Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Giáp Ngọc Hiệu

**Tóm tắt nội dung**

Bài report này là phần tóm tắt lại ý tưởng, nội dung chính, những điều làm được của paper

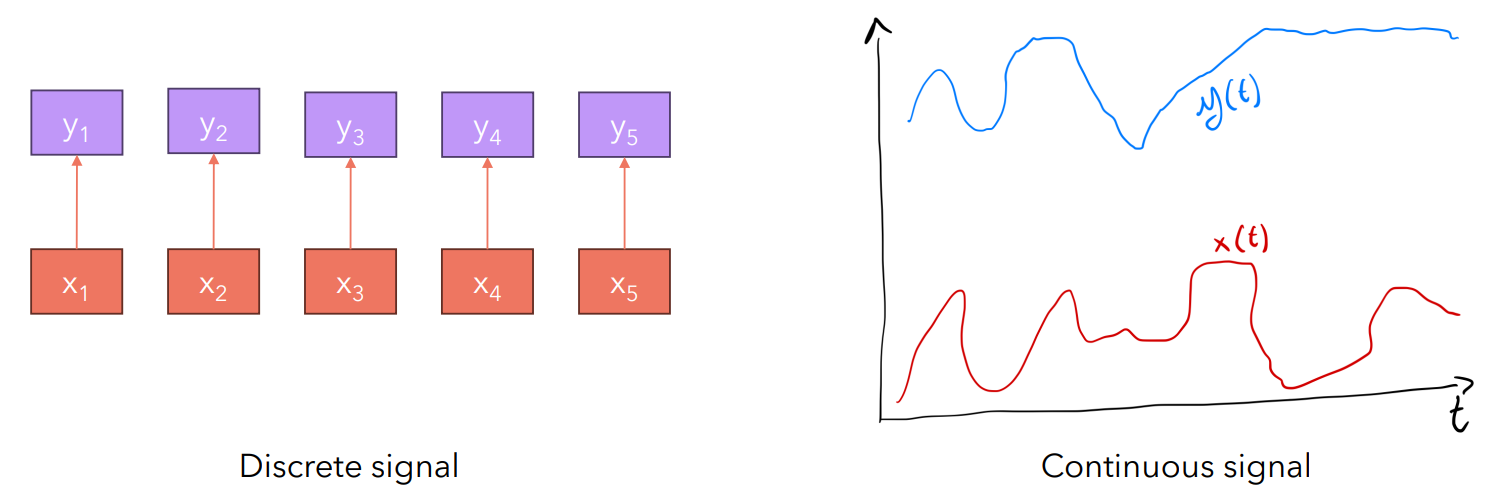
Mamba: Linear-Time Sequence Modeling with Selective State Spaces

1. **Ý tưởng**

Các mô hình nền tảng, hiện đóng vai trò quan trọng cho hầu hết các ứng dụng thú vị trong học sâu, hầu như đều dựa trên kiến ​​trúc Transformer và attention module. Nhiều kiến ​​trúc thời gian dưới bậc hai như linear attention, gated convolution và ecurrent cũng như mô hình structured state space models (SSMs) đã được phát triển để giải quyết sự kém hiệu quả tính toán của Transformers trên các chuỗi dài, nhưng chúng chưa thực hiện tốt trên các phương thức quan trọng như như ngôn ngữ. Tác giả xác định rằng điểm yếu chính của các mô hình như vậy là không có khả năng thực hiện lý luận dựa trên nội dung và thực hiện một số cải tiến. Đầu tiên, chỉ cần để các tham số SSM là chức năng của đầu vào sẽ giải quyết điểm yếu của chúng bằng các phương thức riêng biệt, cho phép mô hình truyền có chọn lọc hoặc quên thông tin dọc theo chiều dài chuỗi tùy thuộc vào token hiện tại. Thứ hai, mặc dù thay đổi này ngăn cản việc sử dụng các phép tích chập hiệu quả, tác giả vẫn thiết kế thuật toán song song nhận biết phần cứng ở recurrent mode. Tác giả tích hợp các SSM chọn lọc này vào một end-to-end neural network architecture được đơn giản hóa mà không cần attention hoặc thậm chí không cần đến các khối MLP (Mamba). Mamba có khả năng suy luận nhanh (5x so với Transformers) và near scaling theo độ dài chuỗi, đồng thời hiệu suất của nó cải thiện trên dữ liệu thực với các chuỗi có độ dài lên tới hàng triệu. Là một neral sequence model backbone, Mamba đạt được hiệu suất tiên tiến trên nhiều phương thức như ngôn ngữ, âm thanh và hệ gen. Về mô hình hóa ngôn ngữ, mô hình Mamba-3B hoạt động tốt hơn Transformers có cùng kích thước và ngang với Transformers có kích thước gấp đôi kích thước của nó, cả trong quá trình pretraining và downstream evaluation.

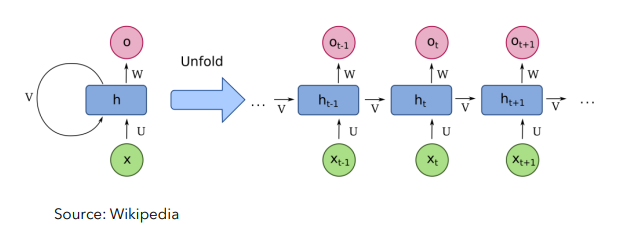
1. **Overview Sequence modeling**

Mục tiêu của sequence model là ánh xạ trình tự đầu vào tới trình tự đầu ra. Chúng ta có thể ánh xạ tín hiệu đầu vào liên tục 𝑥(𝑡) thành tín hiệu đầu ra 𝑦(𝑡) hoặc chuỗi đầu vào rời rạc thành chuỗi đầu ra rời rạc.



**Recurrent Neural Networks (RNN)**

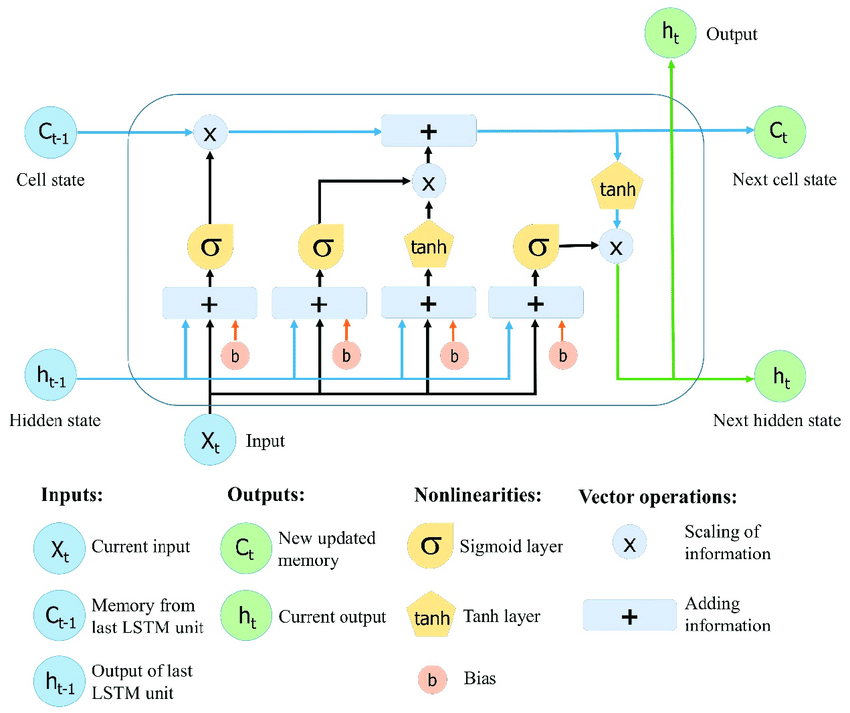
**RNN** là một loại mạng thần kinh được thiết kế để xử lý dữ liệu tuần tự. Ý tưởng cốt lõi đằng sau RNN là việc sử dụng trạng thái ẩn, nắm bắt thông tin về các yếu tố trước đó trong chuỗi. Trạng thái ẩn này được cập nhật tuần tự khi mạng xử lý từng phần tử của đầu vào.



* Context window vô hạn về mặt lí thuyết
* Tính chất tuần tự của RNN dẫn đến thời gian đào tạo chậm - O(N)
* Thời gian suy luận không đổi cho mỗi token.
* Vanishing Gradients: Do quy tắc dây chuyền được sử dụng trong lan truyền ngược, RNN thường gặp phải vấn đề biến mất độ dốc, trong đó độ dốc trở nên cực kỳ nhỏ, cản trở việc học hiệu quả.

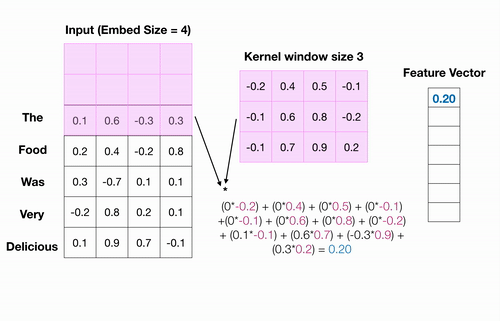
**Long Short-Term Memory (LSTM)**

**LSTM** là một biến thể của RNN giúp giải quyết vấn đề độ dốc biến mất. Họ giới thiệu một cơ chế kiểm soát giúp duy trì sự phụ thuộc lâu dài của dữ liệu. Tuy nhiên, mặc dù giảm thiểu vấn đề độ dốc biến mất ở một mức độ nào đó, nhưng chúng vẫn gặp khó khăn với sự phụ thuộc trong phạm vi rất dài.



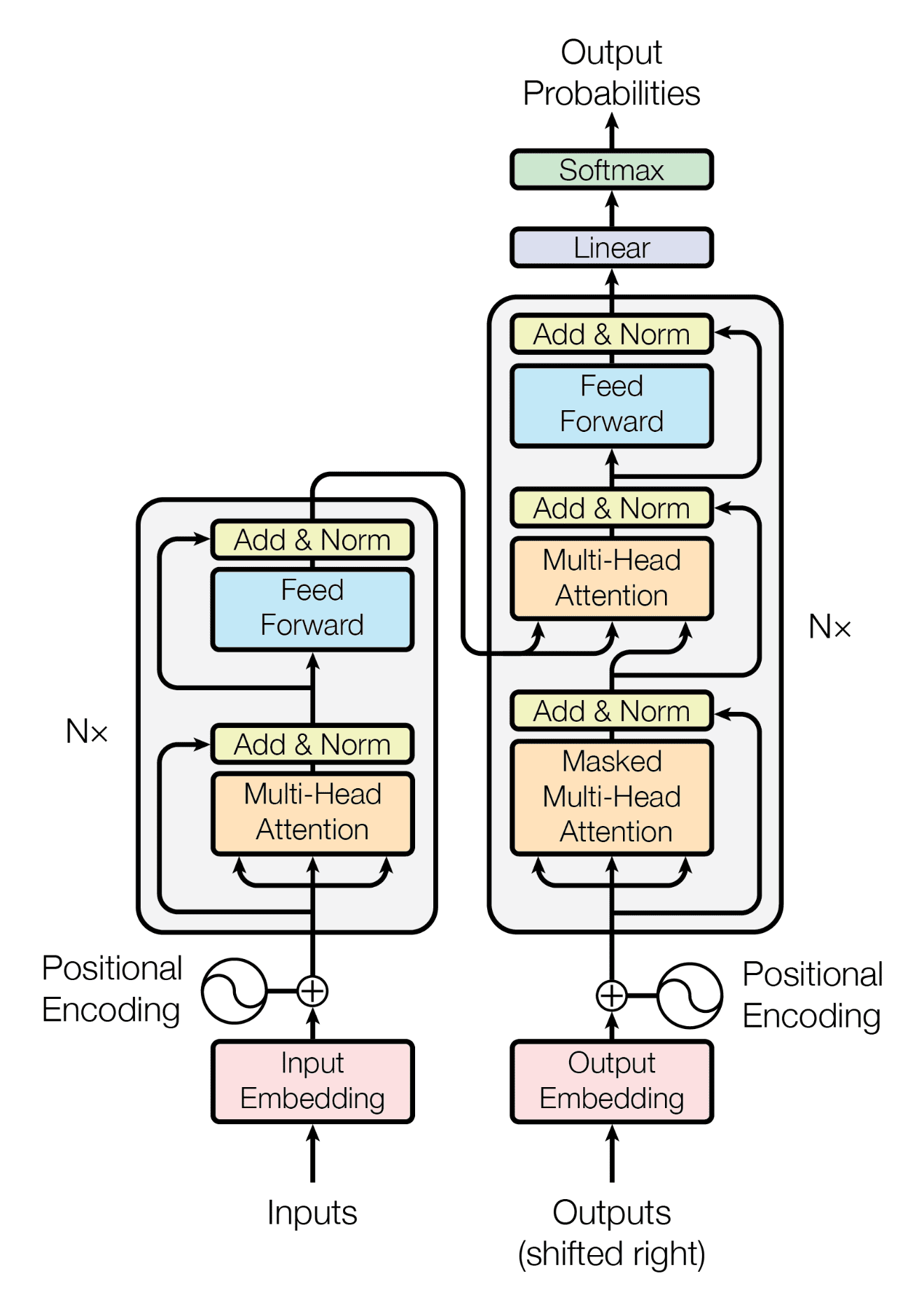
**CNN**

Hình ảnh sau đây minh họa cách filter tích chập trượt qua một câu, ba từ cùng một lúc. Đây được gọi là tích chập 1D vì kernel chỉ di chuyển theo một chiều. Nó tính toán tích số theo từng phần tử của trọng số của mỗi từ, nhân với trọng số được gán cho filter tích chập. Kết quả đầu ra sẽ là một vectơ đặc trưng chứa nhiều giá trị như có trong các phần nhúng đầu vào, do đó kích thước chuỗi đầu vào không quan trọng.



* Context window hữu hạn (tùy thuộc vào kích thước kernel)
* Cần cụ thể hóa kernel trước khi sử dụng.
* Việc huấn luyện và suy luận phụ thuộc vào kích thước kernel.
* Dễ dàng song song hóa.

**Transformers**

****

* Context window hữu hạn
* Thời gian huấn luyện là O(
* Suy luận với 𝑂(𝑁) khi sử dụng KV-Cache, cho mỗi token. Điều này có nghĩa là nếu chúng ta muốn tạo token thứ 10, chúng ta cần thực hiện tích 10 dot product

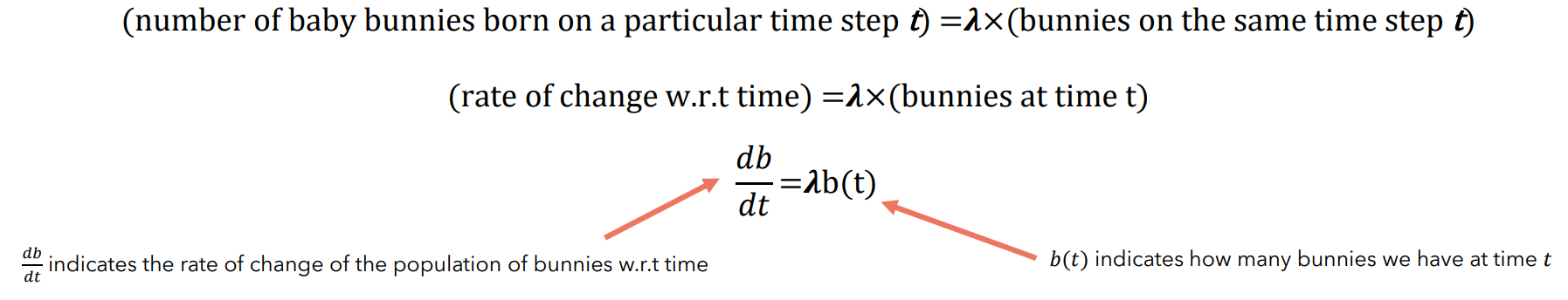
Một cách lý tưởng, chúng ta muốn có một mô hình mà chúng ta có thể:

1. Song song hóa quá trình đào tạo (như Transformer) và có thể mở rộng tuyến tính thành các chuỗi dài (với chi phí tính toán/bộ nhớ là O(N) như RNN)

2. Có thể suy ra từng token có chi phí tính toán/bộ nhớ không đổi (O(1) như RNN)…

1. **Phương pháp**
   1. **Introduction to differential equations**

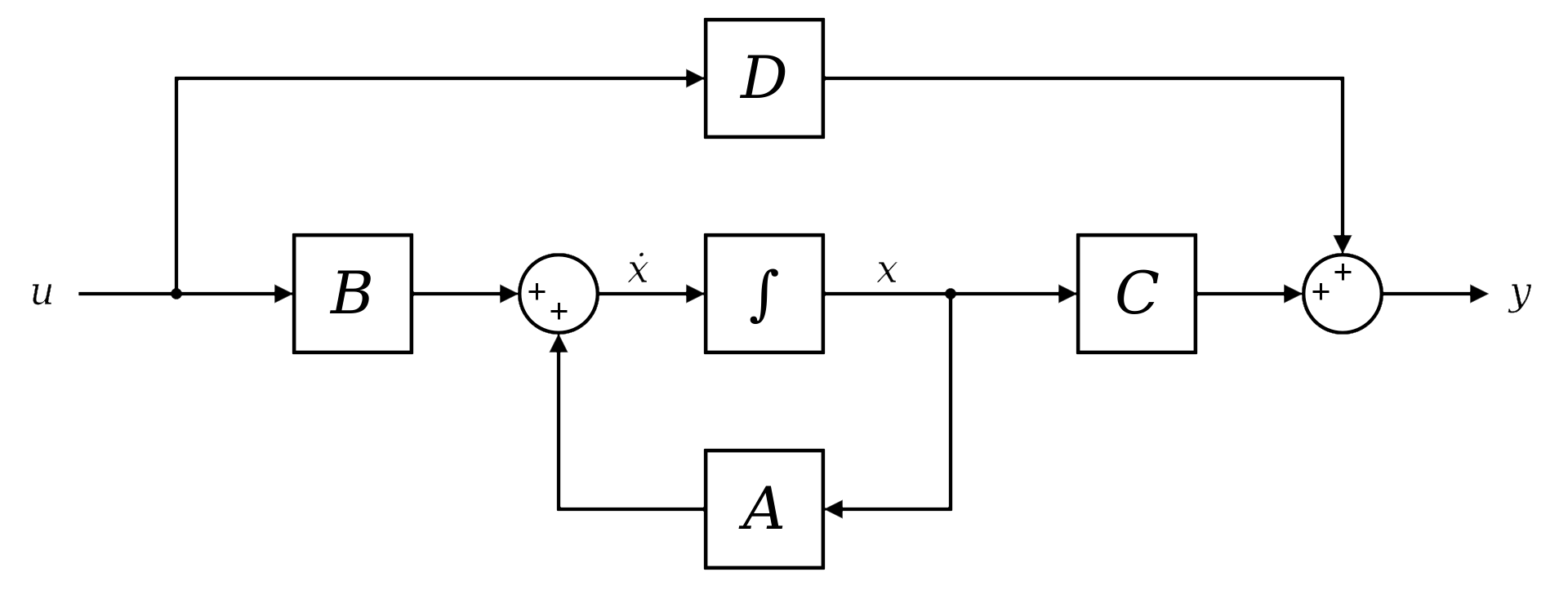
Hãy nói về phương trình vi phân bằng một ví dụ rất đơn giản. Hãy tưởng tượng bạn có một số chú thỏ và quần thể tăng trưởng với tốc độ không đổi 𝝀 tỷ lệ thuận với số lượng thỏ, có nghĩa là mỗi chú thỏ sẽ sinh ra 𝝀 chú thỏ con. Vì vậy, chúng ta có thể nói rằng tốc độ thay đổi của quần thể thỏ như sau:



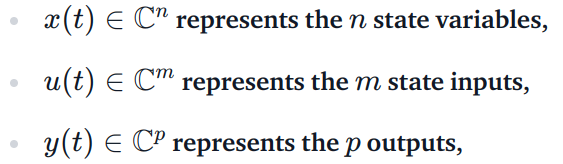
Làm thế nào chúng ta có thể tìm được quần thể tại thời điểm t = 100 khi biết rằng quần thể gồm có 5 con thỏ tại thời điểm t = 0? Chúng ta cần tìm 𝑏(𝑡) mô tả số lượng thỏ của chúng ta theo thời gian. Giải phương trình vi phân có nghĩa là tìm hàm 𝑏(𝑡) làm cho biểu thức trên đúng với mọi giá trị của 𝑡. Chúng ta có thể xác minh rõ ràng rằng nghiệm là 𝑏 𝑡 = 𝑘 , trong đó 𝑘 = 𝑏 (0) = 5, quần thể thỏ ban đầu của chúng ta. Thông thường, chúng ta biểu diễn một phương trình vi phân bằng cách bỏ biến 𝑡 bằng cách viết nó như sau: 𝑏’ = 𝝀b

Chúng ta thường sử dụng các phương trình vi phân để mô hình hóa trạng thái của hệ thống theo thời gian, với mục tiêu tìm ra hàm cho chúng ta trạng thái của hệ thống tại bất kỳ bước thời gian nào, với trạng thái ban đầu của hệ thống tại thời điểm 0.

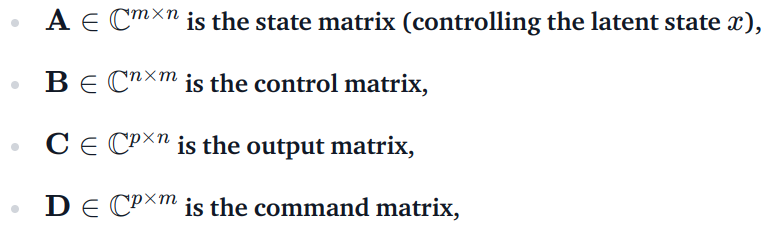
* 1. **Definition of an SSM in deep learning**



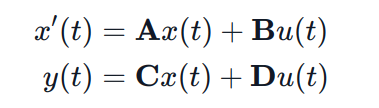
Có thể thấy rằng SSM dựa trên ba biến phụ thuộc vào thời gian t:



Chúng ta cũng có thể thấy rằng nó được tạo thành từ bốn ma trận có thể học được: **A, B, C** và **D**

****

Hình ảnh trên có thể được rút gọn thành hệ phương trình sau (x' chỉ đạo hàm):

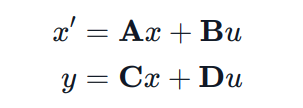


Nó ánh xạ tín hiệu đầu vào 1-D u(t) tới một N-D latent state x(t) trước khi chiếu tới tín hiệu đầu ra 1-D y(t).

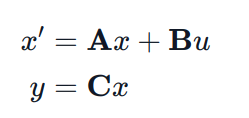
Mục tiêu chỉ đơn giản là sử dụng SSM như black-box trong deep sequence model, trong đó **A,B,C,D** là các tham số được học bằng phương pháp gradient descent. Phần còn lại chúng ta sẽ bỏ qua tham số **D** (giả sử D=0 vì Du có thể được xem là skip connection và dễ tính toán).

SSM ánh xạ đầu vào u(t) thành một vectơ biểu diễn trạng thái x(t) và một đầu ra y(t). Để đơn giản, giả sử đầu vào và đầu ra là một chiều và biểu diễn trạng thái là N chiều. Phương trình đầu tiên xác định sự thay đổi trong x(t) theo thời gian.

Tương tự, vì ngầm hiểu rằng các biến phụ thuộc vào thời gian nên phương trình trước thường được viết dưới dạng sau để đơn giản:



Hệ thống này có thể được làm gọn hơn nữa vì trong deep learning SSMs, Du=0 được coi là ***skip connection*** dễ tính toán.



State space model là tuyến tính và bất biến theo thời gian. Tuyến tính vì các mối quan hệ trong biểu thức trên là tuyến tính và bất biến theo thời gian vì các ma trận tham số A, B, C, D không phụ thuộc vào thời gian (chúng cố định).

* 1. **State space model in Paper**

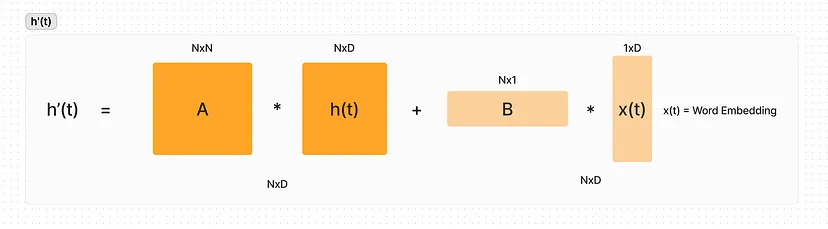
Structured state space sequence models (S4) là một loại sequence model gần đây dành cho học sâu có liên quan rộng rãi đến RNN, CNN cũng như các state space model cổ điển. Chúng được lấy cảm hứng từ một hệ thống liên tục cụ thể ánh xạ hàm hoặc chuỗi 1 chiều 𝑥(𝑡) ∈ ℝ ↦ 𝑦(𝑡) ∈ ℝ thông qua trạng thái tiềm ẩn ℎ(𝑡) ∈

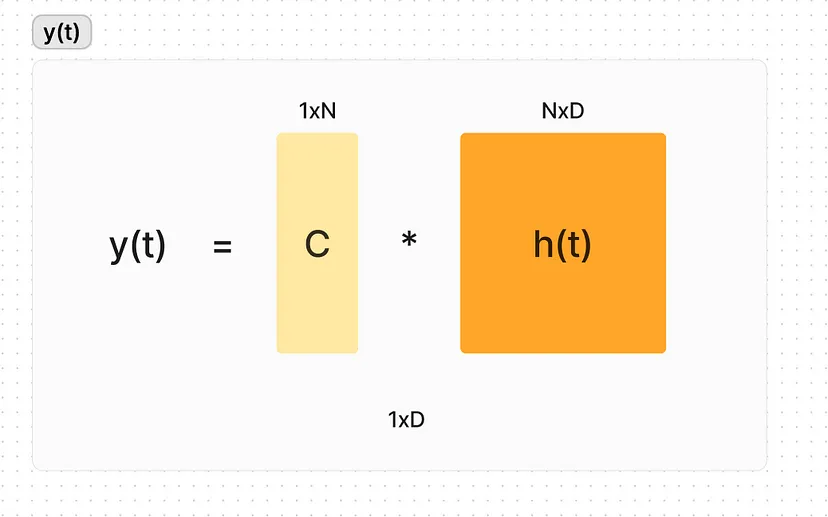


Để tìm tín hiệu đầu ra 𝑦(𝑡) tại thời điểm t, trước tiên chúng ta cần tìm hàm ℎ(𝑡) mô tả trạng thái của hệ thống cho tất cả các bước thời gian. Nhưng điều đó có thể khó giải quyết bằng phương pháp phân tích. Thông thường, chúng ta không bao giờ làm việc với các tín hiệu liên tục mà luôn làm việc với các tín hiệu rời rạc , nên chúng ta cần rời rạc hóa hệ thống của mình.

Ở đây, A, B, C và D là các ma trận xác định động lực của hệ thống, với A biểu thị cách trạng thái phát triển, B biểu thị cách đầu vào ảnh hưởng đến trạng thái, C cách trạng thái được chuyển thành đầu ra và D truyền trực tiếp đầu vào tới đầu ra.

Lưu ý rằng nó khác với standard recurrent network - nó hoàn toàn tuyến tính và không có các phép biến đổi phi tuyến tính mà LSTM hoặc GRU có bên trong chúng.





**Discretization**



Để giải phương trình vi phân, chúng ta cần tìm hàm h(t) làm cho hai vế của phương trình bằng nhau, nhưng hầu hết rất khó tìm được nghiệm giải tích của phương trình vi phân, đó là lý do tại sao chúng ta có thể tính gần đúng nghiệm của một phương trình vi phân. Tìm nghiệm gần đúng của phương trình vi phân có nghĩa là tìm một chuỗi h(0), h(1), h(2), h(3), v.v. mô tả sự phát triển của hệ thống của chúng ta theo thời gian. Vì vậy, thay vì tìm ℎ(𝑡) chúng ta muốn tìm = trong đó ∆ là kích thước bước của chúng ta.

Hãy thử tìm nghiệm gần đúng hàm 𝑏(𝑡) - bằng phương pháp Euler!

1. Đầu tiên chúng ta hãy viết lại mô hình quần thể thỏ của chúng ta:

𝑏′ (𝑡) =𝝀b(t)

2. Đạo hàm của một hàm số là tốc độ thay đổi của hàm số, đó là:

= 𝑏′ (𝑡)

Vì vậy, bằng cách chọn kích thước bước nhỏ ∆ chúng ta có thể loại bỏ giới hạn: ≅ ′ (𝑡). Bằng cách nhân với ∆ và di chuyển các số hạng xung quanh, chúng ta có thể viết thêm:

≅ 𝑏′ (𝑡) ∆ + 𝑏(𝑡)

3. Sau đó, chúng ta có thể thế mô hình quần thể thỏ vào biểu thức trước đó để thu được: 𝑏(𝑡 + ∆) ≅ 𝝀 𝑏(𝑡) ∆ + 𝑏(𝑡)

4. Chúng ta đã thu được một công thức truy hồi!

Hãy sử dụng công thức này để ước tính trạng thái của quần thể thỏ theo thời gian: 𝑏(𝑡 + ∆) ≅ 𝝀 𝑏(𝑡) ∆ + 𝑏(𝑡)

Đặt 𝝀 = 2, ∆ = 1.

Ví dụ: nếu bắt đầu với quần thể 5 con thỏ tại thời điểm t = 0, ta có thể tính diễn biến của quần thể như sau:

• Biết quần thể tại thời điểm t = 0, ta tính được quần thể tại thời điểm t = 1: 𝑏 (1) = ∆ 𝝀 b(0) + 𝑏(0) = 1 × 2 × 5 + 5 = 15

• Biết quần thể tại thời điểm t = 1, ta tính được quần thể tại thời điểm t = 2: 𝑏 (2) = ∆ 𝝀 b(1)+ 𝑏(1) = 1 × 2 × 15 + 15 = 45

• Biết quần thể tại thời điểm t = 2, ta tính được quần thể tại thời điểm t = 3: 𝑏 (3) = ∆ 𝝀 b(2)+𝑏 (2) = 1 × 2 × 45 + 45 = 135

• Kích thước bước ∆ càng nhỏ ,phép tính gần đúng của nghiệm giải tích 𝑏 (𝑡) = 5 càng tốt.

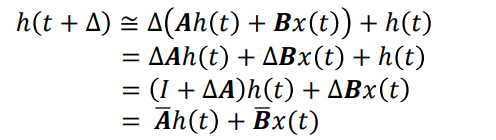
Tương tự, chúng ta cũng có thể rời rạc hóa state space model để có thể tính toán sự phát triển của trạng thái theo thời gian bằng cách sử dụng công thức lặp **recurrent**.

1. Bằng cách sử dụng định nghĩa đạo hàm, chúng ta biết rằng:

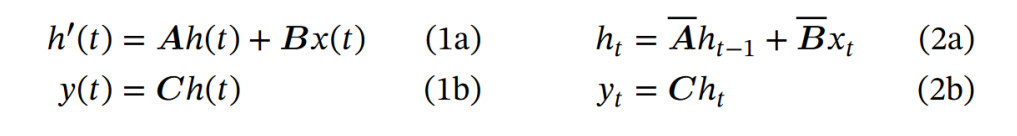
h (𝑡 + ∆) ≅ ∆ h′(t) + ℎ (t)

2. Đây là (continuous) state space model: h′ (𝑡) = 𝑨ℎ (𝑡) + 𝑩𝑥 (𝑡)

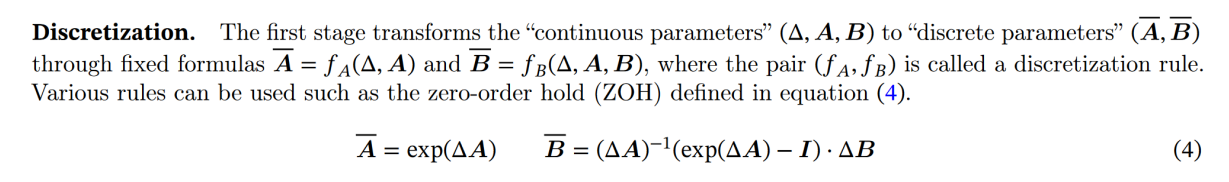
3. Chúng ta có thể thay thế state space model vào biểu thức đầu tiên để có được kết quả sau:



Bây giờ chúng ta có một recurrent formula cho phép chúng ta tính toán lặp lại trạng thái của hệ thống từng bước một, biết trạng thái ở bước thời gian trước đó. Ma trận và là các tham số rời rạc của mô hình. Điều này cũng cho phép chúng ta tính toán đầu ra 𝑦(𝑡) của hệ thống cho các bước thời gian rời rạc.



Trong bài báo thay vì phương pháp Euler, tác giả sử dụng quy tắc Zero-Order Hold (ZOH) để rời rạc hóa hệ thống.



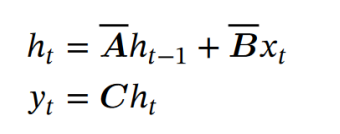
**Computation**

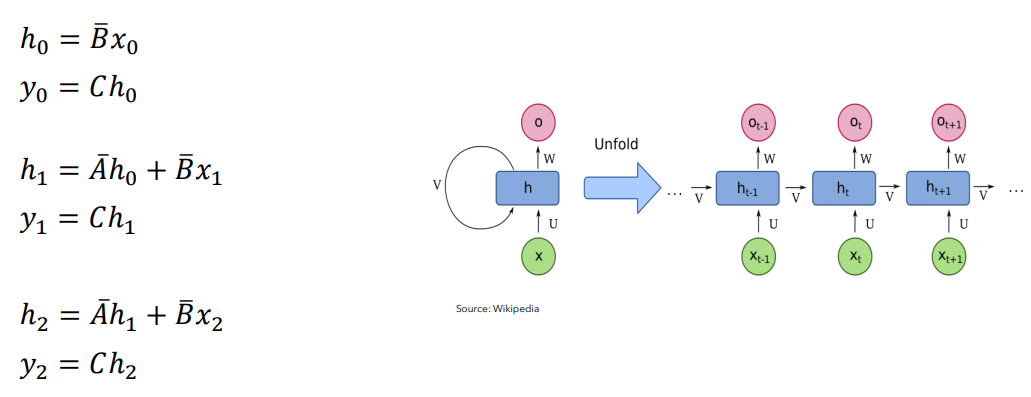
Sau khi các tham số đã được chuyển đổi từ (∆, A, B, C) ↦ (A, B, C), mô hình có thể được tính toán theo hai cách, dưới dạng linear recurrence hoặc global convolution

Thông thường, mô hình sử dụng convolutional mode để huấn luyện song song hiệu quả (trong đó toàn bộ chuỗi đầu vào được nhìn thấy trước đó) và chuyển sang recurrent mode để suy luận tự hồi quy hiệu quả (trong đó các đầu vào được nhìn thấy từng bước một).

**Recurrent computation**

Bây giờ chúng ta đã có recurrent formula, làm cách nào chúng ta có thể sử dụng nó để tính toán đầu ra của hệ thống cho các bước thời gian khác nhau? Giả sử, để đơn giản, trạng thái ban đầu của hệ thống là 0.

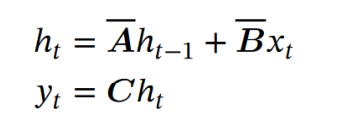


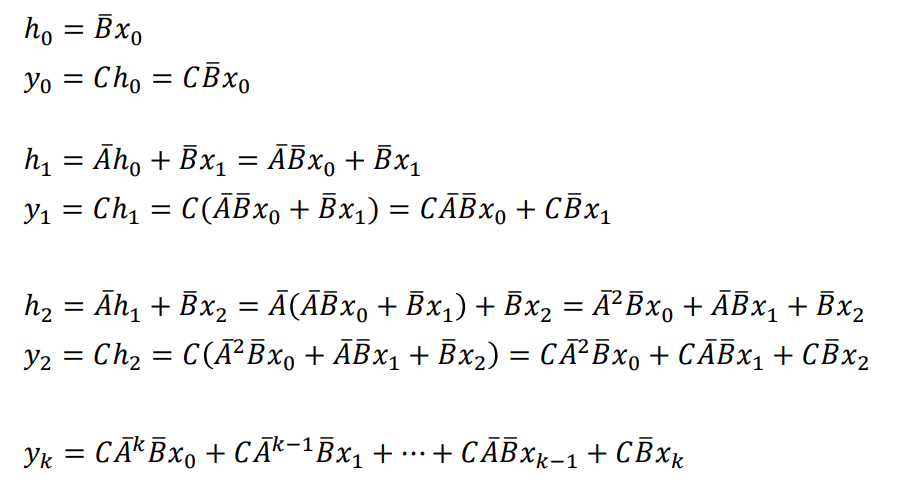


Recurrent formula rất tốt cho việc suy luận vì chúng ta có thể tính toán một token tại một thời điểm với yêu cầu bộ nhớ/tính toán không đổi. Điều này làm cho nó phù hợp trong quá trình suy luận trong Large Language Model, trong đó chúng ta muốn tạo một token tại một thời điểm dựa trên prompt và các token được tạo trước đó.

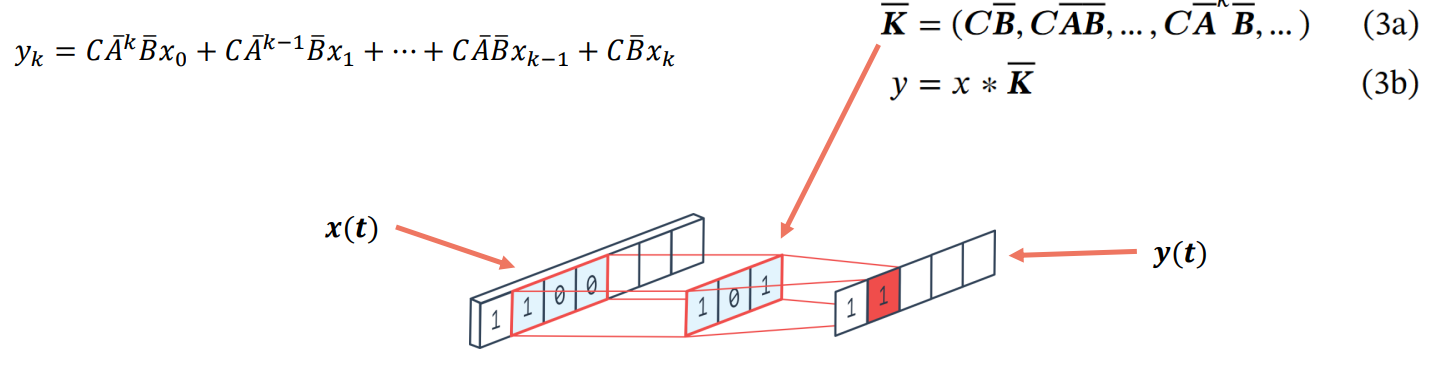
Recurrent formula không tốt cho việc đào tạo, vì trong quá trình đào tạo, chúng ta đã có tất cả các token của đầu vào và mục tiêu, vì vậy chúng ta muốn song song hóa tính toán nhiều nhất có thể, giống như Transformer đã làm!

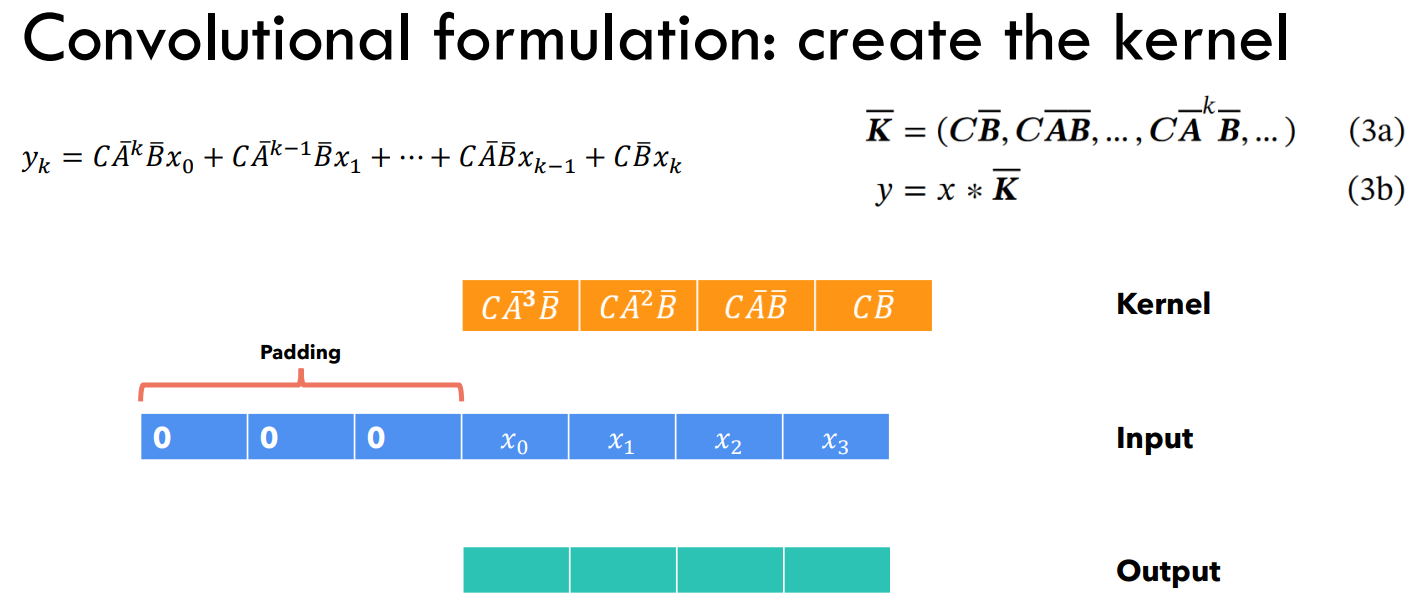
**Convolutional computation**

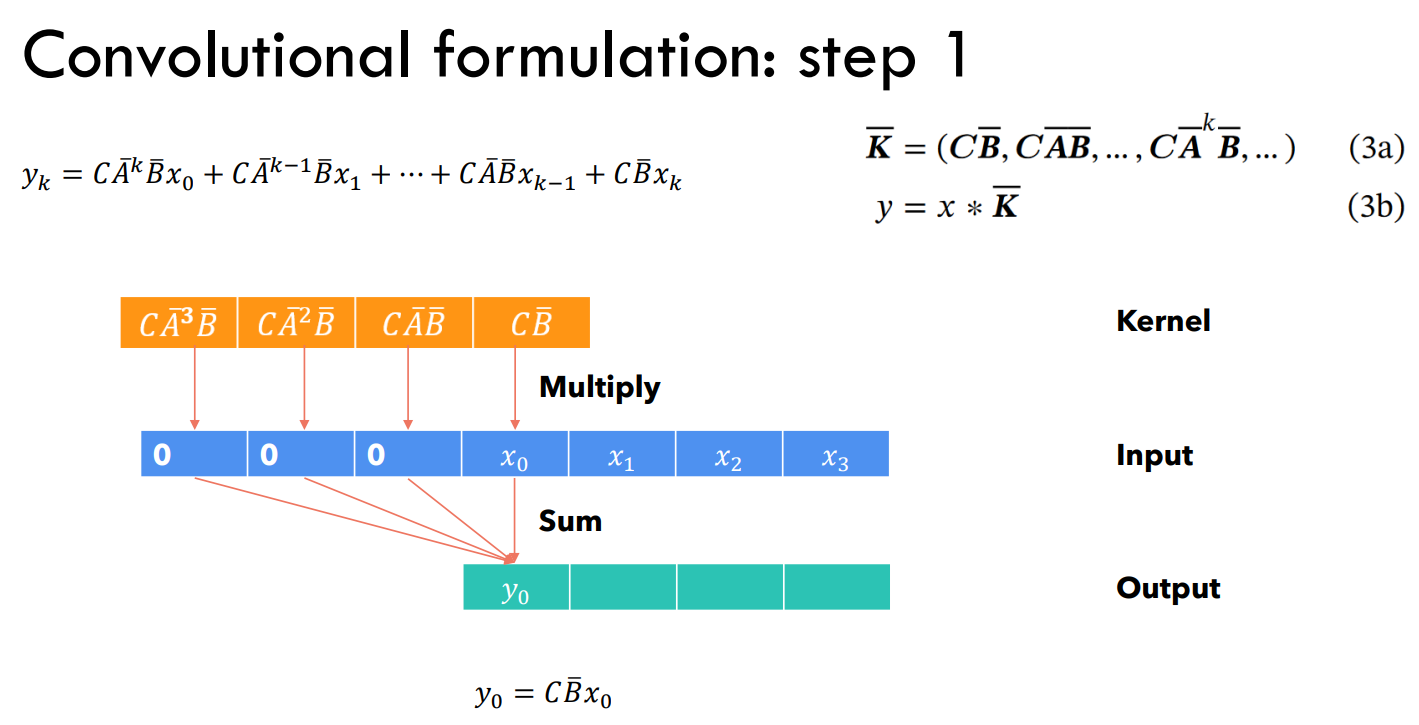
****

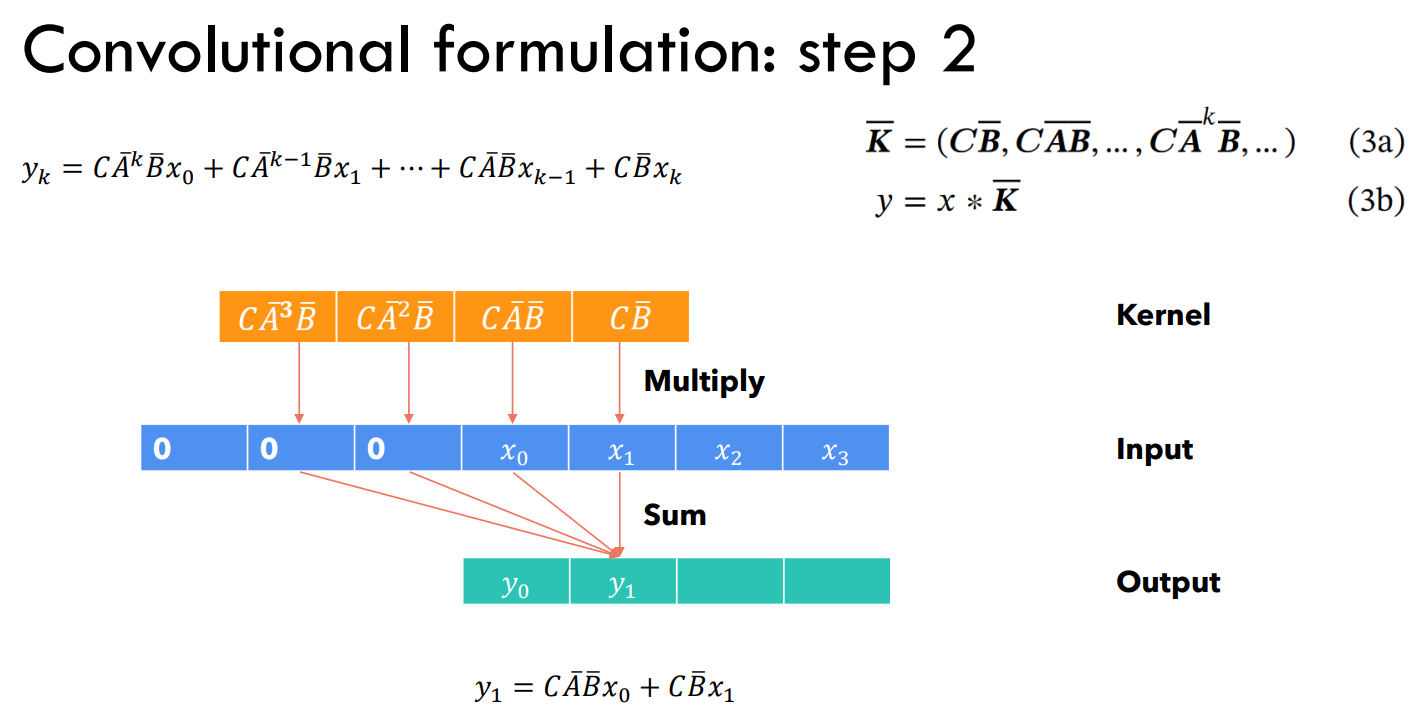
****

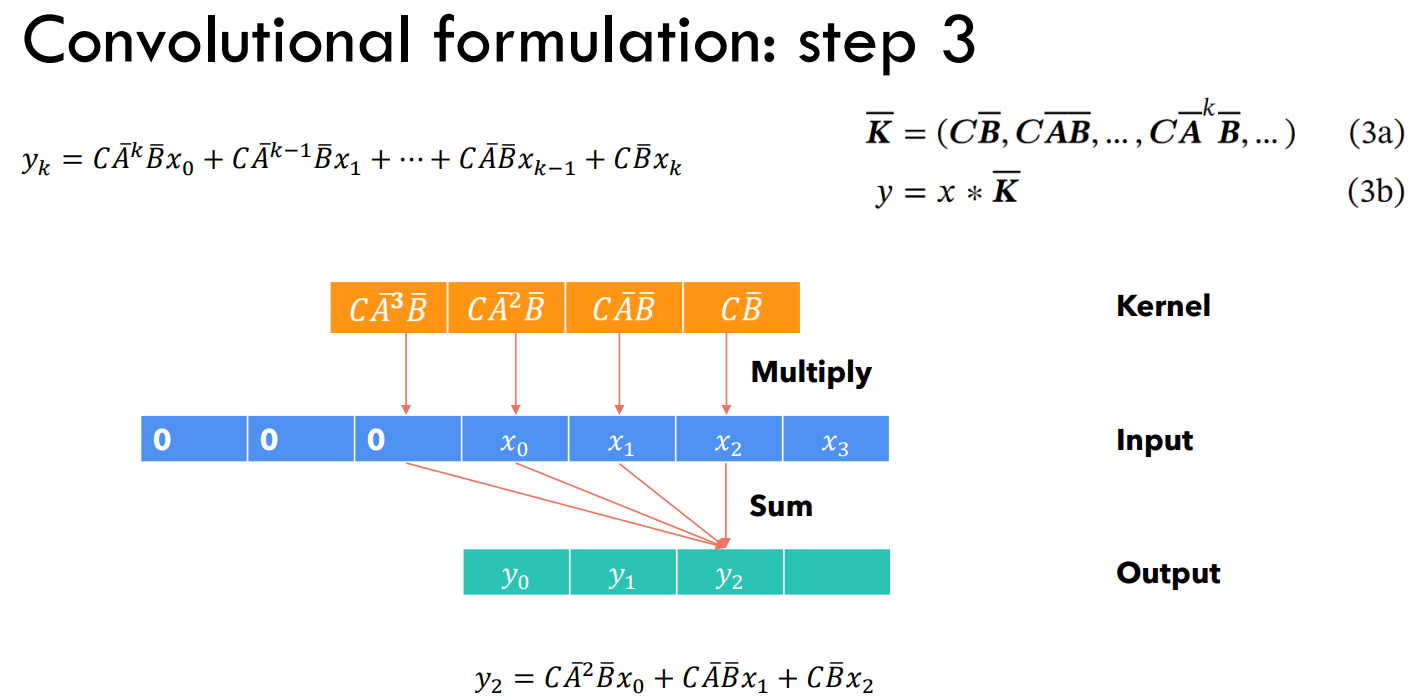
Bằng cách sử dụng công thức được rút ra, chúng ta nhận thấy một điều thú vị: đầu ra của hệ thống có thể được tính bằng cách sử dụng tích chập của hạt nhân với đầu vào 𝑥(𝑡).

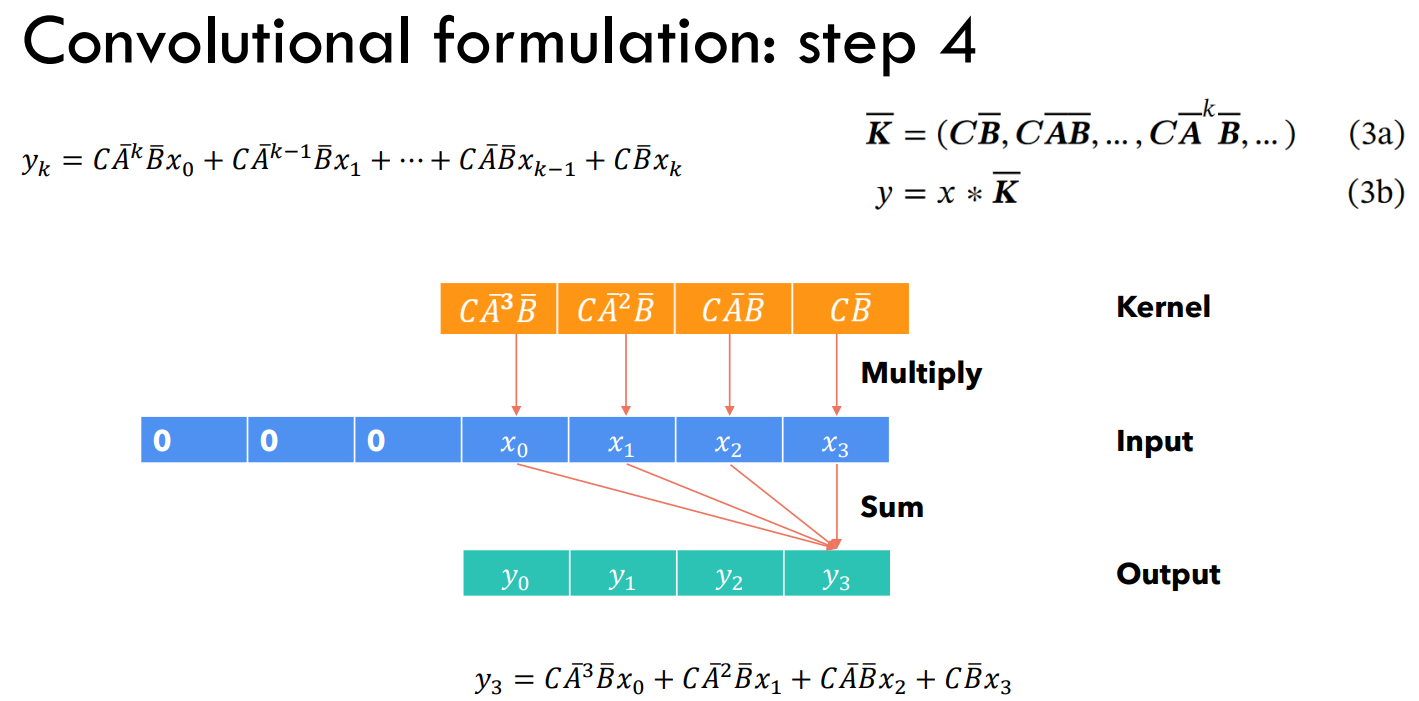












**Convolutional/Recurrent computation**

Điều tốt nhất về phép tính tích chập là nó có thể được song song hóa, vì đầu ra không phụ thuộc vào mà chỉ phụ thuộc vào kernel và đầu vào. Tuy nhiên, việc cụ thể hóa (xây dựng) kernel có thể tốn kém về mặt tính toán, kể cả từ quan điểm bộ nhớ.

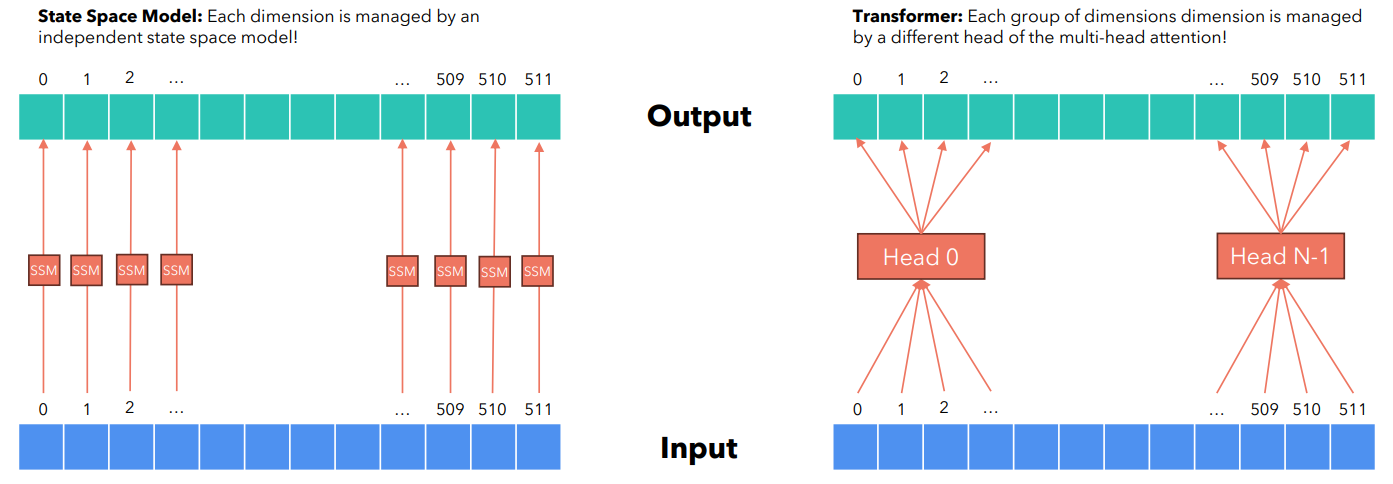
1. Chúng ta có thể sử dụng tính toán tích chập để thực hiện huấn luyện vì chúng ta đã có tất cả chuỗi đầu vào của token và nó có thể dễ dàng song song hóa.

2. Chúng ta có thể sử dụng recurrent formula để thực hiện suy luận, mỗi lần một token, sử dụng lượng tính toán và bộ nhớ không đổi cho mỗi token.

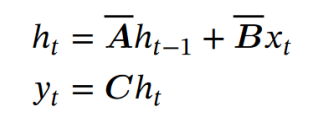
**From 1 dimension to multiple dimensions**

State space model mà chúng ta đã nghiên cứu cho đến nay chỉ tính toán một đầu ra cho mỗi token đầu vào . Làm thế nào chúng ta có thể làm việc khi tín hiệu đầu vào/đầu ra là một vectơ ?

Tất nhiên, chúng ta có thể song song hóa tất cả các hoạt động này bằng cách làm việc trên các lô đầu vào. Bằng cách này, các tham số A, B, C, D và đầu vào x(t) và y(t) trở thành vectơ và ma trận. Bằng cách này, việc tính toán sẽ được thực hiện song song cho tất cả các kích thước.



**The importance of the A matrix**



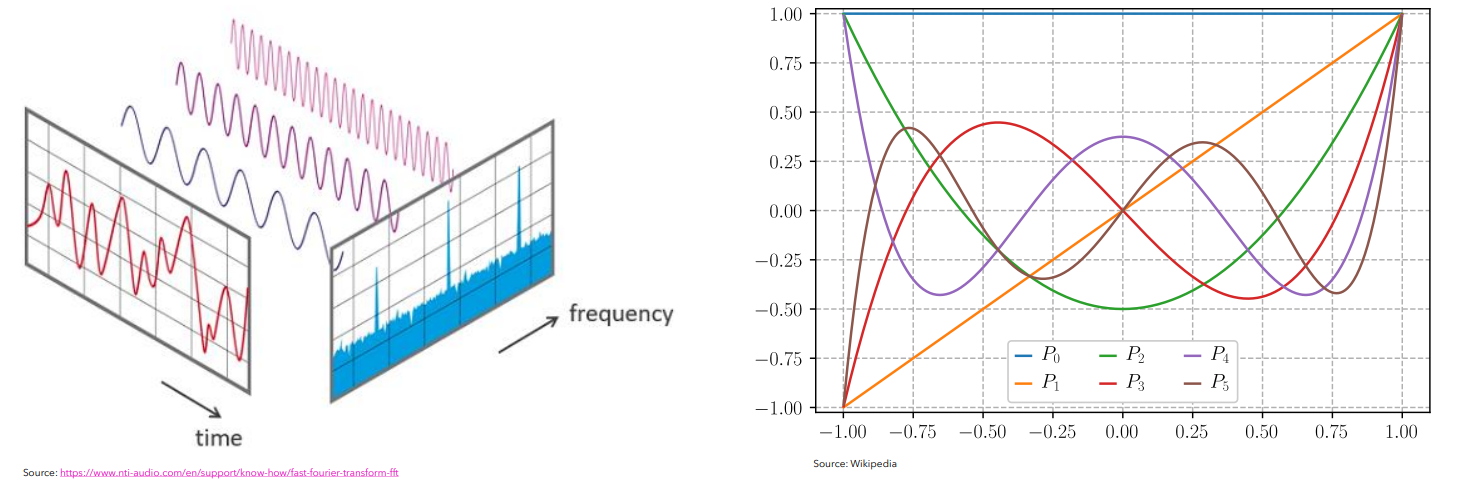
Ma trận A có thể được coi một cách trực quan là ma trận “nắm bắt” thông tin từ trạng thái trước đó để xây dựng trạng thái mới. Nó cũng quyết định cách thông tin này được sao chép kịp thời.

Điều này có nghĩa là chúng ta cần cẩn thận về cấu trúc của ma trận A, nếu không nó có thể không nắm bắt tốt lịch sử của tất cả các đầu vào đã thấy cho đến nay, điều này cần thiết để tạo ra đầu ra tiếp theo. Điều này rất quan trọng đối với các mô hình ngôn ngữ: token tiếp theo do mô hình tạo ra phải phụ thuộc vào các token trước đó.

Để ma trận A hoạt động tốt, tác giả lựa chọn sử dụng lý thuyết HIPPO.

**A little intuition from Fourier transformation**

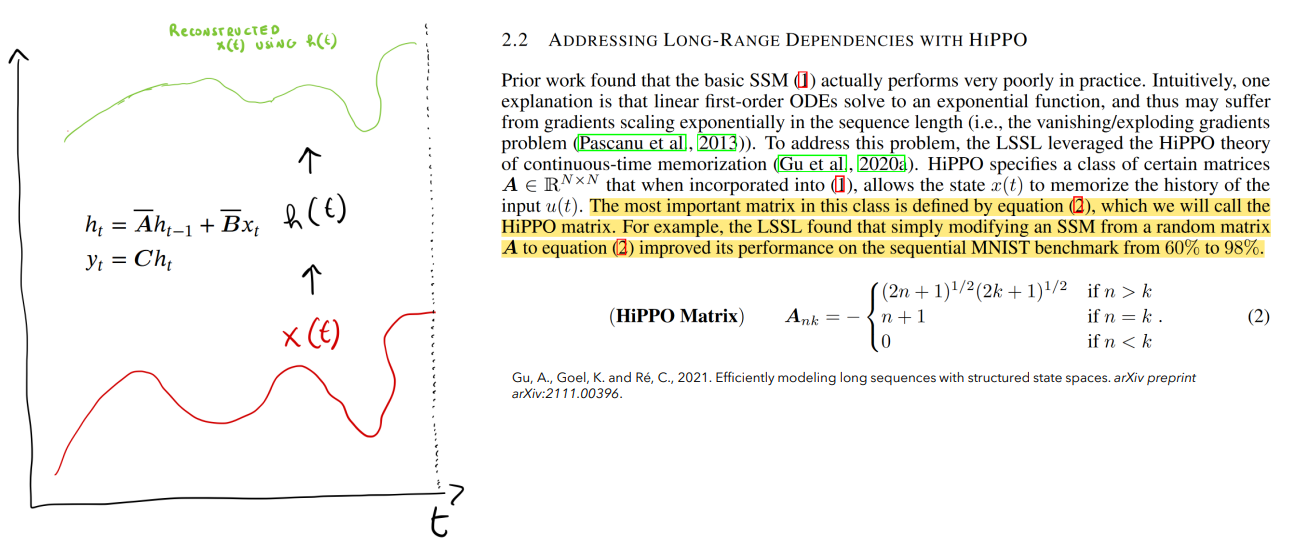
Phép biến đổi Fourier cho phép chúng ta phân tách tín hiệu thành các hàm hình sin, sao cho tổng của các hàm này gần đúng với tín hiệu ban đầu. Với lý thuyết HIPPO, chúng ta làm điều tương tự, nhưng thay vì sử dụng hàm hình sin, chúng ta sử dụng đa thức Legendre



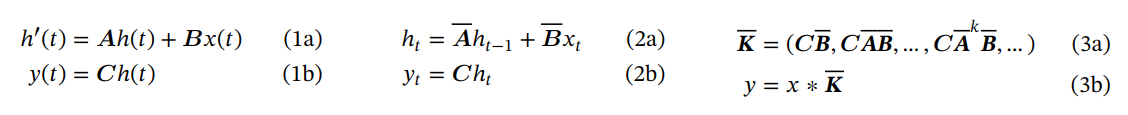
**HIPPO in detail**

Với lý thuyết HIPPO, chúng ta xây dựng ma trận A theo cách nó xấp xỉ tất cả tín hiệu đầu vào được thấy cho đến nay thành một vectơ hệ số (biểu diễn đa thức Legendre).

Sự khác biệt với phép biến đổi Fourier là thay vì xây dựng hoàn hảo tất cả tín hiệu ban đầu cho đến nay, chúng ta xây dựng rất chính xác tín hiệu gần đây hơn, trong khi tín hiệu cũ bị phân rã theo cấp số nhân (như EMA). Vì vậy, trạng thái ℎ(𝑡) thu thập nhiều thông tin hơn về các token được xem gần đây hơn các token cũ hơn.



**Linear Time Invariance (LTI)**

****

Một tính chất quan trọng của các phương trình (1) đến (3) là động lực học của mô hình không đổi theo thời gian. Nói cách khác (∆, A, B, C) và do đó ( , ) cũng cố định cho tất cả các bước thời gian. Thuộc tính này được gọi là bất biến thời gian tuyến tính (LTI), có mối liên hệ sâu sắc với phép truy hồi và tích chập. Một cách không chính thức, tác giả coi LTI SSMs tương đương với bất kỳ phép lặp tuyến tính (2a) hoặc tích chập (3b) nào và sử dụng LTI làm thuật ngữ chung cho các lớp mô hình này.

Cho đến nay, tất cả các structured SSMs đều là LTI (ví dụ: được tính dưới dạng tích chập).

* 1. **Selective State Space Models**

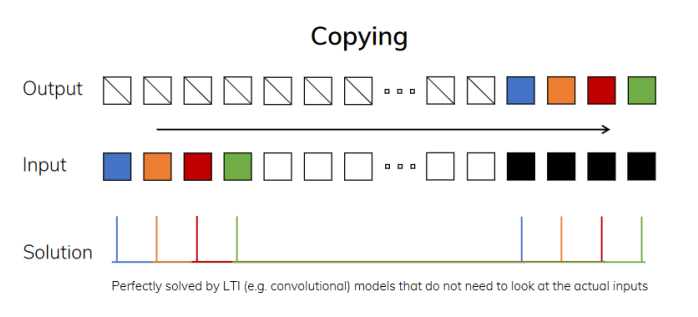
**Motivation: Selection as a Means of Compression**

Từ recurrent view, động lực không đổi của chúng (ví dụ: chuyển đổi ( , ) trong (2)) không thể cho phép chúng chọn thông tin chính xác từ ngữ cảnh của chúng hoặc ảnh hưởng đến trạng thái ẩn được truyền dọc theo chuỗi theo cách phụ thuộc vào đầu vào. Từ convolutional view, người ta biết rằng tích chập toàn cục có thể giải quyết nhiệm vụ sao chép thông thường vì nó chỉ yêu cầu nhận thức về thời gian, nhưng lại gặp khó khăn với nhiệm vụ sao chép chọn lọc vì thiếu nhận thức về nội dung. Cụ thể hơn, khoảng cách giữa đầu vào và đầu ra là khác nhau và không thể được mô hình hóa bằng các kernel tích chập tĩnh.

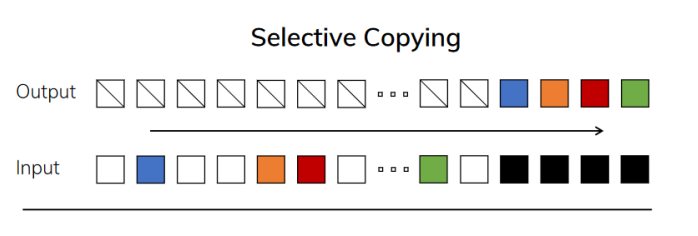
Tóm lại, sự đánh đổi giữa hiệu suất và hiệu quả của các mô hình chuỗi được đặc trưng bởi khả năng nén tốt của trạng thái: mô hình hiệu suất cao phải có một trạng thái nhỏ, trong khi mô hình hiệu quả phải có một trạng thái chứa đựng đầy đủ thông tin cần thiết từ ngữ cảnh. Một nguyên tắc cơ bản trong xây dựng mô hình chuỗi là sự chọn lọc: khả năng tập trung vào hoặc loại bỏ đầu vào vào một trạng thái tuần tự.

**Improving SSMs with Selection**

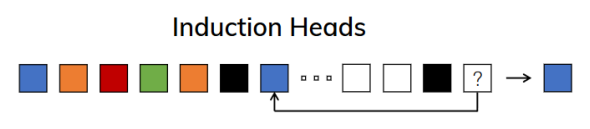
Một phương pháp kết hợp cơ chế lựa chọn vào các mô hình là để các tham số của chúng ảnh hưởng đến tương tác dọc theo chuỗi (ví dụ: recurrent dynamic của RNN hoặc convolution kernel của CNN) phụ thuộc vào đầu vào.



* Viết lại từng token đầu vào, nhưng thay đổi theo thời gian.
* Điều này có thể được thực hiện bởi SSM và độ trễ thời gian có thể được học bằng phép tích chập.



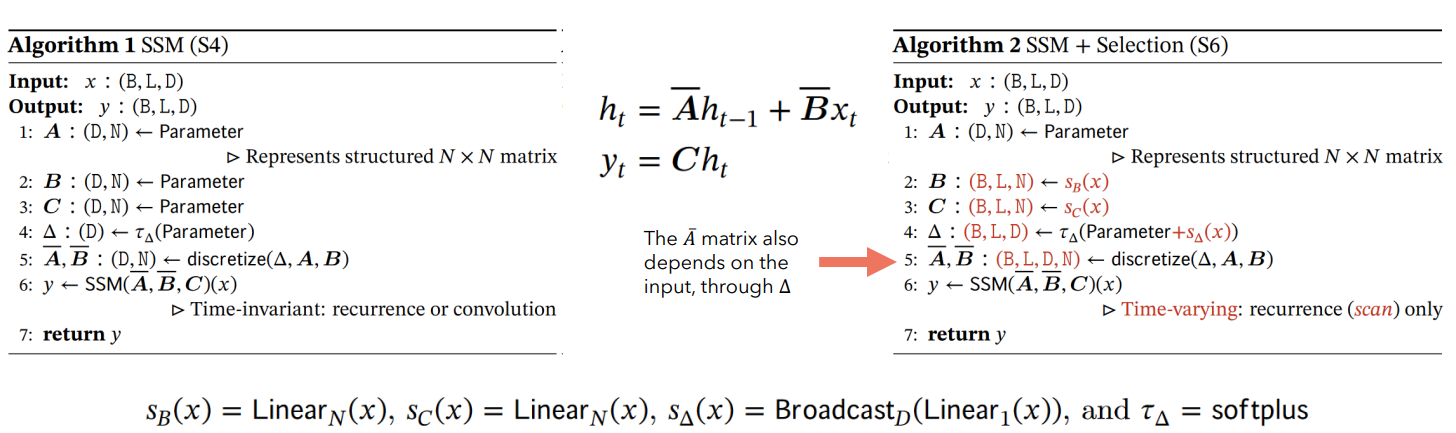
* Đưa ra nhận xét trên Twitter, hãy viết lại nhận xét bằng cách xóa tất cả các từ xấu (token màu trắng).
* Điều này không thể được thực hiện bởi SSM vì nó yêu cầu suy luận nhận biết nội dung, điều mà SSM không thể làm được vì chúng bất biến theo thời gian (có nghĩa là các tham số A, B, C, D giống nhau cho mọi token mà nó tạo ra).

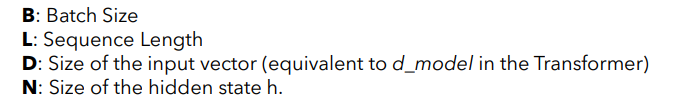


* Ví dụ: với “few-shot” prompt, chúng ta có thể “dạy” các nhiệm vụ mới cho LLM và cách thực hiện chúng. Với mô hình dựa trên Transformer, nhiệm vụ này có thể được thực hiện “dễ dàng”, bởi vì các mô hình dựa trên Transformer có thể truy cập các token trước đó khi tạo token hiện tại, vì vậy chúng có thể “gọi lại lịch sử trước đó”.

Nhiệm vụ này không thể được thực hiện bởi SSM bất biến theo thời gian, bởi vì nó không thể “chọn” token trước đó để gọi lại từ lịch sử của nó.

**Mamba: a selective SSM**

****

****

Không thể đánh giá Mamba bằng cách sử dụng tích chập, bởi vì các tham số của mô hình khác nhau đối với từng token đầu vào và ngay cả khi muốn chạy tích chập, sẽ cần xây dựng L (độ dài chuỗi) các convolutional kernels khác nhau L (độ dài chuỗi)

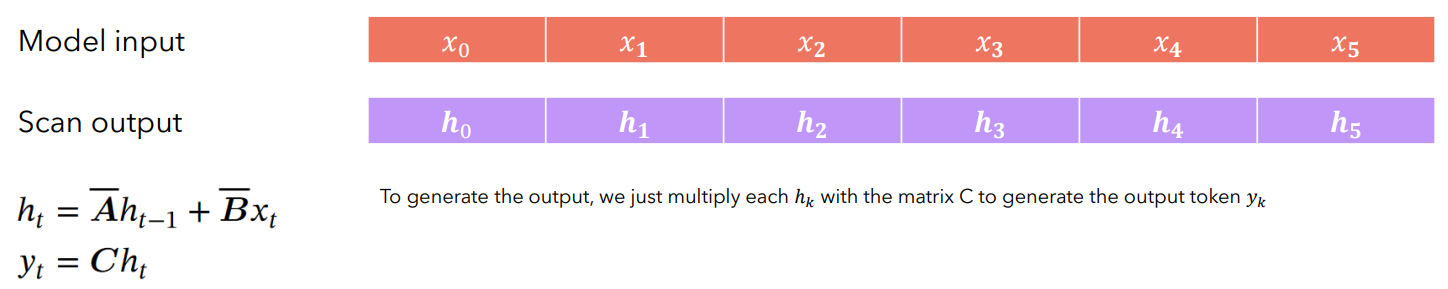
SSM + Selection (S6 🐍) không thực hiện tích chập. Nó bổ sung thêm các cơ chế lựa chọn S(x) và S(x) Các phép toán tuyến tính để giúp mô hình ghi nhớ và quên những gì.

Cơ chế lựa chọn chỉ đơn giản là các lớp tuyến tính bao bọc các tham số B, C và delta.

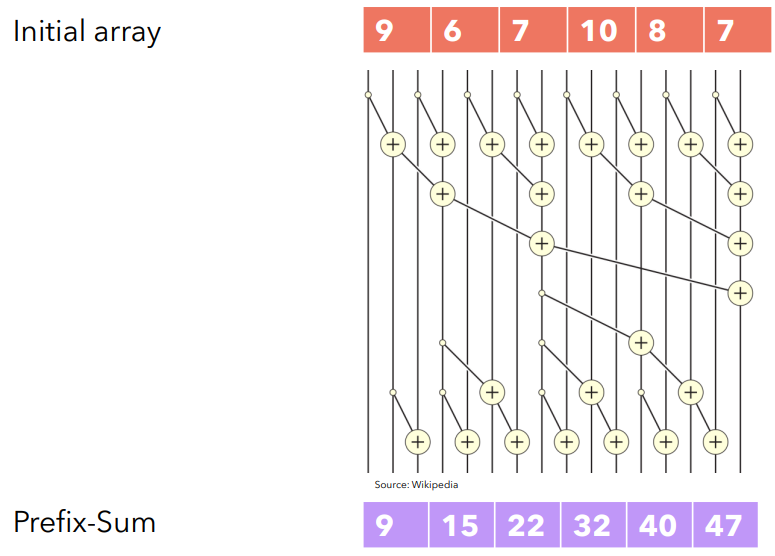
**The scan operation**

Công thức truy hồi của mô hình SSM cũng có thể được coi là thao tác quét (scan), trong đó mỗi trạng thái là tổng của trạng thái trước đó và đầu vào hiện tại.

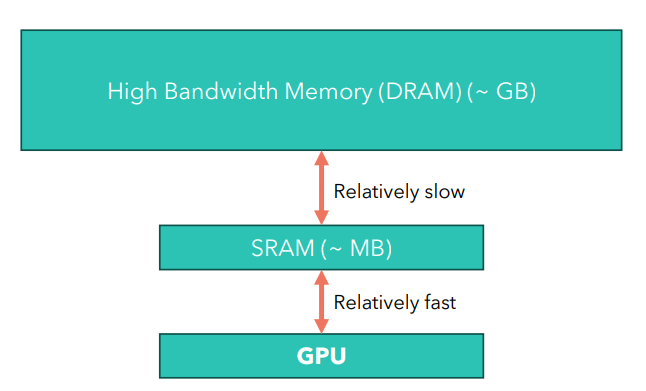
Chúng ta có thể dễ dàng tính toán nó bằng vòng lặp for trong thời gian tuyến tính.



Chúng ta có thể sinh ra nhiều luồng để thực hiện binary operation song song, đồng bộ hóa ở mỗi bước. Độ phức tạp về thời gian thay vì 𝑶(𝑵) được giảm xuống còn 𝑶(𝑵/𝑻) trong đó 𝑻 là số luồng song song.



**Exploiting the memory hierarchy**

****

Bởi vì GPU không nhanh chóng trong việc di chuyển tensor (dữ liệu) giữa các vùng nhớ, nhưng lại rất nhanh chóng trong việc thực hiện các phép toán tính toán, nên đôi khi vấn đề trong thuật toán của chúng ta không phải là số lượng phép toán chúng ta thực hiện, mà là số lượng tensors chúng ta di chuyển qua lại trong các cấp độ bộ nhớ khác nhau. Trong trường hợp này, chúng ta nói rằng phép toán đó bị hạn chế bởi việc thực hiện các thao tác I/O, hay nói cách khác, phép toán này là IObound.

Ý tưởng chính là tận dụng các thuộc tính của bộ tăng tốc hiện đại (GPU) để hiện thực hóa trạng thái ℎ chỉ ở các mức phân cấp bộ nhớ hiệu quả hơn. Đặc biệt, hầu hết các phép toán (ngoại trừ phép nhân ma trận) đều bị giới hạn bởi băng thông bộ nhớ. Điều này bao gồm hoạt động scan của tác giả và tác giả sử dụng kernel fusion để giảm lượng IO bộ nhớ, dẫn đến tăng tốc đáng kể so với triển khai tiêu chuẩn.

Cụ thể, thay vì chuẩn bị đầu vào scan ( , ) có kích thước (𝙱, 𝙻, 𝙳, 𝙽) trong GPU HBM (bộ nhớ băng thông cao), tác giả tải trực tiếp các tham số SSM (∆, A, B, C) từ slow HBM sang fast SRAM, thực hiện phép rời rạc hóa và truy hồi trong SRAM, sau đó ghi kết quả đầu ra cuối cùng có kích thước (𝙱, 𝙻, 𝙳) trở lại HBM

**Mamba: Kernel fusion**

Khi thực hiện thao tác tensor, khung học sâu (ví dụ: PyTorch) sẽ tải tensor vào bộ nhớ nhanh (SRAM) của GPU, thực hiện thao tác (ví dụ: nhân ma trận) rồi lưu lại kết quả vào Bộ nhớ băng thông cao của GPU. Điều gì sẽ xảy ra nếu chúng ta thực hiện nhiều thao tác trên cùng một tensor (ví dụ: 3 thao tác)? Sau đó, khung học sâu sẽ thực hiện trình tự sau:

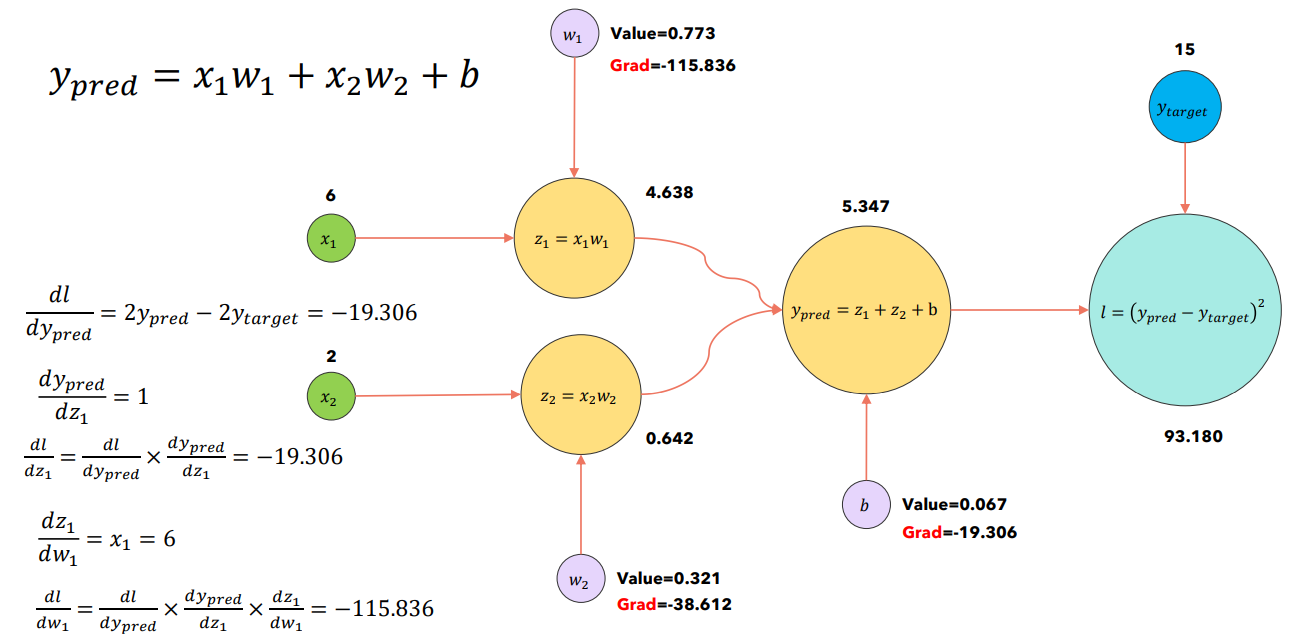
1. Tải đầu vào từ HBM vào SRAM, tính toán thao tác đầu tiên (nhân CUDA tương ứng với thao tác đầu tiên) rồi lưu lại kết quả vào HBM

2. Tải kết quả trước đó từ HBM vào SRAM, tính toán thao tác thứ hai (nhân CUDA tương ứng với thao tác thứ hai) rồi lưu lại kết quả vào HBM 3. Tải kết quả trước đó từ HBM vào SRAM, tính toán thao tác thứ ba (nhân CUDA tương ứng với thao tác thứ ba) và sau đó lưu lại kết quả vào SRAM.

Như có thể thấy, tổng thời gian bị chiếm dụng bởi các hoạt động sao chép mà chúng ta đang thực hiện, vì chúng ta biết rằng GPU sao chép dữ liệu tương đối chậm hơn so với các hoạt động tính toán. Để thực hiện một chuỗi các thao tác nhanh hơn, chúng ta có thể hợp nhất các hạt nhân CUDA để tạo ra một hạt nhân CUDA tùy chỉnh thực hiện lần lượt ba thao tác mà không cần sao chép kết quả trung gian vào HBM và chỉ sao chép kết quả cuối cùng.

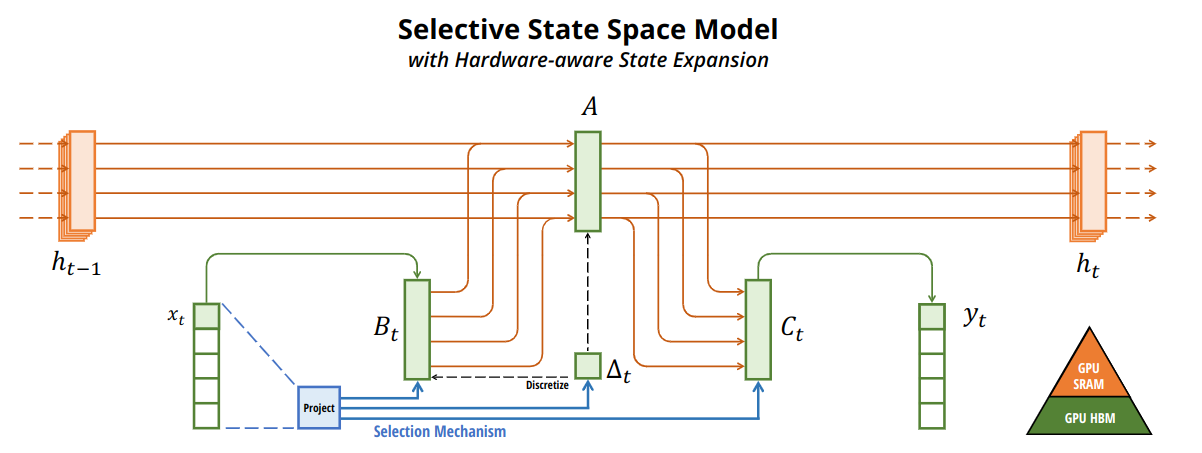
**Mamba: Recomputation**

Khi đào tạo một mô hình deep learning, nó sẽ được chuyển đổi thành biểu đồ tính toán. Khi thực hiện lan truyền ngược, để tính toán độ dốc tại mỗi nút, chúng ta cần lưu trữ các giá trị đầu ra của bước chuyển tiếp như hình dưới đây:



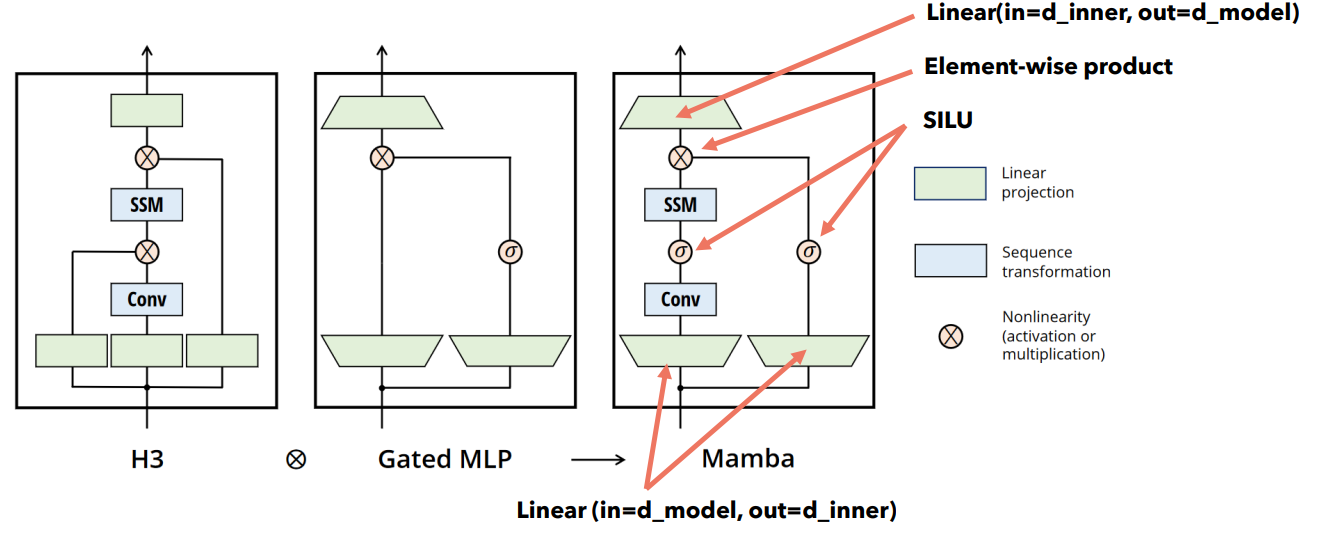
Vì việc lưu các kích hoạt (activations) vào bộ đệm và sau đó sử dụng lại chúng trong quá trình lan truyền ngược có nghĩa là chúng ta cần lưu chúng vào HBM và sau đó sao chép chúng trở lại từ HBM trong quá trình lan truyền ngược, nên việc tính toán lại chúng trong quá trình lan truyền ngược có thể nhanh hơn!

**Mamba: the Mamba Block**

****

Hình 1: SSM có cấu trúc ánh xạ độc lập từng kênh (ví dụ: 𝐷 = 5) của đầu vào 𝑥 đến đầu ra 𝑦 thông qua trạng thái tiềm ẩn có chiều cao hơn ℎ (ví dụ: 𝑁 = 4). Các SSM trước đó tránh hiện thực hóa trạng thái hiệu quả lớn này (𝐷𝑁, nhân kích thước lô 𝐵 và độ dài chuỗi 𝐿) thông qua các đường tính toán thay thế thông minh yêu cầu tính bất biến theo thời gian: các tham số (∆, A, B, C) không đổi theo thời gian. Cơ chế lựa chọn của tác giả bổ sung thêm các động lực phụ thuộc vào đầu vào, điều này cũng yêu cầu thuật toán nhận biết phần cứng cẩn thận để chỉ hiện thực hóa các trạng thái mở rộng ở các mức hiệu quả hơn trong hệ thống phân cấp bộ nhớ GPU.

Mamba được xây dựng bằng cách xếp chồng nhiều lớp của khối Mamba, được hiển thị bên dưới. Điều này rất giống với các lớp xếp chồng lên nhau của mô hình Transformer. Kiến trúc Mamba bắt nguồn từ kiến ​​trúc Hungry Hungry Hippo (H3).

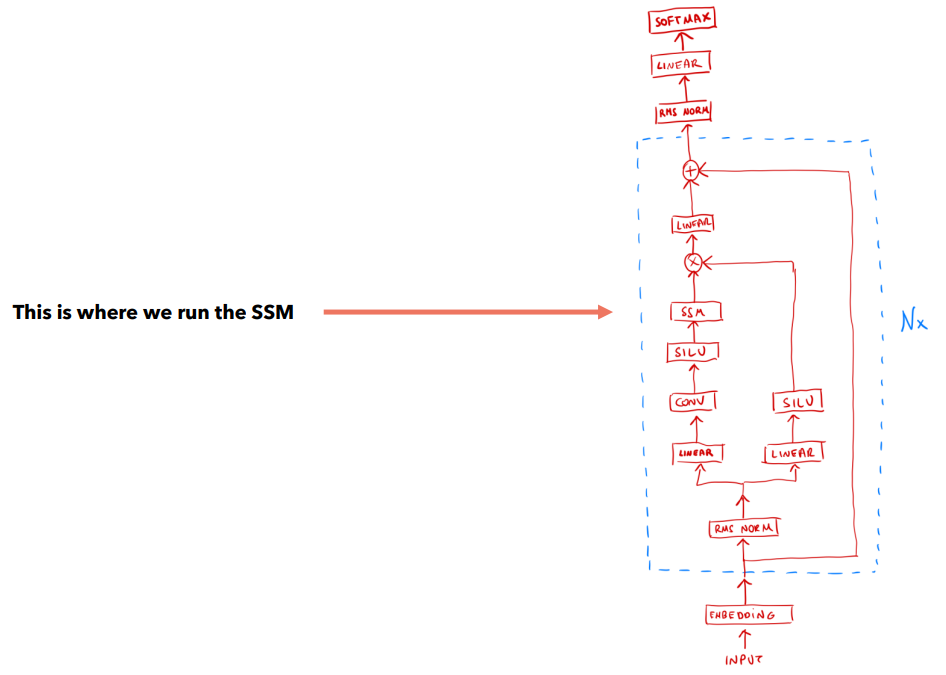


Các khối SSM chọn lọc có thể được kết hợp dưới dạng các phép biến đổi độc lập thành mạng thần kinh, giống như cách bạn thực hiện với Ô RNN như LSTM hoặc GRU. Kiến trúc đầy đủ của khối Mamba được trình bày bên trên và không chỉ có mô-đun SSM đã đề cập ở trên. Có các phép chiếu tuyến tính, tích chập và phi tuyến tính xung quanh khối SSM trong khối Mamba lớn hơn.

Đầu tiên, chiếu đầu vào lên thông qua một lớp tuyến tính giúp mở rộng kích thước của đầu vào, thêm một residual connection ở phía bên phải tương tự như transformer.

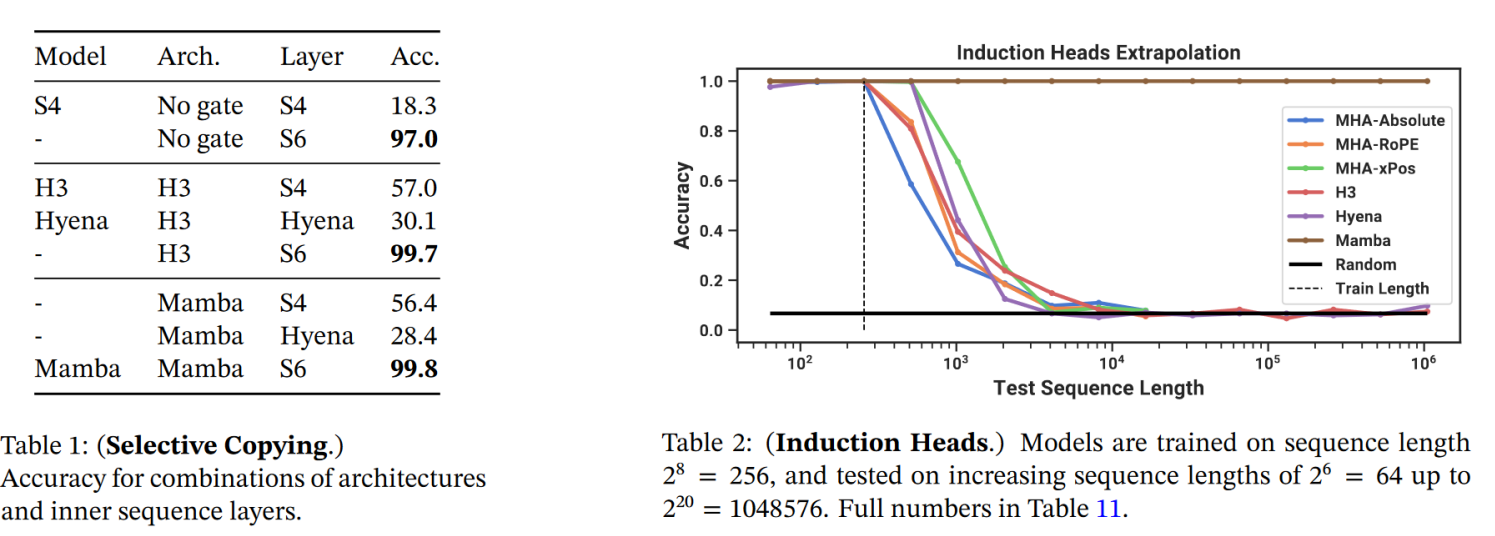
Sau đó, chạy tích chập 1D trên lớp tuyến tính, chuyển nó qua chức năng kích hoạt SiLU / Swish, trước khi nó đến khối SSM mà chúng ta đã nói ở trên.

Sau đó, residual path kết nối trở lại với đầu ra của SSM và chúng thu nhỏ kích thước trở lại giống như đầu vào với lớp tuyến tính cuối cùng.

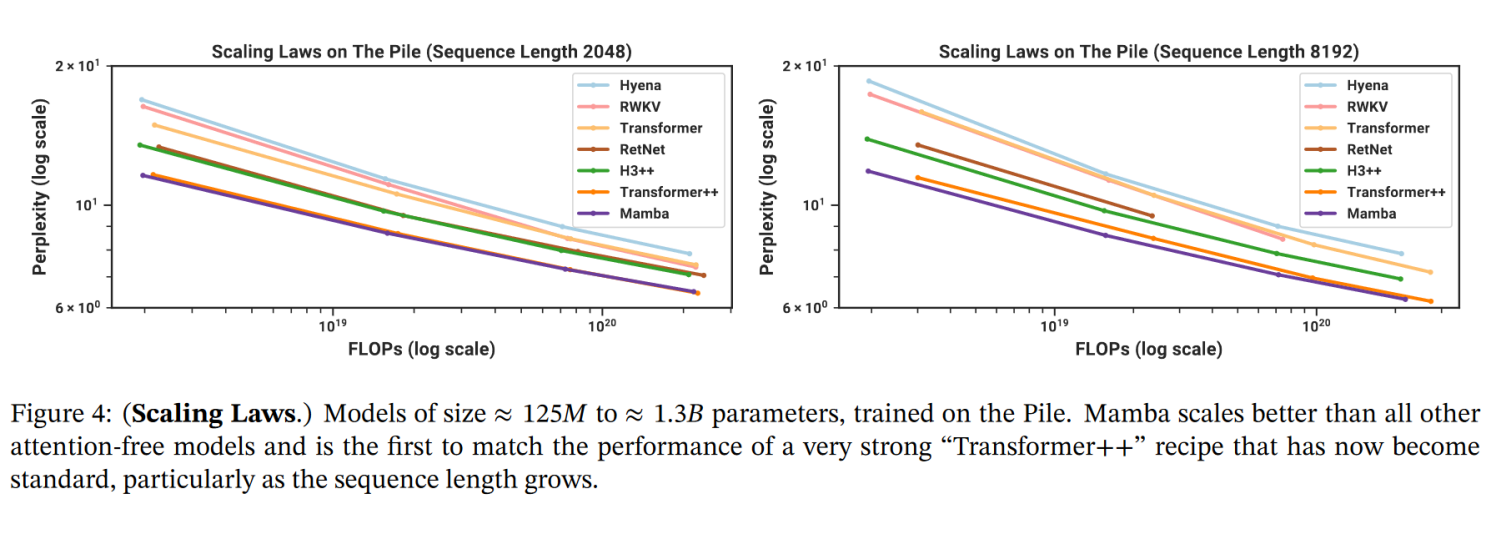


1. **Kết quả**

**Mamba: the performance**

****

Bảng 2 cho thấy Mamba - hay chính xác hơn là lớp SSM chọn lọc của nó, có khả năng giải quyết nhiệm vụ một cách hoàn hảo nhờ khả năng ghi nhớ có chọn lọc token liên quan trong khi bỏ qua mọi thứ khác ở giữa. Nó khái quát hoàn hảo thành các chuỗi có độ dài hàng triệu, hoặc dài hơn 4000× so với những gì nó thấy trong quá trình đào tạo, trong khi không có phương pháp nào khác vượt quá 2×.

****

Hình 4 thể hiện quy luật chia tỷ lệ theo giao thức Chinchilla (Hoffmann và cộng sự 2022) tiêu chuẩn, trên các mô hình từ tham số ≈ 125𝑀 đến ≈ 1.3𝐵. Mamba là mô hình không cần chú ý đầu tiên phù hợp với hiệu suất của công thức Transformer rất mạnh (Transformer++) hiện đã trở thành tiêu chuẩn, đặc biệt khi độ dài chuỗi tăng lên.

1. **Note**

RNN nén bối cảnh thành một trạng thái nhỏ hơn, và sau đó sử dụng nó để dự đoán trình tự đầu ra

Attention hoàn toàn không nén bối cảnh, nó cung cấp cho mô hình toàn quyền truy cập vào lịch sử

Trực giác đằng sau việc sử dụng SSM trong mạng thần kinh là chuyển đổi chuỗi đầu vào thành không gian chiều cao hơn (trạng thái tiềm ẩn), trong đó động lực của nó có thể được nắm bắt hiệu quả hơn trước khi chiếu nó xuống đầu ra mong muốn. Các ma trận A, B và C chuyển đổi dữ liệu đầu vào thành không gian tiềm ẩn phát triển theo thời gian, cho phép mô hình nắm bắt được các phụ thuộc theo thời gian. Phiên bản rời rạc của SSM giúp cho việc áp dụng khái niệm thời gian liên tục này cho dữ liệu thời gian rời rạc, giống như các chuỗi trong các tác vụ học máy, trở nên khả thi về mặt tính toán.

Một kết nối quan trọng: cơ chế cổng cổ điển của RNN là một ví dụ về cơ chế lựa chọn cho SSM.

* Lọc có chọn lọc các token nhiễu không liên quan
* Đặt lại trạng thái để xóa lịch sử không liên quan
* Việc sử dụng các độ phân giải bước “delta” khác nhau giúp cân bằng lượng thông tin chúng ta muốn lưu từ đầu vào hiện tại so với việc cập nhật trạng thái từ một cửa sổ lớn hơn.

Những đóng góp của tác giả trong bài viết này cho SSM như sau:

* Một cơ chế lựa chọn cho phép mô hình lọc ra những thông tin không liên quan và ghi nhớ thông tin liên quan vô thời hạn.
* Một thuật toán nhận biết phần cứng tính toán mô hình định kỳ nhưng không hiện thực hóa ở trạng thái mở rộng, tối ưu hóa bố cục bộ nhớ GPU.

Sự kết hợp của hai kỹ thuật này mang lại các tính chất sau:

* Kết quả chất lượng cao về ngôn ngữ và dữ liệu khác với chuỗi dài
* Đào tạo và suy luận nhanh
  + Bộ nhớ và tính toán quy mô tuyến tính theo độ dài chuỗi trong quá trình đào tạo
  + Suy luận liên quan đến việc hủy kiểm soát từng phần tử của mô hình với thời gian không đổi trên mỗi bước, không có bộ đệm của các phần tử trước đó
* Ngữ cảnh dài - cải thiện hiệu suất trên dữ liệu thực lên tới độ dài chuỗi 1 triệu

1. **Tham khảo**

<https://arxiv.org/abs/2312.00752>

<https://blog.oxen.ai/mamba-linear-time-sequence-modeling-with-selective-state-spaces-arxiv-dives/>

<https://www.youtube.com/watch?v=8Q_tqwpTpVU>

<https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3>

<https://huggingface.co/blog/lbourdois/get-on-the-ssm-train>

<https://srush.github.io/annotated-s4/#discrete-time-ssm-the-recurrent-representation>

-------------------Ending Khóoo ---------------------