Titans: Learning to Memorize at Test Time

Ali Behrouz[†], Peilin Zhong[†], and Vahab Mirrokni[†]

[†]Google Research

{alibehrouz, peilinz, mirrokni}@google.com

Abstract

Over more than a decade there has been an extensive research effort of how effectively utilize recurrent models and attentions. While recurrent models aim to compress the data into a fixed-size memory (called hidden state), attention allows attending to the entire context window, capturing the direct dependencies of all tokens. This more accurate modeling of dependencies, however, comes with a quadratic cost, limiting the model to a fixed-length context. We present a new neural long-term memory module that learns to memorize historical context and helps an attention to attend to the current context while utilizing long past information. We show that this neural memory has the advantage of a fast parallelizable training while maintaining a fast inference. From a memory perspective, we argue that attention due to its limited context but accurate dependency modeling performs as a short-term memory, while neural memory due to its ability to memorize the data, acts as a long-term, more persistent, memory. Based on these two modules, we introduce a new family of architectures, called Titans, and present three variants to address how one can effectively incorporate memory into this architecture. Our experimental results on language modeling, common-sense reasoning, genomics, and time series tasks show that Titans are more effective than Transformers and recent modern linear recurrent models. They further can *effectively* scale to larger than 2M context window size with higher accuracy in needle-in-haystack tasks compared to baselines.

1 Introduction

"The true art of memory is the art of attention!"

— Samuel Johnson, 1787

Transformers, pure attention-based architectures (Vaswani et al. 2017), have been firmly established as state-of-the-art models in sequence modeling, mainly due to their in-context learning and ability to learn at scale (Kaplan et al. 2020). The primary building blocks of Transformers–attention modules—function as associative memory blocks (Bietti et al. 2024), where they learn to store key-value associations and retrieve them by computing pairwise similarity between queries (i.e., search signals) and keys (i.e., contexts). Accordingly, by design, the output of a Transformer is exclusively conditioned on the direct dependencies of tokens in the *current* context window. This accurate modeling of dependencies, however, comes with quadratic time and memory complexity in terms of the context length. In complex real-world tasks (e.g., language modeling (N. F. Liu et al. 2024), video understanding (C.-Y. Wu et al. 2019), long-term time series forecasting (H. Zhou et al. 2021)), the context window can become extremely large, making the applicability of Transformers challenging in these downstream tasks.

To overcome the scalability issue of Transformers, recent studies aim to design different variants of linear Transformers (Kacham, Mirrokni, and P. Zhong 2024; Katharopoulos et al. 2020; S. Yang, B. Wang, Shen, et al. 2024), where softmax is replaced by a kernel function in the attention (see §2.1 for details), resulting in a significant drop in memory consumption. Despite efficiency and the ability to scale to longer context, linear Transformers do not show competitive performance compared to Transformers as the kernel trick makes the model a linear recurrent network, in which the data is compressed into a matrix-valued states (Katharopoulos et al. 2020). This, however, brings a contradictory fact about linear recurrent (or linear Transformers) models: On one hand, we use these linear models to enhance scalability and efficiency (linear vs. quadratic complexity), whose advantages is appeared for very long context; On the other hand, a very long context cannot be properly compressed in a small vector-valued or matrix-valued states (S. Wang 2024).

Titans: Learning to Memorize at Test Time

Ali Behrouz[†], Peilin Zhong[†], and Vahab Mirrokni[†]

[†]Google Research
{alibehrouz, peilinz, mirrokni}@google.com

Abstract

Over more than a decade there has been an extensive research effort of how effectively utilize recurrent models and attentions. While recurrent models aim to compress the data into a fixed-size memory (called hidden state), attention allows attending to the entire context window, capturing the direct dependencies of all tokens. This more accurate modeling of dependencies, however, comes with a quadratic cost, limiting the model to a fixed-length context. We present a new neural long-term memory module that learns to memorize historical context and helps an attention to attend to the current context while utilizing long past information. We show that this neural memory has the advantage of a fast parallelizable training while maintaining a fast inference. From a memory perspective, we argue that attention due to its limited context but accurate dependency modeling performs as a short-term memory, while neural memory due to its ability to memorize the data, acts as a long-term, more persistent, memory. Based on these two modules, we introduce a new family of architectures, called Titans, and present three variants to address how one can effectively incorporate memory into this architecture. Our experimental results on language modeling, common-sense reasoning, genomics, and time series tasks show that Titans are more effective than Transformers and recent modern linear recurrent models. They further can *effectively* scale to larger than 2M context window size with higher accuracy in needle-in-haystack tasks compared to baselines.

"The true art of memory is the art of attention!"

— Samuel Johnson, 1787

1 Introduction

Transformers, pure attention-based architectures (Vaswani et al. 2017), have been firmly established as state-of-the-art models in sequence modeling, mainly due to their in-context learning and ability to learn at scale (Kaplan et al. 2020). The primary building blocks of Transformers–attention modules—function as associative memory blocks (Bietti et al. 2024), where they learn to store key-value associations and retrieve them by computing pairwise similarity between queries (i.e., search signals) and keys (i.e., contexts). Accordingly, by design, the output of a Transformer is exclusively conditioned on the direct dependencies of tokens in the *current* context window. This accurate modeling of dependencies, however, comes with quadratic time and memory complexity in terms of the context length. In complex real-world tasks (e.g., language modeling (N. F. Liu et al. 2024), video understanding (C.-Y. Wu et al. 2019), long-term time series forecasting (H. Zhou et al. 2021)), the context window can become extremely large, making the applicability of Transformers challenging in these downstream tasks.

To overcome the scalability issue of Transformers, recent studies aim to design different variants of linear Transformers (Kacham, Mirrokni, and P. Zhong 2024; Katharopoulos et al. 2020; S. Yang, B. Wang, Shen, et al. 2024), where softmax is replaced by a kernel function in the attention (see §2.1 for details), resulting in a significant drop in memory consumption. Despite efficiency and the ability to scale to longer context, linear Transformers do not show competitive performance compared to Transformers as the kernel trick makes the model a linear recurrent network, in which the data is compressed into a matrix-valued states (Katharopoulos et al. 2020). This, however, brings a contradictory fact about linear recurrent (or linear Transformers) models: On one hand, we use these linear models to enhance scalability and efficiency (linear vs. quadratic complexity), whose advantages is appeared for very long context; On the other hand, a very long context cannot be properly compressed in a small vector-valued or matrix-valued states (S. Wang 2024).

Furthermore, beyond efficiency, most existing architectures–ranging from Hopfield Networks (Hopfield 1982) to LSTMs (Jürgen Schmidhuber and Hochreiter 1997) and Transformers (Vaswani et al. 2017)–face challenges when dealing with generalization, length extrapolation, and/or reasoning (Anil et al. 2022; Qin, Y. Zhong, and Deng 2024), all of which are inseparable parts of many hard real-world tasks. Although these architectures draw inspiration from the human brain, each of which are missing: (1) a crucial component for learning process–such as short-term memory, long-term memory, meta-memory, attending to current context, etc. (Cowan 2008); (2) how these components are interconnected systems that can operate independently; and/or (3) the ability to *actively* learn from data and memorize the abstraction of past history. We argue that in an effective learning paradigm, similar to human brain, there are *distinct* yet interconnected modules, each of which is responsible for a component crucial to the learning process.

Memory Perspective

Memory is a fundamental mental process and is an inseparable component of human learning (Terry 2017). Without a properly functioning memory system, humans and animals would be restricted to basic reflexes and stereotyped behaviors. Accordingly, memory has been the inspiration for many seminal research in machine learning literature; e.g., Hopfield Networks (Hopfield 1982), LSTMs (Jürgen Schmidhuber and Hochreiter 1997), and Transformers (Vaswani et al. 2017).

Taking inspiration from the common definitions of memory and learning in neuropsychology literature (Okano, Hirano, and Balaban 2000), most existing architectures consider memory as a neural update caused by an input, and define learning as a process for acquiring effective and useful memory, given an objective. In this perspective, Recurrent Neural Networks (RNNs) (Williams and Zipser 1989) can be defined as models with a vector-valued memory module $\mathcal{M}$ (also called hidden state) with two main steps: Given a new input x_t at time t, the model (1) updates the memory using a function $f(\mathcal{M}_{t-1}, x_t)$ (with compression); and (2) retrieves the corresponding memory of input using a function $g(\mathcal{M}_t, x_t)$ (see §2.1 for details). Similarly, Transformers can be seen as architectures with a growing memory and two similar steps. That is, the pair of key and value matrices acts as the model's memory, and the model: (1) updates the memory by appending the key and value to the memory (without compression), and (2) retrieves query vectors' corresponding memory by finding the similarity of query and key vectors, which is then used to weight the value vectors for the output.

This perspective, can help us better understand existing paradigms, their critical differences, and design more effective architectures. For example, the main difference between Transformers (Vaswani et al. 2017) and *linear* Transformers (Katharopoulos et al. 2020) is the memory structure as well as the memory updating step, in which linear Transformers compress the historical data into a fixed-size matrix-valued memory while Transformers keep all historical data (within the context length) without any compression. While both linear Transformers and linear RNNs (including state space models) compress the information in memory update step, the critical difference lies in the structure of the memory, where linear RNNs (vs. linear Transformers) use a vector-valued memory (vs. matrix-valued memory). Therefore, this perspective motivates us to ask: (Q1) What constitute a good structure for the memory? (Q2) What is a proper memory update mechanism? and (Q3) What is a good memory retrieval process?

Revisiting our understanding of human memory, it is neither a unitary process nor it serves a single function (Cowan 2008). In fact, memory is a confederation of systems–e.g., short-term, working, and long-term memory–each serving a different function with different neural structures, and each capable of operating independently (Willingham 1997). This fact motivates us to ask: (Q4) How to design an efficient architecture that incorporates different interconnected memory modules. Finally, storing a memory is a neural process that requires to encode and store the abstraction of the past. It can be over-simplification to assume a single vector or a matrix, whose parameters are encoding the data in a linear manner, are enough for storing long-term history. (Q5) Is a deep memory module needed to effectively store/remember long past?

Contributions and Roadmap

In this paper, we aim to answer the above five questions by designing a long-term neural memory module, that can efficiently and effectively learn to memorize at test time. Building upon its design, we discuss how it can be incorporated into an architecture.

Neural Memory (§3). We present a (deep) neural long-term memory that (as a meta in-context model) learns how to memorize/store the data into its parameters at test time. Inspired by human long-term memory system (Mandler 2014), we design this memory module so an event that violates the expectations (being surprising) is more memorable. To this end, we measure the surprise of an input with the gradient of the neural network with respect to the input in *associative memory loss* (see §3.1 for details). To better handle the limited memory, we present a decaying mechanism that consider the proportion of memory size and the amount of data surprise, resulting in better memory management. We show that this decay mechanism is in fact the generalization of forgetting mechanism in modern recurrent models (Dao and Gu 2024; Gu and Dao 2024; S. Yang, Kautz, and Hatamizadeh 2024). Interestingly, we find that this mechanism is equivalent to optimizing a meta neural network with mini-batch gradient descent, momentum, and weight decay. Building upon tensorizing mini-batch gradient descent to use more matmul operations (Yu Sun et al. 2024), we present a fast and parallelizable algorithm to train our deep neural long-term memory.

Titans Architectures (§4). After designing the long-term neural memory, an important remaining question is how to effectively and efficiently incorporate memory into a deep learning architecture. We present Titans, a family of deep models that consists of three hyper-heads: (1) Core: this module consists of the short-term memory, and is responsible for the main flow of processing the data (we use attention with limited window size); (2) Long-term Memory: this branch is our neural long-term memory module that is responsible to store/remember long past; (3) Persistent Memory: this is a set of learnable but date-independent parameters that encodes the knowledge about a task. Finally, as a proof of concept, we present three variants of Titans, in which we incorporate memory as: (i) a context, (ii) a layer, and (iii) a gated branch.

Experimental Results (§5). We perform experimental evaluations on language modeling, commonsense reasoning, recall-intensive, needle in haystack, time series forecasting, and DNA modeling tasks. We observe that our Titan architecture outperforms all modern recurrent models as well as their hybrid variants (combining with sliding-window attention) across a comprehensive set of benchmarks. Furthermore, Titans outperforms Transformers with the same context window, and show competitive performance with Transformers that use the entire context. This results are achieved while, contrary to Transformers, Titans scale to larger than 2M context window size.

2 Preliminaries

In this section, we discuss the notation and some background concepts that we use though the paper. We let $x \in \mathbb{R}^{N \times d_{\text{in}}}$ be the input, $\mathcal{M}$ be a neural network (neural memory module), $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ be the query, key and value of the attention mechanism, and $\mathbf{M}$ be the attention mask. When segmenting the sequence, we use $\mathrm{S}^{(i)}$ to refer to the i-th segment. Through the paper, we abuse the notation and use subscripts to refer to a specific element of a matrix, vector, or segments. For example, we let $\mathrm{S}^{(i)}_j$ be the j-th token in the i-th segment. The only exception is subscripts with t, which we reserved to index recurrence over time, or the state of a neural network at time t. Given a neural network $\mathcal{N}$ and a data sample x, we use $\mathcal{N}(x)$ (resp. $\mathcal{N}^*(x)$) to refer to the forward pass with (resp. without) weight adjustment. Also, we abuse the notation and use $\mathcal{N}^{(k)}$ to refer to the k-th layer of the neural network. In the following, we first, discuss the backgrounds for attention and its efficient variants followed by a review of modern linear RNNs. Finally, we discuss a memory perspective of these architectures that motivates us to design Titans.

2.1 Backgrounds

Attention. Transformers (Vaswani et al. 2017) as the de facto backbone for many deep learning models are based on attention mechanism. Given input $x \in \mathbb{R}^{N \times d_{\text{in}}}$, causal attention computes output $\mathrm{y} \in \mathbb{R}^{N \times d_{\text{in}}}$ based on softmax over input dependent key, value, and query matrices:

$$\mathbf{Q} = x\mathbf{W}_{\mathbf{Q}}, \qquad \mathbf{K} = x\mathbf{W}_{\mathbf{K}}, \qquad \mathbf{V} = x\mathbf{W}_{\mathbf{V}}, \tag{1}$$

$$\mathrm{y}_i = \sum_{j=1}^{i} \frac{\exp\left(\mathbf{Q}_i^\top \mathbf{K}_j / \sqrt{d_{\text{in}}}\right) \mathbf{V}_j}{\sum_{\ell=1}^{i} \exp\left(\mathbf{Q}_i^\top \mathbf{K}_\ell / \sqrt{d_{\text{in}}}\right)}, \tag{2}$$

where $\mathbf{W}_{\mathbf{Q}}, \mathbf{W}_{\mathbf{K}},$ and $\mathbf{W}_{\mathbf{V}} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$ are learnable parameters. Despite the power and effectiveness in recall, transformers need at least $N \times d$ operators to calculate the output, resulting in larger memory consumption and lower-throughput for longer sequences.

Efficient Attentions. To improve the memory consumption and throughput of softmax attention for longer sequences, various studies focused on I/O aware implementations of attention (Dao 2024; Dao, D. Fu, et al. 2022), designing more