

# Học lồng ghép: ảo tưởng về các kiến trúc học sâu

## Nested learning: the illusion of deep learning architectures



**Ali Behrouz**  
Google Research  
USA  
alibehrouz@google.com

**Meisam Razaviyayn**  
Google Research  
USA  
razaviyayn@google.com

**Peiling Zhong** Google  
Research USA  
peilingz@google.com

**Vahab Mirrokni**  
Google Research  
USA  
mirrokni@google.com

*Link bài báo:*

- <https://research.google/blog/introducing-nested-learning-a-new-ml-paradigm-for-continual-learning/>
- <https://abehrouz.github.io/files/NL.pdf>

*Viết hoá bởi:*

- **Bùi Huỳnh Kinh Luân** - [luanbhk@gmail.com](mailto:luanbhk@gmail.com)

# Tóm tắt

Trong những thập kỷ qua, việc phát triển các kiến trúc nơ-ron mạnh mẽ hơn và đồng thời thiết kế các thuật toán tối ưu hóa để huấn luyện chúng một cách hiệu quả đã là trọng tâm của các nỗ lực nghiên cứu nhằm nâng cao năng lực của các mô hình học máy. Bất chấp những tiến bộ gần đây, đặc biệt là trong việc phát triển các mô hình ngôn ngữ (Language Models - LMs), vẫn tồn tại những thách thức cơ bản và các câu hỏi chưa có lời giải về cách các mô hình như vậy có thể liên tục học/ ghi nhớ, tự cải thiện và tìm ra các "giải pháp hiệu quả".

Trong bài báo này, chúng tôi trình bày một hệ hình học tập mới, được gọi là **Nested Learning (NL) - học lồng ghép**, giúp biểu diễn mô hình một cách nhất quán thông qua tập hợp các bài toán tối ưu hóa lồng nhau, đa cấp độ và (hoặc) song song, trong đó mỗi bài toán có "luồng ngữ cảnh" (context flow) riêng.

Học lồng ghép chỉ ra rằng các phương pháp học sâu hiện có học từ dữ liệu thông qua việc nén luồng ngữ cảnh của chính chúng, và giải thích cơ chế này sinh khả năng "học trong ngữ cảnh" (in-context learning) ở các mô hình lớn. Học lồng ghép đề xuất một hướng đi (một chiều hướng mới cho học sâu) để thiết kế các thuật toán học tập có khả năng biểu đạt cao hơn với nhiều "cấp độ" hơn, dẫn đến khả năng học trong ngữ cảnh bậc cao.

Bên cạnh tính hợp lý về mặt khoa học thần kinh và bản chất "hộp trắng" (white-box) minh bạch về mặt toán học, chúng tôi khẳng định tầm quan trọng của phương pháp này thông qua ba đóng góp cốt lõi:

1. **Các bộ tối ưu hóa sâu (Deep Optimizers):** Dựa trên học lồng ghép, chúng tôi chỉ ra rằng các bộ tối ưu hóa dựa trên gradient nổi tiếng (ví dụ: Adam, SGD kết hợp Momentum,...) thực chất là các mô-đun bộ nhớ liên kết (associative memory modules) nhằm mục đích nén các gradient bằng phương pháp giảm gradient (gradient descent). Từ hiểu biết này, chúng tôi trình bày một tập hợp các bộ tối ưu hóa có khả năng biểu đạt cao hơn với bộ nhớ sâu và/hoặc các quy tắc học mạnh mẽ hơn.
2. **Các mô hình Titans tự sửa đổi (Self-Modifying Titans):** Tận dụng những hiểu biết của học lồng ghép về các thuật toán học tập, chúng tôi trình bày một mô hình chuỗi mới lạ biết cách tự sửa đổi bản thân bằng việc học thuật toán cập nhật của chính nó.

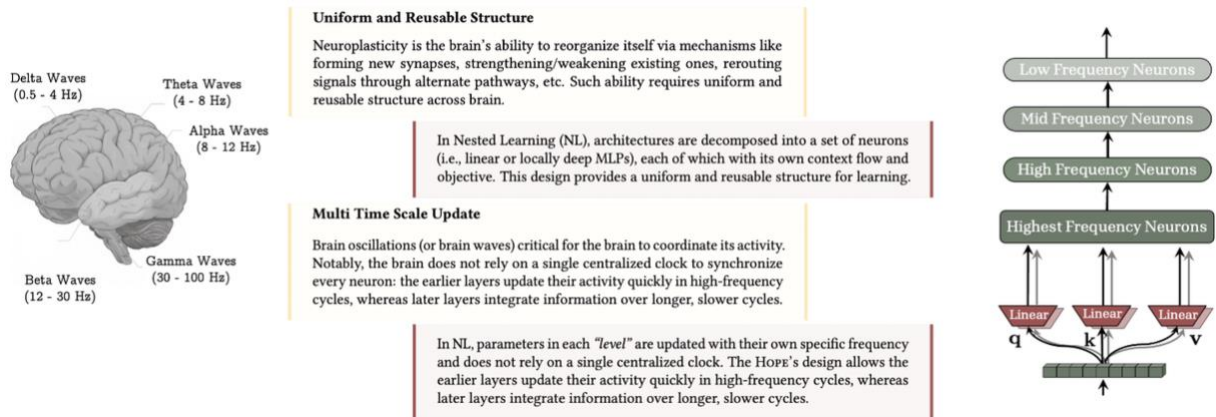
3. **Hệ thống bộ nhớ liên tục (Continuum Memory System):** Chúng tôi trình bày một công thức mới cho hệ thống bộ nhớ, giúp khái quát hóa quan điểm truyền thống về "bộ nhớ dài hạn/ ngắn hạn". Kết hợp mô hình chuỗi tự sửa đổi với hệ thống bộ nhớ liên tục, chúng tôi giới thiệu một mô-đun học tập có tên là HOPE, cho thấy những kết quả đầy hứa hẹn trong các tác vụ mô hình hóa ngôn ngữ, học liên tục và lập luận ngữ cảnh dài.

# Mục lục

Tóm tắt.....	3
Mục lục .....	5
1. Giới thiệu .....	6
2. Học lồng ghép .....	12
3. HOPE: một mô-đun học tập tự tham chiếu (Self-Referential Learning Module) với bộ nhớ liên tục (Continuum Memory) .....	26
4. Thực nghiệm .....	29
Bảng tổng hợp các thuật ngữ chuyên ngành quan trọng.....	30

# 1. Giới thiệu

Phiên bản này của bài báo đã được tóm tắt đáng kể để phù hợp với giới hạn số trang cho bản in chính thức (camera-ready) của hội nghị NeurIPS. Do đó, một số tài liệu, thực nghiệm, thảo luận và phương pháp đã được chuyển xuống phần phụ lục; điều này có thể khiến một số phần trở nên khó theo dõi hoặc gây ra sự thiếu nhất quán. Để tránh tình trạng này, xin vui lòng tham khảo phiên bản trên arXiv [1] của chúng tôi (sẽ có sẵn vào ngày 13 tháng 11).



Hình 1: Cấu trúc đồng nhất và có thể tái sử dụng, cũng như cơ chế cập nhật đa thang thời gian (multi time-scale) trong não bộ, là những thành phần then chốt giúp mở khóa khả năng học tập liên tục ở con người. Học lồng ghép (NL) cho phép cập nhật đa thang thời gian đối với từng thành phần của não bộ, đồng thời chỉ ra rằng các kiến trúc nổi tiếng như Transformers thực chất là các lớp tuyến tính (linear layers) với tần suất cập nhật khác nhau.

Trong nhiều thập kỷ qua, nghiên cứu AI đã tập trung vào việc thiết kế các thuật toán học máy học từ dữ liệu [2–5] hoặc kinh nghiệm [6–8]; thường bằng cách tối ưu hóa một hàm mục tiêu  $L(\theta)$  trên các tham số  $\theta \in \Theta$  bằng các phương pháp dựa trên gradient.

Trong khi các kỹ thuật học máy truyền thống đòi hỏi kỹ thuật chế tác tỉ mỉ và kiến thức chuyên môn sâu để thiết kế các bộ trích xuất đặc trưng (feature extractors), làm hạn chế khả năng xử lý và học trực tiếp từ dữ liệu tự nhiên [9], thì học biểu diễn sâu (deep representation learning) đã cung cấp một giải pháp thay thế hoàn toàn tự động để khám phá các biểu diễn cần thiết cho tác vụ.

Kể từ đó, học sâu đã trở thành một phần không thể tách rời của các mô hình tính toán quy mô lớn, với những thành công mang tính bước ngoặt trong hóa học và sinh học [10], trò chơi [11, 12], thị giác máy tính [13, 14], cũng như hiểu ngôn ngữ tự nhiên và đa phương thức [15–17].

Việc xếp chồng nhiều lớp, như cách thực hiện trong các mô hình học sâu, cung cấp cho mô hình dung lượng lớn hơn, khả năng biểu đạt tốt hơn trong việc đại diện cho các đặc trưng phức tạp, và khối lượng tính toán nội tại nhiều hơn (ví dụ: số lượng #FLOPS) [18–20]. Tất cả những điều này đều là các đặc tính quan trọng và đáng mong đợi đối với các tác vụ tĩnh, đòi hỏi các dự đoán trong cùng phân phối (in-distribution predictions) trên một tập dữ liệu đã được cố định trước.

Tuy nhiên, thiết kế sâu này không phải là giải pháp vạn năng cho mọi thách thức và không thể hỗ trợ khả năng biểu đạt của mô hình ở nhiều khía cạnh, ví dụ:

- (i) Độ sâu tính toán (computational depth) của các mô hình sâu có thể không thay đổi khi thêm nhiều lớp [21, 22], khiến khả năng triển khai các thuật toán phức tạp của chúng không có sự khác biệt so với các phương pháp nông (shallow approaches) truyền thống [23];
- (ii) Dung lượng của một số lớp tham số có thể chỉ cho thấy sự cải thiện không đáng kể khi tăng độ sâu/độ rộng của mô hình [24];
- (iii) Quá trình huấn luyện có thể hội tụ về một giải pháp chưa tối ưu (suboptimal), chủ yếu do việc lựa chọn bộ tối ưu hóa hoặc các siêu tham số của nó chưa tối ưu; và
- (iv) Khả năng thích ứng nhanh với tác vụ mới, học tập liên tục, và/hoặc khái quát hóa đối với dữ liệu ngoài phân phối (out-of-distribution) của mô hình có thể không thay đổi khi xếp chồng thêm nhiều lớp và đòi hỏi những thiết kế kỹ lưỡng hơn.

Trọng tâm của các nỗ lực nhằm vượt qua những thách thức nêu trên và nâng cao năng lực của các mô hình học sâu tập trung vào:

- (1) Phát triển các lớp tham số có khả năng biểu đạt cao hơn (tức là các kiến trúc neuron) [13, 25–28];
- (2) Giới thiệu các hàm mục tiêu (objectives) có thể mô hình hóa các tác vụ tốt hơn [29–32];

- (3) Thiết kế các thuật toán tối ưu hóa hiệu quả/hiệu năng hơn để tìm ra các giải pháp tốt hơn hoặc có khả năng kháng quên (resilience to forgetting) tốt hơn [33–36]; và
- (4) Mở rộng quy mô kích thước mô hình để tăng cường khả năng biểu đạt, khi đã có những lựa chọn "đúng đắn" về kiến trúc, hàm mục tiêu và thuật toán tối ưu hóa [24, 37, 38].

Tổng hợp lại, những tiến bộ này cùng các phát hiện mới về quy luật mở rộng quy mô (scaling patterns) của các mô hình sâu đã thiết lập nên nền móng để xây dựng các Mô hình ngôn ngữ lớn (LLMs).

Sự phát triển của các LLM đánh dấu một cột mốc then chốt trong nghiên cứu học sâu: một sự chuyển dịch hệ hình (paradigm shift) từ các mô hình chuyên biệt cho từng tác vụ sang các hệ thống đa năng hơn với nhiều năng lực nảy sinh (emergent capabilities) là kết quả của việc mở rộng quy mô các kiến trúc "đúng đắn" [38, 39].

Bất chấp mọi thành công và năng lực đáng kinh ngạc trong các tập hợp tác vụ đa dạng [15, 40, 41], các LLM phần lớn đều ở trạng thái tĩnh sau giai đoạn triển khai ban đầu. Điều này có nghĩa là chúng thực hiện thành công các tác vụ đã học trong quá trình tiền huấn luyện hoặc hậu huấn luyện, nhưng không thể liên tục tiếp thu các năng lực mới nằm ngoài ngữ cảnh tức thời của chúng.

Thành phần có khả năng thích ứng duy nhất của các LLM là khả năng học trong ngữ cảnh (in-context learning) – một đặc tính (được biết là nảy sinh) của LLM cho phép thích ứng nhanh với ngữ cảnh và từ đó thực hiện các tác vụ zero-shot hoặc few-shot [38].

Ngoài khả năng học trong ngữ cảnh, các nỗ lực gần đây nhằm khắc phục bản chất tĩnh của các LLM thường gặp phải các vấn đề như: tồn kém về mặt tính toán, đòi hỏi các thành phần bên ngoài, thiếu tính khái quát hóa, và/ hoặc có thể bị quên thảm khốc (catastrophic forgetting) [42–44]. Thực trạng này đã khiến các nhà nghiên cứu đặt câu hỏi: liệu có cần xem xét lại cách thiết kế các mô hình học máy, và liệu có cần một hệ hình học tập mới vượt ra ngoài việc xếp chồng các lớp để khai phá năng lực của LLM trong các thiết lập học tập liên tục hay không.

Các mô hình hiện tại chỉ trải nghiệm hiện tại tức thời. Để so sánh tương đồng và minh họa rõ hơn bản chất tĩnh của các LLM, chúng tôi sử dụng ví dụ về chứng quên thuận chiều

(anterograde amnesia) - một tình trạng thần kinh khiến người bệnh không thể hình thành các ký ức dài hạn mới sau khi khởi phát rối loạn, trong khi các ký ức cũ vẫn còn nguyên vẹn [45].

Tình trạng này giới hạn kiến thức và trải nghiệm của người bệnh trong một cửa sổ ngắn ngủi của hiện tại và quá khứ xa - giai đoạn trước khi khởi phát rối loạn - dẫn đến việc họ liên tục trải nghiệm hiện tại tức thời như thể đó luôn là điều mới mẻ.

Hệ thống xử lý bộ nhớ của các LLM hiện tại cũng gặp phải một mô hình tương tự. Kiến thức của chúng bị giới hạn ở: hoặc là ngữ cảnh tức thời nằm vừa trong cửa sổ ngữ cảnh (context window) của chúng, hoặc là kiến thức trong các lớp MLP nơi lưu trữ quá khứ xa, trước thời điểm "kết thúc quá trình tiền huấn luyện".

Sự so sánh này đã thôi thúc chúng tôi tìm kiếm cảm hứng từ các tài liệu về sinh lý học thần kinh (neurophysiology) và cách não bộ củng cố (consolidate) các ký ức ngắn hạn của nó:

## **1.1. Quan điểm từ não bộ con người và động lực sinh lý thần kinh**

Não bộ con người hoạt động cực kỳ hiệu quả trong việc học tập liên tục (hay còn gọi là quản lý ngữ cảnh hiệu quả - effective context management). Khả năng này thường được cho là nhờ vào tính dẻo thần kinh (neuroplasticity) - năng lực đáng kinh ngạc của não bộ trong việc tự thay đổi để phản hồi lại các trải nghiệm mới, ký ức, quá trình học tập và thậm chí là các tổn thương [46, 47].

Các nghiên cứu gần đây ủng hộ quan điểm rằng sự hình thành trí nhớ dài hạn liên quan đến ít nhất hai quá trình củng cố (consolidation) riêng biệt nhưng bổ sung cho nhau [48–50]:

- (1) Giai đoạn củng cố "trực tuyến" (online) nhanh chóng (còn gọi là củng cố khớp thần kinh - synaptic consolidation) diễn ra ngay lập tức hoặc ngay sau khi học, kể cả khi đang thức. Đây là thời điểm các dấu vết ký ức mới và ban đầu còn mong manh được ổn định và bắt đầu chuyển từ kho lưu trữ ngắn hạn sang dài hạn;
- (2) Quá trình củng cố "ngoại tuyến" (offline) (còn gọi là củng cố hệ thống - systems consolidation) lặp lại việc "phát lại" (replay) các mẫu vừa được mã hóa.

Quá trình này diễn ra trong các đợt sóng gợn nhọn (sharp-wave ripples - SWRs) tại hồi hải mã, phối hợp với các thoi ngủ vỏ não (cortical sleep spindles) và các dao động chậm, giúp tăng cường, tổ chức lại bộ nhớ và hỗ trợ việc chuyển giao thông tin sang các vùng vỏ não [51–53].

Quay trở lại sự so sánh với chứng quên thuận chiều (anterograde amnesia), các bằng chứng chỉ ra rằng tình trạng này có thể ảnh hưởng đến cả hai giai đoạn, nhưng đặc biệt là giai đoạn củng cố trực tuyến. Nguyên nhân chủ yếu là do hồi hải mã (hippocampus) đóng vai trò là "cửa ngõ" để mã hóa các trí nhớ tường thuật (declarative memories) mới, do đó tổn thương tại đây đồng nghĩa với việc thông tin mới sẽ không bao giờ được lưu trữ vào bộ nhớ dài hạn.

Như đã đề cập ở trên, thiết kế của các LLM, và cụ thể hơn là các kiến trúc nền tảng (backbones) dựa trên Transformer, cũng gặp phải tình trạng tương tự sau giai đoạn tiền huấn luyện. Cụ thể, thông tin được cung cấp trong ngữ cảnh không bao giờ tác động đến các tham số bộ nhớ dài hạn (ví dụ: các lớp truyền thẳng - feedforward layers). Vì vậy, mô hình không có khả năng tiếp thu kiến thức hoặc kỹ năng mới, trừ khi thông tin đó vẫn đang được lưu trữ trong bộ nhớ ngắn hạn (ví dụ: cơ chế chú ý - attention).

Vì lẽ đó, mặc dù giai đoạn thứ hai cũng quan trọng không kém, hoặc thậm chí quan trọng hơn đối với việc củng cố ký ức, và sự thiếu vắng nó có thể làm hỏng quá trình cũng như gây mất trí nhớ [54, 55], nhưng trong công trình này, chúng tôi tập trung vào giai đoạn đầu tiên: củng cố bộ nhớ như một quá trình trực tuyến. Chúng tôi cung cấp thêm các thảo luận về quan điểm não bộ con người và mối liên hệ của nó với NL trong phụ lục A.

### Các ký hiệu (Notations):

Chúng tôi đặt  $x \in \mathbb{R}^{N \times d^{in}}$  là đầu vào,  $\mathcal{M}^t$  đại diện cho trạng thái của bộ nhớ/mô hình  $\mathcal{M}$  tại thời điểm  $t$ , trong đó  $K$  là các ma trận Khóa (Keys),  $V$  là các ma trận Giá trị (Values), và  $Q$  là các ma trận Truy vấn (Query). Chúng tôi sử dụng các chữ cái thường in đậm kèm chỉ số dưới  $t$  để chỉ vector tương ứng với đầu vào  $t$  (tức là  $k_t$ ,  $v_t$ , và  $q_t$ ). Chúng tôi quy ước thêm rằng phân phối của bất kỳ thực thể  $f$  nào được ký hiệu là  $p(f)$ .

Xuyên suốt bài báo, chúng tôi sử dụng các MLP đơn giản với  $L_M \geq 1$  lớp và kết nối phần dư (residual connection) làm kiến trúc của mô-đun bộ nhớ  $\mathcal{M}(\cdot)$ . Khi cần thiết, chúng tôi tham số hóa mô-đun bộ nhớ bằng  $\theta_M \supseteq \{W_1, W_2, \dots, W_{L_M}\}$ , tập hợp này ít nhất bao gồm các tham số

của các lớp tuyến tính trong MLP. Chúng tôi sử dụng chỉ số trên đặt trong ngoặc đơn để chỉ các tham số ở các cấp độ học lồng ghép khác nhau (tần suất cập nhật khác nhau - different update frequency): ví dụ,  $W^{(\ell)}$ .

Ghi chú kỹ thuật:

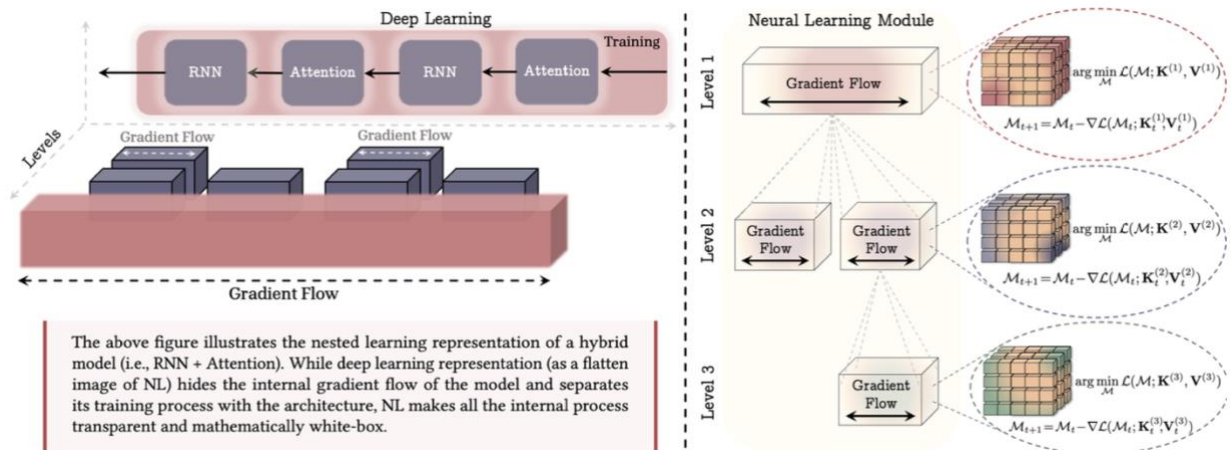
- Đoạn này đã bổ sung định nghĩa về phân phối xác suất  $p(f)$  và làm rõ cách tham số hóa  $\theta_M$  so với đoạn văn bị thiếu trước đó.
- Thuật ngữ residual connection thường được dịch là "kết nối phần dư" hoặc "kết nối tắt" trong các tài liệu AI tiếng Việt.

## 2. Học lồng ghép

Phần này thảo luận về các động lực, định nghĩa hình thức và các hàm ý tổng quan ở cấp độ cao của học lồng ghép (NL).

Chúng tôi bắt đầu bằng việc công thức hóa bộ nhớ liên kết (associative memory), sau đó thông qua các ví dụ từng bước, chúng tôi xây dựng trực giác về sự phân rã kiến trúc (architecture decomposition) và mối liên hệ của nó với việc mô hình hóa một mạng nơ-ron như một hệ thống tích hợp các bài toán tối ưu hóa.

Mục tiêu của chúng tôi trước tiên là chỉ ra cách các phương pháp và khái niệm hiện có trong học sâu nằm trong hệ hình NL như thế nào. Sau đó, chúng tôi trình bày các công thức mới vượt ra ngoài các phương pháp truyền thống và/hoặc cung cấp những hiểu biết sâu sắc (insights) về cách cải thiện các thuật toán và thiết kế hiện có.



Hình 2: Hệ hình học lồng ghép (Nested Learning Paradigm) biểu diễn một mô hình học máy và quy trình huấn luyện của nó như một tập hợp các bài toán tối ưu hóa lồng nhau.

- **(Trái) Một ví dụ về kiến trúc lai:** Trong khi quan điểm học sâu, vốn là hình ảnh được "làm phẳng" của NL, không cung cấp cái nhìn sâu sắc về độ sâu tính toán trong các khối, thì NL biểu diễn một cách minh bạch tất cả các luồng gradient bên trong.
- **(Phải) Một mô-đun học nơ-ron (A Neural Learning Module):** Một mô hình tính toán học cách nén luồng ngữ cảnh (context flow) của chính nó. Ví dụ, cấp độ đầu tiên tương

ứng với vòng lặp huấn luyện ngoài cùng nhất của mô hình, thường được gọi là bước "tiền huấn luyện" (pre-training).

## 2.1. Bộ nhớ liên kết

Bộ nhớ liên kết (Associative Memory) - khả năng hình thành và truy xuất các mối liên hệ giữa các sự kiện - là một quá trình tư duy cơ bản và là một thành phần không thể tách rời của quá trình học tập ở con người [56]. Trong các tài liệu nghiên cứu, các khái niệm về sự ghi nhớ và sự học tập thường được sử dụng hoán đổi cho nhau; tuy nhiên, trong văn liệu tâm lý học thần kinh, hai khái niệm này được phân biệt một cách rõ ràng. Cụ thể hơn, dựa theo các tài liệu tâm lý học thần kinh [57], chúng tôi xây dựng hệ thống thuật ngữ của mình dựa trên các định nghĩa sau đây về bộ nhớ và học tập:

### *Học tập (learnin) so với ghi nhớ (memorization)*

---

*Bộ nhớ là một sự cập nhật thần kinh (neural update) do một đầu vào gây ra, còn học tập là quá trình để thu nhận bộ nhớ hiệu quả và hữu ích.*

---

Trong công trình này, mục tiêu của chúng tôi trước hết là chỉ ra rằng tất cả các thành phần của một mô hình chuỗi tính toán, bao gồm cả các bộ tối ưu hóa và các mạng nơ-ron, đều là các hệ thống bộ nhớ liên kết giúp nén luồng ngữ cảnh của chính chúng. Nhìn chung, bộ nhớ liên kết là một toán tử ánh xạ một tập hợp các khóa tới một tập hợp các giá trị. Chúng tôi áp dụng định nghĩa tổng quát về bộ nhớ liên kết của Behrouz và cộng sự [58]:

**Định nghĩa 1 (Bộ nhớ liên kết - Associative Memory):** Cho một tập hợp các khóa  $\mathcal{K} \subseteq \mathbb{R}^{d_k}$  và các giá trị  $\mathcal{V} \subseteq \mathbb{R}^{d_v}$ , bộ nhớ liên kết là một toán tử  $\mathcal{M} : \mathcal{K} \rightarrow \mathcal{V}$  thực hiện ánh xạ giữa hai tập hợp khóa  $\mathcal{K}$  và giá trị  $\mathcal{V}$ . Để học được ánh xạ này từ dữ liệu, một hàm mục tiêu  $\tilde{\mathcal{L}}(\cdot; \cdot)$  được dùng để đo lường chất lượng của ánh xạ và  $\mathcal{M}$  có thể được định nghĩa như sau:

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

Trong khi bản thân toán tử là một bộ nhớ và việc ánh xạ đóng vai trò như một quá trình ghi nhớ (tức là, ghi nhớ các mối liên hệ giữa các sự kiện trong ngữ cảnh), thì việc thu nhận một

toán tử hiệu quả như vậy dựa trên dữ liệu lại là một quá trình học tập. Cần lưu ý rằng, ở đây, các khóa và giá trị có thể là bất kỳ sự kiện tùy ý nào mà bộ nhớ hướng tới việc ánh xạ, và không chỉ giới hạn ở các token. Ở phần sau của mục này, chúng tôi sẽ thảo luận về việc với một luồng ngữ cảnh cho trước, các khóa và giá trị có thể là các token, các gradient, các chuỗi con,... Hơn nữa, mặc dù thuật ngữ bộ nhớ liên kết phổ biến hơn trong các tài liệu khoa học thần kinh và tâm lý học thần kinh, nhưng công thức nêu trên cũng có mối liên hệ chặt chẽ với nén dữ liệu và biểu diễn chiều thấp. Nghĩa là, ta có thể diễn giải quá trình tối ưu hóa trong Phương trình 1 như là quá trình huấn luyện của một mạng  $\mathcal{M}(\cdot)$  nhằm mục đích nén các ánh xạ vào trong các tham số của nó, và từ đó biểu diễn chúng trong một không gian có số chiều thấp hơn.

Trong mô hình hóa chuỗi, nơi các khóa và giá trị là các token đầu vào (ví dụ: văn bản đã được token hóa), việc lựa chọn hàm mục tiêu và quá trình tối ưu hóa để giải quyết Phương trình 1 có thể dẫn đến các kiến trúc mô hình hóa chuỗi khác biệt (xem [59] và [58]), chẳng hạn như cơ chế chú ý softmax toàn cục/ cục bộ [27], hoặc các mô hình hồi quy hiện đại khác [28, 60, 61]. Cách công thức hóa đơn giản này về các mô hình chuỗi mang lại cho chúng ta sự hiểu biết tốt hơn về quy trình nội tại của chúng, đồng thời cung cấp một công cụ để so sánh đơn giản năng lực mô hình hóa của chúng dựa trên hàm mục tiêu và quá trình tối ưu hóa. Trong phần tiếp theo, thông qua các ví dụ từng bước, chúng tôi sẽ thảo luận về việc công thức này có thể được áp dụng như thế nào cho tất cả các thành phần của một kiến trúc nơ-ron (bao gồm cả quá trình tối ưu hóa của nó trong giai đoạn tiền huấn luyện). Trên thực tế, một mô hình chính là một hệ thống tích hợp của các bộ nhớ đa cấp, lồng ghép, và (hoặc) song song, trong đó mỗi bộ nhớ có "luồng ngữ cảnh" riêng của nó.

**Một ví dụ đơn giản về huấn luyện MLP:** Chúng tôi bắt đầu với một ví dụ đơn giản, trong đó chúng tôi nhằm mục đích huấn luyện một MLP 1 lớp (được tham số hóa bởi  $W$ ) cho tác vụ  $\mathcal{T}$  và trên tập dữ liệu  $\mathcal{D}_{train} = \{x_1, \dots, x_{|\mathcal{D}_{train}|}\}$  bằng việc tối ưu hóa hàm mục tiêu  $\mathcal{L}(\cdot; \cdot)$  với phương pháp giảm gradient (gradient descent). Trong trường hợp này, quá trình huấn luyện tương đương với bài toán tối ưu hóa sau đây:

$$W^* = \arg \min_W \mathcal{L}(W; \mathcal{D}_{train})$$

Việc tối ưu hóa bài toán này bằng phương pháp giảm gradient dẫn đến một quy tắc cập nhật trọng số tương đương với:

$$W_{t+1} = W_t - \eta_{t+1} \nabla_{W_t} \mathcal{L}(W_t; x_{t+1})$$

$$W_{t+1} = W_t - \eta_{t+1} \nabla_{y_{t+1}} \mathcal{L}(W_t; x_{t+1}) \otimes x_{t+1}$$

Trong đó  $x_{t+1} \sim \mathcal{D}_{train}$

Trong đó  $y_{t+1} = W_{x_{t+1}}$  là đầu ra của mô hình đối với đầu vào  $x_{t+1}$ . Dựa trên công thức này, ta có thể đặt  $u_{t+1} = \nabla_{y_{t+1}} \mathcal{L}(W_t; x_{t+1})$  và diễn đạt lại quá trình lan truyền ngược (backpropagation) như là lời giải cho một bài toán tối ưu hóa nhằm tìm kiếm một bộ nhớ liên kết tối ưu, thực hiện ánh xạ các điểm dữ liệu đầu vào  $\mathcal{D}_{train} = \{x_t\}_{t=1}^{|\mathcal{D}_{train}|}$  tới các giá trị  $u_{t+1} = \nabla_{y_{t+1}} \mathcal{L}(W_t; x_{t+1})$  tương ứng của chúng. Cụ thể, chúng tôi đặt  $\mathcal{M}(\cdot) = W_t \cdot$  là tham số hóa của bộ nhớ, và sử dụng độ tương đồng tích vô hướng (dot-product similarity) để đo lường chất lượng ánh xạ của  $W_t$  giữa  $x_{t+1}$  và  $\nabla_{y_{t+1}} \mathcal{L}(W_t; x_{t+1})$ :

$$W_{t+1} = \arg \min_W \langle W_{x_{t+1}}, u_{t+1} \rangle + \frac{1}{2\eta_{t+1}} \|W - W_t\|_2^2$$

$$W_{t+1} = \arg \min_W \langle W_{x_t}, \nabla_{y_{t+1}} \mathcal{L}(W_t; x_{t+1}) \rangle + \frac{1}{2\eta_{t+1}} \|W - W_t\|_2^2$$

Trong công thức trên,  $u_{t+1} = \nabla_{y_{t+1}} \mathcal{L}(W_t; x_{t+1})$  có thể được diễn giải như một tín hiệu bất ngờ cục bộ (Local Surprise Signal - LSS) trong không gian biểu diễn, giúp định lượng sự sai lệch giữa đầu ra hiện tại và cấu trúc mà hàm mục tiêu  $\mathcal{L}(\cdot; \cdot)$  áp đặt. Do đó, công thức này diễn giải giai đoạn huấn luyện của mô hình như một quá trình thu nhận bộ nhớ hiệu quả, thực hiện ánh xạ các mẫu dữ liệu tới tín hiệu bất ngờ cục bộ (LSS) của chúng trong không gian biểu diễn – được định nghĩa là sự sai lệch giữa đầu ra hiện tại và cấu trúc được áp đặt bởi hàm mục tiêu  $\mathcal{L}(\cdot; \cdot)$ .

Theo đó, trong ví dụ này, mô hình của chúng ta có một luồng gradient đơn nhất trên các mẫu dữ liệu, luồng này chỉ hoạt động trên tập dữ liệu  $\mathcal{D}_{train} = \{x_1, \dots, x_{|\mathcal{D}_{train}|}\}$  và sẽ bị đóng băng đối với bất kỳ mẫu dữ liệu nào sau đó (hay còn gọi là thời gian suy luận hoặc kiểm thử).

Tiếp theo, trong ví dụ trên, chúng tôi thay thế thuật toán giảm gradient bằng biến thể dựa trên đà (momentum-based) được cải tiến của nó, dẫn đến quy tắc cập nhật như sau:

$$W_{t+1} = W_t - m_{t+1}$$

$$m_{t+1} = m_t - \eta_{t+1} \nabla_{W_t} \mathcal{L}(W_t; x_{t+1}) = m_t - \nabla_{y_{t+1}} \mathcal{L}(W_t; x_{t+1}) \otimes x_{t+1}$$

Trong phương trình 8, xét trạng thái trước đó của phương trình 7 (tại thời điểm  $t$ ), giá trị của  $\nabla_{W_t} \mathcal{L}(W_t; x_{t+1})$  hoặc tương tự là  $\nabla_{y_{t+1}} \mathcal{L}(W_t; x_{t+1})$  độc lập với phân đệ quy trong phương trình 8, và do đó có thể được tính toán trước.

Theo đó, chúng tôi đặt  $u_{t+1} = \nabla_{W_t} \mathcal{L}(W_t; x_{t+1})$ , và như vậy Phương trình 8 có thể được viết lại thành:

$$W_{t+1} = W_t - m_{t+1}$$

$$m_{t+1} = \arg \min_m - \langle m, \nabla_{W_t} \mathcal{L}(W_t; x_{t+1}) \rangle + \eta_{t+1} \|m - m_t\|_2^2$$

$$m_{t+1} = \arg \min_m - \langle m_{x_{t+1}}, \nabla_{y_{t+1}} \mathcal{L}(W_t; x_{t+1}) \rangle + \eta_{t+1} \|m - m_t\|_2^2$$

Trong đó bài toán tối ưu hóa tại phương trình 10 tương đương với một bước giảm gradient với tốc độ học thích ứng  $\eta_{t+1}$ .

Dựa trên các công thức này, ta có thể diễn giải thành phần momentum theo một trong hai cách:

1. Là một bộ nhớ liên kết không có khóa (key-less) giúp nén các gradient vào trong các tham số của nó; hoặc
2. Là một bộ nhớ liên kết học cách ánh xạ các điểm dữ liệu tới giá trị LSS tương ứng của chúng.

Thật thú vị, công thức này hé lộ rằng phương pháp giảm gradient kết hợp momentum thực chất là một quá trình tối ưu hóa hai cấp độ, trong đó bộ nhớ được tối ưu hóa bởi thuật toán giảm gradient đơn giản. Quá trình này có mối liên hệ chặt chẽ với các chương trình trọng số nhanh (Fast Weight Programs - FWPs) [62], trong đó quá trình cập nhật trọng số (tức Phương trình

9) đóng vai trò là mạng chậm (slow network), có trọng số momentum được sinh ra bởi một mạng nhanh (fast network - tức Phương trình 10).

Tổng kết các ví dụ trên, chúng tôi nhận thấy rằng quá trình huấn luyện một MLP 1 lớp với:

- (1) Giảm gradient: Là một bộ nhớ liên kết 1 cấp độ, học cách ánh xạ các điểm dữ liệu tới giá trị LSS tương ứng của chúng; và
- (2) Giảm gradient kết hợp momentum: Là một bộ nhớ liên kết (hoặc quá trình tối ưu hóa) 2 cấp độ, trong đó cấp bên trong học cách lưu trữ các giá trị gradient vào các tham số của nó, và sau đó cấp bên ngoài cập nhật trọng số chậm (tức  $W_t$ ) bằng giá trị từ bộ nhớ cấp bên trong.

Mặc dù đây là những ví dụ đơn giản nhất xét về cả kiến trúc lẫn thuật toán tối ưu hóa, nhưng một câu hỏi được đặt ra là liệu có thể đưa ra kết luận tương tự trong các thiết lập phức tạp hơn hay không.

**Một ví dụ về phân rã kiến trúc (Architectural Decomposition):** Trong ví dụ tiếp theo, chúng tôi thay thế mô-đun MLP bằng một cơ chế chú ý tuyến tính (linear attention) [60]. Cụ thể, chúng tôi hướng tới việc huấn luyện một lớp chú ý tuyến tính đơn cho tác vụ  $\mathcal{T}$  trên một chuỗi dữ liệu  $\mathcal{D}_{train} = \{x_1, \dots, x_{|\mathcal{D}_{train}|}\}$  bằng cách tối ưu hóa hàm mục tiêu  $\mathcal{L}$  với phương pháp giảm gradient. Hãy cùng nhắc lại công thức của cơ chế chú ý tuyến tính chưa chuẩn hóa:

$$k_t = x_t W_k, v_t = x_t W_v, q_t = x_t W_q$$

$$\mathcal{M}_t = \mathcal{M}_{t-1} + v_t k_t^\top$$

$$y_t = \mathcal{M}_t q_t$$

Như đã thảo luận trong các nghiên cứu trước đây [58, 59], hệ thức truy hồi trong phương trình 13 có thể được diễn đạt lại như là quá trình tối ưu hóa của một bộ nhớ liên kết giá trị ma trận  $\mathcal{M}(\cdot)$ , trong đó, nó nhằm mục đích nén các ánh xạ của khóa và giá trị vào trong các tham số của nó.

Cụ thể hơn, xét định nghĩa 1, nếu chúng ta đặt  $\tilde{\mathcal{L}}(\mathcal{M}_{t-1}; k_t, v_t) := -\langle \mathcal{M}_{t-1}; k_t, v_t \rangle$  và tiến hành tối ưu hóa bộ nhớ bằng phương pháp giảm gradient, thì quy tắc cập nhật bộ nhớ sẽ là: (Lưu ý rằng  $\nabla \tilde{\mathcal{L}}(\mathcal{M}_{t-1}; k_t, v_t) = v_t k_t^\top$  và chúng ta đặt tốc độ học  $\eta_t = 1$ ).

$$\mathcal{M}_{t+1} = \arg \min_{\mathcal{M}} \langle \mathcal{M} k_{t+1}, v_{t+1} \rangle + \|\mathcal{M} - \mathcal{M}_t\|_2^2$$

Với giảm gradient (với tốc độ học  $\eta_t = 1$ )

$$\Rightarrow \mathcal{M}_{t+1} = \mathcal{M}_t - \nabla \tilde{\mathcal{L}}(\mathcal{M}_t; k_{t+1}, v_{t+1}) = \mathcal{M}_t + v_{t+1} k_{t+1}^\top$$

Điều này tương đương với quy tắc cập nhật của cơ chế chú ý tuyến tính chưa chuẩn hóa trong phương trình 13. Ngoài ra, hãy lưu ý rằng như chúng ta đã quan sát trong ví dụ đầu tiên, việc huấn luyện một lớp tuyến tính bằng phương pháp giảm gradient là một bài toán tối ưu hóa 1 lớp của bộ nhớ liên kết (xem phương trình 3). Do đó, quá trình huấn luyện/cập nhật tổng quát của các lớp chiếu (projection layers) (tức là  $W_k, W_v$ , và  $W_q$ .) bản thân nó cũng là một quá trình tối ưu hóa bộ nhớ liên kết.

Theo đó, thiết lập này - tức là việc huấn luyện một cơ chế chú ý tuyến tính bằng giảm gradient – có thể được xem như một **quá trình tối ưu hóa hai cấp độ**:

1. **Vòng lặp bên ngoài** (còn gọi là quá trình huấn luyện) tối ưu hóa các lớp chiếu bằng giảm gradient.
2. **Vòng lặp bên trong** tối ưu hóa bộ nhớ bên trong của  $\mathcal{M}_t$  bằng giảm gradient.

Cần lưu ý rằng, như đã thảo luận ở trên, ở đây chúng ta có hai bộ nhớ liên kết, và do đó mỗi bộ nhớ có quá trình tối ưu hóa và luồng gradient riêng.

- Nghĩa là, trong việc tối ưu hóa các tham số **cấp bên ngoài** của  $W_k, W_v$ , và  $W_q$  không có gradient nào đối với tham số  $\mathcal{M}(\cdot)$  và do đó không có sự lan truyền ngược (backpropagation) qua nó.
- Tương tự, ở **cấp bên trong**, không có sự lan truyền ngược qua các lớp chiếu và chúng được coi là bị đóng băng (frozen).

Hơn nữa, đáng chú ý là trong ví dụ này, công thức trên cũng liên quan chặt chẽ đến quan điểm FWP (Fast Weight Programs) về các cơ chế chú ý tuyến tính [63], trong đó các phép chiếu được coi là trọng số chậm (slow weights), và việc cập nhật bộ nhớ trong Phương trình 13 là quy tắc cập nhật trọng số nhanh (fast weight).

**Phân rã kiến trúc (Architectural Decomposition) với nhiều cấp độ hơn:** Trong cả hai ví dụ trên, chúng tôi đã thảo luận về các trường hợp đơn giản, nơi chúng có thể được diễn giải thành các quy trình tối ưu hóa 2 cấp độ, điều này cũng trùng khớp với các diễn giải theo hướng FWPs của chúng.

Tuy nhiên, trong thực tế, chúng ta cần sử dụng các thuật toán tối ưu hóa mạnh mẽ hơn để huấn luyện mô hình, và/hoặc sử dụng quy tắc cập nhật truy hồi (recurrent update rule) mạnh mẽ hơn cho bộ nhớ.

Lấy một ví dụ đơn giản, giả sử chúng ta sử dụng phương pháp giảm gradient kết hợp momentum để huấn luyện một mô hình chú ý tuyến tính (linear attention model). Trong các ví dụ trên, chúng tôi đã chỉ ra cách thành phần chú ý tuyến tính có thể được phân rã thành hai bài toán tối ưu hóa lồng nhau. Tương tự, ở đây mô hình có thể được biểu diễn như một bài toán tối ưu hóa 2 cấp độ, trong đó:

1. **Cấp độ bên trong** tối ưu hóa bộ nhớ để nén ngữ cảnh bằng phương pháp giảm gradient (xem phương trình 15), và
2. **Cấp độ bên ngoài** tối ưu hóa các lớp chiếu bằng phương pháp giảm gradient kết hợp momentum.

Thú vị thay, từ ví dụ đầu tiên, chúng ta biết rằng bản thân thuật toán "giảm gradient kết hợp momentum" thực chất là một bài toán tối ưu hóa 2 cấp độ, trong đó thành phần momentum chính là một bộ nhớ liên kết giúp nén các gradient trong quá khứ vào các tham số của nó.

## 2.2. Các bài toán tối ưu hóa lồng ghép

Trong phần trước, chúng tôi đã đưa ra các ví dụ để minh họa cách ta có thể phân rã một mô hình học máy thành một tập hợp các bài toán tối ưu hóa lồng ghép hoặc đa cấp độ. Tiếp theo, trước tiên chúng tôi hướng tới việc trình bày một công thức hình thức cho các bài toán học lồng

ghép, và sau đó định nghĩa **mô-đun học nơ-ron (Neural Learning Module)** - một hệ thống tính toán tích hợp có khả năng học từ dữ liệu.

Như chúng ta đã quan sát trong phần trước, mặc dù chúng ta đã phân rã mô hình thành một tập hợp các quy trình tối ưu hóa, nhưng vẫn chưa rõ liệu ta có thể xác định một sự phân cấp (hoặc trật tự) trên các bài toán này, và biểu diễn mô hình theo định dạng này một cách duy nhất hay không.

Lấy cảm hứng từ sự phân cấp của sóng não vốn chỉ báo tần suất xử lý thông tin của từng bộ phận (đã thảo luận trong phần 1), chúng tôi sử dụng tốc độ cập nhật của từng bài toán tối ưu hóa để sắp xếp thứ tự các thành phần thành nhiều cấp độ. Để đạt được mục tiêu này, chúng tôi quy ước một bước cập nhật trên một điểm dữ liệu là đơn vị thời gian, và định nghĩa tần suất cập nhật của từng thành phần như sau:

**Định nghĩa 2 (tần suất cập nhật - Update Frequency):** Đối với bất kỳ thành phần  $A$  nào, dù là thành phần tham số (ví dụ: các trọng số có thể học hoặc thành phần momentum trong phương pháp giảm gradient kết hợp momentum) hay thành phần phi tham số (ví dụ: khối chú ý), chúng tôi định nghĩa tần suất của nó, ký hiệu là  $f_A$ , là số lần cập nhật trên mỗi đơn vị thời gian.

Dựa trên tần suất cập nhật nêu trên, chúng ta có thể sắp xếp thứ tự các thành phần của một thuật toán học máy dựa trên toán tử  $(\cdot > \cdot)$ . Chúng tôi coi  $A$  là nhanh hơn  $B$  và ký hiệu  $A > B$  nếu:

- $f_A > f_B$ , hoặc
- $f_A = f_B$  nhưng việc tính toán trạng thái của  $B$  tại thời điểm  $t$  yêu cầu phải tính toán trạng thái của  $A$  tại thời điểm  $t$  trước.

Trong định nghĩa này, khi  $A \not> B$  và  $B \not> A$ , chúng tôi đặt  $A \stackrel{f}{=} B$ , điều này chỉ ra rằng  $A$  và  $B$  có cùng tần suất cập nhật, nhưng quá trình tính toán của chúng độc lập với nhau (Sau này, chúng tôi sẽ cung cấp một ví dụ về trường hợp này trong bộ tối ưu hóa AdamW).

Dựa trên toán tử trên, chúng tôi sắp xếp các thành phần thành một tập hợp có thứ tự các "cấp độ" (levels), trong đó:

1. Các thành phần trong cùng một cấp độ có cùng tần suất cập nhật, và
2. Cấp độ càng cao thì tần suất cập nhật càng thấp.

Đáng chú ý, dựa trên định nghĩa trên, mỗi thành phần đều có bài toán tối ưu hóa riêng và do đó có ngữ cảnh riêng. Trong khi chúng ta tối ưu hóa mục tiêu nội tại của thành phần bằng các bộ tối ưu hóa dựa trên gradient, thì phát biểu trên tương đương với việc có **luồng gradient riêng biệt (exclusive gradient flow)** cho mỗi thành phần trong mô hình. Tuy nhiên, trong trường hợp tổng quát, ta có thể sử dụng giải pháp phi tham số (như chúng tôi sẽ thảo luận sau về cơ chế chú ý).

**Mô-đun học nơ-ron:** Dựa trên định nghĩa nêu trên về các bài toán học lồng ghép, chúng tôi định nghĩa mô-đun học nơ-ron là một phương thức biểu diễn mới cho các mô hình học máy. Phương thức này mô tả mô hình như một hệ thống các thành phần liên kết với nhau, trong đó mỗi thành phần sở hữu luồng gradient riêng của nó. Cần lưu ý rằng, mang tính trực giao (orthogonal) với học sâu (deep learning), học lồng ghép cho phép chúng ta định nghĩa các mô hình học nơ-ron với nhiều cấp độ hơn, dẫn đến một kiến trúc có khả năng biểu đạt cao hơn.

---

*Học lồng ghép cho phép các mô hình tính toán được cấu thành từ nhiều cấp độ (đa lớp) có thể học và xử lý dữ liệu với các mức độ trừu tượng và thang thời gian khác nhau.*

---

Tiếp theo, chúng tôi khảo sát các bộ tối ưu hóa và các kiến trúc học sâu nổi tiếng từ góc độ học lồng ghép, đồng thời cung cấp các ví dụ về cách thức NL có thể giúp tăng cường các thành phần này.

## 2.3. Các bộ tối ưu hóa dưới dạng mô-đun học tập

Trong phần này, chúng tôi bắt đầu bằng việc tìm hiểu cách các bộ tối ưu hóa nổi tiếng cùng các biến thể của chúng thực chất là những trường hợp đặc biệt của học lồng ghép (nested learning). Hãy nhắc lại phương pháp giảm gradient kết hợp momentum:

$$W_{i+1} = W_i + m_{i+1}$$

$$m_{i+1} = \alpha_{i+1}m_i - \eta_t \nabla \mathcal{L}(W_i; x_i)$$

trong đó ma trận (hoặc vector)  $m_i$  là momentum tại trạng thái  $i$ , còn  $\alpha_i$  và  $\eta_i$  lần lượt là các hệ số momentum và tốc độ học thích ứng. Giả sử  $\alpha_{i+1} = 1$ , thành phần momentum có thể được xem như kết quả của việc tối ưu hóa hàm mục tiêu sau đây bằng phương pháp giảm gradient:

$$\min_m \langle m \nabla \mathcal{L}(W_i; x_i)^\top, I \rangle$$

Cách diễn giải này cho thấy momentum thực sự có thể được coi là một mô-đun siêu bộ nhớ (meta memory module) học cách ghi nhớ các gradient của hàm mục tiêu vào trong các tham số của nó. Phát triển từ trực giác này, trong...

Cách diễn giải này cho thấy momentum thực sự có thể được xem như một **mô-đun siêu bộ nhớ (meta memory module)** học cách ghi nhớ các gradient của hàm mục tiêu vào trong các tham số của nó. Phát triển từ trực giác này, trong phần C.4, chúng tôi chứng minh rằng Adam với một sửa đổi nhỏ chính là **bộ nhớ liên kết tối ưu** cho các gradient của mô hình. Tiếp theo, chúng tôi chỉ ra cách góc nhìn này có thể dẫn tới việc thiết kế các bộ tối ưu hóa có khả năng biểu đạt cao hơn:

### Mở rộng: Sự liên kết có khả năng biểu đạt cao hơn

Như đã thảo luận trước đó, momentum là một bộ nhớ liên kết phi giá trị (value-less) và do đó có khả năng biểu đạt hạn chế. Để giải quyết vấn đề này, tuân theo định nghĩa ban đầu của bộ nhớ liên kết (tức là, ánh xạ các khóa tới các giá trị), chúng tôi đặt tham số giá trị  $v_i = P_i$ , và do đó momentum hướng tới việc cực tiểu hóa biểu thức sau:

$$\min_m \langle m \nabla \mathcal{L}(W_i; x_i)^\top, P_i \rangle$$

Sử dụng phương pháp giảm gradient, dẫn đến quy tắc cập nhật:

$$W_{i+1} = W_i + m_{i+1}$$

$$m_{i+1} = \alpha_{i+1}m_i - \eta_t P_i \nabla \mathcal{L}(W_i; x_i)$$

Công thức này tương đương với việc sử dụng tiền điều kiện hóa (preconditioning) cho Momentum GD.

Thực tế, tiền điều kiện hóa có nghĩa là thành phần momentum là một bộ nhớ liên kết học cách nén các ánh xạ giữa  $P_i$  và thành phần gradient  $\nabla\mathcal{L}(W_i; x_i)$ .

Mặc dù bất kỳ lựa chọn tiền điều kiện hóa hợp lý nào (ví dụ: các đặc trưng ngẫu nhiên) đều có thể cải thiện khả năng biểu đạt của phiên bản GD kết hợp momentum ban đầu – vốn thực chất là một bộ nhớ phi giá trị (tức là, ánh xạ tất cả các gradient tới một giá trị duy nhất) – nhưng góc nhìn trên mang lại trực giác rõ ràng hơn về việc loại tiền điều kiện hóa nào là hữu ích hơn. Cụ thể, momentum hoạt động như một bộ nhớ nhằm mục đích ánh xạ các gradient tới các giá trị tương ứng của chúng, và do đó một hàm của các gradient (ví dụ: thông tin về Hessian) có thể cung cấp cho bộ nhớ các ánh xạ ý nghĩa hơn.

### **Mở rộng: Các hàm mục tiêu có khả năng biểu đạt cao hơn**

Như đã được thảo luận bởi Behrouz và cộng sự [58], việc tối ưu hóa một hàm mục tiêu bên trong dựa trên độ tương đồng tích vô hướng sẽ dẫn đến quy tắc cập nhật dạng Hebbian (Hebbian-like update rule). Quy tắc này có thể khiến bộ nhớ trở nên kém hiệu quả hơn.

Một sự mở rộng tự nhiên của hàm mục tiêu nội tại này là sử dụng hàm mất mát hồi quy  $\ell_2(\cdot)$  (để đo lường độ khớp của ánh xạ khóa - giá trị (key – value) tương ứng) và cực tiểu hóa hàm mất mát  $\|m\nabla\mathcal{L}(W_i; x_i)^\top - P_i\|_2^2$ , dẫn đến quy tắc cập nhật sau:

$$W_{i+1} = W_i + m_{i+1}$$

$$m_{i+1} = (\alpha_{i+1}I - \nabla\mathcal{L}(W_i; x_i)^\top \nabla\mathcal{L}(W_i; x_i))m_i - \eta_t P_i \nabla\mathcal{L}(W_i; x_i)$$

Bản cập nhật này dựa trên quy tắc delta (delta-rule) [64]. Nhờ đó, nó cho phép bộ nhớ (momentum) quản lý dung lượng giới hạn của mình tốt hơn và ghi nhớ chuỗi các gradient trong quá khứ một cách hiệu quả hơn.

### **Mở rộng: Bộ nhớ có khả năng biểu đạt cao hơn**

Như đã thảo luận trước đó, momentum có thể được xem như một mô hình siêu bộ nhớ sử dụng một lớp tuyến tính (tức là, giá trị ma trận) để nén các giá trị gradient trong quá khứ. Do bản chất tuyến tính của momentum, mục tiêu nội tại của nó chỉ có thể học được các hàm tuyến tính của các gradient quá khứ. Để tăng dung lượng học tập của mô-đun này, một giải pháp thay thế là sử dụng các mô-đun học tập bền vững (persistent learning modules) mạnh mẽ khác: cụ thể là, thay thế bộ nhớ giá trị ma trận tuyến tính của momentum bằng một mạng MLP. Nhờ đó, momentum, với vai trò là bộ nhớ lưu trữ các gradient quá khứ, sẽ có dung lượng lớn hơn để nắm bắt các động lực cơ bản (underlying dynamics) của các gradient. Để đạt được mục đích này, chúng tôi mở rộng công thức trong phương trình 17 như sau:

$$W_{i+1} = W_i + m_{i+1}(u_i) \text{ và } m_{i+1} = \alpha_{i+1}m_i - \eta_t \nabla \mathcal{L}^{(2)}(m_i; u_i, I)$$

Trong đó  $u_i = \nabla \mathcal{L}(W_i; x_i)$  và  $\nabla \mathcal{L}^{(2)}(\cdot)$  là hàm mục tiêu nội tại của momentum (ví dụ: độ tương đồng tích vô hướng  $\langle m(u_i^\top), 1 \rangle$ ). Chúng tôi gọi biến thể này là **giảm gradient momentum sâu (Deep Momentum Gradient Descent - DMGD)**.

### Mở rộng: Các đầu ra phi tuyến

Phát triển từ góc nhìn trên, khi chúng ta xem momentum như một kiến trúc nơ-ron, một kỹ thuật phổ biến để nâng cao năng lực biểu diễn của mô-đun bộ nhớ momentum là sử dụng tính phi tuyến (non-linearity) trên đầu ra của nó [28, 65]. Cụ thể, chúng tôi viết lại phương trình 23 như sau:

$$W_{i+1} = W_i + \sigma(m_{i+1}(u_i)) \text{ và } m_{i+1} = \alpha_{i+1}m_i - \eta_t \nabla \mathcal{L}^{(2)}(m_i; u_i, I)$$

Trong đó  $\sigma(\cdot)$  là một hàm phi tuyến tùy ý.

Lấy một ví dụ, nếu chúng ta đặt  $\sigma(\cdot) = \text{Newton-Schulz}(\cdot)$ , trong đó  $\text{Newton-Schulz}(\cdot)$  là phương pháp lặp Newton-Schulz [66], và  $m(\cdot)$  là một lớp tuyến tính; thì bộ tối ưu hóa thu được sẽ tương đương với **bộ tối ưu hóa Muon** [34].

### Vượt ra ngoài phương pháp lan truyền ngược (Backpropagation) đơn giản

Như đã thảo luận trước đó trong phần 2.1, quá trình tiền huấn luyện và lan truyền ngược là một dạng bộ nhớ liên kết, trong đó dữ liệu đầu vào được ánh xạ tới "tín hiệu bất ngờ" (surprise) gây ra bởi đầu ra dự đoán của nó  $\nabla_{y_t} \mathcal{L}(W_t; x_t)$ :

$$W_{t+1} = W_t - \eta_{t+1} \nabla_{W_t} \mathcal{L}(W_t; x_t) = W_t - \eta_{t+1} \nabla_{y_t} \mathcal{L}(W_t; x_t) \otimes x_t$$

Trong đó  $x_t \sim \mathcal{D}_{train}$

Điều này, nhìn từ góc độ bộ nhớ liên kết, tương đương với một bước giảm gradient trong quá trình tối ưu hóa của:

$$\min_W \langle W x_t, \nabla_{y_t} \mathcal{L}(W_t; x_t) \rangle$$

Như đã thảo luận trong phụ lục C, công thức trên dẫn đến việc bỏ qua các mối phụ thuộc của các mẫu dữ liệu như  $x_t$ . Để mở rộng thành một công thức mạnh mẽ hơn, trong đó có tính đến các mối phụ thuộc giữa các điểm dữ liệu (điều này cực kỳ quan trọng khi chúng ta sử dụng bộ tối ưu hóa trong không gian token vì chúng không độc lập với nhau), chúng tôi sử dụng hàm mục tiêu hồi quy  $L_2$  với một bước giảm gradient như sau:

$$\min_W \left\| \langle W x_t, \nabla_{y_t} \mathcal{L}(W_t; x_t) \rangle \right\|_2^2$$

Công thức này dẫn đến một biến thể mới của phương pháp giảm gradient, có thể được đơn giản hóa như sau:

$$W_{t+1} = W_t (I - x_t x_t^\top) - \eta_{t+1} \nabla_{W_t} \mathcal{L}(W_t; x_t)$$

$$W_{t+1} = W_t (I - x_t x_t^\top) - \eta_{t+1} \nabla_{y_t} \mathcal{L}(W_t; x_t) \otimes x_t$$

Trong đó  $x_t \sim \mathcal{D}_{train}$

Sau đó, chúng tôi sử dụng bộ tối ưu hóa này làm bộ tối ưu hóa nội tại cho kiến trúc HOPE của mình.

### 3. HOPE: một mô-đun học tập tự tham chiếu (Self-Referential Learning Module) với bộ nhớ liên tục (Continuum Memory)

Các kiến trúc nền tảng hiện có bao gồm:

1. **Một mô-đun bộ nhớ làm việc** (working memory module) (ví dụ: attention), chịu trách nhiệm chủ động tổng hợp thông tin dọc theo chiều dài chuỗi; và
2. **Một lớp truyền thẳng** (feed-forward layer) (ví dụ: MLP) giúp tổng hợp thông tin qua các đặc trưng và đóng vai trò là bộ nhớ bền vững hoặc kho lưu trữ kiến thức của giai đoạn tiền huấn luyện.

Từ góc độ NL (Học lòng ghép), tiền huấn luyện là giai đoạn mà cấp độ ngoài cùng nhất của mô-đun học tập được cập nhật trên luồng ngữ cảnh giới hạn của nó. Theo đó, trong thiết lập học liên tục, giai đoạn tiền huấn luyện này cũng rất ít khi được cập nhật theo thời gian, và do đó kho lưu trữ kiến thức tương ứng của nó cũng rất ít cần được cập nhật.

Dựa trên trực giác này, chúng tôi mở rộng quan điểm truyền thống về hệ thống "bộ nhớ dài hạn/ ngắn hạn" và đề xuất một lớp truyền thẳng lưu trữ kiến thức cho **từng cấp độ** (miền tần số).

Từ định nghĩa về tần suất, **hệ thống bộ nhớ liên tục (Continuum Memory System - CMS)** được công thức hóa thành một chuỗi các khối MLP  $MLP^{(f_1)}(\cdot), \dots, MLP^{(f_k)}(\cdot)$ , trong đó mỗi khối liên kết với một kích thước đoạn dữ liệu (chunk size) là

$$C^{(\ell)} := \frac{\max_{\ell} C^{(\ell)}}{f_{\ell}}$$

Sao cho với đầu vào  $x = \{x_1, \dots, x_T\}$ , đầu ra của chuỗi được tính toán như sau (chúng tôi lược bỏ các bước chuẩn hóa để đảm bảo tính rõ ràng):

$$y_t = MLP^{(f_k)} \left( MLP^{(f_{k-1})} \left( \dots MLP^{(f_1)}(x_t) \right) \right)$$

Trong đó các tham số của khối MLP thứ  $\ell$ , tức là  $\theta^{(f_\ell)}$ , được cập nhật sau mỗi  $C^{(\ell)}$  bước:

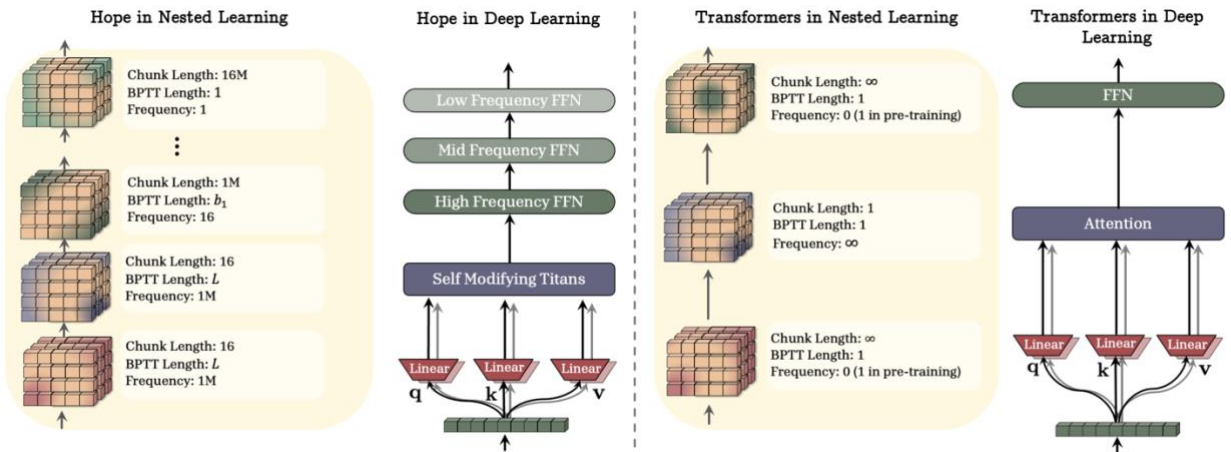
$$\theta_{i+1}^{(f_\ell)} = \theta_i^{(f_\ell)} - \begin{cases} \sum_{t=i-C^{(\ell)}}^i \eta_i^{(\ell)} \int (\theta_t^{(f_\ell)}; x_t) & \text{nếu } i \equiv 0 \pmod{C^{(\ell)}} \\ 0 & \text{trường hợp khác} \end{cases}$$

Trong Phụ lục B.1, chúng tôi thảo luận về các biến thể khác nhau của công thức này, bao gồm cả các lớp MLP lồng ghép hoàn toàn (fully nested MLP layers).

Ở đây,  $\eta_i^{(\ell)}$  là các tốc độ học tương ứng với  $\theta^{(f_\ell)}$ , và  $f(\cdot)$  là thành phần lõi của một bộ tối ưu hóa tùy ý (ví dụ:  $\nabla \mathcal{L}(\theta_t^{(f_\ell)}; x_t)$  trong phương pháp giảm gradient).

Khối Transformer thông thường [27] là một trường hợp đặc biệt của công thức này, trong đó  $k = 1$ . Đáng chú ý là phương trình 31 cung cấp một diễn giải quan trọng: các tham số  $\theta_t^{(f_\ell)}$  chịu trách nhiệm nén ngữ cảnh của chính chúng vào trong các tham số của chúng, và do đó chúng là đại diện cho kiến thức trừu tượng về ngữ cảnh của mình.

**HOPE:** Chúng tôi trình bày thêm một mô-đun học tập tự tham chiếu (self-referential learning module) dựa trên Titans [28] và biến thể giảm gradient của chúng tôi trong phần B.1. Việc kết hợp mô hình chuỗi tự tham chiếu này với hệ thống bộ nhớ liên tục tạo nên kiến trúc HOPE.



Hình 3: So sánh kiến trúc nền tảng HOPE với Transformer (Các thành phần chuẩn hóa và các thành phần phụ thuộc dữ liệu tiềm năng đã được lược bỏ để đảm bảo tính rõ ràng).

Bảng 1: Hiệu năng của HOPE và các mô hình cơ sở trên các tác vụ mô hình hóa ngôn ngữ và lập luận thường thức. Các mô hình lai được đánh dấu bằng \*.

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. ↑
HOPE (ours)	26.05	29.38	35.40	64.62	40.11	51.19	56.92	28.49	38.33	60.12	46.90
760M params / 30B tokens											
Transformer++	25.21	27.64	35.78	66.92	42.19	51.95	60.38	32.46	39.51	60.37	48.69
RetNet	26.08	24.45	34.51	67.19	41.63	52.09	63.17	32.78	38.36	57.92	48.46
DeltaNet	24.37	24.60	37.06	66.93	41.98	50.65	64.87	31.39	39.88	59.02	48.97
TTT	24.17	23.51	34.74	67.25	43.92	50.99	64.53	33.81	40.16	59.58	47.32
Samba*	20.63	22.71	39.72	69.19	47.35	52.01	66.92	33.20	38.98	61.24	51.08
Titans (LMM)	20.04	21.96	37.40	69.28	48.46	52.27	66.31	35.84	40.13	62.76	51.56
HOPE (ours)	20.53	20.47	39.02	70.13	49.21	52.70	66.89	36.05	40.71	63.29	52.26
1.3B params / 100B tokens											
Transformer++	18.53	18.32	42.60	70.02	50.23	53.51	68.83	35.10	40.66	57.09	52.25
RetNet	19.08	17.27	40.52	70.07	49.16	54.14	67.34	33.78	40.78	60.39	52.02
DeltaNet	17.71	16.88	42.46	70.72	50.93	53.35	68.47	35.66	40.22	55.29	52.14
Samba*	16.13	13.29	44.94	70.94	53.42	55.56	68.81	36.17	39.96	62.11	54.00
Titans (LMM)	15.60	11.41	49.14	73.09	56.31	59.81	72.43	40.82	42.05	60.97	56.82
HOPE (ours)	15.11	11.63	50.01	73.29	56.84	60.19	72.30	41.24	42.52	61.46	57.23

Chú thích:

- Bảng này so sánh hiệu năng của mô hình HOPE với các mô hình đối chuẩn (baselines) hàng đầu hiện nay (như Transformer++, RetNet, DeltaNet, TTT, Samba, Titans). So sánh được thực hiện trên hai quy mô kích thước mô hình và dữ liệu huấn luyện khác nhau:
  - 760M params / 30B tokens: Mô hình kích thước 760 triệu tham số, huấn luyện trên 30 tỷ token.
  - 1.3B params / 100B tokens: Mô hình kích thước 1.3 tỷ tham số, huấn luyện trên 100 tỷ token.
- Chứng minh luận điểm chính của bài báo:
  - **Kiến trúc HOPE hiệu quả hơn:** Việc kết hợp Học lồng ghép (Nested Learning) và Bộ nhớ liên tục giúp mô hình xử lý ngôn ngữ và lập luận tốt hơn so với cơ chế Attention thuần túy (Transformer) hay các mạng RNN hiện đại (Titans, RetNet).
  - **Khả năng toàn diện:** HOPE không chỉ tốt ở việc dự đoán từ tiếp theo (Language Modeling) mà còn rất mạnh ở các tác vụ đòi hỏi tư duy và hiểu biết thường thức (Reasoning tasks).

## 4. Thực nghiệm

Do giới hạn về không gian trình bày trong bài báo chính, chúng tôi chỉ báo cáo kết quả đánh giá mô hình HOPE trên các tác vụ mô hình hóa ngôn ngữ và lập luận thường thức. Tuy nhiên, chúng tôi cung cấp một tập hợp kết quả phong phú hơn trong phần phụ lục, bao gồm các thực nghiệm về bộ tối ưu hóa, sự nảy sinh của khả năng học trong ngữ cảnh (in-context learning), khả năng học liên tục của HOPE, các nghiên cứu cắt bỏ (ablation studies), các tác vụ ngữ cảnh dài,... Chi tiết về các thiết lập thực nghiệm và các bộ dữ liệu khác được sử dụng nằm trong phụ lục G.

**Mô hình hóa ngôn ngữ (Language Modeling) và lập luận thường thức (Common-sense Reasoning):** Chúng tôi tuân theo các nghiên cứu mô hình hóa chuỗi gần đây [28, 67, 68] và báo cáo kết quả của HOPE cùng các mô hình cơ sở với các kích thước 340M, 760M và 1.3B trên các tác vụ mô hình hóa ngôn ngữ cũng như các tác vụ hạ nguồn (downstream tasks) về lập luận thường thức.

Các kết quả này được báo cáo trong bảng 1 HOPE thể hiện hiệu năng rất tốt trên tất cả các quy mô và các tác vụ đánh giá chuẩn (benchmark tasks), vượt trội hơn cả Transformer và các mạng nơ-ron hồi quy hiện đại gần đây, bao gồm Gated DeltaNet và Titans.

Khi so sánh HOPE với Titans và Gated DeltaNet, chúng ta có thể thấy rằng việc thay đổi động các phép chiếu khóa (key), giá trị (value) và truy vấn (query) dựa trên ngữ cảnh, cũng như việc sử dụng một mô-đun bộ nhớ sâu (deep memory module), có thể tạo ra một mô hình có độ hỗn loạn (perplexity) thấp hơn và độ chính xác cao hơn trong các kết quả đánh giá chuẩn.

# Bảng tổng hợp các thuật ngữ chuyên ngành quan trọng

## Các khái niệm cốt lõi của bài báo

Thuật ngữ	Tiếng Việt	Giải thích
Nested Learning (NL)	Học lồng ghép	Hệ hình học tập mới, xem mô hình học máy là tập hợp các bài toán tối ưu hóa lồng vào nhau theo thời gian/ tần suất.
Nested Optimization	Tối ưu hóa lồng ghép	Cấu trúc trong đó quá trình tối ưu hóa này chứa bên trong nó một quá trình tối ưu hóa khác (ví dụ: Inner-loop và Outer-loop).
Context Flow	Luồng ngữ cảnh	Dòng chảy thông tin hoặc gradient mà một thành phần cụ thể của mô hình xử lý.
Associative Memory	Bộ nhớ liên kết	Hệ thống lưu trữ và truy xuất thông tin dựa trên sự liên kết (ánh xạ) giữa Khóa (Key) và Giá trị (Value).
Deep Optimizer	Bộ tối ưu hóa sâu	Bộ tối ưu hóa được tham số hóa bởi một mạng nơ-ron (như MLP) thay vì các công thức toán học cố định, giúp nó "học cách tối ưu".
Self-Referential	Tự tham chiếu	Khả năng của mô hình tự quan sát và sửa đổi thuật toán cập nhật của chính mình.
Update Frequency	Tần suất cập nhật	Tốc độ mà một thành phần thay đổi trạng thái.
Continuum Memory System (CMS)	Hệ thống bộ nhớ liên tục	Hệ thống bộ nhớ không chia tách rạch ròi ngắn hạn/ dài hạn, mà là một chuỗi các lớp với tần suất cập nhật giảm dần.

## Tối ưu hóa & toán học

Thuật ngữ	Tiếng Việt	Giải thích
Gradient Descent	Giảm Gradient	Thuật toán tối ưu hóa cơ bản, cập nhật trọng số theo hướng ngược chiều đạo hàm của hàm mất mát.

Backpropagation	Lan truyền ngược	Thuật toán tính toán gradient của hàm mất mát đối với các trọng số, đi từ lớp cuối ngược về lớp đầu.
Momentum	Đà (Momentum)	Kỹ thuật giúp tăng tốc gradient descent bằng cách tích lũy các vector gradient quá khứ (được ví như "bộ nhớ" của optimizer).
Preconditioning	Tiền điều kiện hóa	Kỹ thuật biến đổi hình học của không gian tối ưu hóa để hội tụ nhanh hơn.
Hessian Matrix	Ma trận Hessian	Ma trận các đạo hàm riêng bậc hai, mô tả độ cong của bề mặt hàm lỗi.
Local Surprise Signal (LSS)	Tín hiệu bất ngờ cục bộ	Sự sai lệch giữa dự đoán của mô hình và thực tế (Gradient), kích thích việc cập nhật bộ nhớ.
Fast Weights vs. Slow Weights	Trọng số nhanh vs. Chậm	Trọng số nhanh thay đổi theo từng mẫu dữ liệu (như activation/attention), trọng số chậm được học qua thời gian dài.
Newton-Schulz Iteration	Phép lặp Newton-Schulz	Thuật toán xấp xỉ nghịch đảo ma trận hoặc căn bậc hai ma trận mà không cần phân rã SVD (dùng trong Muon optimizer).

## Kiến trúc & mạng nơ-ron

Thuật ngữ	Tiếng Việt	Giải thích
Transformer	Mạng Transformer	Kiến trúc dựa trên cơ chế Attention, hiện đang thống trị lĩnh vực xử lý ngôn ngữ tự nhiên.
Linear Attention	Chú ý tuyến tính	Biến thể của Attention loại bỏ Softmax để giảm độ phức tạp tính toán.
Multi-Layer Perceptron (MLP)	Mạng Perceptron đa lớp	Mạng nơ-ron truyền thẳng cơ bản, gồm các lớp kết nối đầy đủ (Dense/Linear layers).
Feed-forward Layer	Lớp truyền thẳng	Lớp mạng nơi thông tin đi theo một chiều, thường dùng để lưu trữ tri thức tĩnh (trong Transformer).
Residual Connection	Kết nối phần dư (Kết nối tắt)	Đường dẫn cộng trực tiếp đầu vào vào đầu ra của một lớp giúp huấn luyện mạng sâu dễ hơn.

Key / Value / Query (K, V, Q)	Khóa / Giá trị / Truy vấn	Ba thành phần cơ bản trong cơ chế Attention (để truy xuất thông tin).
----------------------------------	------------------------------	--

## Khoa học thần kinh & học máy

Thuật ngữ	Tiếng Việt	Giải thích
Neuroplasticity	Tính dẻo thần kinh	Khả năng thay đổi cấu trúc và chức năng của não bộ để thích nghi với trải nghiệm mới.
Synaptic Consolidation	Củng cố khớp thần kinh	Quá trình củng cố ký ức nhanh chóng ngay sau khi học (giai đoạn Online).
Systems Consolidation	Củng cố hệ thống	Quá trình củng cố ký ức lâu dài, chuyển thông tin từ hồi hải mã sang vỏ não (giai đoạn Offline/Ngủ).
Catastrophic Forgetting	Quên thảm khốc	Hiện tượng mô hình học tri thức mới thì quên sạch tri thức cũ.
In-context Learning	Học trong ngữ cảnh	Khả năng của LLM thực hiện tác vụ mới chỉ dựa trên ví dụ trong prompt mà không cần cập nhật trọng số.
Zero-shot / Few-shot	Zero-shot / Few-shot	Khả năng thực hiện tác vụ mà không cần (zero) hoặc chỉ cần vài (few) ví dụ mẫu.
Out-of-distribution (OOD)	Ngoài phân phối	Dữ liệu kiểm thử khác biệt hoàn toàn so với dữ liệu mà mô hình đã được huấn luyện.