	VIETTEL AI RACE	Public 113
	ATTENTION VÀ SỰ HÌNH THÀNH CỦA MÔ HÌNH TRANSFORMER	Lần ban hành: 1

## 1. Định nghĩa:

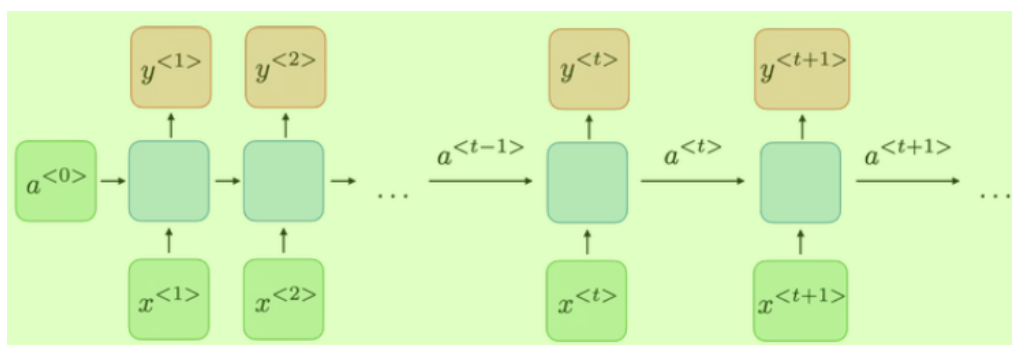
Attention là một kỹ thuật được sử dụng trong các mạng neural, kỹ thuật này được sử dụng trong các mô hình thực hiện các task như dịch máy hay ngôn ngữ tự nhiên. BERT và GPT là 2 mô hình điển hình có sử dụng Attention. Attention là thành phần chính tạo nên sự đình đám của mô hình Transformer, mô hình này chính là sự đột phá trong các bài toán xử lý của NLP so với các mạng neural hồi quy. Vậy Attention là gì mà tại sao nó lại là sự khác biệt đến vậy, hãy cùng tôi đi tìm hiểu trong bài viết ngày hôm nay với tiêu đề “Attention và sự hình thành của mô hình Transformer”

## 2. Động lực cho sự phát triển của Attention

### 2.1. Recurrent Neural Network (RNN) và sự hạn chế đáng kể

#### 2.1.1. Ý tưởng cốt lõi của RNN

Con người chúng ta không thể bắt đầu suy nghĩ của mình tại tất cả các thời điểm, cũng giống như việc bạn đang đọc bài viết này, bạn hiểu mỗi chữ ở đây dựa vào các chữ mà bạn đã đọc và hiểu trước đó, chứ không phải đọc xong là quên chữ đó đi rồi đến lúc gặp thì lại phải đọc và tiếp thu lại. Giống như trong bài toán của chúng ta. Các mô hình mạng nơ-ron truyền thống lại không thể làm được việc trên. Vì vậy mạng nơ-ron hồi quy (RNN) được sinh ra để giải quyết việc đó. Mạng này chứa các vòng lặp bên trong cho phép nó lưu lại các thông tin đã nhận được. RNN là một thuật toán quan trọng trong xử lý thông tin dạng chuỗi hay nói cách khác là dạng xử lý tuần tự.



Cấu trúc cơ bản của RNN

	VIETTEL AI RACE	Public 113
	ATTENTION VÀ SỰ HÌNH THÀNH CỦA MÔ HÌNH TRANSFORMER	Lần ban hành: 1

Vậy như nào là xử lý tuần tự - Xử lý tuần tự là mỗi block sẽ lấy thông tin của block trước và input hiện tại làm đầu vào

Tại mỗi bước  $t$ , giá trị kích hoạt  $a^t$  và đầu ra  $y^t$  được biểu diễn như sau:

$$a^t = g_1.(W_{aa}.a^{t-1} + W_{ax}.x^t + b_a)$$

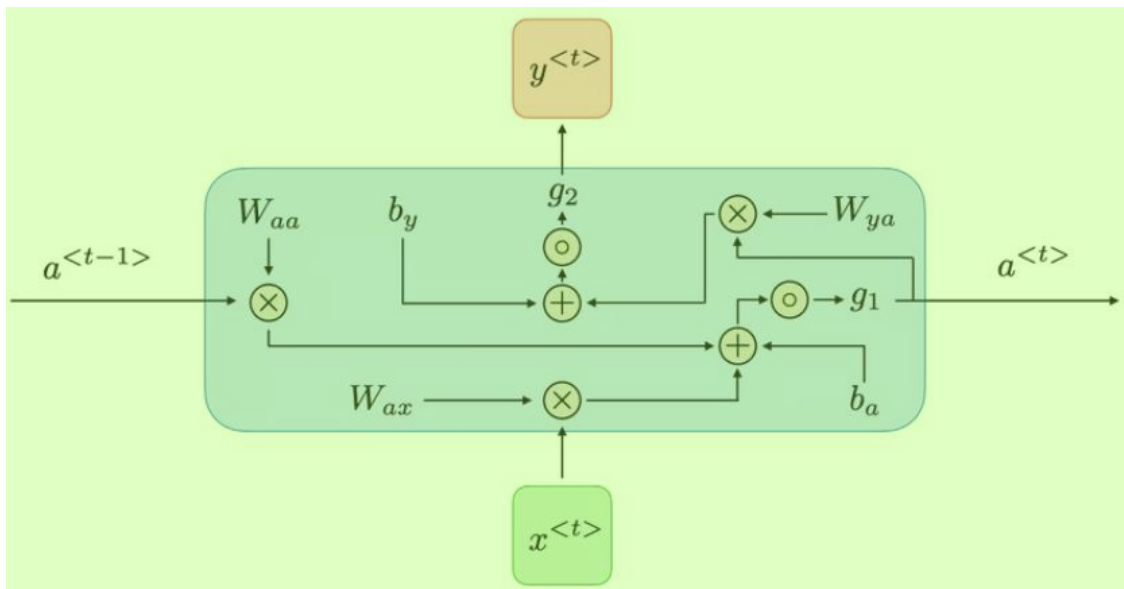
ta có thể viết gọn lại như sau:

$$a^t = g_1.((W_{aa}W_{ax}) \begin{pmatrix} a^{t-1} \\ x^t \end{pmatrix} + b_a)$$

$$a^t = g_1.(W \begin{pmatrix} a^{t-1} \\ x^t \end{pmatrix} + b_a)$$

Từ với  $a^t$  ta có công thức tính đầu ra tương ứng  $y^t$ :

$$y^t = g_2.(W_{ya}.a^t + b_y)$$



Cấu trúc một block trong RNN

### 2.1.2. Ưu điểm và nhược điểm của RNN

	<b>VIETTEL AI RACE</b>	Public 113
	<b>ATTENTION VÀ SỰ HÌNH THÀNH CỦA MÔ HÌNH TRANSFORMER</b>	Lần ban hành: 1

<b>Ưu điểm</b>	<b>Nhược điểm</b>
Khả năng xử lý các chuỗi đầu vào có độ dài khác nhau	Tính toán khá chậm
Kích cỡ mô hình không bị tăng lên theo kích thước đầu vào	Khó truy cập lại thông tin đã đi qua ở một khoảng thời gian dài trước đó - hay còn gọi là bị quên thông tin khi gặp nhiều thông tin mới
Quá trình tính toán có sử dụng thông tin trước đó	Phải thực hiện tuần tự nên không tận dụng triệt để được khả năng tính toán song song của GPU
Trọng số được chia sẻ trong suốt quá trình học	Vanishing gradient

## 2.2. Vấn đề gặp phải của Long Short Term Memory (LSTM) <sup>3</sup>

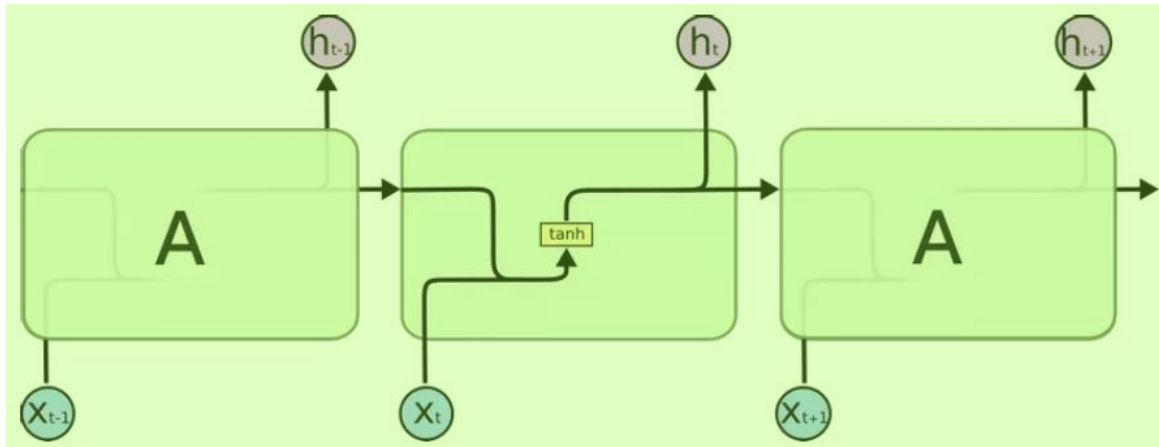
### 2.2.1 Ý tưởng cốt lõi của LSTM (Long short term memory) <sup>4</sup>

LSTM là một dạng đặc biệt của RNN, nó có khả năng học các thông tin ở xa. Về cơ bản thì LSTM và RNN không khác nhau là mấy nhưng LSTM có cải tiến một số phép tính trong 1 hidden state và nó đã hiệu quả. Hiệu quả như nào thì chúng ta hãy cùng đọc tiếp nhé!

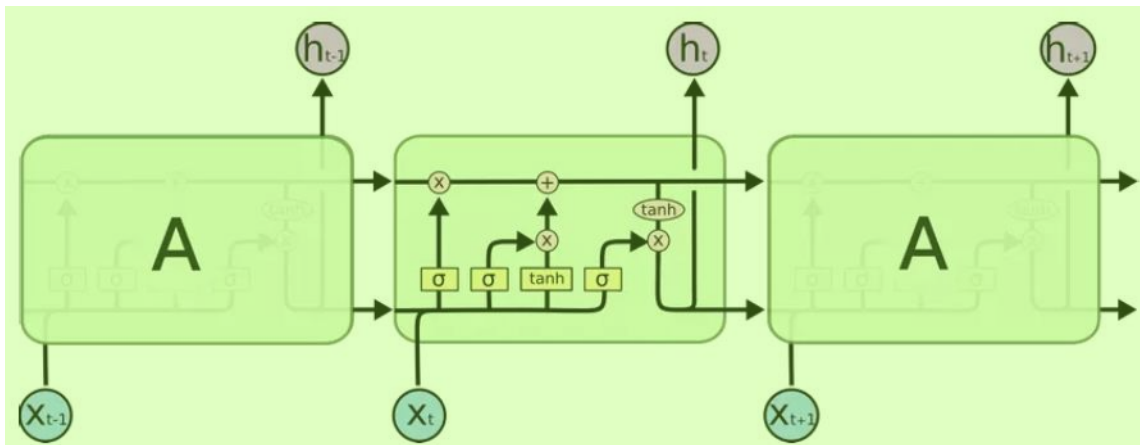
	VIETTEL AI RACE	Public 113
	ATTENTION VÀ SỰ HÌNH THÀNH CỦA MÔ HÌNH TRANSFORMER	Lần ban hành: 1

Cấu trúc của LSTM không khác gì RNN, nhưng sự cải tiến ở đây nằm ở phần tính toán trong từng hidden state như sau: Thay vì chỉ có một tầng mạng nơ-ron, LSTM thiết kế với 4 tầng mạng nơ-ron tương tác với nhau một cách rất đặc biệt.

Dưới đây là 2 hình ảnh biểu diễn sự khác nhau giữa RNN và LSTM




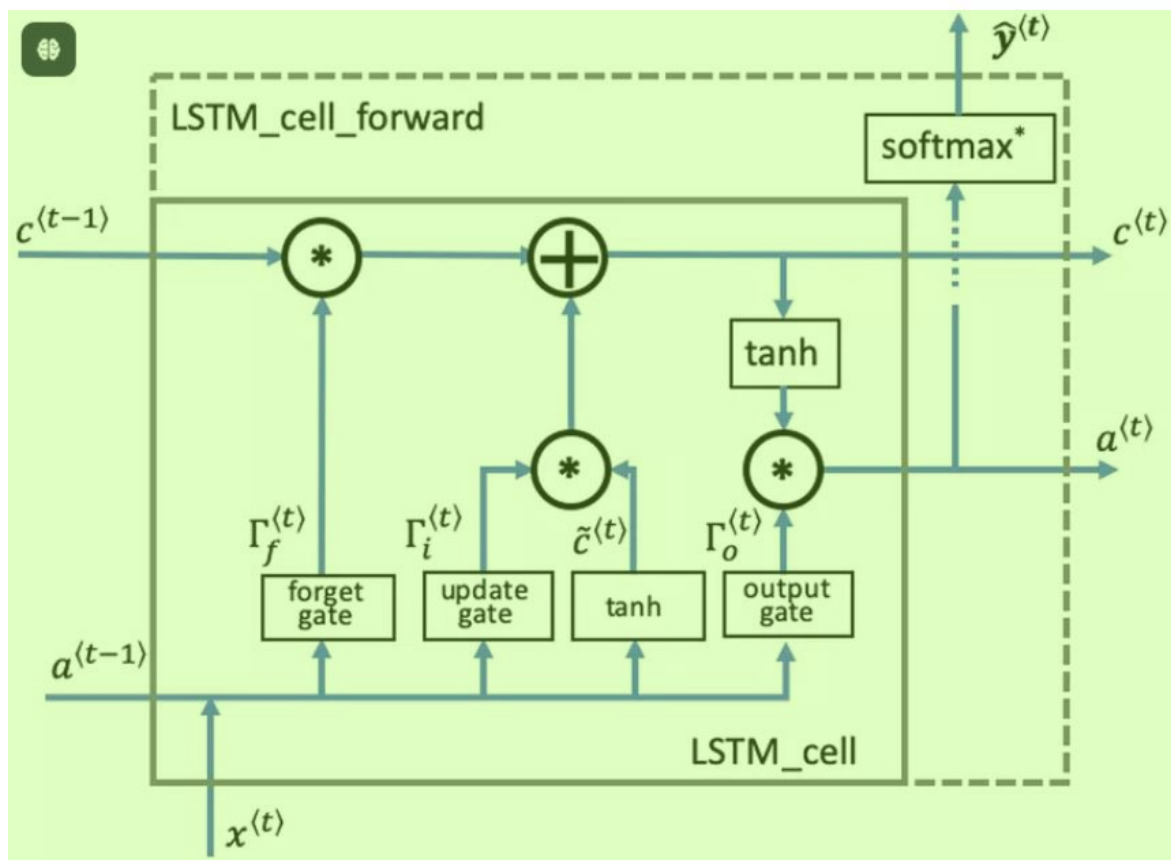
RNN with tanh function



LSTM with tanh and sigmoid functions

Chìa khóa để giúp LSTM có thể truyền tải thông tin giữa các hidden state một cách xuyên suốt chính là cell state (hình dưới):

	VIETTEL AI RACE	Public 113
	ATTENTION VÀ SỰ HÌNH THÀNH CỦA MÔ HÌNH TRANSFORMER	Lần ban hành: 1



LSTM cell state

Forget gate  $\Gamma_f$

Đầu ra là hàm sigmoid chứa các giá trị từ 0 đến 1.

Nếu forget gate có giá trị bằng 0, LSTM sẽ "quên" trạng thái được lưu trữ trong đơn vị tương ứng của trạng thái cell trước đó.

Nếu cổng quên có giá trị bằng 1, LSTM sẽ chủ yếu ghi nhớ giá trị tương ứng ở trạng thái được lưu trữ.

$$\Gamma_f^{(t)} = \sigma(\mathbf{W}_f[\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_f)$$

Candidate value  $\tilde{c}^{(t)}$

	VIETTEL AI RACE	Public 113
	ATTENTION VÀ SỰ HÌNH THÀNH CỦA MÔ HÌNH TRANSFORMER	Lần ban hành: 1

Chứa thông tin có thể được lưu trữ từ time step hiện tại.

$$\tilde{\mathbf{c}}^{(t)} = \tanh \left( \mathbf{W}_c [\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_c \right)$$

Update gate  $\Gamma_i$

Quyết định xem phần thông tin nào của  $\tilde{\mathbf{c}}^{(t)}$  có thể thêm vào  $\mathbf{c}^{(t)}$ .

$$\Gamma_i^{(t)} = \sigma(\mathbf{W}_i [\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_i)$$

Cell state  $\mathbf{c}^{(t)}$

Là bộ nhớ trong của LSTM. Cell state như 1 băng tải truyền các thông tin cần thiết xuyên suốt cả quá trình, qua các nút mạng và chỉ tương tác tuyến tính 1 chút. Vì vậy thông tin có thể truyền đi thông suốt mà không bị thay đổi.

$$\mathbf{c}^{(t)} = \Gamma_f^{(t)} * \mathbf{c}^{(t-1)} + \Gamma_i^{(t)} * \tilde{\mathbf{c}}^{(t)}$$

	VIETTEL AI RACE	Public 113
	ATTENTION VÀ SỰ HÌNH THÀNH CỦA MÔ HÌNH TRANSFORMER	Lần ban hành: 1

## Output gate $\Gamma_o$

Cổng điều chỉnh lượng thông tin đầu ra của cell hiện tại và lượng thông tin truyền tới trạng thái tiếp theo.

$$\Gamma_o^{(t)} = \sigma(\mathbf{W}_o[\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_o)$$

## Hidden state $\mathbf{a}^{(t)}$

Được sử dụng để xác định ba cổng ( $\Gamma_f, \Gamma_u, \Gamma_o$ ) của time step tiếp theo.

$$\mathbf{a}^{(t)} = \Gamma_o^{(t)} * \tanh(\mathbf{c}^{(t)})$$

## Prediction $\mathbf{y}_{pred}^{(t)}$

Dự đoán trong trường hợp sử dụng này là phân loại, vì vậy bạn sẽ sử dụng softmax.

$$\mathbf{y}_{pred}^{(t)} = \text{softmax}(\mathbf{W}_y \mathbf{a}^{(t)} + \mathbf{b}_y)$$