future studies. For the variants of our model, we use different number of memory levels with different frequencies, which are grouped based on the lowest frequency. Note that the lowest-frequency memory corresponds to the most persistent memory of the model and so we expect models with higher lowest frequency to be more adaptive.

The results are reported in Figure 7. Hope with any number of levels and also with any lowest frequency outperforms both ICL baseline as well as the efficient DuoAttention. Furthermore, comparing the Hope's variants with themselves, the results suggest that: (1) The more levels of memory can help with in-context learning capability of the model and enhances its long-term memory and so long-context understanding; and (2) The higher the lowest-frequency the lower the performance. As discussed above, the lowest-frequency memory is the most persistent memory of the model and so increasing it means that the model has a weaker but more adaptive long-term memory. Due to the fact that the frequency of update directly affects the efficiency of the model, one might find "Lowest Frequency = 2K" an optimal setup as it offers significantly more efficient forward pass, while achieves close performance with "Lowest Frequency = 512".

Learning a New Language In-Context. Throughout the paper, we argued that pre-training can be seen as an in-context

learning process, where the context is the entire pre-training data. Accordingly, when we use more levels in the neural learning module, or performing in-context learning on different context flows, we expect the model to show better adaptability and continual learning capabilities. To this end, with combining two existing benchmarks of MTOB (Tanzer et al. 2024) and Manchu (Pei et al. 2025), we design a new continual learning task that the LLM learns two new languages in context and is expected to translate phrases to English. Then for our Continual Translation of a Novel Language (CTNL) task, we consider two setups: (1) Learning and testing the performance of the model on each language separately (in red color). This is a baseline we use to measure the catastrophic forgetting when comparing with the next setting; and (2) The model, first, learns these two languages in a sequential manner (in blue color) and then is asked to translate the phrases to English. We use ICL as the baseline, and to study the importance of multi-level design of Hope, we use different variants of Hope with 1, 2, and 3 additional levels of memory, which we refer to as Hope-1, Hope-2, and Hope-3, respectively.

The results are reported in Table 8. More specifically, each point is the performance of one model in one of the setups such that the $\boldsymbol{x}$-axis (resp. $\boldsymbol{y}$-axis) indicates the model ChRF in translation of Manchu $\rightarrow$ English (resp. Kalamang $\rightarrow$ English). In the first setup (in-context translation without continual learning), all Hope's variants performs better or on-par compared to ICL, supporting the importance of CMS design. In the second setup (i.e., continual translation), however, ICL faces dramatic performance drop and almost rely on its capabilities achieved in its pre-training (i.e., catastrophic forgetting about the knowledge in the context). On the other hand, increasing the memory levels in the Hope shows clear improvement and Hope-3 almost recovers the ICL capability in the first setup, without continual learning. These results further support the importance of CMS design in continual learning and the ability of the model to adapt itself to the new tasks.

9.2 Hope: Long Context Understanding

In the previous section, we evaluated the performance of Hope-Attention, when the MLP blocks are adapted to perform in-context learning. In this part, we evaluate the performance of Hope in long context understanding, when it starts learning from scratch. To this end, we use about 50B tokens from a mixture of FineWeb-Edu (Penedo et al. 2024) and long-context documents with a vocabulary size of 32K to train all the models from scratch. All models are optimized using AdamW with tuned learning rate for each model and with the default optimizer configuration in Behrouz et al. (2025c). We focus on two popular benchmarks of RULER (Hsieh et al. 2024) and BABILong (Kuratov et al. 2024):

Needle-in-a-Haystack (NIAH) Tasks. In the first part, we focus on the needle-in-a-haystack with different setups of: (1) single needle but different types (i.e., pass-key, number, and uuid), (2) multi-key, (3) multi-query, and (4) multi-value, all follows Hsieh et al. (2024). As for the baselines, we use RetNet (Sun et al. 2023) and DeltaNet (Schlag et al. 2021) as the representative of the models *purely* based on Hebbian- and Delta-rule, and experimented with diverse set of modern *linear* recurrent models and so used the linear models with the best performance: i.e., RWKV-7 (Peng et al. 2025a) and Comba (Hu et al. 2025). As another group of baselines, we also compare with *deep* memory modules with dot-product and L_2 regression objectives: i.e., DLA (Behrouz et al. 2025a), and Titans (Behrouz et al. 2025c).

The results are reported in Table 1. Comparing with other attention-free models, Hope achieves the best performance across all tasks and levels of difficulties. Particularly, comparing to linear memories, deep memory modules show better performance in longer sequences, mainly due to their higher memory capacity to compress more tokens. Comparing Hope with Titans, the superior performance of Hope, specifically in longer context length supports the importance of both self-referential update as well as the CMS design. Finally, we also evaluate the performance of Hope-Attention and compare it with Transformers to better understand the contribution of CMS design. The results indicate that Hope-Attention achieves a better performance compared to Transformers, supporting the advantage of having CMS in Hope-Attention design.

BABILong. Next, we evaluate the Hope's performance on BABILong benchmark (Kuratov et al. 2024) and compare it with (1) large models such as GPT4 and GPT4o-mini (Achiam et al. 2023); (2) middle-size Llama-8B model (Dubey et al. 2024) with its RAG augmented version; and (3) the state-of-the-art small models in this tasks: i.e., RMT (Bulatov et al. 2022), ARMT (Rodkin et al. 2024), and Titans (Behrouz et al. 2025c). We follow the original setup of the benchmark and fine-tune the small models with the same process as Kuratov et al. (2024).

The results are reported in Table 9. Large models show significant performance drop with increasing the sequence length, where all fail around 128K-256K context length. The RAG augmented model also show a drop with increasing the context, but it is able to relatively maintain its performance after 256K context length. Among fine-tuned models, Titans, ARMT, and Hope show competitive results until 1M context length, but the performance of both Titans and ARMT drop fast after

Table 2: Performance of models on language modeling and common-sense reasoning tasks.

Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	SIQA acc ↑	BoolQ acc ↑	Avg. ↑
760M params / 30B tokens											
Transformer++	24.18	24.27	37.1	67.2	43.8	53.0	65.6	33.4	39.1	61.7	50.11
Samba*	21.07	22.85	39.2	68.9	47.8	53.1	65.8	34.9	38.9	63.1	51.46
RetNet	25.77	24.19	34.5	66.8	41.2	51.9	63.6	32.5	38.8	56.2	48.19
DeltaNet	24.52	24.38	36.8	67.3	44.5	51.8	64.2	32.7	39.6	60.1	49.63
RWKV-7	23.75	23.08	37.1	67.3	47.6	52.2	64.7	34.2	39.4	61.9	50.55
Comba	22.41	22.19	37.5	66.9	48.2	52.4	65.1	34.1	40.1	62.8	50.89
TTT	24.17	23.51	34.7	67.3	43.9	51.0	64.5	33.8	40.2	59.6	47.32
Miras (Memora)	22.28	22.31	38.2	67.8	49.3	53.3	63.6	36.1	40.9	63.0	51.53
DLA	23.12	22.09	36.1	68.0	47.9	52.7	65.8	34.6	39.1	59.6	50.48
Titans	20.08	21.52	38.1	69.1	48.5	52.7	66.2	35.7	40.3	62.8	51.68
Hope	18.68	20.07	38.8	69.2	49.1	53.6	66.8	36.1	41.2	63.4	52.28
1.3B params / 100B tokens											
Transformer++	17.92	17.73	42.6	71.4	52.3	54.1	69.9	36.5	41.8	58.4	53.38
Samba*	16.15	13.21	45.2	71.5	53.8	55.8	69.1	36.7	40.6	63.0	54.46
RetNet	18.91	17.04	41.2	71.3	49.1	55.2	67.5	34.1	41.4	61.0	52.60
DeltaNet	18.62	17.10	41.6	70.1	49.4	52.7	67.6	35.2	39.7	54.8	51.39
RWKV-7	18.44	15.96	46.7	72.4	54.9	57.5	71.6	38.2	40.7	60.4	55.30
Comba	18.16	14.87	46.9	73.1	54.5	57.7	72.0	39.1	40.2	60.6	55.39
TTT	18.42	14.51	46.8	72.9	55.2	59.0	71.8	39.5	39.8	59.6	55.58
Miras (Memora)	15.90	12.04	48.7	73.1	56.0	57.4	71.5	37.9	40.2	61.3	55.76
DLA	16.31	12.29	44.5	70.6	53.9	54.2	69.6	36.0	40.8	60.2	53.72
Titans	15.60	11.41	49.1	73.1	56.3	59.8	72.4	40.8	42.1	61.0	56.82
Hope	14.39	10.08	51.0	73.9	57.5	61.2	73.8	42.7	42.8	61.4	58.04

* is a hybrid of attention + linear RNN (Ren et al. 2024).

that point. Hope maintains its good performance even for 10M context length, mainly due to its CMS design. It is notable that the performance of all small models, including Hope, can drop significantly when used without fine-tuning. The reason is, compressing large context (e.g., 10M), in addition to a powerful memory management in the high-frequency level, requires enough capacity to compress 10M tokens or at least tokens that are needed for the final answer. The fine-tuning step helps the models to adjust their lower-frequency levels to adapt fast and so properly manage their memory in the higher-frequency levels.

9.3 Hope: Language Modeling and Common-Sense Reasoning

In this section, we aim to study Hope as a backbone of a language model and evaluate it on common language modeling and common-sense reasoning tasks with the setup of:

- Datasets: We evaluate Hope and baselines on Wikitext (Merity et al. 2017), LMB (Paperno et al. 2016), PIQA (Bisk et al. 2020), HellaSwag (Zellers et al. 2019), WinoGrande (Sakaguchi et al. 2021), ARC-easy (ARC-e) and ARC-challenge (ARC-c) (Clark et al. 2018), SIQA (Sap et al. 2019), and BoolQ (Clark et al. 2019) benchmarks.
- Baselines: As for the baselines, similar to Section 9.2, we use RetNet (Sun et al. 2023) and DeltaNet (Schlag et al. 2021) as the representatives of the models that are *purely* based on Hebbian- or Delta-rule, and two modern *matrix-valued* recurrent models with the best performance compared to others: i.e., RWKV-7 (Peng et al. 2025a) and Comba (Hu et al. 2025). As another group of baselines, we compare with attention-free *deep* memory modules with diverse internal attentional bias of dot-product, L_2, and L_p regression: i.e., TTT (Sun et al. 2024), Miras (Behrouz et al. 2025b), DLA (Behrouz et al. 2025a) and Titans (Behrouz et al. 2025c). Finally, we also compare with Transformers (Vaswani et al. 2017; Dubey et al. 2024) as well as the hybrid of attention and linear RNN, Samba (Ren et al. 2024).
- Training: We train models with about 760M and 1.3B parameters, trained with 30B and 100B tokens, respectively, from a mixture of FineWeb-Edu (Penedo et al. 2024) and long-context documents with a vocabulary size of 32K to

Table 5: Accuracies of various models on the formal language recognition tasks.

		Non-Star-Free Regular						Counter					
	Parallel	Parity		$(aa)^*$		$(abab)^*$		$a^n b^n$		$a^n b^n c^n$		Shuffle-2	
Model	Training	Bin0	Bin1	Bin0	Bin1	Bin0	Bin1	Bin0	Bin1	Bin0	Bin1	Bin0	Bin1
LSTM	✗	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Transformer	✓	46.4	0.0	0.0	0.0	0.0	0.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Linear	✓	78.1	0.0	0.0	0.0	0.0	0.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
DeltaNet	✓	98.2	10.1	0.0	0.0	0.0	0.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
SRWM	✗	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Hope	✓	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**

train all the models from scratch. All models are optimized using AdamW with tuned learning rate for each model and with the default optimizer configuration in Behrouz et al. (2025c).

The results are reported in Table 2. Hope outperforms all the baselines on the average performance in both language modeling and common-sense reasoning benchmarks. Interestingly, with scaling the parameters, Hope show higher performance gain compare to other attention-free models.

9.4 Hope: In-context Recall Tasks and MAD Synthetic Benchmark

In-context recall is often referred to as one of the challenging benchmarks for attention-free models. In this section, we follow Arora et al. (2024) and perform experiments on SWDE (Lockard et al. 2019), NQ (Kwiatkowski et al. 2019), DROP (Dua et al. 2019), FDA (Arora et al. 2023), SQUAD (Rajpurkar et al. 2016), and TQA (Kembhavi et al. 2017) to evaluate the effectiveness of Hope's design. We use the same set of baselines and experimental setup as the above previous section. The results are reported in Table 3. Transformers achieve the best performance, while Hope show competitive results, outperforming all the attention-free baselines and closing the gap with Transformers.

Table 3: The performance of Hope and baselines in short in-context recall tasks. Hope outperforms all attention-free models and close the gap with Transformers.

	SWDE	NQ	DROP	FDA	SQUAD	TQA
Transformers	71.4	22.0	23.9	67.3	39.4	59.1
RWKV-7	52.3	17.8	21.7	32.8	28.5	56.2
Comba	53.9	19.1	21.9	35.5	30.2	56.4
Titans	60.8	20.3	22.0	37.6	31.8	57.5
Hope	65.9	21.2	22.8	41.9	33.0	57.7

Table 4: Performance of Hope and baselines on the synthetic benchmark of MAD (Poli et al. 2024). Hope outperforms all the baselines, including Transformers.

	Compress.	ICR	Fuzzy ICR	Selective Copying	Memory
Transformers	49.4	100	47.9	96.2	83.7
RWKV-7	45.1	100	32.8	95.6	82.2
Comba	46.3	100	32.8	96.4	82.9
Titans	49.8	100	50.0	99.4	83.4
Hope	51.2	100	52.1	99.7	85.2

We also study the performance of Hope on MAD benchmark (Poli et al. 2024), which is a synthetic benchmark, evaluating the performance of models in recall, memorization, compression, and copying tasks. The results are reported in Table 4. Hope achieves the best results compared to baselines.

9.5 Language Recognition Tasks

One of the critical limitations of Transformers is on non-parallelizable tasks, where the recurrence plays a significant rule in achieving proper performance. An example of such tasks is state tracking problem, where the model needs to track its state given a sequence of instructions (movement, etc.), and several studies have shown that Transformers both in theory and practice significantly underperforms models with non-linear recurrence (Merrill et al. 2024; Grazzi et al. 2025). In this section, we focus on formal language recognition tasks and follow the construction of the benchmark by Irie et al. (2023). The results are reported in Table 5. Hope achieves the perfect score on all the tasks, similar to other non-linear recurrent models; e.g., LSTM Schmidhuber et al. 1997 and SRWM (Irie et al. 2022b). The main advantage of Hope, however, is the fact

Table 6: Ablation Study on Hope. All components of Hope are positively contributing to its performance.

Model	Language Modeling ppl ↓	Reasoning acc ↑
Hope	12.24	58.1
w/o DGD	13.41	56.5
w/o Momentum	13.58	56.9
w/o weight decay	13.71	57.2
w/o CMS	13.04	57.3
w/o inner-projection $\boldsymbol{k}$	13.77	56.9
w/o inner-projection $\boldsymbol{v}$	13.90	55.1
w/o inner-projection $\boldsymbol{q}$	12.19	57.4

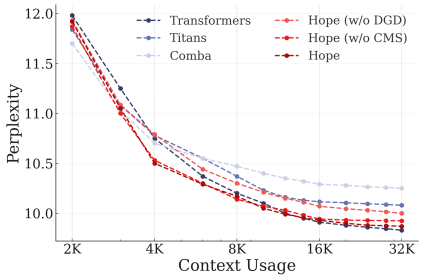

Figure 10: The effect of context usage on the perplexity of the model. We expect the perplexity of a model with powerful memory management to decrease with more context.

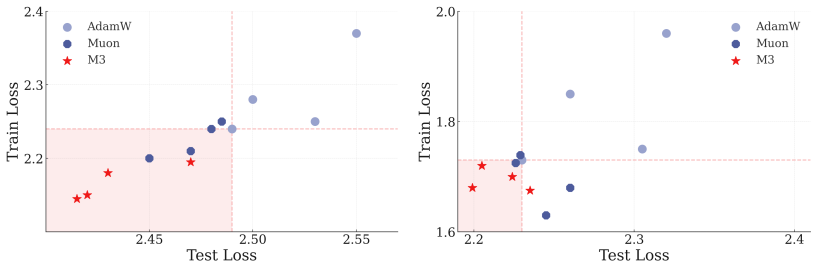

Figure 11: ViT test and train loss on ImageNet-21K (Ridnik et al. 2021), trained with AdamW, Muon, and our M3 optimizers.

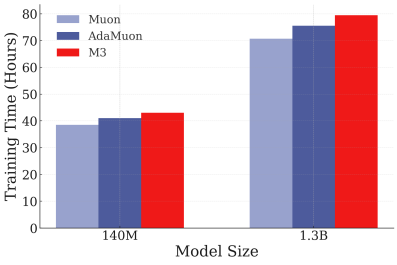

Figure 12: Training time of models with 140M and 1.3B parameters with Muon, AdaMuon, and M3 optimizers.

that when it is needed, it has parallelizable training and so can scale to larger scales for language modeling tasks.

9.6 Hope: Ablation Studies and Scaling

In this section, we first study the importance of design choices we have made for the Hope architecture by performing ablation studies, where we remove or change one of the Hope's components at a time. Note that, in the previous experiments, we have evaluated the significance of some components: E.g., Table 8 and Figure 7 have already been shown the effect of number of levels as well as the frequency of update on the performance of Hope in continual learning. In this section, to compare different variants, we use the average perplexity of models in language modeling as well as the average accuracy in common-sense reasoning tasks. The results are reported in Table 6. (1) The first row, replaces Delta Gradient Descent with a simple gradient descent in the design of self-modifying Titans; (2) The second row removes the momentum term in the self-modifying Titans; (3) removes the weight decay; (4) removes the CMS from Hope's architecture; (5, 6, 7) transfer the projections for $\boldsymbol{k}, \boldsymbol{v}, \boldsymbol{q}$ from higher-frequency level to the lowest-frequency level. All the components of Hope contribute to its superior performance in these tasks and removing or changing each of them can damage the model's perplexity in language modeling and/or accuracy in common-sense reasoning tasks.

9.7 Expressive Optimizers

In this section, we evaluate the performance of our M3 optimizer from both aspects of effectiveness in finding the solution and also efficiency of training in large scales:

ImageNet. In this experiment, we focus on ViT (Dosovitskiy et al. 2021) architecture for vision tasks and pre-train it on ImageNet-21K, which consists of 11M images corresponding to 10,450 classes. We use a patch size of 16, MLP dimension of 1536 and 3072 for the two scales of 24M and 86M models, respectively. We control all other components and finetuned hyperparameters for each optimizer separately to make sure a fair comparison. We vary the optimizer for training the ViT to understand which of the optimizers find more effective solutions in the same number of training steps. The training and

test loss of trained models with different optimizers are reported in Figure 11. Our M3 shows the best training/test loss compared to both AdamW and Muon.

Efficiency in Large Models. Due to the small size of the model that is needed for ImageNet, we also perform an efficiency comparison when training a language model. To this end, we use two scales of 140M and 1.3B and train the same Transformer model using different optimizers of Muon (Jordan et al. 2024), AdaMuon (Si et al. 2025), and our M3. The results are reported in Figure 12. Our M3 optimizer, due to the use of multiple momentums (memories) is reletively slower compare to the Muon optimizer, and show on par efficiency with AdaMuon.

10 Conclusion

In this paper, we introduced *Nested Learning (NL)* a new learning paradigm, in which modern machine learning systems are modeled as inter-connected, multi-level optimization problems, each with its own context flow and update frequency. Within this view, both architectures and optimizers are instances of nested systems of associative memories that compress their own context (either is tokens, gradients, or higher-level signals) into internal parameters. This perspective reframes pre-training, in-context learning, and continual learning as manifestations of the same underlying mechanism: learning to compress and reuse context at different levels and time scales. This perspective recasts backpropagation, momentum, and preconditioning as associative memory mechanisms, and explains a wide family of existing methods as specific design points in a larger (previously hidden) space.

Building on NL's viewpoint, we derived generalized gradient-based updates: e.g., Delta Gradient Descent, Delta Momentum, Multi-scale Momentum Muon (M3), and reinterpreted modern sequence architectures nested associative memories. To enhance the memory processing, we introduced Continuum Memory System (CMS), a new formulation for memory that generalizes the traditional viewpoint of "long-term/short-term memory blocks". Our Hope architecture based on a self-modifying Titans and CMS improves continual learning and long-context reasoning capabilitites, while remaining competitive as a general backbone.

Is Catastrophic Forgetting Solved? While Hope and CMS have shown promising results in reducing catastrophic forgetting in the tasks we empirically studied, the undesirable phenomenon of catastrophic forgetting is not "solved" in general. From nested learning viewpoint on learning and backpropagation process, catastrophic forgetting is a natural consequence of compression, where the limited capacity of the network forces the model to forget so that it retains capacity for new information. We view NL as a roadmap rather than a destination: it suggests that progress on continual learning, long-context reasoning, modern optimizers, and self-modifying models will come from better exploiting the extra design axis of *levels* rather than from ever-deeper static networks.

References

[1] Walter Pitts. "The linear theory of neuron networks: The dynamic problem". In: *The bulletin of mathematical biophysics* 5 (1943), pp. 23–31.

[2] Warren S McCulloch and Walter Pitts. "The statistical organization of nervous activity". In: *Biometrics* 4.2 (1948), pp. 91–99.

[3] Warren S McCulloch. "The brain computing machine". In: *Electrical Engineering* 68.6 (1949), pp. 492–497.

[4] William Beecher Scoville and Brenda Milner. "Loss of recent memory after bilateral hippocampal lesions". In: *Journal of neurology, neurosurgery, and psychiatry* 20.1 (1957), p. 11.

[5] Arthur L Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of research and development* 3.3 (1959), pp. 210–229.

[6] Seppo Linnainmaa. "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". PhD thesis. Master's Thesis (in Finnish), Univ. Helsinki, 1970.

[7] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[8] Erkki Oja. "Simplified neuron model as a principal component analyzer". In: *Journal of mathematical biology* 15.3 (1982), pp. 267–273.

[9] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[10] Geoffrey E Hinton and David C Plaut. "Using fast weights to deblur old memories". In: *Proceedings of the ninth annual conference of the Cognitive Science Society*. 1987, pp. 177–186.

[11] DL Prados and SC Kak. "Neural network capacity using delta rule". In: *Electronics Letters* 25.3 (1989), pp. 197–199.

[12] Juergen Schmidhuber. "Learning to control fast-weight memories: An alternative to recurrent nets. Accepted for publication in". In: *Neural Computation* (1992).

[13] Tim VP Bliss and Graham L Collingridge. "A synaptic model of memory: long-term potentiation in the hippocampus". In: *Nature* 361.6407 (1993), pp. 31–39.

[14] Jürgen Schmidhuber. "A 'self-referential'weight matrix". In: *International conference on artificial neural networks*. Springer. 1993, pp. 446–450.

[15] Juergen Schmidhuber, Jieyu Zhao, and Marco Wiering. *Simple principles of metalearning*. 1996.

[16] Uwe Frey and Richard GM Morris. "Synaptic tagging and long-term potentiation". In: *Nature* 385.6616 (1997), pp. 533–536.

[17] Juergen Schmidhuber and Sepp Hochreiter. "Long Short-term Memory". In: *Neural Computation MIT-Press* (1997).

[18] Amos Storkey. "Increasing the capacity of a hopfield network without sacrificing functionality". In: *International Conference on Artificial Neural Networks*. Springer. 1997, pp. 451–456.

[19] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. Vol. 1. 1. 1998.

[20] Jonathan H. Connell and Sridhar Mahadevan. "Robot Learning". In: *Robotica* 17.2 (1999), pp. 229–235. DOI: 10.1017/S0263574799271172.

[21] Sean PA Drummond, Gregory G Brown, J Christian Gillin, John L Stricker, Eric C Wong, and Richard B Buxton. "Altered brain response to verbal learning following sleep deprivation". In: *Nature* 403.6770 (2000), pp. 655–657.

[22] Hideyuki Okano, Tomoo Hirano, and Evan Balaban. "Learning and memory". In: *Proceedings of the National Academy of Sciences* 97.23 (2000), pp. 12403–12404.

[23] Jürgen Schmidhuber. "Gödel machines: self-referential universal problem solvers making provably optimal self-improvements". In: *arXiv preprint cs/0309048* (2003).

[24] Gyorgy Buzsaki and Andreas Draguhn. "Neuronal oscillations in cortical networks". In: *science* 304.5679 (2004), pp. 1926–1929.

[25] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

[26] Alvaro Pascual-Leone, Amir Amedi, Felipe Fregni, and Lotfi B Merabet. "The plastic human brain cortex". In: *Annu. Rev. Neurosci.* 28.1 (2005), pp. 377–401.

[27] David J Foster and Matthew A Wilson. "Reverse replay of behavioural sequences in hippocampal place cells during the awake state". In: *Nature* 440.7084 (2006), pp. 680–683.

[28] Lisa Marshall, Halla Helgadóttir, Matthias Mölle, and Jan Born. "Boosting slow oscillations during sleep potentiates memory". In: *Nature* 444.7119 (2006), pp. 610–613. DOI: 10.1038/nature05278.

[29] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. "Dbpedia: A nucleus for a web of open data". In: *international semantic web conference*. Springer. 2007, pp. 722–735.

[30] Timothy J. Buschman and Earl K. Miller. "Top-down versus bottom-up control of attention in the prefrontal and posterior parietal cortices". In: *Science* 315.5820 (2007), pp. 1860–1862. DOI: 10.1126/science.1138071.
[31] Daoyun Ji and Matthew A Wilson. "Coordinated memory replay in the visual cortex and hippocampus during sleep". In: *Nature neuroscience* 10.1 (2007), pp. 100–107.
[32] Seung-Schik Yoo, Peter T Hu, Ninad Gujar, Ferenc A Jolesz, and Matthew P Walker. "A deficit in the ability to form new human memories without sleep". In: *Nature neuroscience* 10.3 (2007), pp. 385–392.
[33] Nicholas J Higham. *Functions of matrices: theory and computation*. SIAM, 2008.
[34] Michael V Johnston. "Plasticity in the developing brain: implications for rehabilitation". In: *Developmental disabilities research reviews* 15.2 (2009), pp. 94–101.
[35] Adrien Peyrache, Mehdi Khamassi, Karim Benchenane, Sidney I Wiener, and Francesco P Battaglia. "Replay of rule-learning related neural patterns in the prefrontal cortex during sleep". In: *Nature neuroscience* 12.7 (2009), pp. 919–926.
[36] Susanne Diekelmann and Jan Born. "The memory function of sleep". In: *Nature Reviews Neuroscience* 11.2 (2010), pp. 114–126. DOI: 10.1038/nrn2762.
[37] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).
[38] Juergen Fell and Nikolai Axmacher. "The role of phase synchronization in memory processes". In: *Nature reviews neuroscience* 12.2 (2011), pp. 105–118.
[39] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *Cited on* 14.8 (2012), p. 2.
[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).
[41] Hong-Viet V. Ngo, Thomas Martinetz, Jan Born, and Matthias Mölle. "Auditory closed-loop stimulation of the sleep slow oscillation enhances memory". In: *Neuron* 78.3 (2013), pp. 545–553. DOI: 10.1016/j.neuron.2013.03.006.
[42] Dzmitry Bahdanau. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).
[43] James F Cavanagh and Michael J Frank. "Frontal theta as a mechanism for cognitive control". In: *Trends in cognitive sciences* 18.8 (2014), pp. 414–421.
[44] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).
[45] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes." In: *ICLR*. Ed. by Yoshua Bengio and Yann LeCun. 2014. URL: http://dblp.uni-trier.de/db/conf/iclr/iclr2014.html#KingmaW13.
[46] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. "On the number of linear regions of deep neural networks". In: *Advances in neural information processing systems* 27 (2014).
[47] Pascal Fries. "Rhythms for cognition: communication through coherence". In: *Neuron* 88.1 (2015), pp. 220–235.
[48] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.
[49] Bernhard P. Staresina, Til Ole Bergmann, Mathilde Bonnefond, Roemer van der Meij, Ole Jensen, Lorena Deuker, Christian E. Elger, Nikolai Axmacher, and Jürgen Fell. "Hierarchical nesting of slow oscillations, spindles and ripples in the human hippocampus during sleep". In: *Nature Neuroscience* 18.11 (2015), pp. 1679–1686. DOI: 10.1038/nn.4119.
[50] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. "Using fast weights to attend to the recent past". In: *Advances in neural information processing systems* 29 (2016).
[51] Timothy Dozat. "Incorporating nesterov momentum into adam". In: (2016).
[52] Andrew C. Heusser, David Poeppel, Youssef Ezzyat, and Lila Davachi. "Episodic sequence memory is supported by a theta–gamma phase code". In: *Nature Neuroscience* 19.10 (2016), pp. 1374–1380. DOI: 10.1038/nn.4374.
[53] Mikael Lundqvist, Pawel Herman, Scott L. Brincat, Timothy J. Buschman, and Earl K. Miller. "Gamma and beta bursts underlie working memory". In: *Neuron* 90.1 (2016), pp. 152–164. DOI: 10.1016/j.neuron.2016.02.028.
[54] Denis Paperno, German Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernandez. "The LAMBADA dataset: Word prediction requiring a broad discourse context". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Katrin Erk and Noah A. Smith. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1525–1534. DOI: 10.18653/v1/P16-1144. URL: https://aclanthology.org/P16-1144/.
[55] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. "Exponential expressivity in deep neural networks through transient chaos". In: *Advances in neural information processing systems* 29 (2016).

[56] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. "Squad: 100,000+ questions for machine comprehension of text". In: *arXiv preprint arXiv:1606.05250* (2016).

[57] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alex Smola. "Stochastic variance reduction for nonconvex optimization". In: *International conference on machine learning*. PMLR. 2016, pp. 314–323.

[58] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[59] Thomas B Christophel, P Christiaan Klink, Bernhard Spitzer, Pieter R Roelfsema, and John-Dylan Haynes. "The distributed nature of working memory". In: *Trends in cognitive sciences* 21.2 (2017), pp. 111–124.

[60] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *International conference on machine learning*. PMLR. 2017, pp. 1126–1135.

[61] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. "Deep learning scaling is predictable, empirically". In: *arXiv preprint arXiv:1712.00409* (2017).

[62] Aniruddha Kembhavi, Minjoon Seo, Dustin Schwenk, Jonghyun Choi, Ali Farhadi, and Hannaneh Hajishirzi. "Are you smarter than a sixth grader? textbook question answering for multimodal machine comprehension". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern recognition*. 2017, pp. 4999–5007.

[63] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[64] Takashi Kitamura, Sachie K Ogawa, Dheeraj S Roy, Teruhiro Okuyama, Mark D Morrissey, Lillian M Smith, Roger L Redondo, and Susumu Tonegawa. "Engrams and circuits crucial for systems consolidation of a memory". In: *Science* 356.6333 (2017), pp. 73–78.

[65] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. "Pointer Sentinel Mixture Models". In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=Byj72udxe.

[66] W Scott Terry. *Learning and memory: Basic principles, processes, and procedures*. Routledge, 2017.

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[68] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. "signSGD: Compressed optimisation for non-convex problems". In: *International conference on machine learning*. PMLR. 2018, pp. 560–569.

[69] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. "Think you have solved question answering? try arc, the ai2 reasoning challenge". In: *arXiv preprint arXiv:1803.05457* (2018).

[70] Jianqing Fan. *Local polynomial modelling and its applications: monographs on statistics and applied probability 66*. Routledge, 2018.

[71] Vineet Gupta, Tomer Koren, and Yoram Singer. "Shampoo: Preconditioned stochastic tensor optimization". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1842–1850.

[72] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[73] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. "BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 2924–2936. DOI: 10.18653/v1/N19-1300. URL: https://aclanthology.org/N19-1300/.

[74] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. "DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs". In: *arXiv preprint arXiv:1903.00161* (2019).

[75] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. "Learning deep representations by mutual information estimation and maximization". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=Bklr3j0cKX.

[76] Jens G Klinzing, Niels Niethard, and Jan Born. "Mechanisms of systems memory consolidation during sleep". In: *Nature neuroscience* 22.10 (2019), pp. 1598–1610.

[77] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. "Natural questions: a benchmark for question answering research". In: *Transactions of the Association for Computational Linguistics* 7 (2019), pp. 453–466.

[78] Stefan Larson, Anish Mahendran, Joseph J Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K Kummerfeld, Kevin Leach, Michael A Laurenzano, Lingjia Tang, et al. "An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 1311–1316.

[79] Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. "Openceres: When open information extraction meets the semi-structured web". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019, pp. 3047–3056.

[80] Tsendsuren Munkhdalai, Alessandro Sordoni, Tong Wang, and Adam Trischler. "Metalearned neural memory". In: *Advances in Neural Information Processing Systems* 32 (2019).

[81] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. "Social IQa: Commonsense Reasoning about Social Interactions". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 4463–4473. DOI: 10.18653/v1/D19-1454. URL: https://aclanthology.org/D19-1454/.

[82] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. "HellaSwag: Can a Machine Really Finish Your Sentence?" In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluis Marquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 4791–4800. DOI: 10.18653/v1/P19-1472. URL: https://aclanthology.org/P19-1472/.

[83] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. "Piqa: Reasoning about physical commonsense in natural language". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 2020, pp. 7432–7439.

[84] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[85] Inigo Casanueva, Tadas Temcinas, Daniela Gerz, Matthew Henderson, and Ivan Vulic. "Efficient Intent Detection with Dual Sentence Encoders". In: *ACL 2020* (2020), p. 38.

[86] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. "Orthogonal gradient descent for continual learning". In: *International conference on artificial intelligence and statistics*. PMLR. 2020, pp. 3762–3773.

[87] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[88] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. "Scaling laws for neural language models". In: *arXiv preprint arXiv:2001.08361* (2020).

[89] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. "Transformers are rnns: Fast autoregressive transformers with linear attention". In: *International conference on machine learning*. PMLR. 2020, pp. 5156–5165.

[90] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. "On the Variance of the Adaptive Learning Rate and Beyond". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=rkgz2aEKDr.

[91] Noam Shazeer. "Glu variants improve transformer". In: *arXiv preprint arXiv:2002.05202* (2020).

[92] Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. "A dataset of information-seeking questions and answers anchored in research papers". In: *arXiv preprint arXiv:2105.03011* (2021).

[93] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[94] Akihiro Goto, Ayaka Bota, Ken Miya, Jingbo Wang, Suzune Tsukamoto, Xinzhi Jiang, Daichi Hirai, Masanori Murayama, Tomoki Matsuda, Thomas J. McHugh, Takeharu Nagai, and Yasunori Hayashi. "Stepwise synaptic plasticity events drive the early phase of memory consolidation". In: *Science* 374.6569 (2021), pp. 857–863. DOI:

10.1126/science.abj9195. eprint: https://www.science.org/doi/pdf/10.1126/science.abj9195. URL: https://www.science.org/doi/abs/10.1126/science.abj9195.

[95] Kazuki Irie, Imanol Schlag, Robert Csordas, and Juergen Schmidhuber. "Going beyond linear transformers with recurrent fast weight programmers". In: *Advances in neural information processing systems* 34 (2021), pp. 7703–7717.

[96] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. "Highly accurate protein structure prediction with AlphaFold". In: *nature* 596.7873 (2021), pp. 583–589.

[97] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. "ImageNet-21K Pretraining for the Masses". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*. 2021. URL: https://openreview.net/forum?id=Zkj_VcZ6ol.

[98] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. "Winogrande: An adversarial winograd schema challenge at scale". In: *Communications of the ACM* 64.9 (2021), pp. 99–106.

[99] Imanol Schlag, Kazuki Irie, and Juergen Schmidhuber. "Linear transformers are secretly fast weight programmers". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 9355–9366.

[100] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. "What learning algorithm is in-context learning? investigations with linear models". In: *arXiv preprint arXiv:2211.15661* (2022).

[101] Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. "Recurrent memory transformer". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 11079–11091.

[102] Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. "Meta-learning via Language Model In-context Tuning". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022, pp. 719–730.

[103] Alexandre Défossez, Leon Bottou, Francis Bach, and Nicolas Usunier. "A Simple Convergence Proof of Adam and Adagrad". In: *Transactions on Machine Learning Research* (2022). ISSN: 2835-8856. URL: https://openreview.net/forum?id=ZPQhzTSWA7.

[104] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. "Training compute-optimal large language models". In: *arXiv preprint arXiv:2203.15556* (2022).

[105] Kazuki Irie, Francesco Faccio, and Jürgen Schmidhuber. "Neural differential equations for learning to program neural nets through continuous learning rules". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 38614–38628.

[106] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Juergen Schmidhuber. "A modern self-referential weight matrix that learns to modify itself". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 9660–9677.

[107] William Merrill, Ashish Sabharwal, and Noah A Smith. "Saturated transformers are constant-depth threshold circuits". In: *Transactions of the Association for Computational Linguistics* 10 (2022), pp. 843–856.

[108] Dheeraj S Roy, Young-Gyun Park, Minyoung E Kim, Ying Zhang, Sachie K Ogawa, Nicholas DiNapoli, Xinyi Gu, Jae H Cho, Heejin Choi, Lee Kamentsky, et al. "Brain-wide mapping reveals that engrams for a single memory are distributed across multiple brain regions". In: *Nature communications* 13.1 (2022), p. 1799.

[109] Yufeng Zhang, Boyi Liu, Qi Cai, Lingxiao Wang, and Zhaoran Wang. "An analysis of attention via the lens of exchangeability and latent variable models". In: *arXiv preprint arXiv:2212.14852* (2022).

[110] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. "Gpt-4 technical report". In: *arXiv preprint arXiv:2303.08774* (2023).

[111] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. "Language models enable simple systems for generating structured views of heterogeneous data lakes". In: *arXiv preprint arXiv:2304.09433* (2023).

[112] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. "Symbolic discovery of optimization algorithms". In: *Advances in neural information processing systems* 36 (2023), pp. 49205–49233.

[113] Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. "Liquid Structural State-Space Models". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=g4OTKRKfS7R.

[114] Kazuki Irie, Robert Csordas, and Juergen Schmidhuber. "Practical computational power of linear transformers and their recurrent and self-referential extensions". In: *arXiv preprint arXiv:2310.16076* (2023).

[115] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. "CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=iaYcJKpY2B_.

[116] Bo Peng, Eric Alcaide, Quentin Gregory Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Nguyen Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Jiaju Lin, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan S. Wind, Stanisław Wozniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. "RWKV: Reinventing RNNs for the Transformer Era". In: *The 2023 Conference on Empirical Methods in Natural Language Processing*. 2023. URL: https://openreview.net/forum?id=7SaXczaBpG.

[117] Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. "Hyena hierarchy: Towards larger convolutional language models". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 28043–28078.

[118] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. "Are emergent abilities of large language models a mirage?" In: *Advances in neural information processing systems* 36 (2023), pp. 55565–55581.

[119] Aaditya Singh, Stephanie Chan, Ted Moskovitz, Erin Grant, Andrew Saxe, and Felix Hill. "The transient nature of emergent in-context learning in transformers". In: *Advances in neural information processing systems* 36 (2023), pp. 27801–27819.

[120] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. "Retentive network: A successor to transformer for large language models". In: *arXiv preprint arXiv:2307.08621* (2023).

[121] Johannes Von Oswald, Maximilian Schlegel, Alexander Meulemans, Seijin Kobayashi, Eyvind Niklasson, Nicolas Zucchet, Nino Scherrer, Nolan Miller, Mark Sandler, Max Vladymyrov, et al. "Uncovering mesa-optimization algorithms in transformers". In: *arXiv preprint arXiv:2309.05858* (2023).

[122] Wenhai Wang, Zhe Chen, Xiaokang Chen, Jiannan Wu, Xizhou Zhu, Gang Zeng, Ping Luo, Tong Lu, Jie Zhou, Yu Qiao, et al. "Visionllm: Large language model is also an open-ended decoder for vision-centric tasks". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 61501–61513.

[123] Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. "The Surprising Effectiveness of Test-Time Training for Few-Shot Learning". In: *Forty-second International Conference on Machine Learning*. 2024.

[124] Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. "In-context language learning: Architectures and algorithms". In: *arXiv preprint arXiv:2401.12973* (2024).

[125] Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, James Zou, Atri Rudra, and Christopher Re. "Simple linear attention language models balance the recall-throughput tradeoff". In: *Forty-first International Conference on Machine Learning*. 2024. URL: https://openreview.net/forum?id=e93ffDcpH3.

[126] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. "xLSTM: Extended Long Short-Term Memory". In: *arXiv preprint arXiv:2405.04517* (2024).

[127] Aleksandar Botev, Soham De, Samuel L Smith, Anushan Fernando, George-Cristian Muraru, Ruba Haroun, Leonard Berrada, Razvan Pascanu, Pier Giuseppe Sessa, Robert Dadashi, et al. "RecurrentGemma: Moving Past Transformers for Efficient Open Language Models". In: *arXiv preprint arXiv:2404.07839* (2024).

[128] Jonathan Daume, Jan Kamiński, Andrea G. P. Schjetnan, Yousef Salimpour, Umais Khan, Michael Kyzar, Chrystal M. Reed, William S. Anderson, Taufik A. Valiante, Adam N. Mamelak, and Ueli Rutishauser. "Control of working memory by phase–amplitude coupling of human hippocampal neurons". In: *Nature* 629 (2024), pp. 393–401. DOI: 10.1038/s41586-024-07309-z.

[129] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. "The llama 3 herd of models". In: *arXiv e-prints* (2024), arXiv–2407.

[130] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. "RULER: What's the Real Context Size of Your Long-Context Language Models?" In: *First Conference on Language Modeling*. 2024. URL: https://openreview.net/forum?id=kIoBbc76Sy.

[131] K Jordan, Y Jin, V Boza, Y Jiacheng, F Cecista, L Newhouse, and J Bernstein. "Muon: An optimizer for hidden layers in neural networks, 2024b". In: *URL https://kellerjordan. github. io/posts/muon* (2024).

[132] Praneeth Kacham, Vahab Mirrokni, and Peilin Zhong. "PolySketchFormer: Fast Transformers via Sketching Polynomial Kernels". In: *Proceedings of the 41st International Conference on Machine Learning*. Ed. by Ruslan Salakhutdinov,

Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp. Vol. 235. Proceedings of Machine Learning Research. PMLR, 21–27 Jul 2024, pp. 22748–22770. URL: https://proceedings.mlr.press/v235/kacham24a.html.

[133] Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Igorevich Sorokin, Artyom Sorokin, and Mikhail Burtsev. "BABILong: Testing the Limits of LLMs with Long Context Reasoning-in-a-Haystack". In: *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2024. URL: https://openreview.net/forum?id=u7m2CG84BQ.

[134] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. "Deepseek-v3 technical report". In: *arXiv preprint arXiv:2412.19437* (2024).

[135] Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and Qiang Liu. "Longhorn: State space models are amortized online learners". In: *arXiv preprint arXiv:2407.14207* (2024).

[136] William Merrill, Jackson Petty, and Ashish Sabharwal. "The Illusion of State in State-Space Models". In: *Forty-first International Conference on Machine Learning*. 2024. URL: https://openreview.net/forum?id=QZgo9JZpLq.

[137] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. "The fineweb datasets: Decanting the web for the finest text data at scale". In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 30811–30849.

[138] Bo Peng, Daniel Goldstein, Quentin Gregory Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Teddy Ferdinan, Kranthi Kiran GV, Haowen Hou, Satyapriya Krishna, Ronald McClelland Jr., Niklas Muennighoff, Fares Obeid, Atsushi Saito, Guangyu Song, Haoqin Tu, Ruichong Zhang, Bingchen Zhao, Qihang Zhao, Jian Zhu, and Rui-Jie Zhu. "Eagle and Finch: RWKV with Matrix-Valued States and Dynamic Recurrence". In: *First Conference on Language Modeling*. 2024. URL: https://openreview.net/forum?id=soz1SEiPeq.

[139] Michael Poli, Armin W Thomas, Eric Nguyen, Pragaash Ponnusamy, Björn Deiseroth, Kristian Kersting, Taiji Suzuki, Brian Hie, Stefano Ermon, Christopher Ré, et al. "Mechanistic design and scaling of hybrid architectures". In: *arXiv preprint arXiv:2403.17844* (2024).

[140] Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. "Samba: Simple Hybrid State Space Models for Efficient Unlimited Context Language Modeling". In: *arXiv preprint arXiv:2406.07522* (2024).

[141] Ivan Rodkin, Yuri Kuratov, Aydar Bulatov, and Mikhail Burtsev. "Associative recurrent memory transformer". In: *arXiv preprint arXiv:2407.04841* (2024).

[142] Clayton Sanford, Daniel Hsu, and Matus Telgarsky. "Transformers, parallel computation, and logarithmic depth". In: *Forty-first International Conference on Machine Learning*. 2024. URL: https://openreview.net/forum?id=QCZabhKQhB.

[143] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. "Learning to (learn at test time): Rnns with expressive hidden states". In: *arXiv preprint arXiv:2407.04620* (2024).

[144] Garrett Tanzer, Mirac Suzgun, Eline Visser, Dan Jurafsky, and Luke Melas-Kyriazi. "A Benchmark for Learning to Translate a New Language from One Grammar Book". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=tbVWug9f2h.

[145] Wannan Yang, Chen Sun, Roman Huszár, Thomas Hainmueller, Kirill Kiselev, and György Buzsáki. "Selection of experience for memory by hippocampal sharp wave ripples". In: *Science* 383.6690 (2024), pp. 1478–1483.

[146] Ruiqi Zhang, Spencer Frei, and Peter L Bartlett. "Trained transformers learn linear models in-context". In: *Journal of Machine Learning Research* 25.49 (2024), pp. 1–55.

[147] Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhiquan Luo. "Why transformers need adam: A hessian perspective". In: *Advances in neural information processing systems* 37 (2024), pp. 131786–131823.

[148] Lisa Adams, Felix Busch, Tianyu Han, Jean-Baptiste Excoffier, Matthieu Ortala, Alexander Löser, Hugo JWL Aerts, Jakob Nikolas Kather, Daniel Truhn, and Keno Bressem. "Longhealth: A question answering benchmark with long clinical documents". In: *Journal of Healthcare Informatics Research* (2025), pp. 1–17.

[149] Zeyuan Allen-Zhu. "Physics of Language Models: Part 4.1, Architecture Design and the Magic of Canon Layers". In: *SSRN Electronic Journal* (May 2025). https://ssrn.com/abstract=5240330.

[150] Shaden Alshammari, John Hershey, Axel Feldmann, William T Freeman, and Mark Hamilton. "I-Con: A Unifying Framework for Representation Learning". In: *arXiv preprint arXiv:2504.16929* (2025).

[151] Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. "Atlas: Learning to optimally memorize the context at test time". In: *arXiv preprint arXiv:2505.23735* (2025).

[152] Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. "It's All Connected: A Journey Through Test-Time Memorization, Attentional Bias, Retention, and Online Optimization". In: *arXiv preprint arXiv:2504.13173* (2025).

[153] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. "Titans: Learning to Memorize at Test Time". In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025. URL: https://openreview.net/forum?id=8GjSf9Rh7Z.

[154] Franz Louis Cesista. *Heuristic Solutions for Steepest Descent on the Stiefel manifold*. July 2025. URL: %5Curl%7Bhttps://leloykun.github.io/ponder/steepest-descent-stiefel%7D.

[155] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. "Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities". In: *arXiv preprint arXiv:2507.06261* (2025).

[156] Benoit Dherin, Michael Munn, Hanna Mazzawi, Michael Wunder, and Javier Gonzalvo. "Learning without training: The implicit dynamics of in-context learning". In: *arXiv preprint arXiv:2507.16003* (2025).

[157] Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, et al. "Cartridges: Lightweight and general-purpose long context representations via self-study". In: *arXiv preprint arXiv:2506.06266* (2025).

[158] Riccardo Grazzi, Julien Siems, Jörg K.H. Franke, Arber Zela, Frank Hutter, and Massimiliano Pontil. "Unlocking State-Tracking in Linear RNNs Through Negative Eigenvalues". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=UvTo3tVBk2.

[159] Jiaxi Hu, Yongqi Pan, Jusen Du, Disen Lan, Xiaqiang Tang, Qingsong Wen, Yuxuan Liang, and Weigao Sun. "Improving Bilinear RNN with Closed-loop Control". In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025.

[160] Kazuki Irie and Samuel J Gershman. "Fast weight programming and linear transformers: from machine learning to neurobiology". In: *arXiv preprint arXiv:2508.08435* (2025).

[161] Ben Keigwin, Dhruv Pai, and Nathan Chen. *Gram-Space Manifold Muon*. Oct. 2025. URL: %5Curl%7Bhttps://blog.tilderesearch.com/vignettes/gram-space%7D.

[162] Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, et al. "Minimax-01: Scaling foundation models with lightning attention". In: *arXiv preprint arXiv:2501.08313* (2025).

[163] Saleh Momeni, Sahisnu Mazumder, Zixuan Ke, and Bing Liu. "In-context continual learning assisted by an external continual learner". In: *Proceedings of the 31st International Conference on Computational Linguistics*. 2025, pp. 7292–7306.

[164] Renhao Pei, Yihong Liu, Peiqin Lin, François Yvon, and Hinrich Schütze. "Understanding In-Context Machine Translation for Low-Resource Languages: A Case Study on Manchu". In: *arXiv preprint arXiv:2502.11862* (2025).

[165] Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Haowen Hou, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, et al. "RWKV-7" Goose" with Expressive Dynamic State Evolution". In: *arXiv preprint arXiv:2503.14456* (2025).

[166] Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Haowen Hou, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, et al. "Rwkv-7" goose" with expressive dynamic state evolution". In: *arXiv preprint arXiv:2503.14456* (2025).

[167] Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. "Optimizing Test-Time Compute via Meta Reinforcement Finetuning". In: *Forty-second International Conference on Machine Learning*. 2025. URL: https://openreview.net/forum?id=TqODUDsU4u.

[168] Chongjie Si, Debing Zhang, and Wei Shen. "Adamuon: Adaptive muon optimizer". In: *arXiv preprint arXiv:2507.11005* (2025).

[169] Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. "DeltaProduct: Improving State-Tracking in Linear RNNs via Householder Products". In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025. URL: https://openreview.net/forum?id=SoRiaijTGr.

[170] David Silver and Richard S Sutton. "Welcome to the era of experience". In: *Google AI* 1 (2025).

[171] Richard S. Sutton. *The OaK Architecture: A Vision of SuperIntelligence from Experience*. Keynote at the Reinforcement Learning Conference (RLC). Accessed: 2025-12-02. Aug. 2025. URL: https://rlj.cs.umass.edu/rlc-2025.

[172] Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham M. Kakade. "SOAP: Improving and Stabilizing Shampoo using Adam for Language Modeling". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=IDxZhXrpNf.

[173] Ke Alexander Wang, Jiaxin Shi, and Emily B Fox. "Test-time regression: a unifying framework for designing sequence models with associative memory". In: *arXiv preprint arXiv:2501.12352* (2025).

[174] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, junxian guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. "DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=cFu7ze7xUm.

[175] hongzhou yu, Tianhao Cheng, Yingwen Wang, Wen He, Qing Wang, Ying Cheng, Yuejie Zhang, Rui Feng, and Xiaobo Zhang. "FineMedLM-o1: Enhancing Medical Knowledge Reasoning Ability of LLM from Supervised Fine-Tuning to Test-Time Training". In: *Second Conference on Language Modeling*. 2025. URL: https://openreview.net/forum?id=7ZwuGZCopw.

[176] Yifan Zhang, Zhen Qin, and Quanquan Gu. "Higher-order Linear Attention". In: *arXiv preprint arXiv:2510.27258* (2025).

A Generalized Formulation for Nested Learning and Nested Systems

In this section, we discuss the generalized version of nested systems and NSAM, we discussed in Section 3:

Definition 6 ((Generalized) Nested System)**.** *A (ordered) nested system is a system with K (ordered) levels such that each level $1 \leq k \leq K$ consists of a set of optimization problems $\{(\mathcal{L}_i^{(k)}, C_i^{(k)}, \Theta_i^{(k)})\}_{i=1}^{N_k}$, where $\mathcal{L}_i(\cdot;\cdot)$ is the optimization objective in the i-th problem, C_i is its context (the data that is optimized on), Θ_i is the set of its parameters, and each optimization problem is optimized using gradient descent, or equivalently:*

$$\theta_{i_{t+1}}^{(k)} = \arg\min_{\Phi_i^{(k)}} \; \mathcal{L}_i^{(k)}(\Phi_i^{(k)}; x_{t+1}) + \frac{1}{2\eta_{i_{t+1}}^{(k)}} \|\Phi_i^{(k)} - \theta_{i_t}^{(k)}\|_2^2 \qquad \textit{where } x_{t+1} \sim C_i^{(k)}, \quad \textit{and} \quad \Phi_i{}^{(k)} \in \Theta_i^{(k)}. \tag{98}$$

Definition 7 ((Generalized) Nested System of Associative Memories)**.** *A nested system of associative memory (*NSAM*) is a system with K (ordered) levels such that each level $1 \leq k \leq K$ consists of a set of optimization problems $\{(\mathcal{L}_i^{(k)}, C_i^{(k)}, \Theta_i^{(k)})\}_{i=1}^{N_k}$, where $C_i = \{(\boldsymbol{k}_j^{(i)}, \boldsymbol{v}_j^{(i)})\}_{j=1}^{L_i}$ is a set of ground-truth mappings, $\mathcal{L}_i(\cdot;\cdot,\cdot)$ is the optimization objective that measures the quality of memory learned mappings in the i-th problem, Θ_i is the set of memory parameters, and each optimization problem is optimized using gradient descent:*

$$\theta_{i_{t+1}}^{(k)} = \arg\min_{\Phi_i^{(k)}} \; \mathcal{L}_i^{(k)}(\Phi_i^{(k)}; \boldsymbol{k}_{t+1}^{(i)}, \boldsymbol{v}_{t+1}^{(i)}) + \frac{1}{2\eta_{i_{t+1}}^{(k)}} \|\Phi_i^{(k)} - \theta_{i_t}^{(k)}\|_2^2 \qquad \textit{where } (\boldsymbol{k}_{t+1}^{(i)}, \boldsymbol{v}_{t+1}^{(i)}) \sim C_i^{(k)}, \quad \textit{and} \quad \Phi_i{}^{(k)} \in \Theta_i^{(k)}. \tag{99}$$

B Adam, AdaGrad, and Other Similar Optimizers as Associative Memory Modules

Revisiting the momentum term without using the chain rule in backpropagation,

$$\begin{aligned} W_{\ell_{t+1}} &= W_{\ell_t} + \boldsymbol{m}_{\ell_{t+1}} \\ \boldsymbol{m}_{\ell_{t+1}} &= \alpha_{\ell,t+1} \boldsymbol{m}_{\ell_t} - \eta_{\ell,t+1} \nabla_{W_{\ell_t}} \mathcal{L}\left(W_{\ell_t}; \boldsymbol{x}_{t+1}\right), \end{aligned} \tag{100}$$

one can interpret the momentum as a key or value-less associative memory (Behrouz et al. 2025b, see Section 5), where the gradient terms $\nabla_{W_{\ell_t}} \mathcal{L}\left(W_{\ell_t}; \boldsymbol{x}_{t+1}\right)$ are compressed into the momentum. We expect from a powerful momentum term to perfectly memorize all the past gradients in the training process so it can better model the current update to the weights. To this end, one can start from defining a simple objective as:

$$\tilde{\mathcal{L}}_t = \sum_{i=1}^{t} \|\boldsymbol{m}_{\ell_t} \odot \boldsymbol{g}_{\ell_{i+1}} - \boldsymbol{P}_{\ell_t}\|_2^2 \;\; + \;\; \lambda_\ell \|\boldsymbol{m}_{\ell_t}\|_F^2, \tag{101}$$

where $\boldsymbol{g}_{\ell_{t+1}} = -\nabla_{W_{\ell_t}} \mathcal{L}\left(W_{\ell_t}; \boldsymbol{x}_{t+1}\right)$. This objective aims to find a momentum term that does not simply map the gradients to 1 (which also results in limited memory management in momentum), but it maps gradients to a global property of past data samples. The more expressive this global property is, the more accurately the momentum can incorporate the compressed information from past. Accordingly, the objective in Equation 101 admits an optimal associative memory that maps gradients to $\boldsymbol{P}_{\ell_{t+1}}$ as:

$$\begin{aligned} \boldsymbol{m}_{\ell,i}^{(t)*} &= \left[\left(\boldsymbol{H}_{\ell,i}^{(t)} \;+\; \lambda_\ell \, \boldsymbol{I}\right)^{-1}\right] \odot \left(\boldsymbol{M}_{\ell,i}^{(t)}\right) = \left[\left(\boldsymbol{H}_{\ell,i}^{(t)} \;+\; \lambda_\ell \, \boldsymbol{I}\right)^{-1}\right] \odot \tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)} \odot \boldsymbol{P}_{\ell_t}, \qquad \text{where} \\ \boldsymbol{M}_{\ell,i+1}^{(t)} &= \boldsymbol{M}_{\ell,i}^{(t)} \;+\; \beta_1 \, \boldsymbol{g}_{\ell_{i+1}} \odot \boldsymbol{P}_{\ell_t} = \tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)} \odot \boldsymbol{P}_{\ell_t}, \\ \tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)} &= \boldsymbol{M}_{\ell,i}^{(t)} \;+\; \beta_1 \, \boldsymbol{g}_{\ell_{i+1}}, \\ \boldsymbol{H}_{\ell,i+1}^{(t)} &= \boldsymbol{H}_{\ell,i}^{(t)} + \beta_2 \, \boldsymbol{g}_{\ell_{i+1}} \odot \boldsymbol{g}_{\ell_{i+1}} = \boldsymbol{H}_{\ell,i}^{(t)} + \beta_2 \, \boldsymbol{g}_{\ell_{i+1}}^2. \end{aligned} \tag{102}$$

Given this solution, one can write the update step as:

$$W_{\ell_{i+1}} = W_{\ell_i} - \eta_t \, \boldsymbol{m}_{\ell,i}^{(t)*} = W_{\ell_i} - \eta_i \, \left[\left(\boldsymbol{H}_{\ell,i}^{(t)} \;+\; \lambda_\ell \, \boldsymbol{I}\right)^{-1}\right] \odot \tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)} \odot \boldsymbol{P}_{\ell_t}. \tag{103}$$

We start with a simple case, where $\boldsymbol{P}_{\ell_t}$ is the summation of the square of past gradients: i.e., $\boldsymbol{P}_{\ell_t} = \sum_{i=1}^{t} \boldsymbol{g}_{\ell_{i+1}}^2$. With the choice of $\lambda \longrightarrow 0$, Equation 103 recovers simple gradient descent with momentum. That is, let $\lambda = 0$, we have:

$$W_{\ell_{i+1}} = W_{\ell_i} - \eta_t \, \boldsymbol{m}_{\ell,i}^{(t)^*} = W_{\ell_i} - \eta_i \, \left(\boldsymbol{H}_{\ell,i}^{(t)}\right)^{-1} \odot \tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)} \odot \underbrace{\boldsymbol{P}_{\ell_t}}_{\boldsymbol{H}_{\ell,i}^{(t)}/\beta_2} = W_{\ell_i} - \eta_t \beta_2 \, \tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)}, \tag{104}$$

which based on the definition of $\tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)}$, this update rule is equivalent to gradient descent with momentum. Next, we explore a more sophisticated design choice, where we use $\boldsymbol{P}_{\ell_t}$ as the variance of data samples before token $t+1$. In this case, formally, $\boldsymbol{P}_{\ell_t} = \sqrt{\sum_{i=1}^{t} \boldsymbol{g}_{\ell_{i+1}}^2}$ and so the update rule is as follows:

$$W_{\ell_{i+1}} = W_{\ell_i} - \eta_t \, \boldsymbol{m}_{\ell,i}^{(t)^*} = W_{\ell_i} - \eta_i \, \left[\left(\boldsymbol{H}_{\ell,i}^{(t)} \; + \; \lambda_\ell \, \boldsymbol{I}\right)^{-1}\right] \odot \tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)} \odot \underbrace{\boldsymbol{P}_{\ell_t}}_{\boldsymbol{H}_{\ell,i}^{(t)^{\frac{1}{2}}}/\sqrt{\beta_2}} \approx W_{\ell_i} - \frac{\eta_t}{\sqrt{\beta_2}} \frac{\tilde{\boldsymbol{M}}_{\ell,i}^{(t)}}{\boldsymbol{H}_{\ell,i}^{(t)^{1/2}} + \varepsilon} \tag{105}$$

which is equivalent to the popular Adam optimizer (Kingma et al. 2014a). Therefore, Adam is an optimal associative memory given the L_2 regression objective that is defined in Equation 101. In fact, the update component of Adam at each state aims to learn a mapping between the gradients and their variance (as a global property of past data samples). One interesting point about this formulation is about the frequency of elements. Looking at the first and second momentum in Adam optimizer, the frequency of update for both of these memories are the same as both are updated after each sample. Furthermore, the computation of each of them can be done in parallel and so are independent. Accordingly, it is one of the few examples that two components without any internal gradient flow, are placed in the same frequency and so level (see Section 3.2).

Going beyond element-wise update, we reformulate Equation 101 with outer-product operation as:

$$\tilde{\mathcal{L}}_t = \sum_{i=1}^{t} \|\boldsymbol{m}_{\ell_t} \boldsymbol{g}_{\ell_{i+1}} - \boldsymbol{P}_{\ell_t}\|_2^2 \; + \; \lambda_\ell \|\boldsymbol{m}_{\ell_t}\|_F^2, \tag{106}$$

where $\boldsymbol{g}_{\ell_{t+1}} = -\nabla_{W_{\ell_t}} \mathcal{L}\left(W_{\ell_t}; \boldsymbol{x}_{t+1}\right)$. Similar to Equation 102, finding the optimal solution to the above objective is defined as:

$$\boldsymbol{m}_{\ell,i}^{(t)^*} = \left[\left(\boldsymbol{H}_{\ell,i}^{(t)} \; + \; \lambda_\ell \, \boldsymbol{I}\right)^{-1}\right] \left(\boldsymbol{M}_{\ell,i}^{(t)}\right) = \left[\left(\boldsymbol{H}_{\ell,i}^{(t)} \; + \; \lambda_\ell \, \boldsymbol{I}\right)^{-1}\right] \tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)} \boldsymbol{P}_{\ell_t}, \qquad \text{where} \tag{107}$$

$$\boldsymbol{M}_{\ell,i+1}^{(t)} = \boldsymbol{M}_{\ell,i}^{(t)} \; + \; \beta_1 \, \boldsymbol{P}_{\ell_t} \, \boldsymbol{g}_{\ell_{i+1}}^\top = \boldsymbol{P}_{\ell_t} \, \tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)}, \tag{108}$$

$$\tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)} = \boldsymbol{M}_{\ell,i}^{(t)} \; + \; \beta_1 \, \boldsymbol{g}_{\ell_{i+1}}^\top, \tag{109}$$

$$\boldsymbol{H}_{\ell,i+1}^{(t)} = \boldsymbol{H}_{\ell,i}^{(t)} + \beta_2 \, \boldsymbol{g}_{\ell_{i+1}} \boldsymbol{g}_{\ell_{i+1}}^\top. \tag{110}$$

With a similar choice as Adam, i.e., letting $\boldsymbol{P}_{\ell_t}$ be the variance of the gradients so far, $\boldsymbol{P}_{\ell_t} = \sqrt{\sum_{i=1}^{t} \boldsymbol{g}_{\ell_{i+1}} \boldsymbol{g}_{\ell_{i+1}}^\top}$, then the above solution can be simplified as:

$$W_{\ell_{i+1}} = W_{\ell_i} - \eta_t \, \boldsymbol{m}_{\ell,i}^{(t)^*} = W_{\ell_i} - \eta_i \, \left[\left(\boldsymbol{H}_{\ell,i}^{(t)} \; + \; \lambda_\ell \, \boldsymbol{I}\right)^{-1}\right] \underbrace{\boldsymbol{P}_{\ell_t}}_{\boldsymbol{H}_{\ell,i}^{(t)^{\frac{1}{2}}}/\sqrt{\beta_2}} \tilde{\boldsymbol{M}}_{\ell,i+1}^{(t)} \approx W_{\ell_i} - \frac{\eta_t}{\sqrt{\beta_2}} \, \boldsymbol{H}_{\ell,i}^{(t)^{-1/2}} \, \tilde{\boldsymbol{M}}_{\ell,i}^{(t)} \tag{111}$$

The above formulation is the AdaGrad with momentum (Défossez et al. 2022) and so generalizes the AdaGrad (Duchi et al. 2011), i.e., when $\beta_1 = 1$. Similarly, based on the connection of Adam optimizer (Kingma et al. 2014a) with other algorithms such as RMSProp (Hinton et al. 2012), SignSGD and its momentum-based variants (Bernstein et al. 2018), NAdam (Dozat 2016), AMSGrad (Reddi et al. 2016), RAdam (Liu et al. 2020), and Lion (Chen et al. 2023), as well as considering AdaGrad's connection with optimizers such as Shampoo (Gupta et al. 2018) and Soap (Vyas et al. 2025)–i.e., as the approximation of the preconditioning term–we can conclude that all these optimizers can be re-formulated as associative memory that aims to compress the gradients.

C Delta Gradient Descent with Normalization

In Section 4.5 we discussed that gradient descent can be see as an associative memory, and be reformulated as:

$$W_{t+1} = \arg\min_{W} \ \langle W\boldsymbol{x}_t, \nabla_{y_t}\mathcal{L}(W_t;\boldsymbol{x}_t)\rangle + \frac{1}{2\eta_t}\ \|W - W_t\|_2^2, \tag{112}$$

where each step aims at learning the negative of the gradient direction. Due to the fact that this learning rule only use update terms that depends on the current gradient, we defined $\mathbf{u}_t = -\nabla_{y_t}\mathcal{L}(W_t;\boldsymbol{x}_t)$, and extended the above process to Delta Gradient Descent with more expressive objective of $\mathcal{L}_2$ regression loss:

$$W_{t+1} = \arg\min_{W} \ \frac{1}{2}\|W\boldsymbol{x}_t - \mathbf{u}_t\|_2^2 + \frac{1}{2\eta_t}\|W - W_t\|_2^2. \tag{113}$$

We assume that $\boldsymbol{x}_t$ is normalized (e.g. in normalized memory systems or in neural networks with normalization layers, $\|\boldsymbol{x}_t\|_2 = \lambda$). Taking gradient to optimize the above objective,

$$2(W_{t+1}\boldsymbol{x}_t - \nabla_{y_t}\mathcal{L}(W_t,\boldsymbol{x}_t))\boldsymbol{x}_t^\top + 2\eta_t(W_{t+1} - W_t) = 0, \tag{114}$$

which results in:

$$W_{t+1}(\boldsymbol{x}_t\boldsymbol{x}_t^\top + \eta_t I) = \nabla_{y_t}\mathcal{L}(W_t,\boldsymbol{x}_t)\boldsymbol{x}_t^\top + \eta_t W_t, \tag{115}$$

$$\Rightarrow\ W_{t+1} = (\nabla_{y_t}\mathcal{L}(W_t,x_t)x_t^\top + \eta_t W_t)(\boldsymbol{x}_t\boldsymbol{x}_t^\top + \eta_t I)^{-1}. \tag{116}$$

To compute $(\boldsymbol{x}_t\boldsymbol{x}_t^\top + \eta_t I)^{-1}$ term with Sherman-Morrison lemma, we have:

$$(x_t x_t^\top + \eta_t I)^{-1} = \frac{1}{\eta_t}(I - \frac{1}{\lambda^2 + \eta_t}\boldsymbol{x}_t\boldsymbol{x}_t^\top), \tag{117}$$

and so:

$$W_{t+1} = (\nabla_{y_t}\mathcal{L}(W_t,\boldsymbol{x}_t)\boldsymbol{x}_t^\top + \eta_t W_t)\frac{1}{\eta_t}(I - \frac{1}{\lambda^2 + \eta_t}\boldsymbol{x}_t\boldsymbol{x}_t^\top) \tag{118}$$

$$\Rightarrow\ W_{t+1} = W_t\left(I - \frac{1}{\lambda^2 + \eta_t}\boldsymbol{x}_t\boldsymbol{x}_t^\top\right) + \frac{1}{\eta_t}\nabla_{y_t}\mathcal{L}(W_t,\boldsymbol{x}_t)\boldsymbol{x}_t^\top - \underbrace{\frac{1}{\lambda^2\eta_t + \eta_t^2}\nabla_{y_t}\mathcal{L}(W_t,\boldsymbol{x}_t)\boldsymbol{x}_t^\top\boldsymbol{x}_t\boldsymbol{x}_t^\top}_{\frac{\lambda}{\lambda^2\eta_t+\eta_t^2}\nabla_{y_t}\mathcal{L}(W_t,\boldsymbol{x}_t)\boldsymbol{x}_t^\top} \tag{119}$$

$$\Rightarrow\ W_{t+1} = W_t\left(I - \frac{1}{\lambda^2 + \eta_t}\boldsymbol{x}_t\boldsymbol{x}_t^\top\right) - \left(\frac{\lambda}{\lambda^2\eta_t + \eta_t^2} - \frac{1}{\eta_t}\right)\nabla_{y_t}\mathcal{L}(W_t,\boldsymbol{x}_t)\boldsymbol{x}_t^\top \tag{120}$$

$$\Rightarrow\ W_{t+1} = W_t\left(I - \alpha_t\boldsymbol{x}_t\boldsymbol{x}_t^\top\right) - \beta\ \nabla_{y_t}\mathcal{L}(W_t,\boldsymbol{x}_t)\boldsymbol{x}_t^\top \tag{121}$$