ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Bài 6: Transfomers

Hà Nội, 8/2021

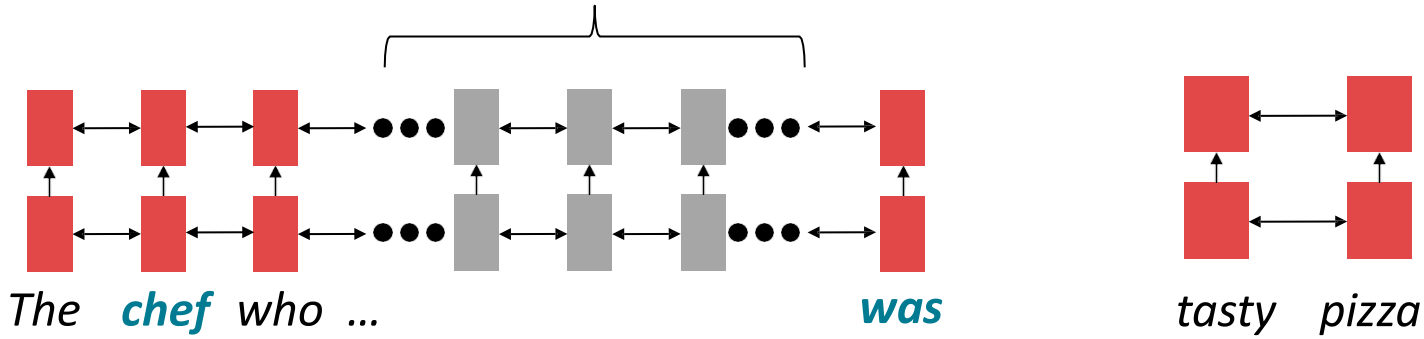
Nội dung

1. Từ RNN tới cơ chế chú ý
2. Giới thiệu Transformers
3. Nhược điểm và một số biến thể của Transformers

Từ RNN tới cơ chế chú ý

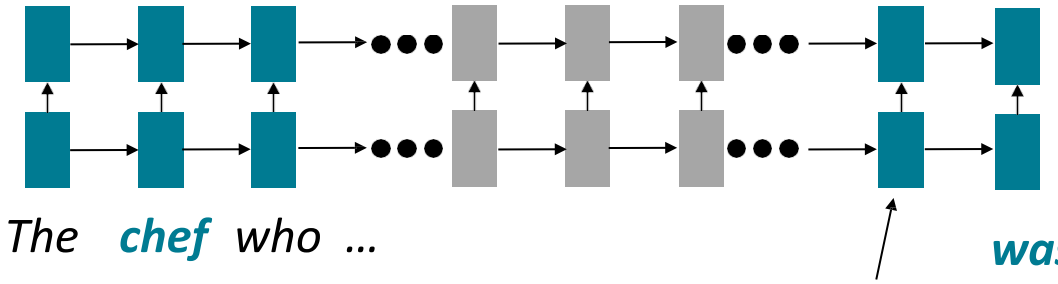
Vấn đề của RNN: khoảng cách tương tác tuyến tính

- RNNs xử lý tuần tự từ trái sang phải
- Mã hoá thông tin cục bộ một cách tuyến tính: nghĩa mỗi từ thường bị ảnh hưởng bởi các từ xung quanh
- Vấn đề: RNNs cần O(*độ dài dãy*) bước để các cặp từ cách xa nhau tương tác



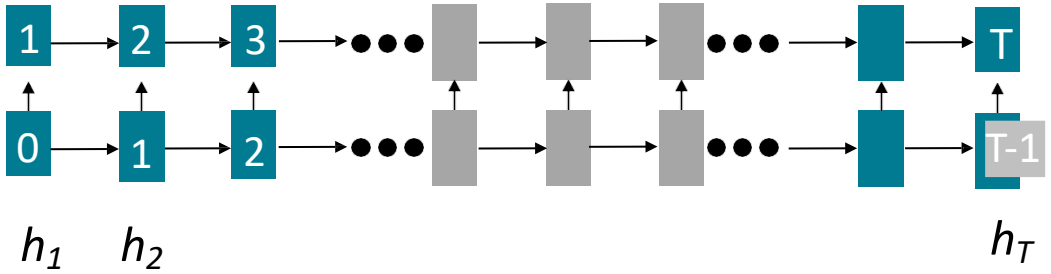
Vấn đề của RNN: khoảng cách tương tác tuyến tính

- O(*độ dài dãy*) bước cho các cặp cách xa nghĩa là:
  - Khó để học mối quan hệ giữa các cặp ở xa (do vấn đề triệt tiêu hoặc bùng nổ gradient!)
  - Thứ tự tuyến tính của các từ không phải là cách đúng khi nghĩ về câu…



Vấn đề của RNN: Thiếu tính song song

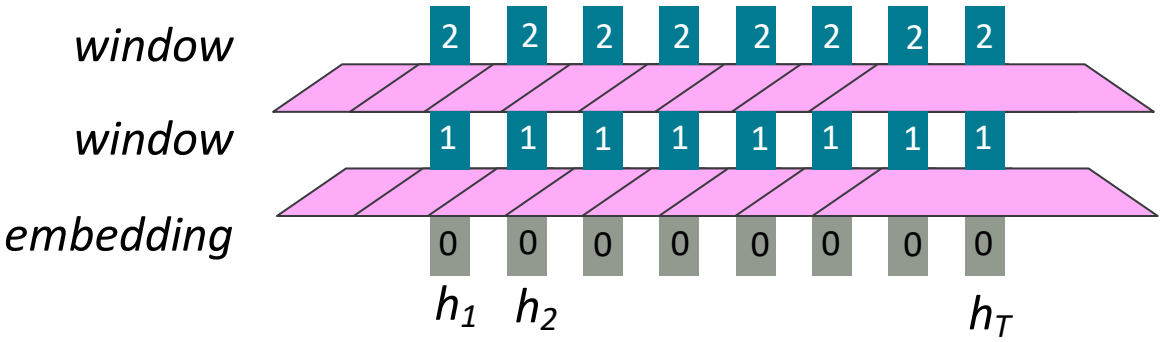
- Tính toán ngược và xuôi đều cần O(*độ dài dãy*) phép toán không song song
  - GPUs có thể xử lý hàng loạt phép tính độc lập cùng một thời điểm!
  - Nhưng các trạng thái ẩn tương lai trong RNN không thể tính toán trước khi các trạng thái ẩn quá khứ được tính
  - Do vậy rất khó để huấn luyện trên các dữ liệu rất lớn!



Con số thể hiện số bước tối thiểu trước khi một trạng thái ẩn có thể được tính toán

Nếu không hồi quy thì dùng cái gì? Dùng cửa sổ từ?

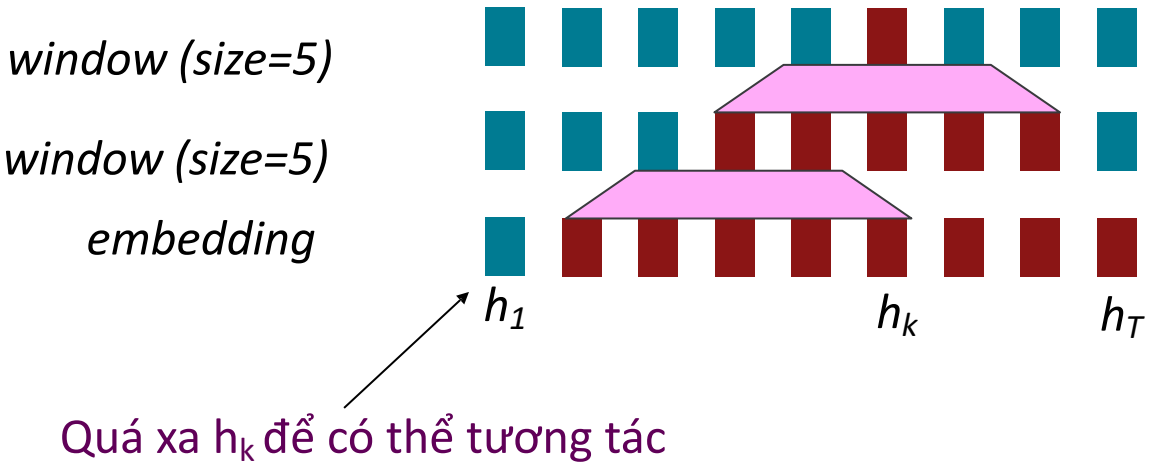
- Mô hình cửa sổ từ tổng hợp thông tin ngữ cảnh cục bộ
  - Còn được gọi là tích chập 1D
  - Số phép toán **không song song** không tăng theo độ dài dãy!



Con số thể hiện số bước tối thiểu trước khi một trạng thái ẩn có thể được tính toán

Nếu không hồi quy thì dùng cái gì? Dùng cửa sổ từ?

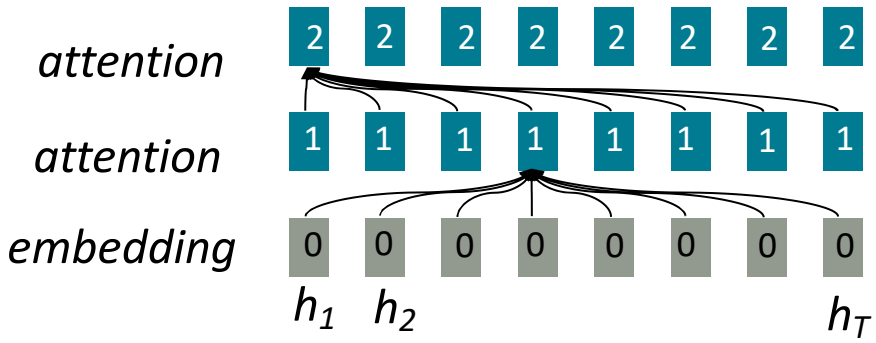
- Mô hình cửa sổ từ tổng hợp thông tin ngữ cảnh cục bộ
- Vậy những quan hệ ở khoảng cách xa thì sao?
  - Chồng nhiều cửa sổ từ cho phép tương tác với các từ ở xa hơn
- Khoảng cách tương tác xa nhất: O(*độ dài dãy / kích thước cửa sổ*)



Màu đỏ thể hiện những trạng thái mà có thể h_k tương tác

Nếu không hồi quy thì dùng cái gì? Dùng cơ chế chú ý?

- Cơ chế chú ý xem mỗi biểu diễn của một từ như là một truy vấn (query) để truy vấn và tương tác thông tin từ một tập các giá trị.
  - Ta đã học cơ chế chú ý từ phần giải mã (decoder) sang phần mã hoá (encoder). Hôm nay ta sẽ trao đổi về cơ chế chú ý trong cùng một câu.
- Số phép toán không song song không tăng theo độ dài dãy.
- Khoảng cách tương tác xa nhất: O(1), vì mọi từ tương tác với nhau tại mọi lớp!



Mọi từ chú ý tới tất cả các từ trong lớp trước; hầu hết cung được lược bỏ cho dễ minh hoạ

Tự chú ý (self-attention)

- Cơ chế chú ý làm việc với queries, keys và values.
  - Queries: $q_1, q_2, \dots, q_T$ trong đó $q_i \in \mathbb{R}^d$
  - Keys: $k_1, k_2, \dots, k_T$ trong đó $k_i \in \mathbb{R}^d$
  - Values: $v_1, v_2, \dots, v_T$ trong đó $v_i \in \mathbb{R}^d$
- Trong **tự chú ý**, queries, keys và values sinh ra từ cùng một nguồn.
- Ví dụ đầu ra của lớp trước là $x_1, x_2, \dots, x_T$ (mỗi véc-tơ tương ứng một từ), ta có thể chọn $q_i = k_i = v_i = x_i$.
- Công thức tự chú ý:

$$e_{ij} = q_i^T k_j$$

Tính tương quan điểm chú ý **key-query**

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_l \exp(e_{il})}$$

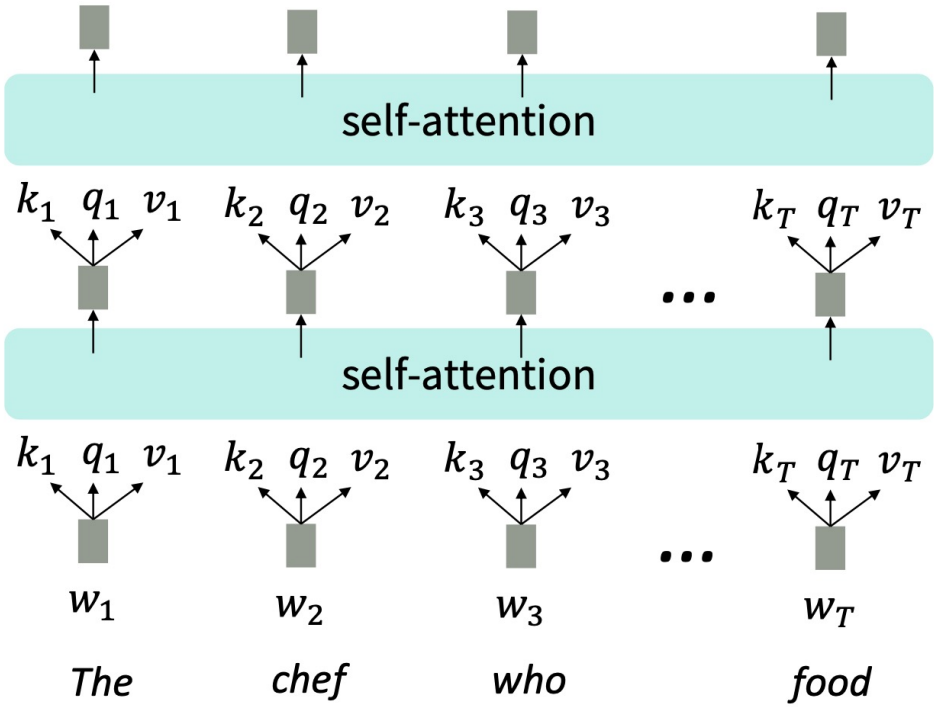
Tính trọng số chú ý từ các hệ số tương quan (dùng softmax)

$$\text{output}_i = \sum_j \alpha_{ij} v_j$$

Tính đầu ra bằng tổng có trọng số của các **values**

Sử dụng Tự chú ý như một khối trong NLP

- Xếp chồng nhiều khối tự chú ý như với LSTM.
- Liệu có thể dùng cơ chế tự chú ý thay thế trực tiếp các lớp hồi quy?
- Không, nó có một vài vấn đề mà ta sẽ xem xét.
- Thứ nhất, tự chú ý hoạt động trên các tập hợp đầu vào, tức là nó không quan tâm tới thứ tự đầu vào.



Tự chú ý không biết thứ tự đầu vào input.

Trở ngại và giải pháp để sử dụng tự chú ý như các khối xây dựng mạng nơ-ron

Trở ngại

- Không có thông tin về thứ tự!

Giải pháp

Giải quyết vấn đề đầu tiên: thự tự dãy

- Vì tự chú ý không xây dựng thông tin về thứ tự, ta cần phải mã hoá thứ tự của câu vào trong keys, queries, và values.
- Giả sử biểu diễn mỗi vị trí trong câu bởi một véc-tơ

$$\mathrm{p_i} \in \mathbb{R}^{\mathrm{d}}, i \in \{1,2, \dots, T\} \text{ là các véc-tơ vị trí}$$

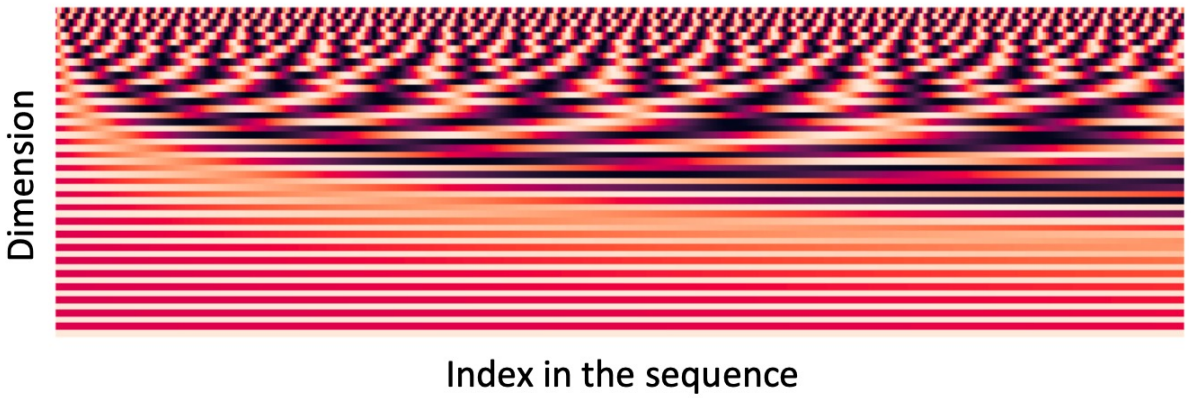
- Ta chưa cần biết $\mathrm{p_i}$ xác định như thế nào!
- Dễ dàng để tích hợp thông tin này vào khối tự chú ý: chỉ cần cộng $\mathrm{p_i}$ vào các đầu vào!
- Giả sử $\tilde{\mathrm{q}}_\mathrm{i}, \tilde{\mathrm{k}}_\mathrm{i}, \tilde{\mathrm{v}}_\mathrm{i}$ là các values, keys, và queries cũ.

$$\mathrm{q_i} = \tilde{\mathrm{q}}_\mathrm{i} + \mathrm{p_i}$$
$$\mathrm{k_i} = \mathrm{k_i} + \mathrm{p_i}$$
$$\mathrm{v_i} = \mathrm{v_i} + \mathrm{p_i}$$

Trong các mạng nhiều lớp tự chú ý, chúng ta thường thêm thông tin vị trí ở lớp đầu tiên! Bạn có thể concat chúng nhưng mọi người thường dùng phép toán cộng…

Biểu diễn thông tin vị trí thông qua hàm sin

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- Ưu điểm:
  - Tính chu kỳ chỉ ra rằng có thể "vị trí tuyết đối" không quá quan trọng
  - Có thể ngoại suy cho những câu dài hơn vì chu kỳ lặp đi lặp lại!
- Nhược điểm:
  - Không học được; việc ngoại suy cũng không thật sự hiệu quả!

Học véc-tơ biểu diễn vị trí

- Học biểu diễn vị trí tuyệt đội: Tất cả $\mathrm{p_i}$ đều là véc-tơ tham số để học! Học ma trận $\boldsymbol{p} \in \mathbb{R}^{\mathrm{dxT}}$, trong đó $\mathrm{p_i}$ là một cột của ma trận đó!

- Ưu điểm:
  - Linh hoạt: mỗi vị trí được học để khớp dữ liệu
- Nhược điểm:
  - Không thể ngoại suy cho các vị trí ngoài dải 1, … , T.
- Hầu hết các hệ thống dùng phương pháp này!

- Một vài công bố thử nghiệm các cách biểu diễn linh hoạt hơn:
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

Trở ngại và giải pháp để sử dụng tự chú ý như các khối xây dựng mạng nơ-ron

Trở ngại

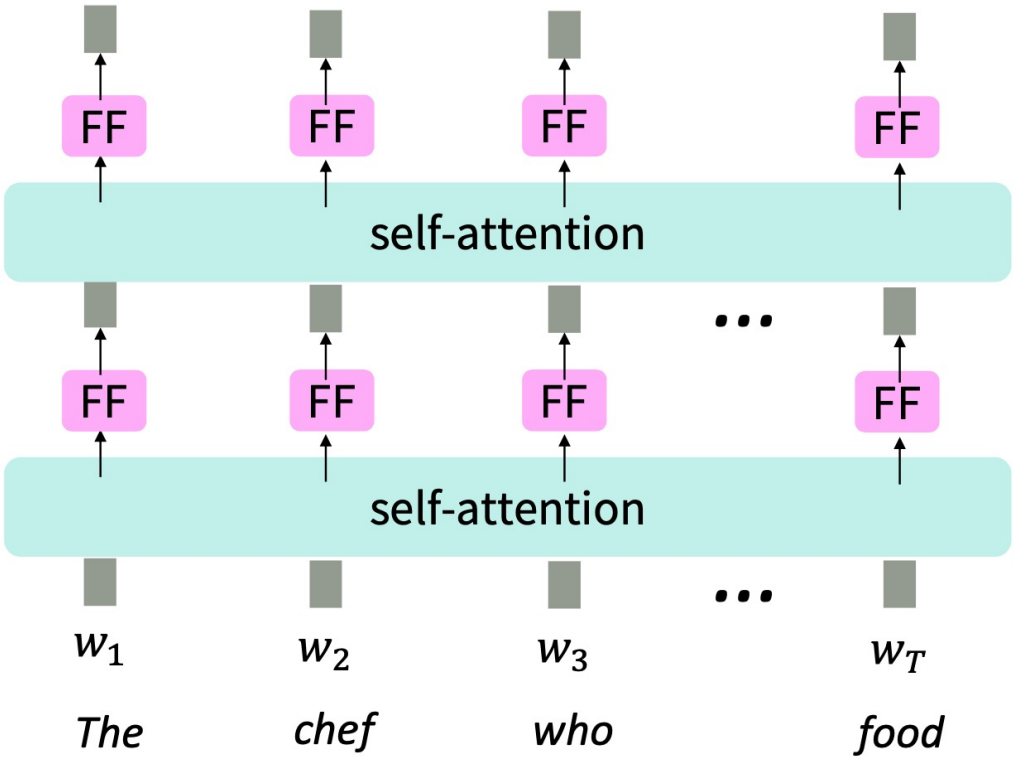
- Không có thông tin về thứ tự!
- Thiếu biến đổi phi tuyến! Hiện mới đơn giản là trung bình có trọng số

Giải pháp

- Thêm thông tin biểu diễn vị trí vào các đầu vào

Thêm biến đổi phi tuyến trong tự chú ý

- Lớp tự chú ý chỉ là tuyến tính. Xếp chồng bao nhiêu lớp vẫn chỉ là tuyến tính.
- Giải quyết dễ dàng: them mạng **feed-forward** để hậu xử lý véc-tơ đầu ra của tự chú ý.



$m_i = MLP(\text{output}_i)$
$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$

Mạng FF xử lý kết quả của lớp tự chú ý

Trở ngại và giải pháp để sử dụng tự chú ý như các khối xây dựng mạng nơ-ron

Trở ngại

- Không có thông tin về thứ tự!
- Thiếu biến đổi phi tuyến! Hiện mới đơn giản là trung bình có trọng số
- Cần đảm bảo "không nhìn vào tương lại" khi đoán một câu
  - Chẳng hạn trong dịch máy
  - Hoặc mô hình ngôn ngữ

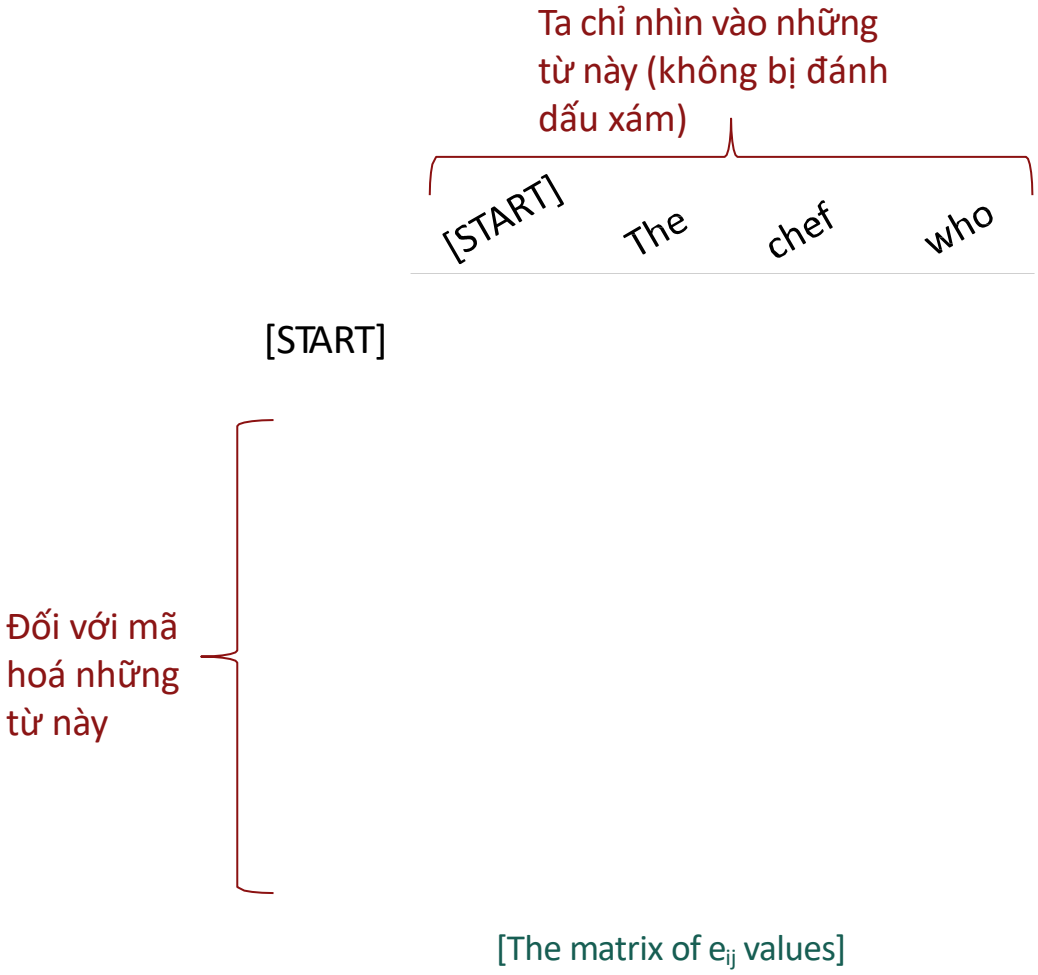
Giải pháp

- Thêm thông tin biểu diễn vị trí vào các đầu vào
- Áp dụng mạng feedforward network cho đầu ra của mỗi lớp tự chú ý.

Đánh dấu phần tương lai

- Để dùng tự chú ý trong phần giải mã, ta phải đảm bảo không sử dụng thông tin tương lai.
- Tại mỗi bước, ta có thể đổi tập keys và queries chỉ gồm các từ quá khứ (không hiệu quả!)
- Để cho phép tính toán song song, ta có thể đánh dấu các từ tương lai bằng cách thiết lập điểm chú ý tới chúng thành $-\infty$.

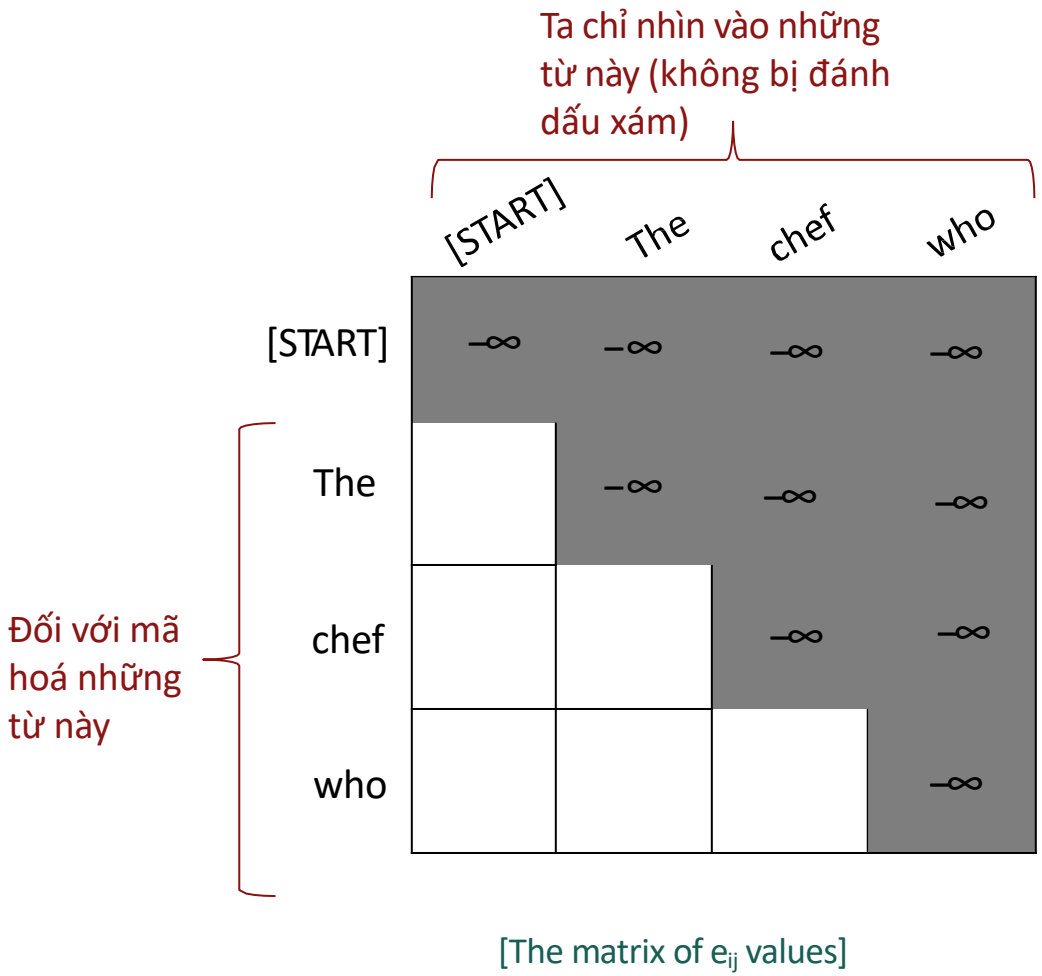
$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$



Đánh dấu phần tương lai

- Để dùng tự chú ý trong phần giải mã, ta phải đảm bảo không sử dụng thông tin tương lai.
- Tại mỗi bước, ta có thể đổi tập keys và queries chỉ gồm các từ quá khứ (không hiệu quả!)
- Để cho phép tính toán song song, ta có thể đánh dấu các từ tương lai bằng cách thiết lập điểm chú ý tới chúng thành $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$



Trở ngại và giải pháp để sử dụng tự chú ý như các khối xây dựng mạng nơ-ron

Trở ngại

- Không có thông tin về thứ tự!
- Thiếu biến đổi phi tuyến! Hiện mới đơn giản là trung bình có trọng số
- Cần đảm bảo "không nhìn vào tương lại" khi đoán một câu
  - Chẳng hạn trong dịch máy
  - Hoặc mô hình ngôn ngữ

Giải pháp

- Thêm thông tin biểu diễn vị trí vào các đầu vào
- Áp dụng mạng feedforward network cho đầu ra của mỗi lớp tự chú ý.
- Đánh dấu phần tương lai bằng cách đặt các trọng số chú ý bằng 0!

Những thứ cần thiết cho một khối tự chú ý

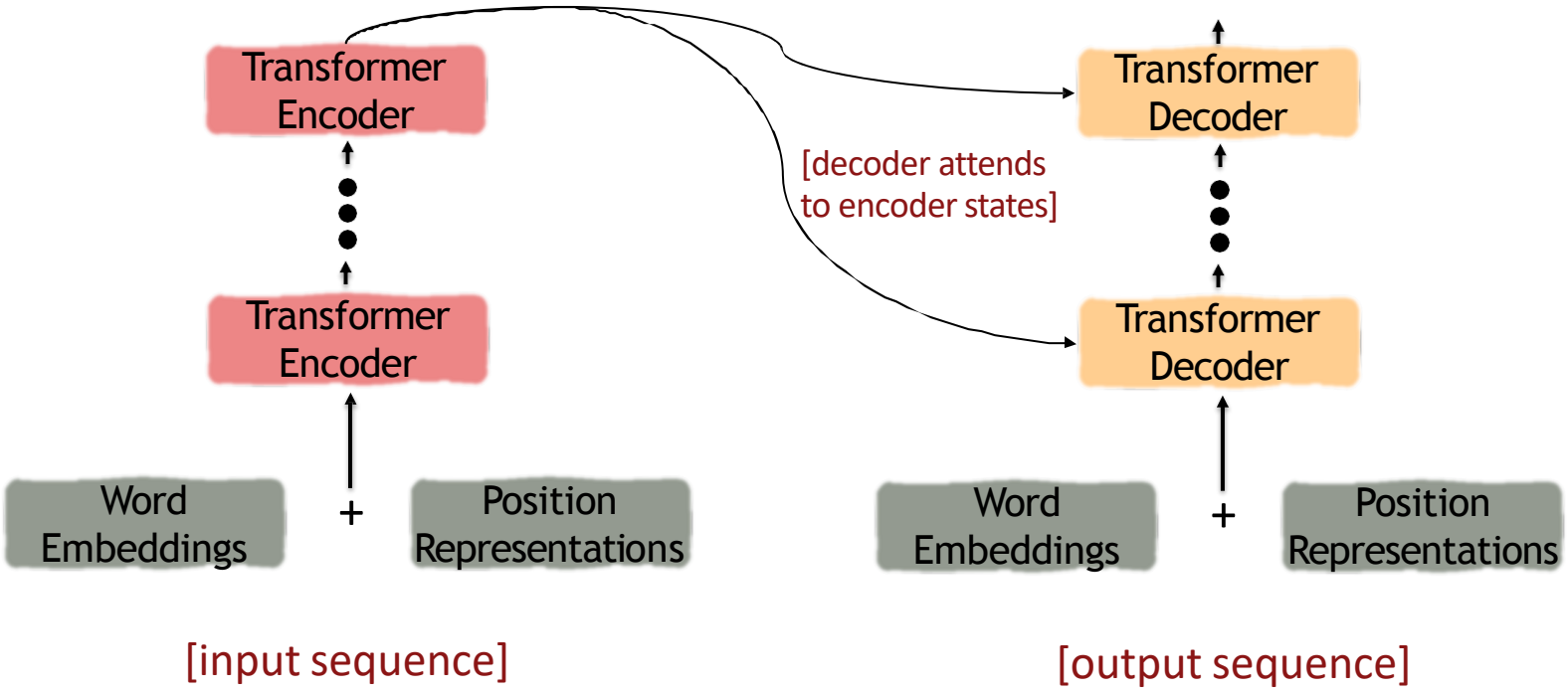
- Lớp tự chú ý:
  - Thành phần cơ bản của khối.
- Biểu diễn vị trí:
  - Thể hiện thứ tự của dãy, do lớp tự chú ý không có thông tin thứ tự của đầu vào.
- Phi tuyến:
  - Áp dụng vào đầu ra của lớp tự chú ý
  - Thường được cài đặt bằng mạng feed-forward.
- Đánh dấu:
  - Để song song hoá trong khi vẫn đảm bảo không nhìn vào tương lai.
  - Giúp thông tin từ tương lại không bị "rò rỉ" về quá khứ.

- Nhưng … đây vẫn chưa phải là mô hình Transformer mà chúng ta vẫn nghe thấy.

Giới thiệu về Transformers

The Transformer Encoder-Decoder [Vaswani et al., 2017]

- Trước tiên xem xét mức tổng quan



The Transformer Encoder-Decoder [Vaswani et al., 2017]

- Còn gì trong phần giải mã (encoder) của Transformer mà ta chưa xem xét?

1. **Key-query-value attention**: làm sao để xây dựng k, q, v từ embedding của từ?
2. Multi-headed attention: chú ý tới nhiều chỗ trong một lớp!
3. Kỹ thuật (trick) hỗ trợ huấn luyện!
   - Kết nối tắt
   - Chuẩn hoá lớp (layer normalization)
   - Scaling the dot product
   - Các kỹ thuật này không tăng cường khả năng của mô hình mà chỉ cải thiện quá trình huấn luyện.

Transformer Encoder: Key-Query-Value Attention

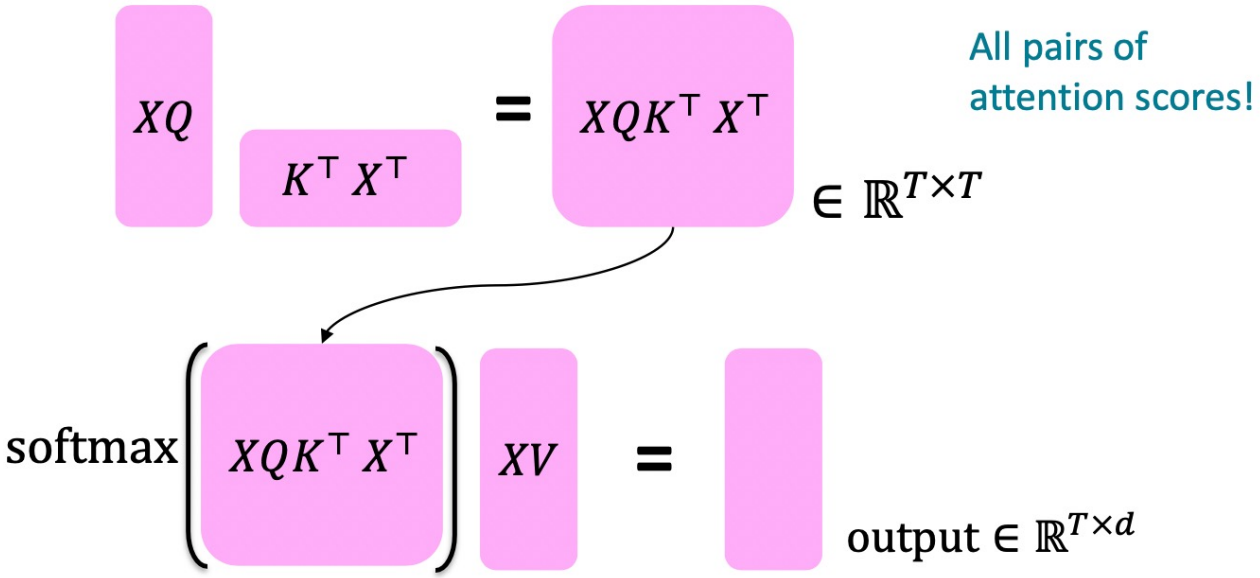
- Tự chú ý là khi keys, queries, và values được sinh ra từ cùng một nguồn. Transformer làm điều này theo một cách riêng:
  - Let $x_1, \dots, x_T$ là véc-tơ đầu vào của Transformer encoder; $x_i \in \mathbb{R}^d$

- Khi đó keys, queries, values xác định như sau:
  - $k_i = Kx_i$, where $K \in \mathbb{R}^{d \times d}$ là ma trận khoá.
  - $q_i = Qx_i$, where Q $\in \mathbb{R}^{d \times d}$ là ma trận truy vấn.
  - $v_i = Vx_i$, where V $\in \mathbb{R}^{d \times d}$ là ma trận giá trị.

- Các ma trận này cho phép biến đổi véc-tơ x theo các cách khác nhau để sử dụng chúng trong 3 vai trò khác nhau tương ứng.

Transformer Encoder: Key-Query-Value Attention

- Tính toán key-query-value attention dưới dạng ma trận.
  - Giả sử $X = [x_1, \ldots, x_T] \in \mathbb{R}^{T \times d}$ là ghép (concat) các véc-tơ đầu vào.
  - Ký hiệu $XK \in \mathbb{R}^{T \times d}$, $XQ \in \mathbb{R}^{T \times d}$, $XV \in \mathbb{R}^{T \times d}$.
  - Đầu ra được xác định như sau: output = softmax $(XQ\ (XK)^\top)\ XV$.



Transformer Encoder: Multi-headed attention

- Sẽ thế nào nếu ta muốn nhìn vào nhiều chỗ trong câu cùng một lúc?
- Vớ từ i, lớp tự chú ý "nhìn" ở đâu $x_i^\top Q^\top K x_j$ có giá trị cao, nhưng có thể chúng ta muốn chú ý vào nhiều vị trí j khác nhau cho các lý do khác nhau?
- Ta sẽ định nghĩa nhiều đầu chú ý bằng nhiều ma trận Q, K, V
- Ký hiệu $\mathrm{Q_l}, \mathrm{K_l}, \mathrm{V_l} \in \mathbb{R}^{\mathrm{dx(d/h)}}$, trong đó h là số đầu, $\mathrm{l} = 1 \ldots h$.
- Mỗi đầu chú ý thực hiện độc lập:

  $\mathrm{output_l} = \mathrm{softmax}(\mathrm{XQ_l K_l^T X^T}) * \mathrm{XV_l}$, trong đó $\mathrm{output_l} \in \mathbb{R}^{\mathrm{d/h}}$
- Sau đó các đầu ra được kết hợp với nhau!

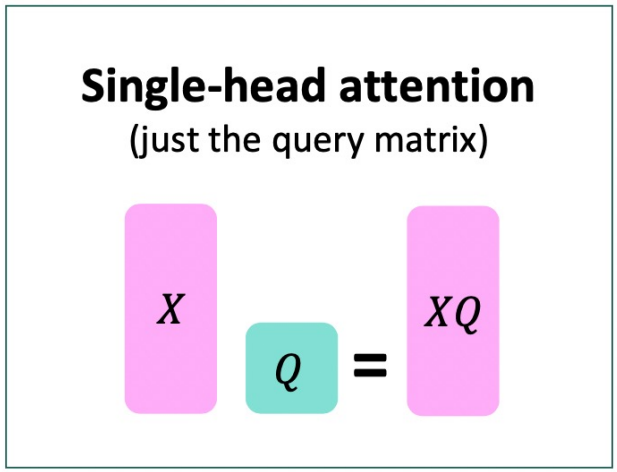
  output = $Y[\mathrm{output}_1; \ldots ; \mathrm{output}_h]$, where $Y \in \mathbb{R}^{d \times d}$

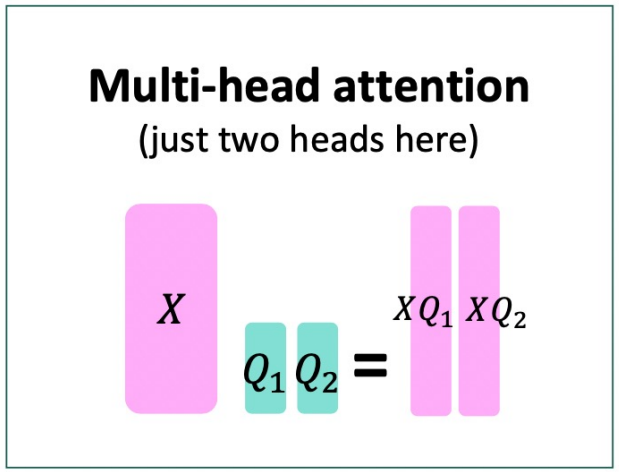
- Mỗi đầu "nhìn" vào những thứ khác nhau, và sinh ra các véc-tơ giá trị khác nhau.

29

Transformer Encoder: Multi-headed attention

- Sẽ thế nào nếu ta muốn nhìn vào nhiều chỗ trong câu cùng một lúc?
- Vớ từ i, lớp tự chú ý "nhìn" ở đâu ${x_i}^\top Q^\top K x_j$ có giá trị cao, nhưng có thể chúng ta muốn chú ý vào nhiều vị trí j khác nhau cho các lý do khác nhau?
- Ta sẽ định nghĩa nhiều đầu chú ý bằng nhiều ma trận Q, K, V
- Ký hiệu $\mathrm{Q_l}, \mathrm{K_l}, \mathrm{V_l} \in \mathbb{R}^{\mathrm{dx(d/h)}}$, trong đó h là số đầu, $\mathrm{l} = 1 \ldots h$.

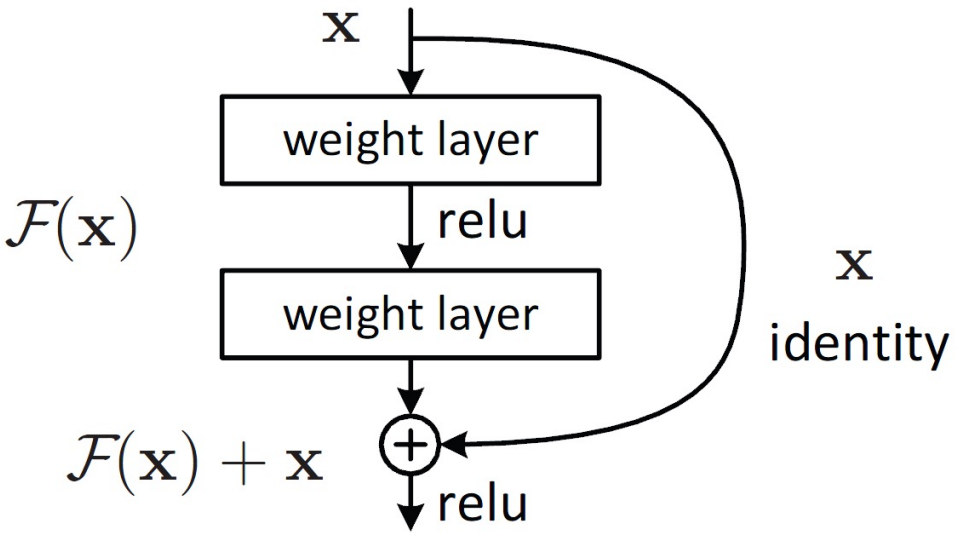


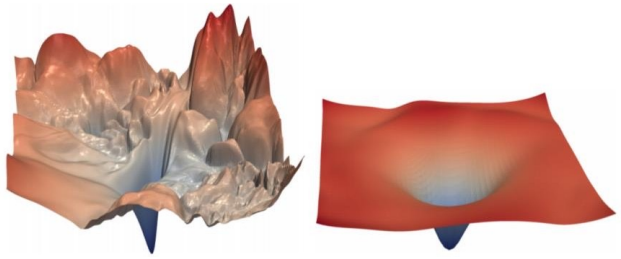


Same amount of computation as single-head self-attention!

Transformer Encoder: Kết nối tắt [He et al., 2016]

- Kết nối tắt giúp huấn luyện mô hình dễ hơn (làm bề mặt hàm mục tiêu mượt hơn)





[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

Transformer Encoder: Chuẩn hoá lớp [Ba et al., 2016]

Batch Normalization for fully-connected networks

$x: N \times D$

Normalize

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

Layer Normalization for fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

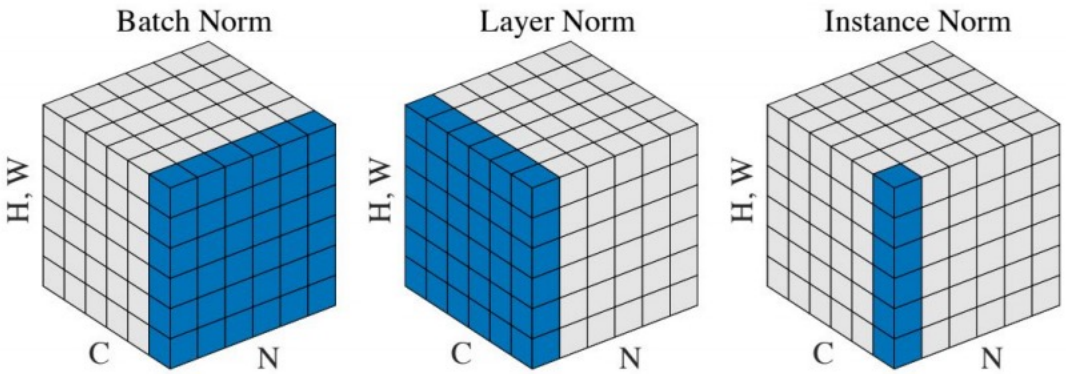
$x: N \times D$

Normalize

$\mu, \sigma: N \times 1$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$



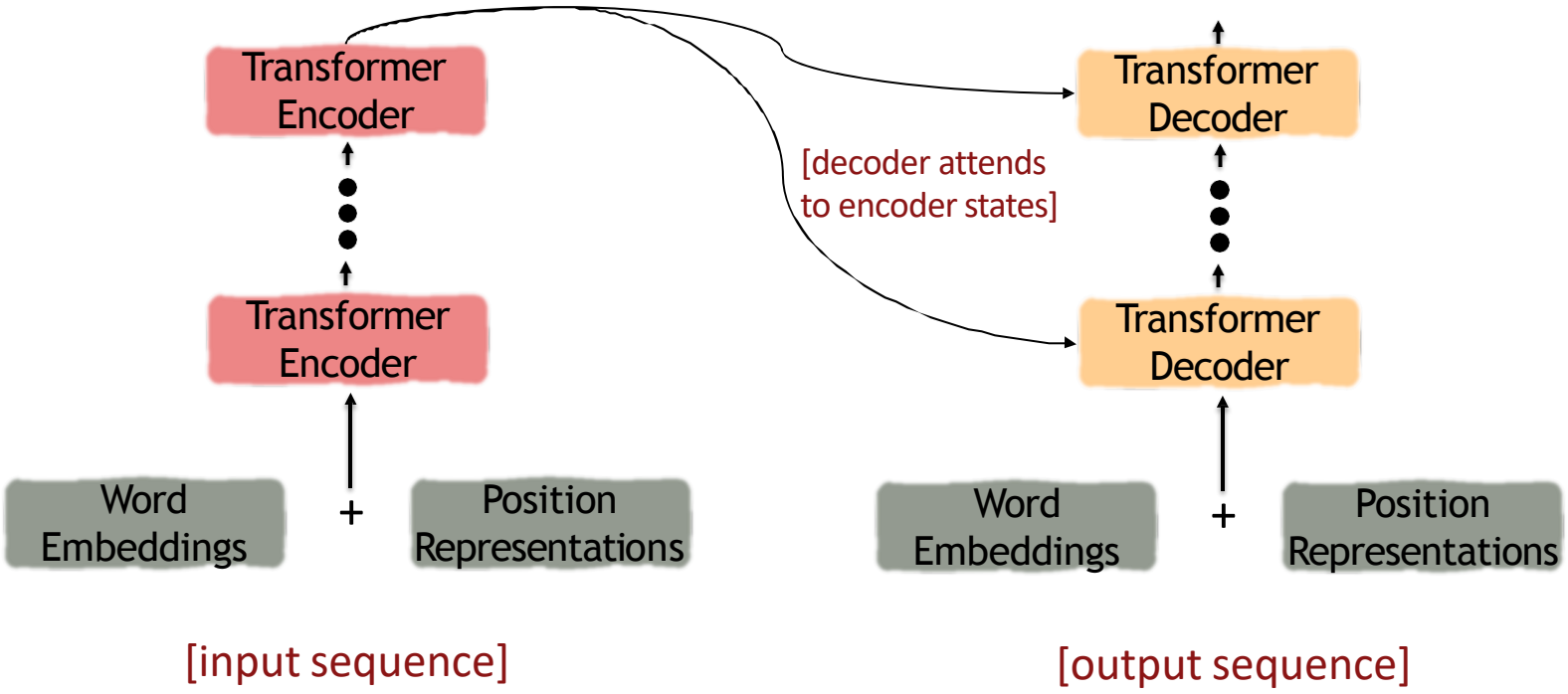
Transformer Encoder: Scaled Dot Product [Vaswani et al., 2017]

- Khi số chiều d lớn, tính vô hướng giữa các véc-tơ cũng có xu hướng lớn theo.
  - Do đó đầu vào của softmax có thể rất lớn, làm hàm bão hoà và triệt tiêu gradient.
- Chia điểm chú ý cho d/h, để làm cho đầu vào của softmax bé lại

$$\text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$$

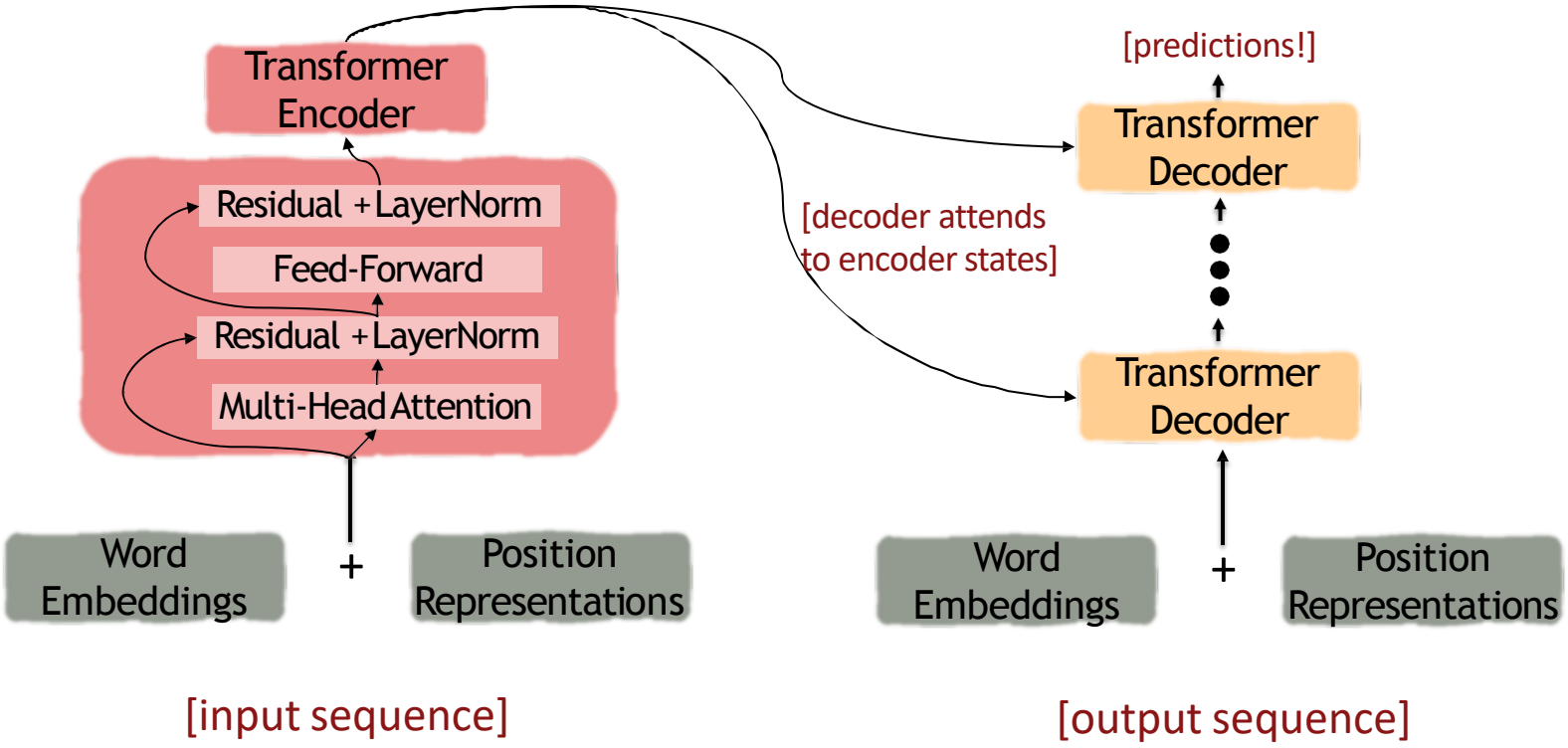
Transformer Encoder-Decoder [Vaswani et al., 2017]

- Xem lại kỹ hơn phần mã hoá



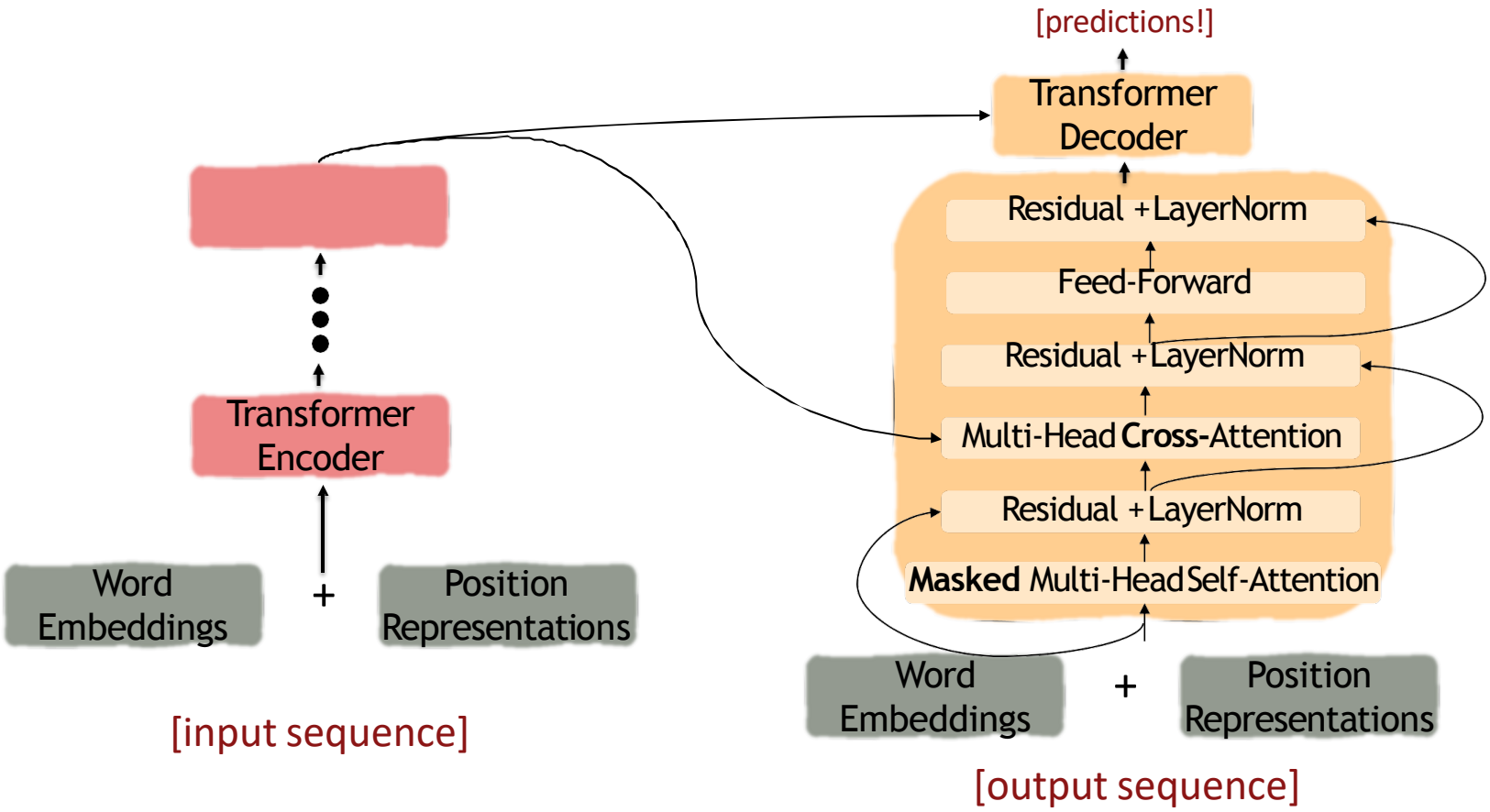
Transformer Encoder-Decoder [Vaswani et al., 2017]

- Xem lại kỹ hơn phần mã hoá



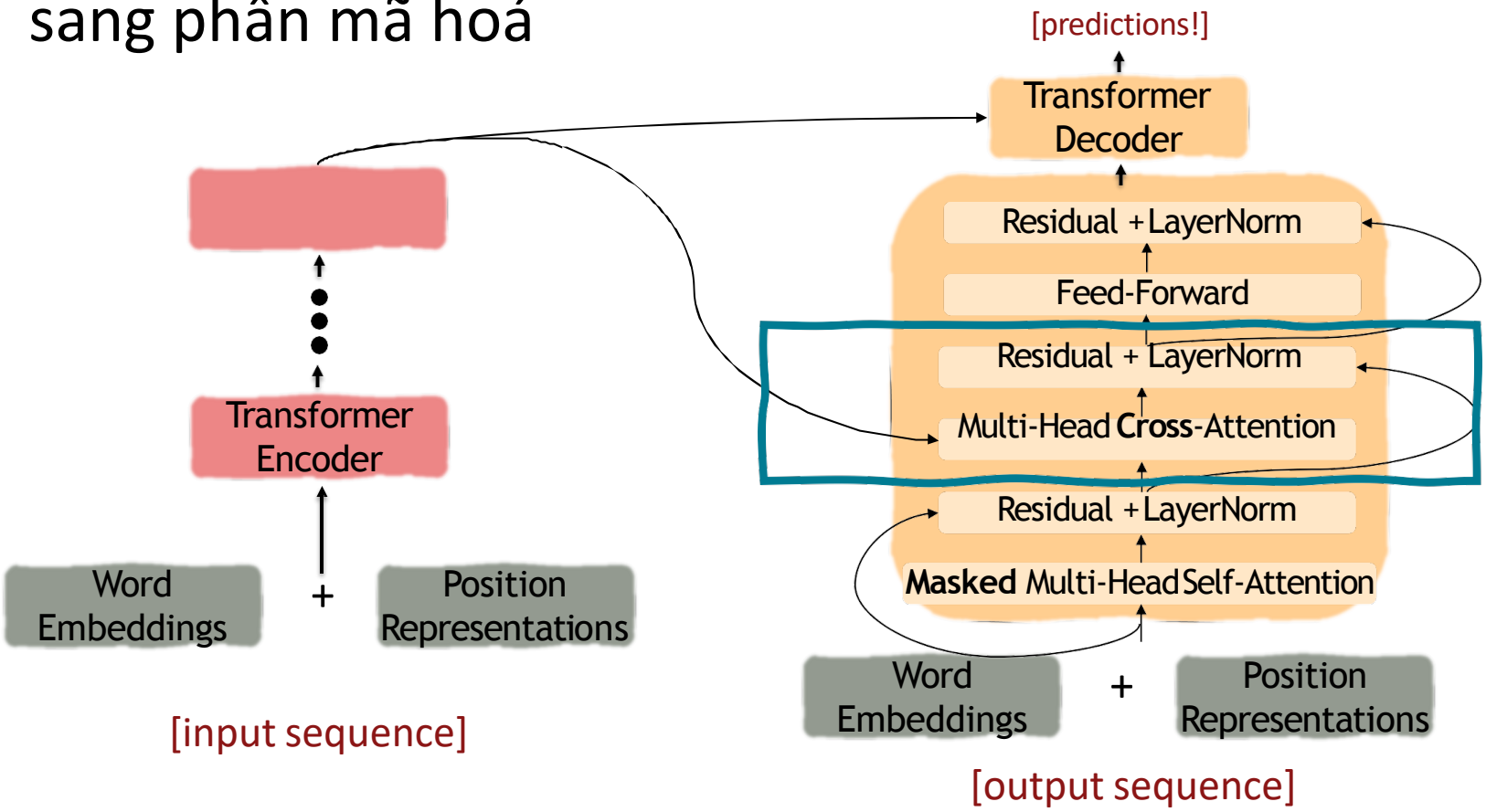
Transformer Encoder-Decoder [Vaswani et al., 2017]

- Xem kỹ phần giải mã



Transformer Encoder-Decoder [Vaswani et al., 2017]

- Chỉ có duy nhất phần mới là chú ý từ phần giải mã sang phần mã hoá



Transformer Decoder: chú ý chéo (Cross-attention)

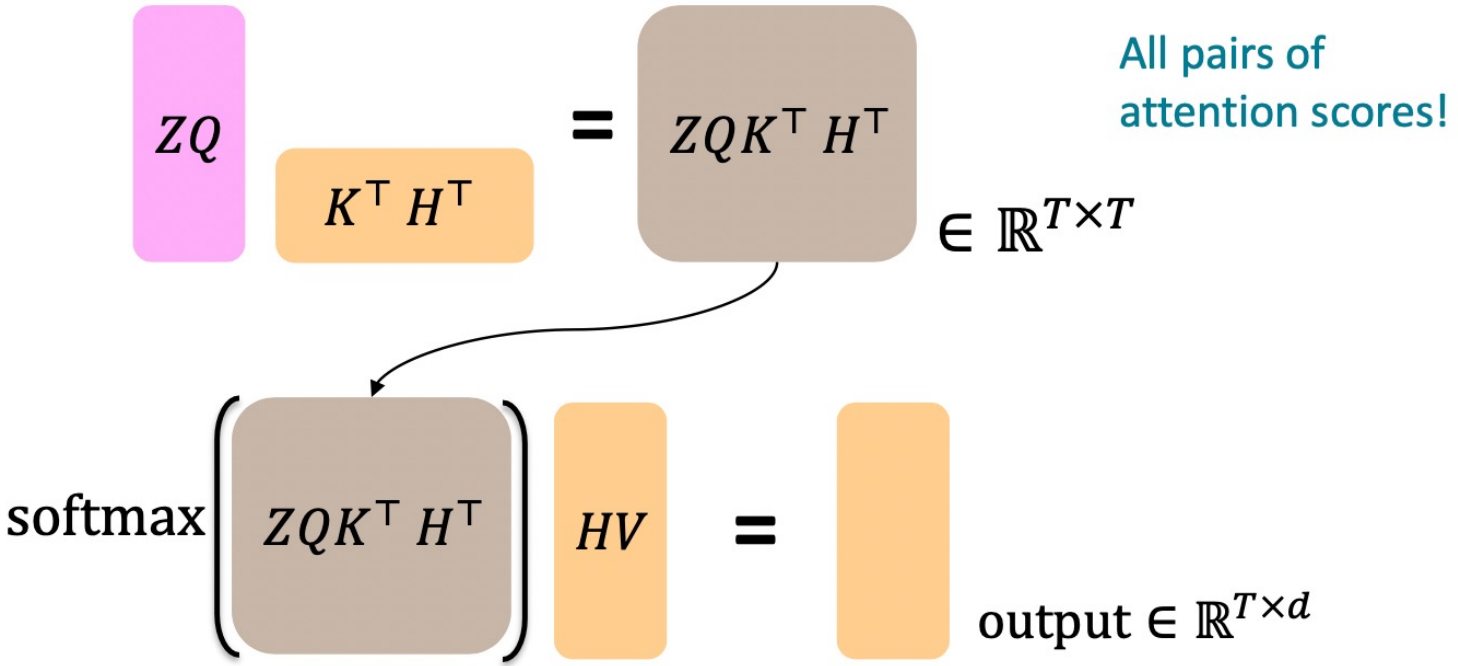
- Giả sử $h_1, \ldots, h_T \in \mathbb{R}^d$ là đầu ra phần mã hoá của Transformer;
- Giả sử $z_1, \ldots, z_T \in \mathbb{R}^d$ là đầu vào của phần giải mã Transformer
- Khi đó keys và values được tính toán từ phần mã hoá (như một bộ nhớ):

  $k_i = Kh_i, v_i = Vh_i$

- Truy vấn được tính toán từ phần giải mã, $q_i = Qz_i$.

Transformer Decoder: chú ý chéo (Cross-attention)

- Tính toán chú ý chéo dưới dạng ma trận
  - Giả sử $\mathrm{H} = [\mathrm{h}_1, \ldots, \mathrm{h}_\mathrm{T}] \in \mathbb{R}^{T \times d}$ là ghép (concat) các véc-tơ mã hoá.
  - Giả sử $\mathrm{Z} = [\mathrm{z}_1, \ldots, \mathrm{z}_\mathrm{T}] \in \mathbb{R}^{T \times d}$ là ghép (concat) các véc-tơ giải mã.
  - Đầu ra được xác định như sau: output = softmax ($\mathrm{Z}Q$ ($\mathrm{H}K$)$^\top$) $\mathrm{H}V$.



Kết quả của Transformers

- Dịch máy

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$

[Test sets: WMT 2014 English-German and English-French]

[Vaswani et al., 2017]

Kết quả của Transformers

- Sinh văn bản

Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

[Liu et al., 2018]; WikiSum dataset

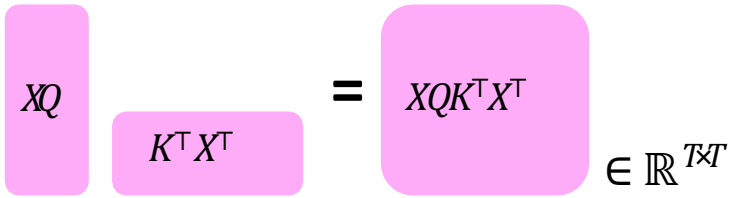
Nhược điểm và một số biến thể của Transformers

Nhược điểm của Transformer

- **Độ phức tạp bình phương trong lớp tự chú ý**:
  - Tính toán tương tác tất cả các cặp $O(T^2)$!
  - Với mô hình hồi quy, độ phức tạp chỉ tuyến tính $O(T)$!
- Vấn đề biểu diễn vị trí:
  - Liệu có thể làm tốt hơn việc biểu diễn đơn giản các vị trí tuyệt đối?
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

Độ phức tạp bình phương như là hàm theo độ dài dãy

- Ưu điểm của tự chú ý so với RNN là tính song song hoá cao.
- Tuy nhiên, số phép toán tăng theo $O(T^2 d)$, trong đó T là độ dài dãy, và d số chiều embedding.

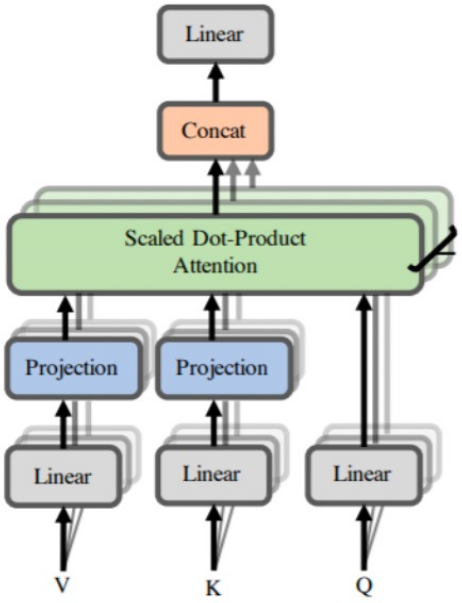


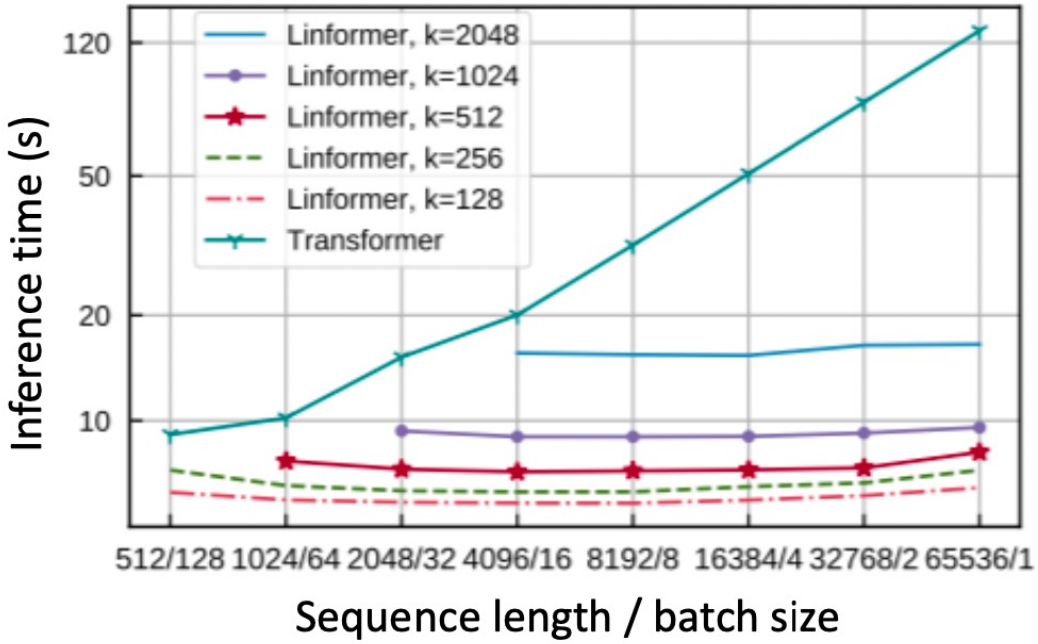
Need to compute all pairs of interactions! $O(T^2 d)$

- Ví dụ d khoảng $\mathbf{1,000}$.
  - Với câu ngắn, $T \leq 30$; $T^2 \leq \mathbf{900}$.
  - Thực tế, ta đặt giới hạn $T = 512$.
  - Nhưng nếu $\boldsymbol{T \geq 10,000}$ thì sao? Ví dụ để xử lý các văn bản dài?

Một số nghiên cứu gần đây giảm độ phức tạp tính toán bình phương của lớp tự chú ý

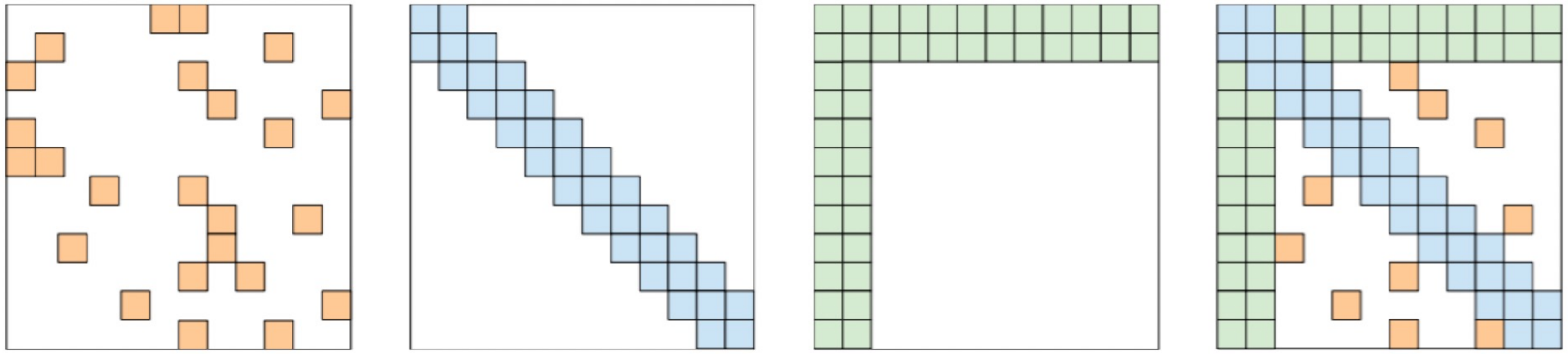
- Ví dụ, Linformer [Wang et al., 2020]





Một số nghiên cứu gần đây giảm độ phức tạp tính toán bình phương của lớp tự chú ý

- Ví dụ, BigBird [Zaheer et al., 2021]



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Tài liệu tham khảo

Khoá cs224n Stanford:
http://web.stanford.edu/class/cs224n/

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!

soict.hust.edu.vn/ fb.com/groups/soict