

Recurrent Neural Network (RNN) Long Short-Term Memory (LSTM)

Mentor: PĐ Hải

Người trình bày: Nguyễn Minh Hiếu

PAYT CLUB PTIT

September 2025

1. Recurrent Neural Networks
2. Long Short-Term Memory
3. Transformers

1: Recurrent Neural Networks

Đặt vấn đề

- Bài toán: Nhận diện hành động trong video 30s.
 - ▶ Input: Video 30s – > trích xuất 30 ảnh (1 FPS)
 - ▶ Output: Phân loại hành động (Đứng, ngồi, chạy,...)
- Đặc điểm dữ liệu:
 - ▶ Các ảnh có **thứ tự thời gian** quan trọng
 - ▶ Đảo thứ tự ảnh có thể **thay đổi hoàn toàn nội dung**
 - ▶ Một ảnh đơn lẻ **không thể mô tả đầy đủ** hành động

- Nếu sử dụng CNN thì có hạn chế:
 - ▶ CNN chỉ xử lý được **từng ảnh riêng lẻ**
 - ▶ không nắm bắt được **mối quan hệ thời gian** giữa các khung hình

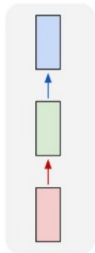
=> Sequences Data

=> RNN ra đời:

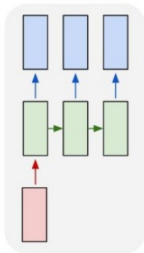
- ▶ Có thể xử lý **dữ liệu dạng chuỗi** có thứ tự
- ▶ Có khả năng **hiểu được ngữ cảnh thời gian** giữa các frame

Phân loại bài toán RNN

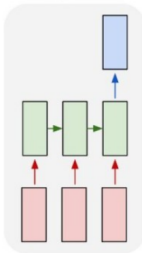
one to one



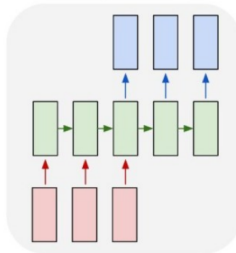
one to many



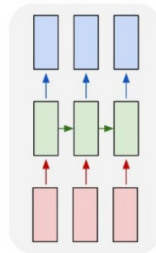
many to one



many to many

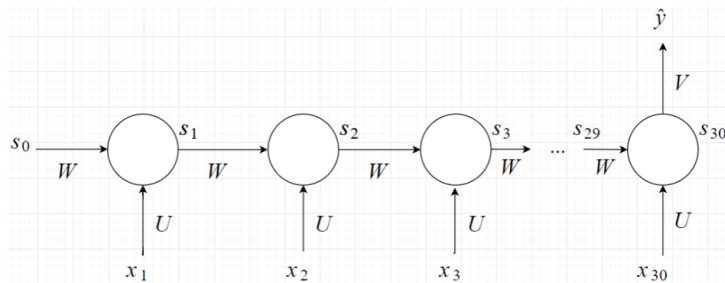


many to many



- **Bài toán:** Nhận diện hành động trong video 30s (Many to one).
- **Input:** Tách video thành 30 ảnh (1 FPS). Các ảnh sẽ được cho qua pretrained model CNN để lấy ra các feature vector có kích thước $nx1$. Vector tương ứng với ảnh ở giây thứ i là x_i .
- **Output:** vector có kích thước $dx1$ (d là số lượng hành động cần phân loại), softmax function được sử dụng như trong bài phân loại ảnh.

Mô hình RNN



- $\mathbf{x}_i \in \mathbb{R}^{n \times 1}$, $\mathbf{s}_i \in \mathbb{R}^{m \times 1}$, $\mathbf{y}_i \in \mathbb{R}^{d \times 1}$,
 $\mathbf{U} \in \mathbb{R}^{m \times n}$, $\mathbf{W} \in \mathbb{R}^{m \times m}$, $\mathbf{V} \in \mathbb{R}^{d \times m}$.

- $s_t = f(U \cdot x_t + W \cdot s_{t-1}) \quad \forall t \geq 1$

- $\hat{y} = g(V \cdot s_{30})$

Backpropagation Through Time (BPTT)

- $\frac{\partial L}{\partial U}, \frac{\partial L}{\partial V}, \frac{\partial L}{\partial W}$.

- $\frac{\partial L}{\partial V} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial V}$

- $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial s_{30}} \cdot \frac{\partial s_{30}}{\partial W}$

- $\frac{\partial s_{30}}{\partial W} = \frac{\partial s'_{30}}{\partial W} + \frac{\partial s_{30}}{\partial s_{29}} \cdot \frac{\partial s_{29}}{\partial W}$

trong đó $\frac{\partial s'_{30}}{\partial W}$ là đạo hàm của s_{30} với W khi coi s_{29} là hằng số đối với W .

- $\frac{\partial L}{\partial W} = \sum_{i=0}^{30} \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial s_{30}} \cdot \frac{\partial s_{30}}{\partial s_i} \cdot \frac{\partial s'_i}{\partial W}$

trong đó: $\frac{\partial s_{30}}{\partial s_i} = \prod_{j=i}^{29} \frac{\partial s_{j+1}}{\partial s_j}$ và $\frac{\partial s'_i}{\partial W}$ là đạo hàm của s_i với W khi coi s_{i-1} là hằng số đối với W .

Vanishing Gradient Problem in RNNs

Giả sử activation là tanh function: $s_t = \tanh(U \cdot x_t + W \cdot s_{t-1})$

Ta có:

$$\frac{\partial s_i}{\partial s_{i-1}} = (1 - s_i^2) \cdot W \Rightarrow \frac{\partial s_{30}}{\partial s_i} = W^{30-i} \cdot \prod_{j=i}^{29} (1 - s_j^2)$$

Với $s_j < 1$, $W < 1 \Rightarrow$ những state xa thì:

$$\frac{\partial s_{30}}{\partial s_i} \approx 0 \quad \text{hay} \quad \frac{\partial L}{\partial W} \approx 0$$

\Rightarrow Đây chính là vấn đề vanishing gradient trong RNNs

2: Long Short-Term Memory

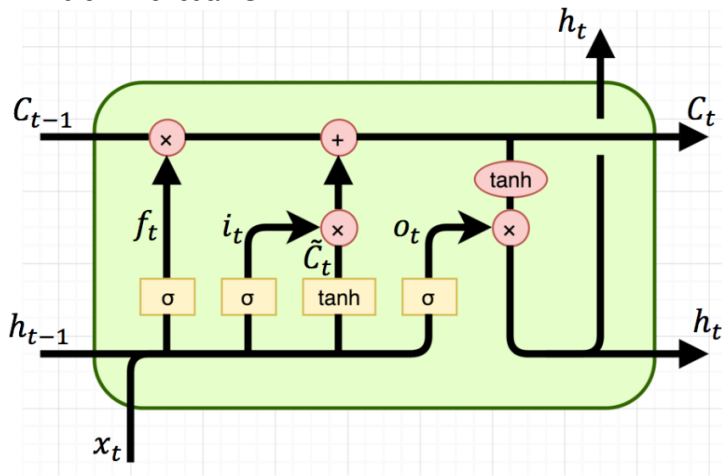
Ý tưởng

- Tôi đang ăn cơm. Mẹ tôi đang quét nhà.
- forget
- input
- output

Mô hình LSTM

- Xét state thứ t của mô hình LSTM:

- ▶ Input: c_{t-1} , h_{t-1} , x_t .
- ▶ Output: c_t , h_t . h_t đóng vai trò khá giống như s_t trong RNN, c_t là điểm mới của LSTM.



Công thức LSTM

- **Forget Gate:**

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1})$$

- **Input Gate:**

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1})$$

- **New Memory Cell:**

$$\tilde{C}_t = \tanh(W_C \cdot x_t + U_C \cdot h_{t-1})$$

- **Final Memory Cell:**

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$$

- **Output Gate:**

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1})$$

- **Hidden State:**

$$h_t = o_t \circ \tanh(C_t)$$

- h_t, \hat{C}_t giống h_t, s_t trong RNN
⇒ **Short-term memory**
- C_t giống như băng chuyền, quyết định thông tin quan trọng cần dùng ở sau sẽ được giữ lại và sử dụng khi cần
⇒ **Long-term memory**

LSTM hạn chế vanishing gradient

- **RNN:** Thành phần chính gây vanishing gradient

$$\frac{\partial s_{t+1}}{\partial s_t} = (1 - s_t^2) \cdot W, \quad \text{với } s_t, W < 1$$

- **LSTM:** Thành phần tương ứng

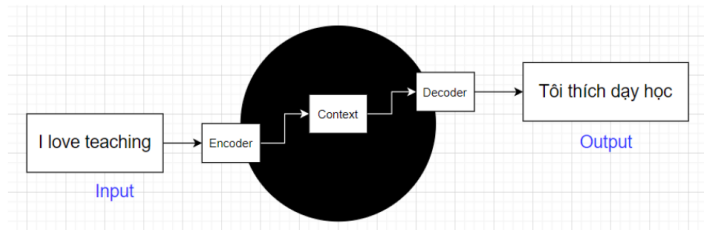
$$\frac{\partial C_t}{\partial C_{t-1}} = f_t, \quad \text{với } 0 < f_t < 1$$

- Về cơ bản LSTM vẫn bị vanishing gradient nhưng **ít hơn** so với RNN
- Khi thông tin quan trọng cần được giữ lại: $f_t \approx 1$

⇒ Tránh được vanishing gradient

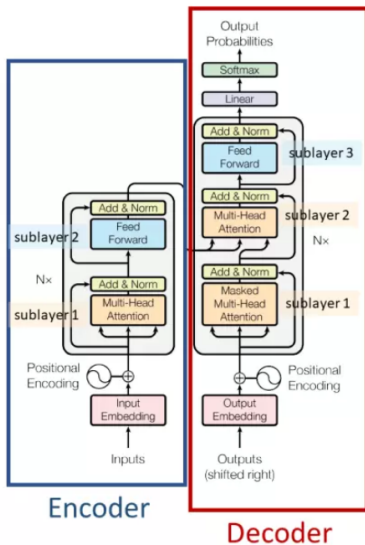
3. Transformers

Sequence to sequence model



- Điểm yếu của phương pháp này là rất khó bắt được sự phụ thuộc xa giữa các từ trong câu và tốc độ huấn luyện chậm do phải xử lý input tuần tự
 - => Cần có cơ chế để lấy được thông tin các từ ở input cho mỗi từ cần dự đoán ở output thay vì chỉ dựa vào context vector
 - => Transformers ra đời.

Transformers model



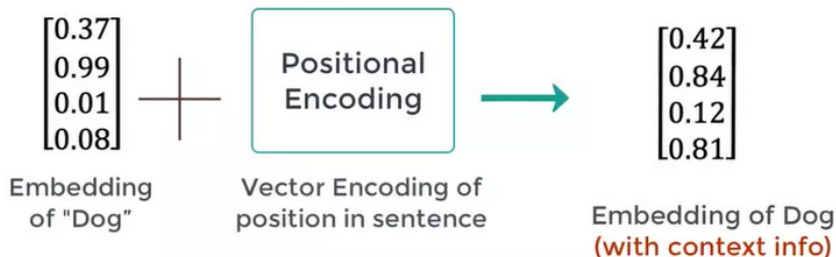
Input Embedding

Input Embedding



Positional Encoding

Trong đó pos là vị trí của từ trong câu, PE là giá trị phần tử thứ i trong embeddings có độ dài d_{model} . Sau đó ta cộng PE vector và Embedding vector:

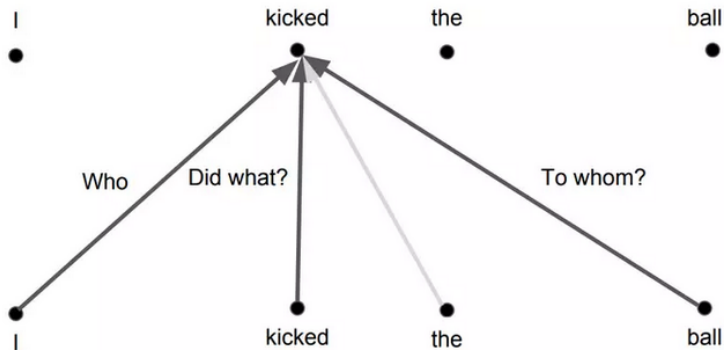


$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

Self-attention

Self-Attention là cơ chế giúp Transformers "hiểu" được sự liên quan giữa các từ trong một câu.

Self-Attention



Công thức tính attention cho 1 từ

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

Trong đó:

- Q (Queries matrix): Truy vấn đến các vector K của các từ trong câu bằng cách nhân chập với những vector này
- K (Keys matrix): Đóng vai trò như một khóa đại diện cho từ
- V : Values matrix
- d_k (Dimension của key vectors): Scale

Các bước tính attention

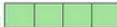
Input

Thinking

Machines

Embedding

x_1 

x_2 

Queries

q_1 


q_2 



W^Q

Keys

k_1 


k_2 



W^K

Values

v_1 

v_2 



W^V

Các bước tính attention

Input

Embedding

Queries

Keys

Values

Score

Divide by $8 (\sqrt{d_k})$

Softmax

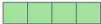
Softmax

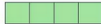
X
Value

Sum

Thinking

Machines

x_1 

x_2 

q_1 

q_2 

k_1 

k_2 

v_1 

v_2 

$q_1 \cdot k_1 = 112$

$q_1 \cdot k_2 = 96$

14

12

0.88

0.12

v_1 

v_2 

z_1 

z_2 

Multi-head attention

Focus

The	→	The big red dog
big	→	The big red dog
red	→	The big red dog
dog	→	The big red dog

Multi-head attention

1) Concatenate all the attention heads



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

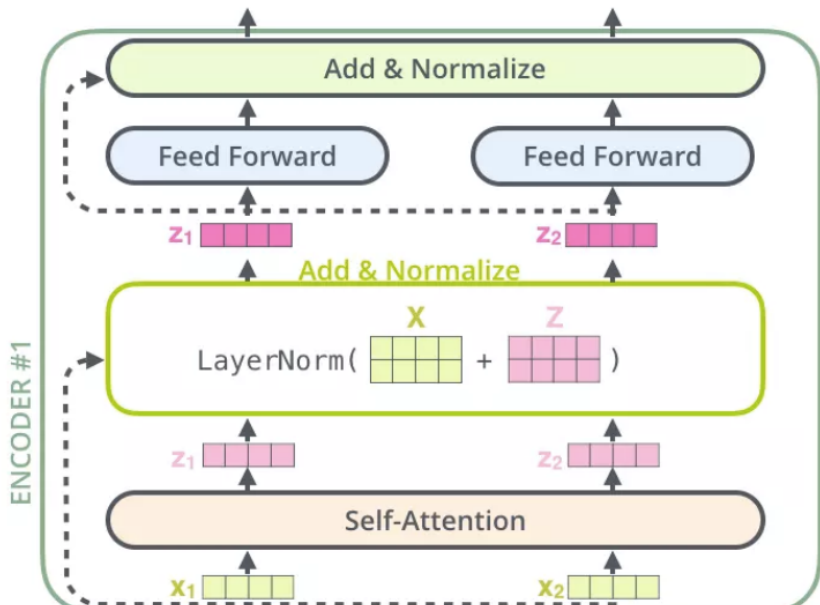


2) Multiply with a weight matrix W^O that was trained jointly with the model

\times



Residuals



- Self-Attention: Học các quan hệ rõ ràng, có thể diễn giải giữa các từ.
- Feed-Forward (FFN): Học các quan hệ ngầm, tiềm ẩn giữa các đặc trưng trong từng vector, những cái mà attention không mô tả được.

$$\text{FFN}(x) = W_2 \cdot \max(0, W_1 x + b_1) + b_2$$

Masked multi-head attention

Decoder sinh từng từ tiếng Pháp dựa trên các từ trước đó. Để tránh việc “nhìn trước” từ tương lai, masked self-attention được dùng, che đi các từ chưa dịch và chỉ cho Decoder truy cập những từ đã sinh ra.



Quá trình decode

- Quá trình decode gần giống encode, nhưng Decoder sinh từng từ một và input bị masked.
- Ở sub-layer 1, chỉ tạo Q từ masked input, còn K,V lấy từ Encoder để dùng trong sub-layer 2 và 3.
- Cuối cùng, vector đi qua Linear + Softmax để dự đoán xác suất từ tiếp theo.