Module I: Titans Memory Architecture (Multi-Layer Memory Module)

Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence

—

Contents

1 Abstract

This module rigorously defines the *Titans Memory Architecture*, a multi-layer memory system designed for test–time adaptation and continual learning within the Eidos framework. The architecture comprises multiple layers of memory including short-term, working, long-term, and personal memory. A memory bank stores key–value pairs, and a similarity-based retrieval mechanism aggregates relevant memory content via attention. A meta–learner then uses this aggregated memory read to produce adaptive parameter updates for the model. The design supports idempotent, reversible, and efficient retrieval and updating under heterogeneous hardware constraints. We present formal definitions, algorithmic descriptions, theoretical guarantees, and integration considerations with the highest academic rigor.

2 Introduction and Motivation

Robust memory mechanisms are essential for adaptive systems, particularly in contexts where long-term dependencies, contextual adjustments, and continual learning are required. The *Titans Memory Architecture* is engineered to support multiple layers of memory:

- **Short-Term Memory:** Captures transient, context-dependent information for immediate tasks.
- **Working Memory:** Maintains intermediate representations relevant to the current task or sequence.
- **Long-Term Memory:** Stores persistent, significant information acquired over extended periods.
- **Personal Memory:** Reflects adaptive, individualized knowledge updated continuously during deployment.

This module enables the system to retrieve and integrate memory efficiently at test time and to update model parameters dynamically based on retrieved information. The design emphasizes modularity, scalability, and hardware-agnostic deployment, ensuring that even models with extensive memory components can be efficiently managed.

3 Preliminaries and Notation

We define the following notation to be used throughout this module:

- Let $r \in \mathbb{R}^{d_L}$ denote a latent representation of an input, computed by an encoder g_θ from a deep model.
- The memory bank is denoted by

$$\mathcal{M} = \{(k_i, v_i) \mid i = 1, \ldots, M_{\mathcal{M}}\},$$

  where:
  - $k_i \in \mathbb{R}^{d_k}$ is the key corresponding to a stored memory element,
  - $v_i \in \mathbb{R}^{d_v}$ is the associated value (which may encode feature corrections, gradient information, or auxiliary signals).

- A similarity function is defined as:

$$s : \mathbb{R}^{d_L} \times \mathbb{R}^{d_k} \to \mathbb{R},$$

for instance, using cosine similarity:

$$s(r, k_i) = \frac{\langle W_s r,\, k_i \rangle}{\|W_s r\| \|k_i\|},$$

where $W_s \in \mathbb{R}^{d_L \times d_k}$ is a learnable projection matrix.

- A temperature parameter $\tau > 0$ is used to scale similarity scores.
- Attention weights over the memory are computed as:

$$\alpha_i(x) = \frac{\exp\big(s(r, k_i)/\tau\big)}{\sum_{j=1}^{M_{\mathcal{M}}} \exp\big(s(r, k_j)/\tau\big)}.$$

- The aggregated memory read is defined as:

$$m(x) = \sum_{i=1}^{M_{\mathcal{M}}} \alpha_i(x)\, v_i \in \mathbb{R}^{d_v}.$$

- A meta–learner is defined as:

$$h : \mathbb{R}^{d_v} \to \mathbb{R}^p,$$

which maps the memory read $m(x)$ to an update vector $\Delta\theta(x)$ for the model parameters.

- The adapted parameters are given by:

$$\theta_x = \theta + \Delta\theta(x),$$

where θ are the base model parameters.

- The architecture is *multi-layered*, partitioning $\mathcal{M}$ into sub-banks:

$$\mathcal{M} = \mathcal{M}_{\text{short}} \cup \mathcal{M}_{\text{working}} \cup \mathcal{M}_{\text{long}} \cup \mathcal{M}_{\text{personal}},$$

each of which may be processed with different weights or retrieval strategies.

4 Formal Definitions and Mathematical Formulation

Definition I.1 (Memory Bank)

The memory bank is defined as:

$$\mathcal{M} = \{(k_i, v_i) \mid i = 1, \ldots, M_{\mathcal{M}}\},$$

where for each i:

- $k_i \in \mathbb{R}^{d_k}$ is the key vector associated with a particular memory unit.
- $v_i \in \mathbb{R}^{d_v}$ is the corresponding value vector, which encodes information such as contextual corrections or learned feature adjustments.

Definition I.2 (Memory Retrieval Mechanism)

Given a latent representation $r \in \mathbb{R}^{d_L}$ derived from an input x by an encoder g_θ, the similarity between r and each memory key k_i is computed as:

$$s(r, k_i) = \frac{\langle W_s r,\, k_i \rangle}{\|W_s r\|\, \|k_i\|},$$

where $W_s \in \mathbb{R}^{d_L \times d_k}$ is a learnable projection. The attention weight for each memory slot is then:

$$\alpha_i(x) = \frac{\exp\big(s(r, k_i)/\tau\big)}{\sum_{j=1}^{M_{\mathcal{M}}} \exp\big(s(r, k_j)/\tau\big)}.$$

The aggregated memory read is defined by:

$$m(x) = \sum_{i=1}^{M_{\mathcal{M}}} \alpha_i(x)\, v_i.$$

Definition I.3 (Meta–Learner for Test-Time Adaptation)

The meta–learner is a function:

$$h : \mathbb{R}^{d_v} \to \mathbb{R}^p,$$

which computes a parameter update:

$$\Delta\theta(x) = h\big(m(x)\big).$$

The model parameters are adapted at test time via:

$$\theta_x = \theta + \Delta\theta(x),$$

where $\theta \in \Theta$ are the base parameters.

Definition I.4 (Multi-Layer Memory Structure)

We partition the memory bank into hierarchical layers:

$$\mathcal{M} = \mathcal{M}_{\text{short}} \cup \mathcal{M}_{\text{working}} \cup \mathcal{M}_{\text{long}} \cup \mathcal{M}_{\text{personal}},$$

where:

- $\mathcal{M}_{\text{short}}$ stores transient, context-specific information.
- $\mathcal{M}_{\text{working}}$ maintains task-related intermediate representations.
- $\mathcal{M}_{\text{long}}$ holds persistent information acquired over extended periods.
- $\mathcal{M}_{\text{personal}}$ captures adaptive, user- or domain-specific knowledge.

Each sub-bank can be processed using a specialized similarity function s_ℓ and may be aggregated via weighted summation:

$$m(x) = \sum_{\ell \in \{\text{short, working, long, personal}\}} w_\ell\, m_\ell(x),$$

where:

$$m_\ell(x) = \sum_{i \in I_\ell} \alpha_i^{(\ell)}(x)\, v_i^{(\ell)},$$

with I_ℓ indexing the memory units in layer ℓ, and w_ℓ are weights (learned or pre-defined) governing the contribution of each layer.

5 Algorithmic Description

The following pseudocode details the operation of the Titans Memory Architecture, including memory retrieval, aggregation, and test–time parameter adaptation.

Algorithm 1 Titans Memory Architecture: Memory Retrieval and Adaptation

Input: Test input $x \in \mathcal{X}$; encoder g_θ; memory banks $\{\mathcal{M}_\ell\}_{\ell \in \{\text{short, working, long, personal}\}}$; temperature τ; meta–learner h

Output: Adapted parameters θ_x or direct prediction update

Compute: Latent representation $r \leftarrow g_\theta(x) \in \mathbb{R}^{d_L}$

for each memory layer ℓ in {short, working, long, personal} **do**

  for each memory unit $(k_i^{(\ell)}, v_i^{(\ell)}) \in \mathcal{M}_\ell$ **do**

    Compute similarity: $s_i^{(\ell)} \leftarrow s_\ell(r, k_i^{(\ell)})$

  end for

  Compute attention weights:

$$\alpha_i^{(\ell)} \leftarrow \frac{\exp\big(s_i^{(\ell)}/\tau\big)}{\sum_{j \in I_\ell} \exp\big(s_j^{(\ell)}/\tau\big)}$$

  Aggregate memory read for layer ℓ:

$$m_\ell(x) \leftarrow \sum_{i \in I_\ell} \alpha_i^{(\ell)}\, v_i^{(\ell)}$$

end for

Combine Layers:

$$m(x) \leftarrow \sum_\ell w_\ell\, m_\ell(x)$$

Meta–Learner Update:

$$\Delta\theta(x) \leftarrow h\big(m(x)\big)$$

Update parameters:

$$\theta_x \leftarrow \theta + \Delta\theta(x)$$

Return: θ_x (or use θ_x to compute prediction $\hat{y} = f_{\theta_x}(x)$)

6 Theoretical Analysis and Guarantees

Theorem I.1 (Convergence and Stability of Memory Read)

Statement: Assume that for each layer ℓ, the similarity function s_ℓ is bounded and the attention weights $\alpha_i^{(\ell)}$ form a probability distribution. Then, for any input x, the aggregated memory read

$$m(x) = \sum_\ell w_\ell \left(\sum_{i \in I_\ell} \alpha_i^{(\ell)}\, v_i^{(\ell)} \right)$$

is a well-defined, bounded vector. Furthermore, if the memory banks are updated in a controlled manner, then the meta–learner update $\Delta\theta(x) = h(m(x))$ converges, and repeated adaptation leads to a stable parameter set θ_x.

Proof Sketch: Since $\alpha_i^{(\ell)} \geq 0$ and $\sum_{i \in I_\ell} \alpha_i^{(\ell)} = 1$, each $m_\ell(x)$ is a convex combination of bounded vectors $v_i^{(\ell)}$. Hence, $m(x)$ is bounded. Under standard assumptions on the contraction properties of h (e.g., Lipschitz continuity with a constant less than one), iterative updates will converge to a fixed point. □

Proposition I.2 (Scalability)

The hierarchical partitioning of $\mathcal{M}$ into layers enables the system to scale with the overall amount of memory. Since each layer is managed separately and combined via weighted summation, the in-memory retrieval operations remain efficient even as the total number of memory units grows.

7 Integration with the Overall Eidos Framework

Module I, the Titans Memory Architecture, is a critical component of the Eidos system. It:

- Provides a multi-layer memory system that supports test–time adaptation and continual learning.
- Interfaces with the Core Model Architectures (Module H) by supplying adaptive updates through the meta–learner.
- Receives latent representations from upstream modules (e.g., contextual embeddings from Module E) and aggregates memory information to influence model parameters.
- Supports real-time updates and retrieval, ensuring that the system remains adaptive to new data and domain shifts.

8 Implementation Considerations

- **Efficient Storage:** Memory banks should be implemented using data structures optimized for sparse retrieval and vector operations (e.g., approximate nearest neighbor search structures).
- **Parallel Retrieval:** Similarity computations and attention weight calculations can be parallelized across memory layers.
- **Dynamic Weighting:** The weights w_ℓ for combining memory layers may be learned or set based on domain knowledge to reflect the relative importance of each memory type.
- **Meta–Learner Design:** The function h should be designed to produce small, stable updates to the model parameters and may itself be a shallow neural network.
- **Resource Constraints:** Considerations for computational resources (e.g., GPU memory) should guide the number of memory units retained in active memory.

9 Conclusion

In this module, we have rigorously defined the Titans Memory Architecture, a multi-layer memory system essential for test–time adaptation and continual learning in the Eidos framework. The module:

- Introduces a memory bank $\mathcal{M}$ partitioned into hierarchical layers (short-term, working, long-term, and personal).
- Defines a similarity-based retrieval mechanism that computes attention weights $\alpha_i(x)$ over memory units.
- Aggregates memory reads from different layers via weighted summation.
- Utilizes a meta–learner h to convert the aggregated memory read into adaptive parameter updates.
- Provides theoretical guarantees regarding the boundedness, convergence, and scalability of the memory retrieval process.

This architecture enables the overall system to adapt dynamically to new data and contextual changes, thereby enhancing model performance and robustness in diverse environments.

10 Module Summary

Completed:

- Module A: Input Processing.
- Module B: Universal Communication & Data Handling Interface and Coordination.
- Module C: Universal Streaming/Handling/Loading/Indexing Module.
- Module D: Multidimensional Vocabulary and Tokenization System.
- Module E: Contextual NLU/NLP Embedding and Multidimensional Tokenization.
- Module F: Deep Knowledge Graphs System (Base and Personal).
- Module G: Infinite RoPE Context Scaling and Dynamic Vocabulary Updating.
- Module H: Core Model Architectures (RWKV and Transformer Modules, Mixture-of-Experts Style).
- Module I: Titans Memory Architecture (Multi-Layer Memory Module).

Remaining Modules:

- Module J: Recursive Adaptive Dynamic Idempotent Feedback and State-Based Runtime Learning and Inference.
- Module K: Universal Training System.
- Module L: Final Decoding and Multimodal Output.