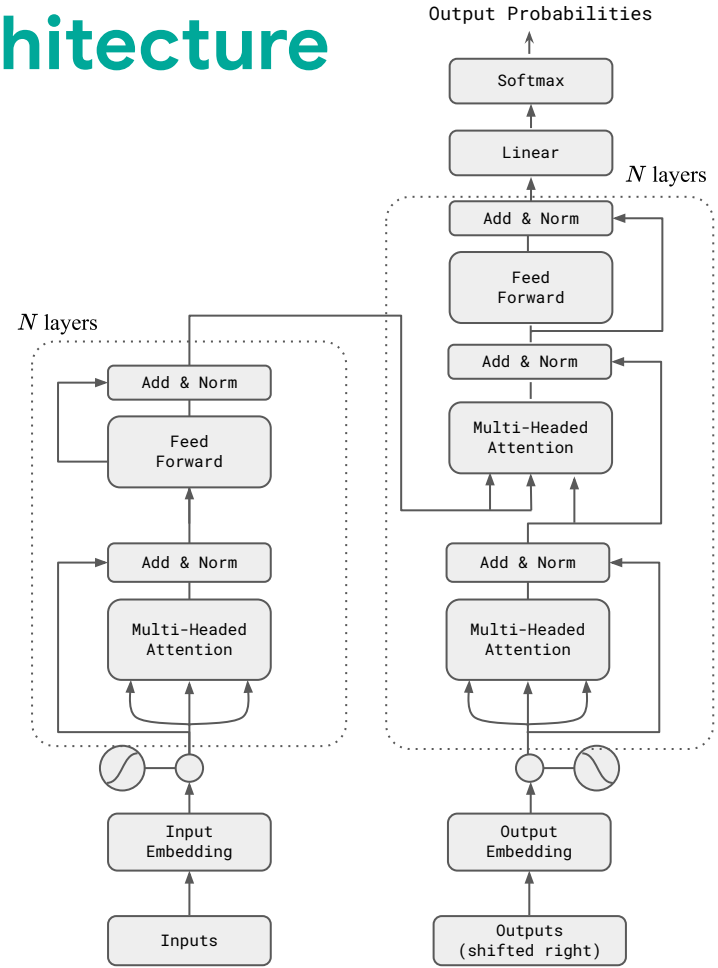
Transformer

Mô hình Transformer

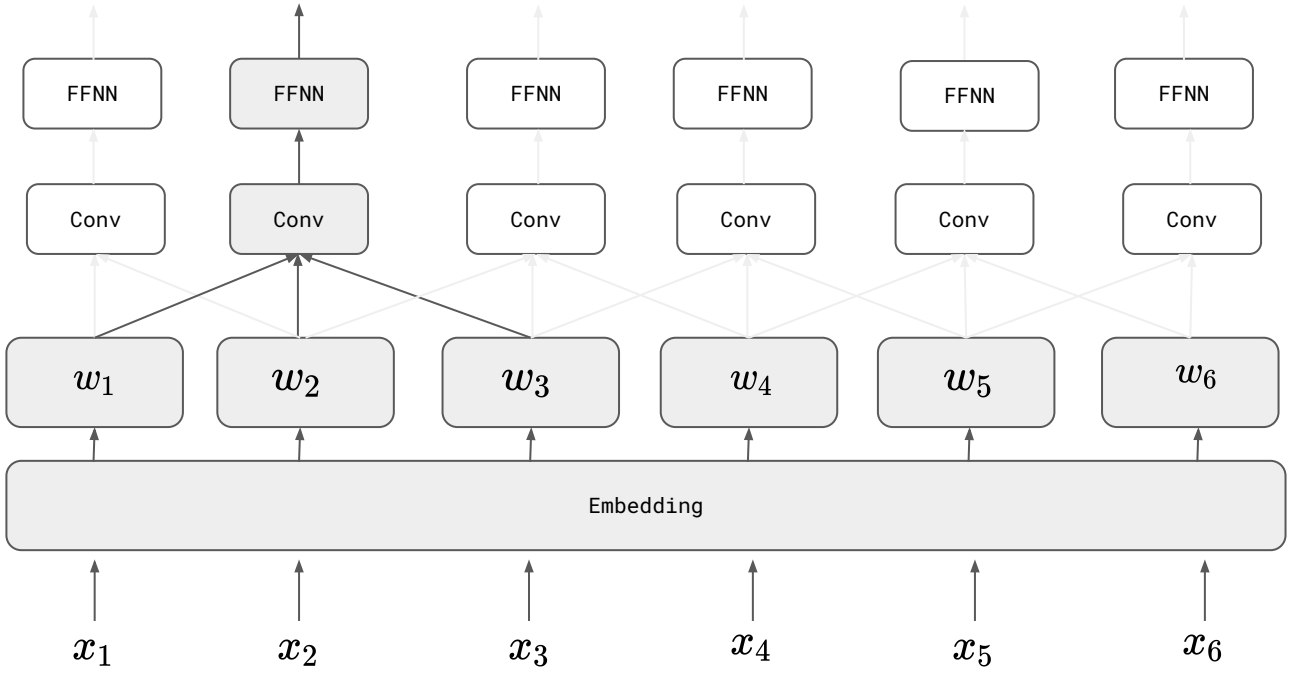
Architecture



1. Lược bỏ kiến trúc Seq2Seq
2. Sử dụng Attention làm lõi chính của model
3. Loss Function vẫn sử dụng Cross Entropy

https://arxiv.org/pdf/1706.03762.pdf

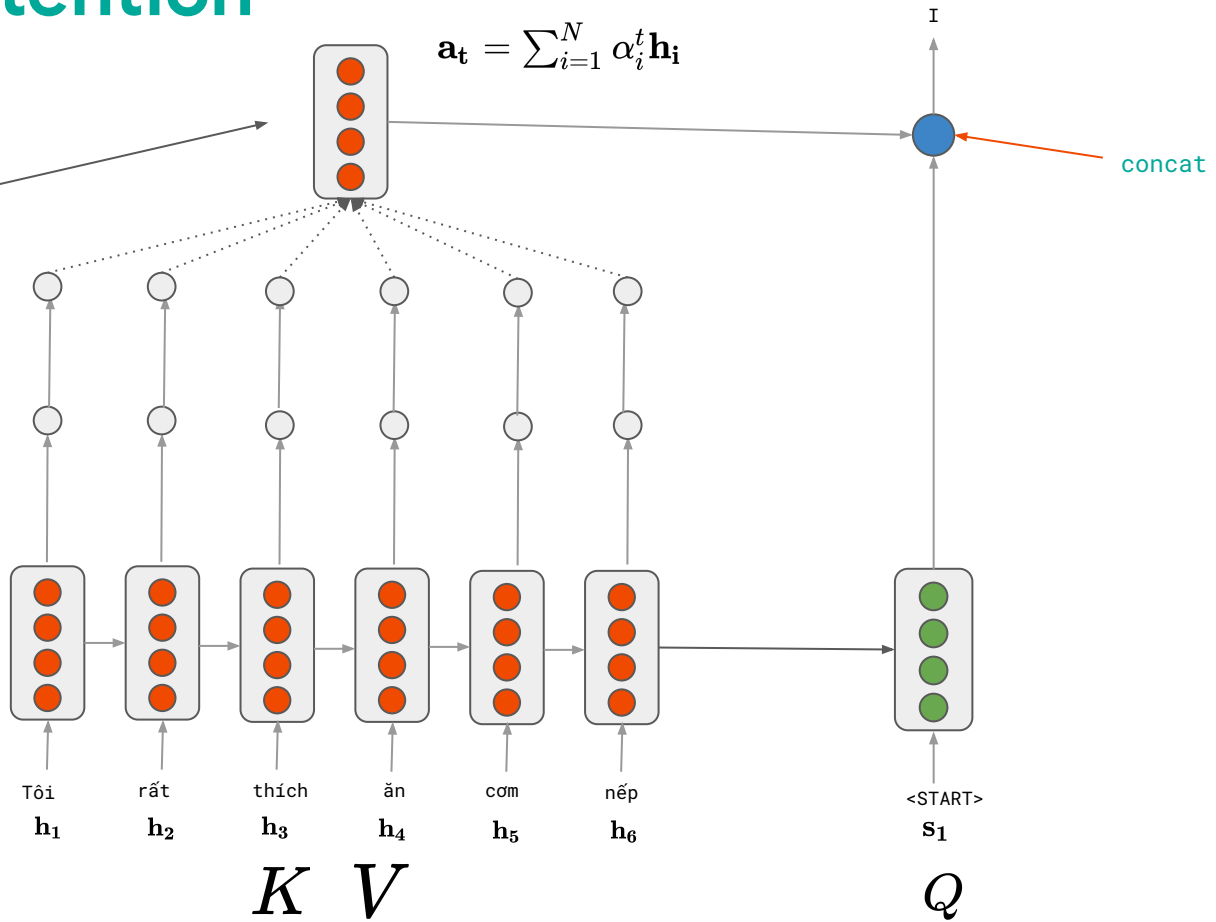
Motivation: Convolutions



- Dễ dàng thực thi song song
- Khai thác tốt phụ thuộc cục bộ (local dependencies)

Motivation: Attention

Tại sao không sử dụng Attention **để biểu diễn cho một từ** ???



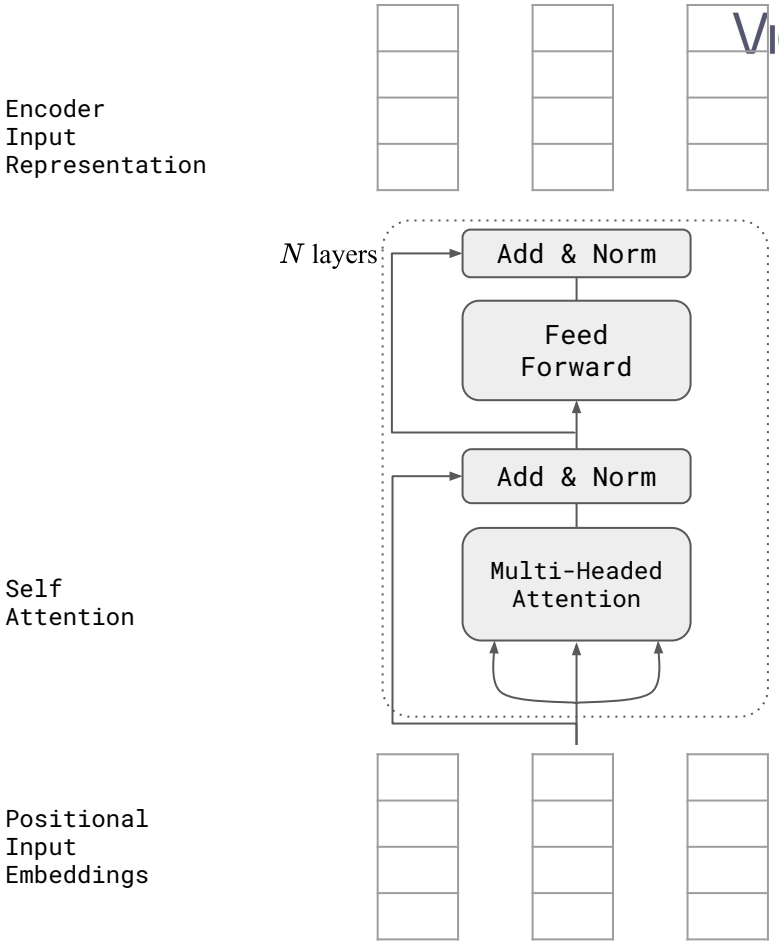
Encoder

Nhiệm vụ của Encoder

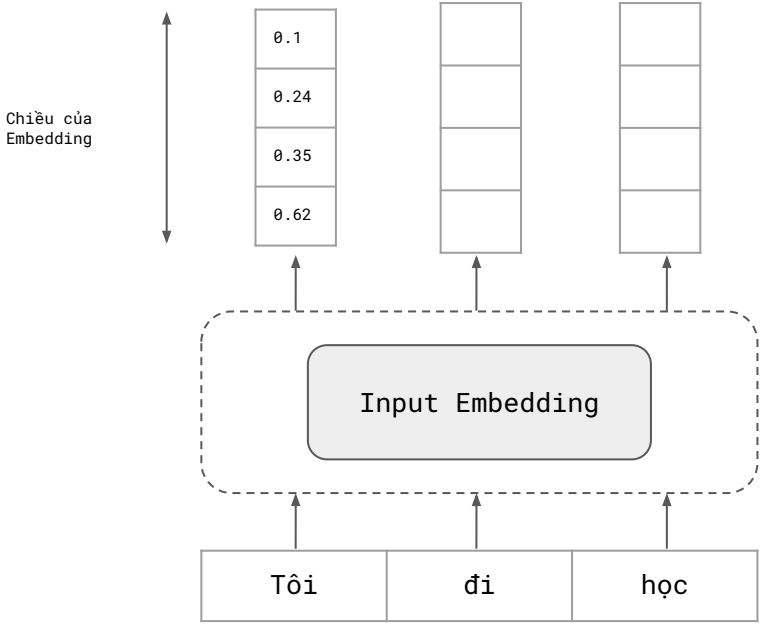
- Học mối tương quan giữa mỗi từ trong câu với các từ còn lại (self Attention)
- Bổ sung mối quan hệ này vào Embedding của từng từ đầu.

Encoder bao gồm 6 layer giống nhau

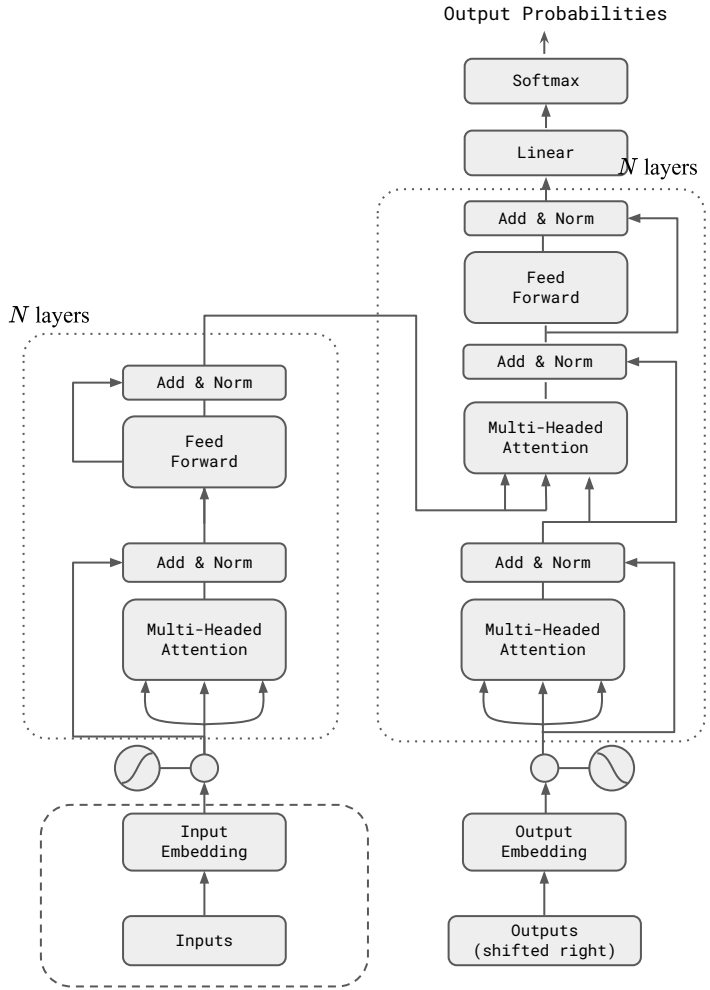
- Mỗi Layer bao gồm 2 phần nhỏ
  - Multi-Headed Attention
  - Position-wise fully connected feed-forward network



Input Embedding

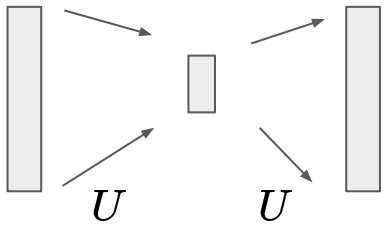


Tương tự như RNN, Seq2Seq mô hình vẫn sử dụng Embedding để mô phỏng từ dưới dạng vector có chiều



Word Embedding with Tying Weights

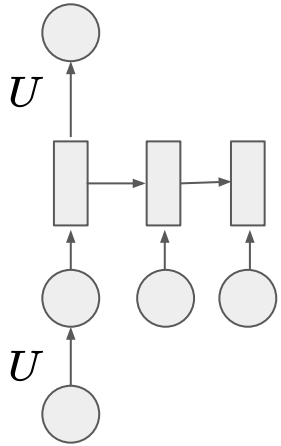
Word2Vec



	Input	Output	Tied
Simlex999	0.30	0.29	0.17
Verb-143	0.41	0.34	0.12
MEN	0.66	0.61	0.50
Rare-Word	0.34	0.34	0.23
MTurk-771	0.59	0.54	0.37

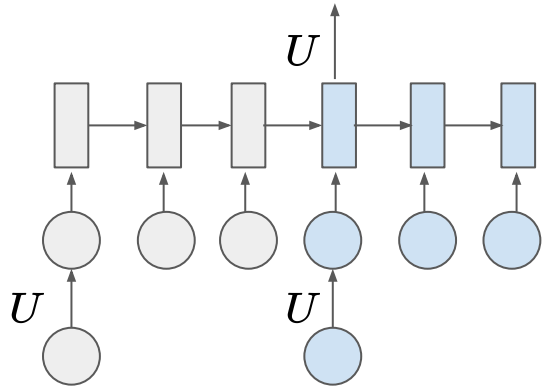
Tied Weight trong mô hình word2vec skip-gram không thực sự hiệu quả

Neural network language models



Trong language model, Output Embedding vượt trội hơn Input Embedding

Neural translation models



Weight tying giúp giảm tham số của mô hình neural translation mà không ảnh hưởng tới hiệu năng

Tying weights là sử dụng chung duy nhất một Embedding cho từ đầu vào cũng như từ đầu ra

https://arxiv.org/pdf/1608.05859.pdf

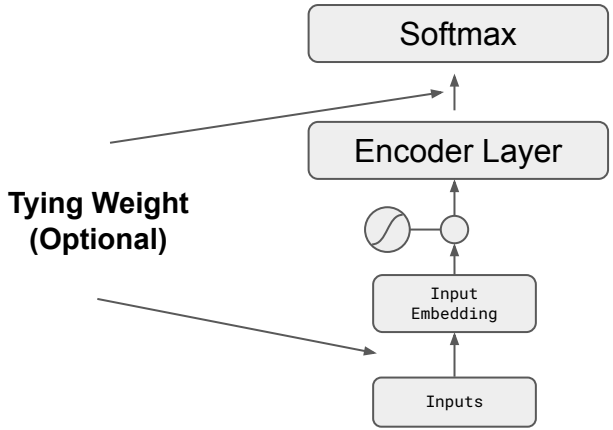
Tying Weights in Transformer (Optional)

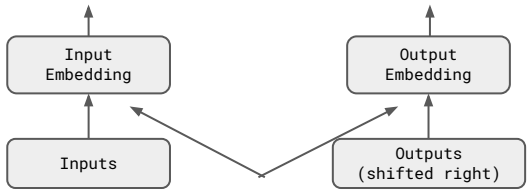
Language pairs	Subwords only in source	Subwords only in target	Subwords in both
EN→FR	2K	7K	85K
EN→DE	3K	11K	80K

Table 1: Shared BPE subwords between pairs of languages.

Việc tying Weights tại Embedding giữa Encoder và Decoder chỉ hiệu quả khi thực hiện bài toán dịch trên các ngôn ngữ tương đồng nhau về từ điển.

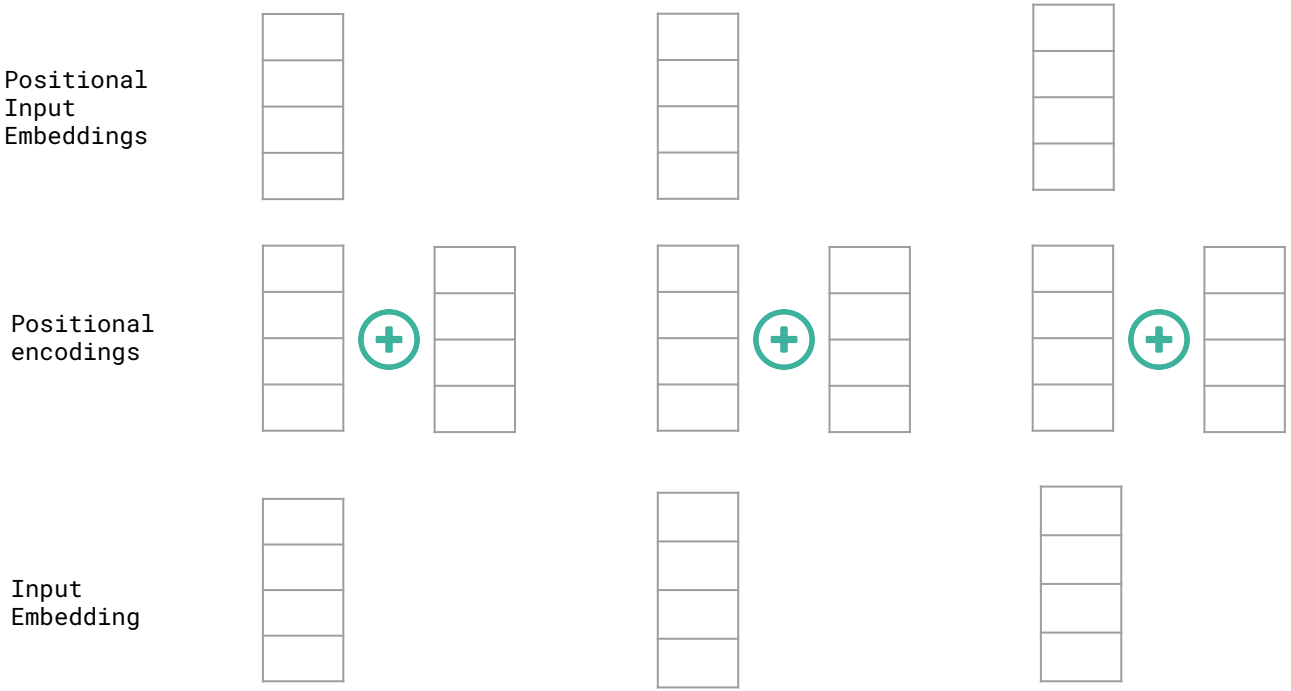
Ví dụ: Tiếng Anh và Tiếng Pháp (cũng như tiếng Đức) **chia sẻ 90% (85%) of BPE subwords**

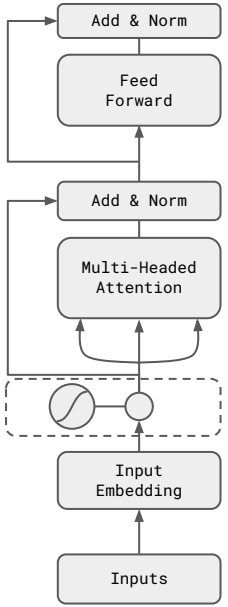




Tying Weight (Optional)

Positional encodings



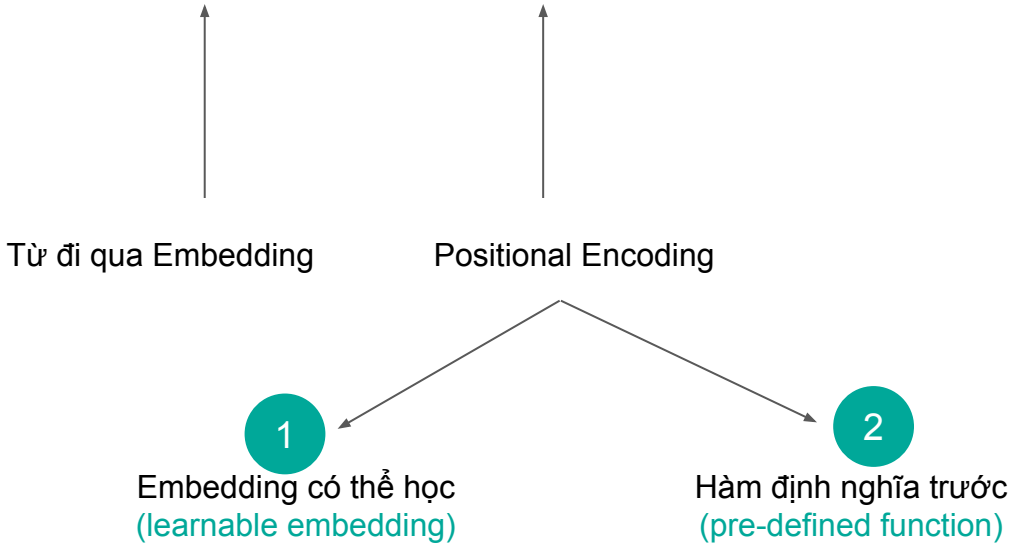


Vì model không có hồi quy (recurrence) hay phép tích chập (convolution), vì thế để đảm bảo thứ tự của chuỗi, chúng ta sẽ phải bổ sung thêm thông tin về trị tương đối và tuyệt đối của các tokens trong chuỗi.

Positional encodings

Bổ sung thông tin về vị trí của từ

$$z_i = WE(x_i) + PE(i)$$



Convolutional Seq2Seq Learning

Attention không ghi nhận thông tin vị trí (position-insensitive)

Sử dụng learned positional embeddings

1. Các từ đầu vào

$$x = (x_1, \ldots, x_m)$$

2. Đưa qua Embedding

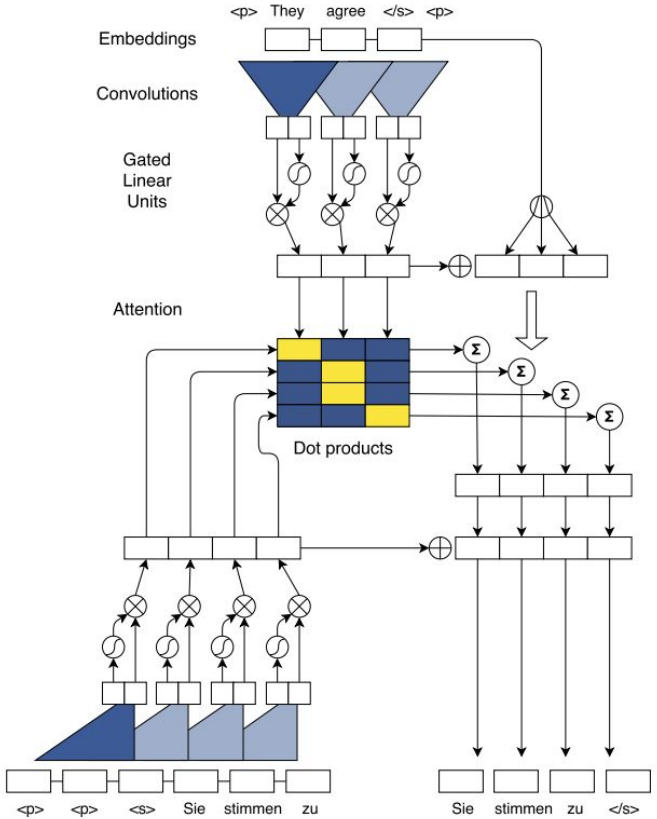
$$w = (w_1, \ldots, w_m)$$

3. Bổ sung thông tin vị trí

$$e = (w_1 + p_1, \ldots, w_m + p_m)$$

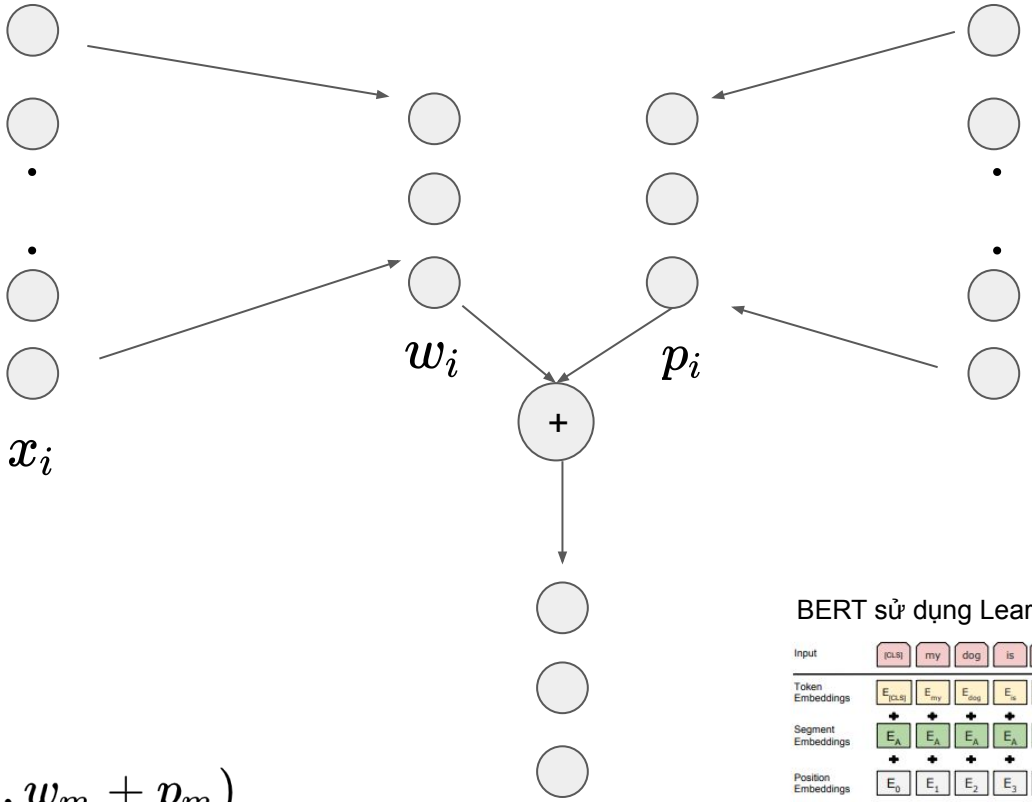
Có thể học

https://arxiv.org/pdf/1705.03122.pdf



Learned positional embeddings

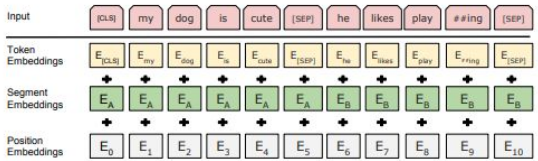
One hot vector
(vocab_size,)



One hot vector
(sentence_length,)

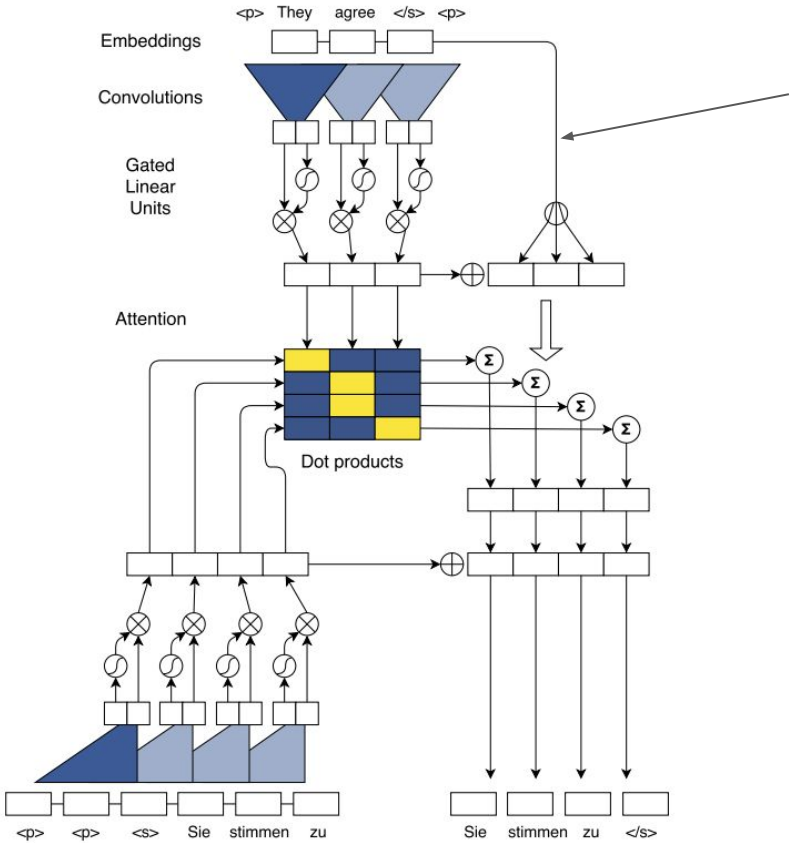
$$e = (w_1 + p_1, \ldots, w_m + p_m)$$

BERT sử dụng Learned positional embeddings



https://arxiv.org/pdf/1810.04805.pdf

Without Position Embeddings



Residual Block

Thú vị: Nếu không sử dụng position embeddings cho đầu vào và đầu ra thì BLEU Score **chỉ giảm 0.5.**

	PPL	BLEU
ConvS2S	6.64	21.7
-source position	6.69	21.3
-target position	6.63	21.5
-source & target position	6.68	21.2

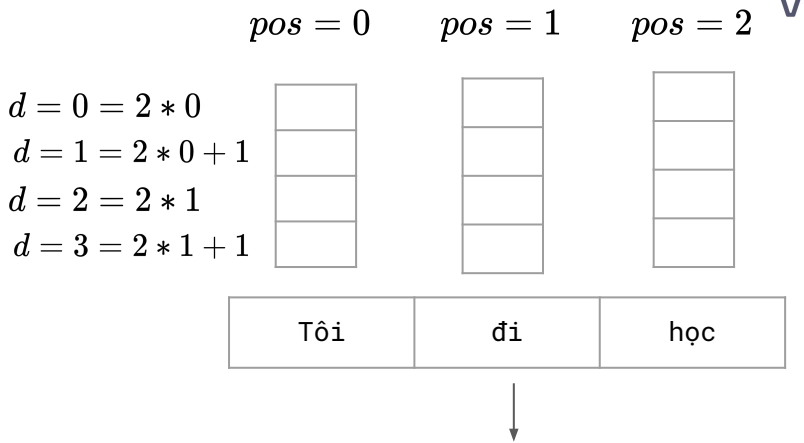
Sinusoid

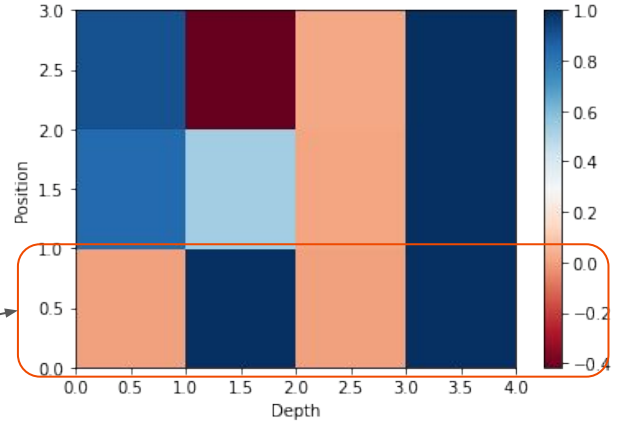
Sinusoidal functions

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}})$$

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

pos: vị trí của token
i: vị trí i trong embedding





Positional encodings cho từ “Tôi”

Sinusoid

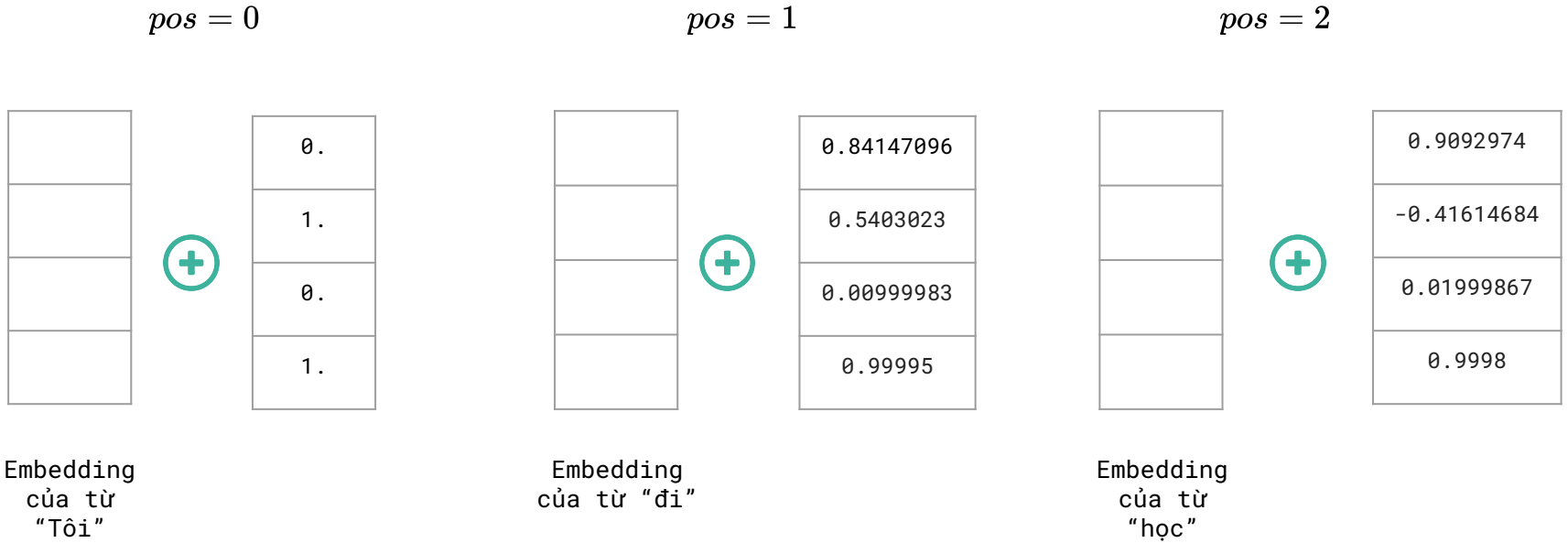
$$d = 0 = 2 * 0$$
$$d = 1 = 2 * 0 + 1$$
$$d = 2 = 2 * 1$$
$$d = 3 = 2 * 1 + 1$$

$pos = 0$	$pos = 1$	$pos = 2$	
0.	0.84147096	0.9092974	$i = 0$
1.	0.5403023	-0.41614684	$i = 0$
0.	0.00999983	0.01999867	$i = 1$
1.	0.99995	0.9998	$i = 1$

$$PE_{(1,1)} = cos(\frac{1}{10000^{\frac{2*0}{4}}})$$

$$PE_{(2,2)} = sin(\frac{2}{10000^{\frac{2*1}{4}}})$$

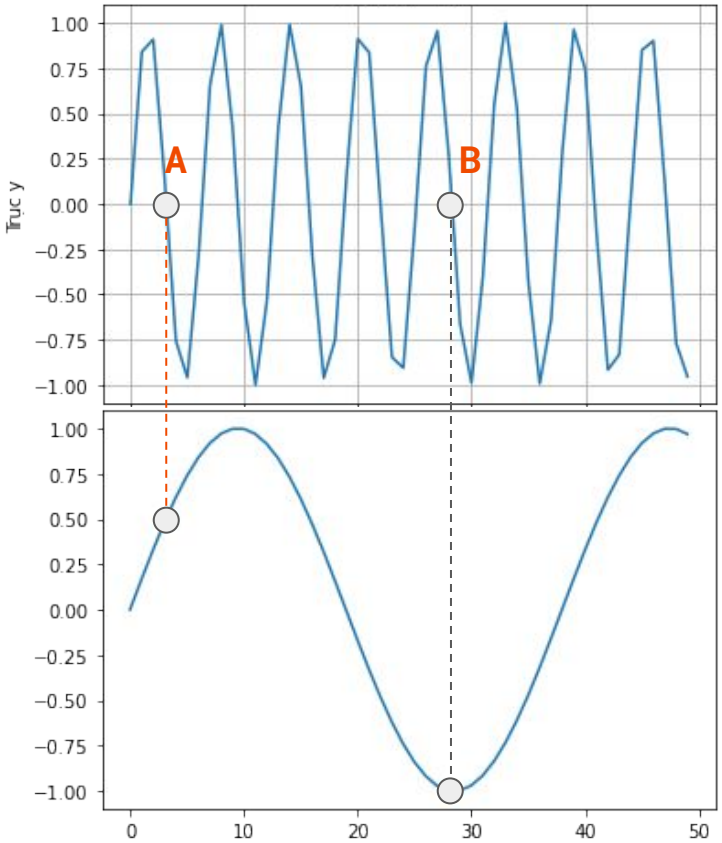
Sinusoid


$pos = 0$
$pos = 1$
$pos = 2$
0.
1.
0.
1.
0.84147096
0.5403023
0.00999983
0.99995
0.9092974
-0.41614684
0.01999867
0.9998
Embedding của từ "Tôi"
Embedding của từ "đi"
Embedding của từ "học"

Sinusoid

d = 0

d = 100



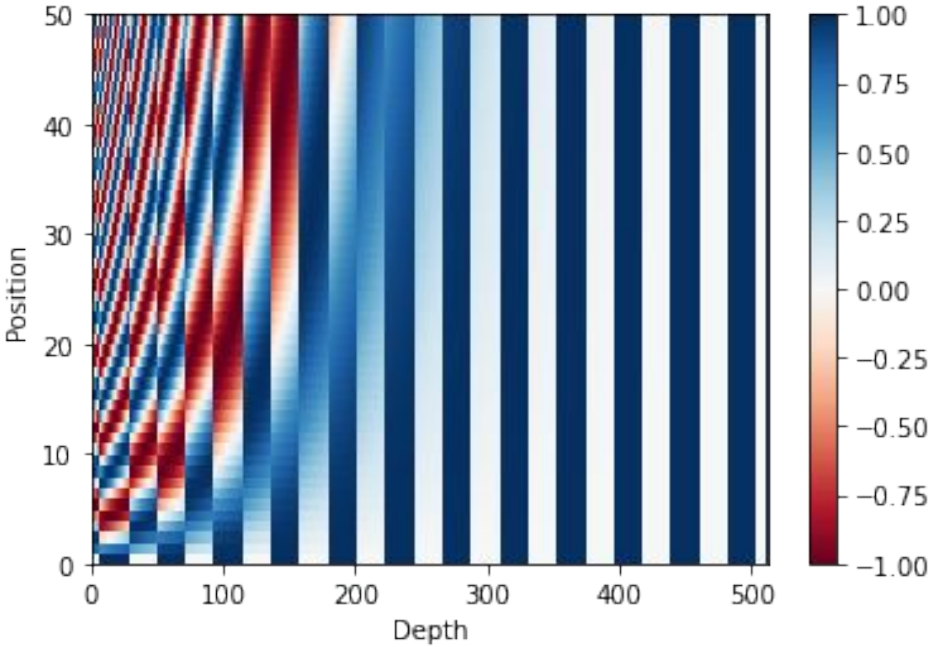
Chiều dài của câu: 52
Chiều của Embedding: 512

Mặc dù 2 vị trí A và B, khi d = 0 trong embedding, giá trị 2 vị trí là tương đồng nhau.

Tuy nhiên với d lớn hơn trong embedding, sự khác nhau giá trị 2 vị trí rõ ràng hơn rất nhiều.

Điều này cho thấy **sự hiệu quả** trong việc sử dụng công thức PE trên vector embedding để tạo ra positional encoding

Sinusoid



Chiều dài của câu: 52
Chiều của Embedding: 512

1 1 0 1
1 1 0 0

Sự giống ở những d ban đầu và khác dần ở các d tiếp theo có thể liên tưởng đến bit.

Sinusoid - Relative Positioning

Sử dụng sinusoidal positional encoding cho phép model lấy thông tin về vị trí tương đối (relative positions) một cách hiệu quả.

> *We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PEpos+k can be represented as a linear function of PEpos.*

Với mỗi cặp **sin-cos** tương ứng, tồn tại ma trận $M \in \mathbb{R}^{2\times 2}$ không phụ thuộc vào vị trí (pos)

$$M \cdot \begin{bmatrix} \sin(\omega_i . pos) \\ \cos(\omega_i . pos) \end{bmatrix} = \begin{bmatrix} \sin(\omega_i . (pos + \phi)) \\ \cos(\omega_i . (pos + \phi)) \end{bmatrix}$$

ϕ offset giữa 2 vị trí

$$\omega_i = \frac{1}{10000^{\frac{2i}{d_{\text{model}}}}}$$

Proof

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Position Embedding Target

Cho position space $\mathcal{P}$ và input embedding space $\mathcal{X}$

Nhiệm vụ của hàm positional embedding là học mapping $f : \mathcal{P} \rightarrow \mathcal{X}$

Absolute Position

Nếu positional embedding có thể học được absolute position, sẽ tồn tại một hàm đảo ngược

$$g : \mathcal{X} \rightarrow \mathcal{P}$$

Type	PE	MAE
Learned	BERT	34.14
	RoBERTa	6.06
	GPT-2	1.03
Pre-Defined	sinusoid	0.0

Table 1: Mean absolute error of the reversed mapping function learned by linear regression.

Hàm sinusoid mô phỏng hoàn hảo vị trí tuyệt đối (absolute position)

Relative Position

Bên cạnh vị trí tuyệt đối, mối quan hệ giữa 2 vị trí tương đối cũng sẽ có thông tin (relative positions)

Xây dựng mô hình logistic để kiểm tra liệu các embeddings có thể mô phỏng thứ tự của 2 vị trí

Type	PE	Error Rate
Learned	BERT	19.72%
	RoBERTa	7.23%
	GPT-2	1.56%
Pre-Defined	sinusoid	5.08%

Transformer Encoders

Transformer Decoder

Table 2: Error rate of the relative position regression.

Position embeddings của Transformer encoders vẫn học rất ít thông tin về vị trí tương đối.

https://arxiv.org/pdf/2010.04903.pdf

Explanation

Tại sao Position embeddings của Transformer Encoders lại ít học thông tin vị trí tương đối cũng như tương đối ???

Absolute Position

Type	PE	MAE
Learned	BERT	34.14
	RoBERTa	6.06
	GPT-2	1.03
Pre-Defined	sinusoid	0.0

Table 1: Mean absolute error of the reversed mapping function learned by linear regression.

Relative Position

Type	PE	Error Rate
Learned	BERT	19.72%
	RoBERTa	7.23%
	GPT-2	1.56%
Pre-Defined	sinusoid	5.08%

Table 2: Error rate of the relative position regression.

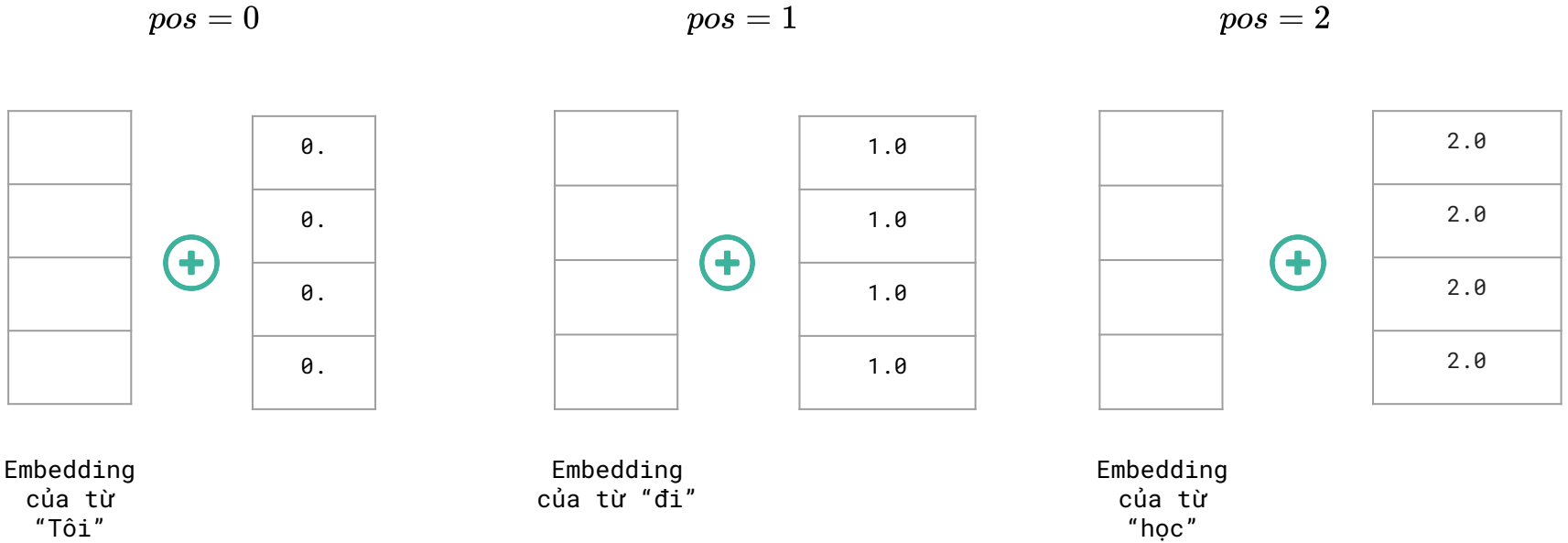
Giải thích: Encoder cực tiểu hàm mất mát của masked language modeling. Cho nên Encoder có thể dự đoán thành công một từ dựa vào những **từ ngay xung quanh** thay vì phải học vị trí tuyệt đối cũng như tương đối của tất cả các từ trong câu.

$$\mathcal{L}(\mathcal{U}) = \sum_i \log P(u_i|u_1, ...u_{i-1}, u_{i+1}, ...u_l; \theta),$$

Trái lại, Decoder cực tiểu hàm mất mát autoregressive language modeling. Cho nên yêu cầu về thông tin vị trí tuyệt đối / tương đối của các từ phía trước cao hơn so với Encoder.

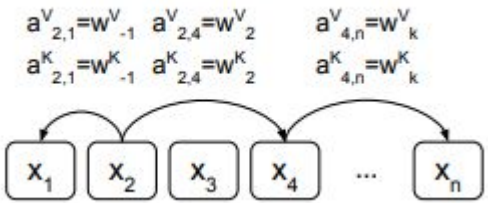
$$\mathcal{L}(\mathcal{U}) = \sum_i \log P(u_i|u_1, ...u_{i-1}; \theta).$$

Positional encodings Quiz



Thiết kế Positional encodings như thế này có ổn hay không???

Further Work



$$e_{ij} = \frac{x_i W^Q (x_j W^K + a^K_{ij})^T}{\sqrt{d_z}}$$

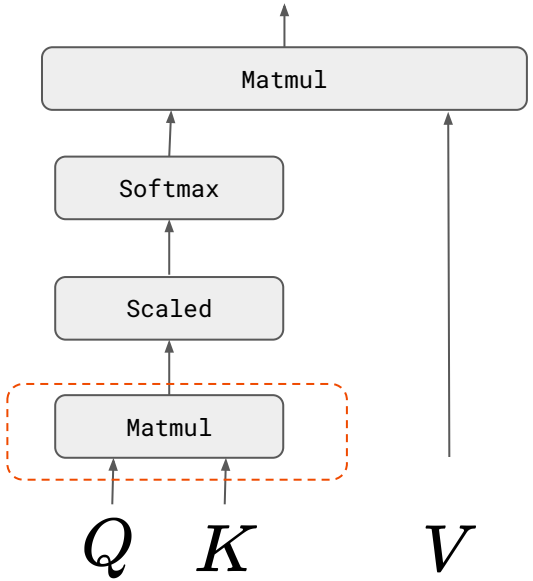
Vector thể hiện mối quan hệ tương đối giữa vị trí i và j

$$z_i = \sum_{j=1}^{n} \alpha_{ij} (x_j W^V + a^V_{ij})$$

Mô phỏng quan hệ vị trí tương đối (relative position) thông qua đồ thị

https://arxiv.org/abs/1803.02155

Scaled dot product attention



Queries Keys Values

Q

Tôi
đi
học

K

Tôi
đi
học

V

Tôi
đi
học

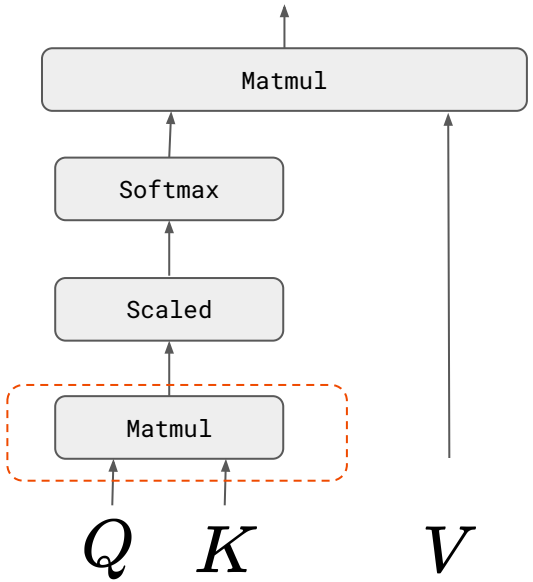
1.	2.	3.	4.
5.	2.	1.	3.
4.	3.	2.	1.

1.	2.	3.	4.
5.	2.	1.	3.
4.	3.	2.	1.

1.	2.	3.	4.
5.	2.	1.	3.
4.	3.	2.	1.

$$Q \in \mathbb{R}^{(\text{length}_q) \times d_k} \quad K \in \mathbb{R}^{(\text{length}_k) \times d_k} \quad V \in \mathbb{R}^{(\text{length}_v) \times d_v}$$

Scaled dot product attention



Q

Tôi
đi
học

1.	2.	3.	4.
5.	2.	1.	3.
4.	3.	2.	1.

K

Tôi
đi
học

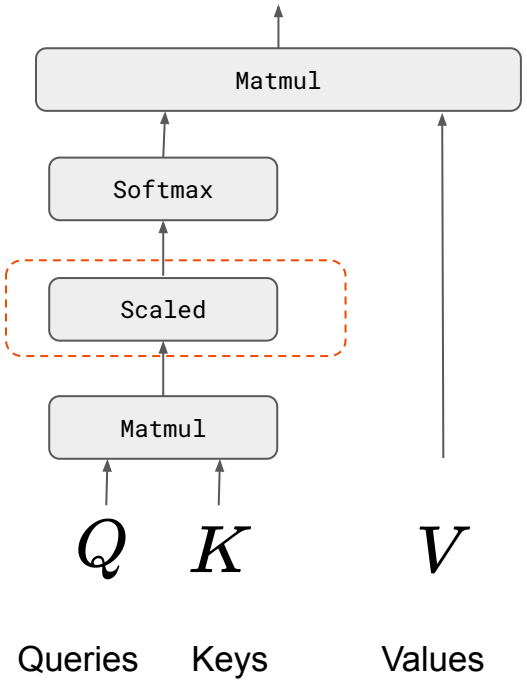
1.	2.	3.	4.
5.	2.	1.	3.
4.	3.	2.	1.

QK^T

	Tôi	đi	học
Tôi	30.	24.	21.
đi	24.	39.	33.
học	21.	33.	30.

$Q \in \mathbb{R}^{(\text{length}_q) \times d_k}$ $K \in \mathbb{R}^{(\text{length}_k) \times d_k}$ $QK^T \in \mathbb{R}^{(\text{length}_q \times \text{length}_k)}$

Scaled dot product attention



QK^T

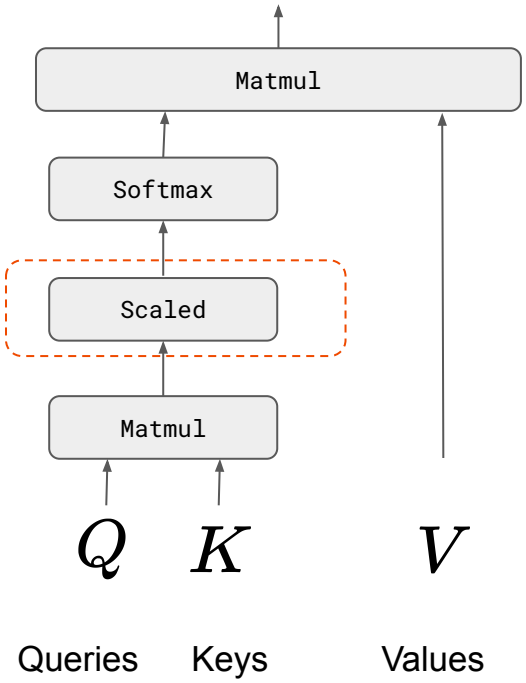
	Tôi	đi	học
Tôi	30.	24.	21.
đi	24.	39.	33.
học	21.	33.	30.

$\frac{QK^T}{\sqrt{d_k}}$

	Tôi	đi	học
Tôi	15.	12.	10.5
đi	12.	19.5	16.5
học	10.5	16.5	15.

Tại sao phải chia cho $\sqrt{d_k}$

Scaled dot product attention



QK^T

$$\frac{QK^T}{\sqrt{d_k}}$$

Tại sao phải chia cho $\sqrt{d_k}$

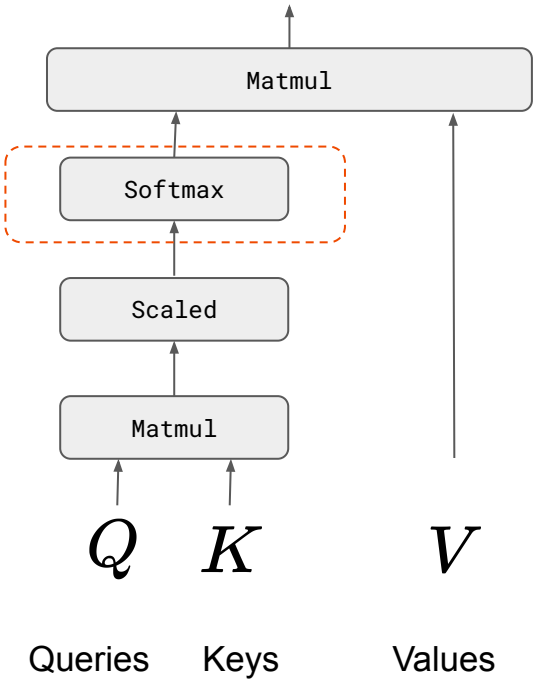
Giải thích

Việc nhân dot product có xu hướng tăng cường độ lớn, có xu hướng làm cho hàm softmax trả về những giá trị rất nhỏ (Hàm softmax có xu hướng tăng cường giá trị lớn, giảm nhẹ giá trị nhỏ).

Nếu q và k là biến ngẫu nhiên độc lập với mean = 0 và variance = 1, thì dot produc của 2 biến này có mean = 0 và variance = d_k

Để tránh tình trạng này chúng ta scale kết quả dot product với giá trị: $\sqrt{d_k}$

Scaled dot product attention



$$\frac{QK^T}{\sqrt{d_k}}$$

	Tôi	đi	học
Tôi	15.	12.	10.5
đi	12.	19.5	16.5
học	10.5	16.5	15.

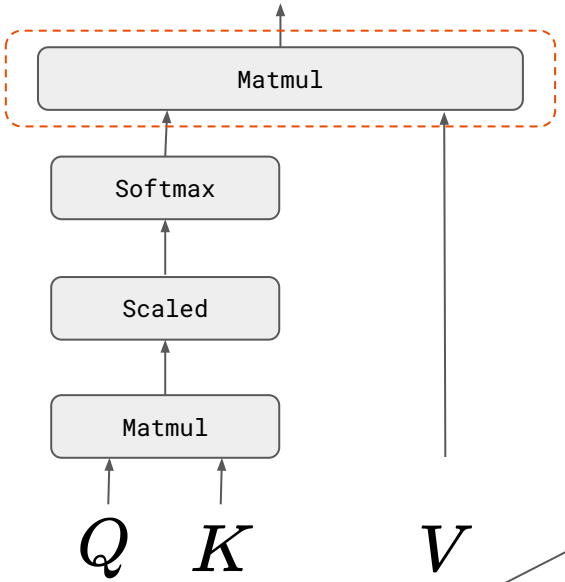
Sự tương xứng của "Tôi" với các từ còn lại

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$ Attention Weights

Tổng các giá trị bằng 1

	Tôi	đi	học
Tôi	9.42599405e-01	4.69292610e-02	1.04713335e-02
đi	5.26576432e-04	9.52072524e-01	4.74008998e-02
học	2.02246585e-03	8.15920960e-01	1.82056574e-01

Scaled dot product attention



$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

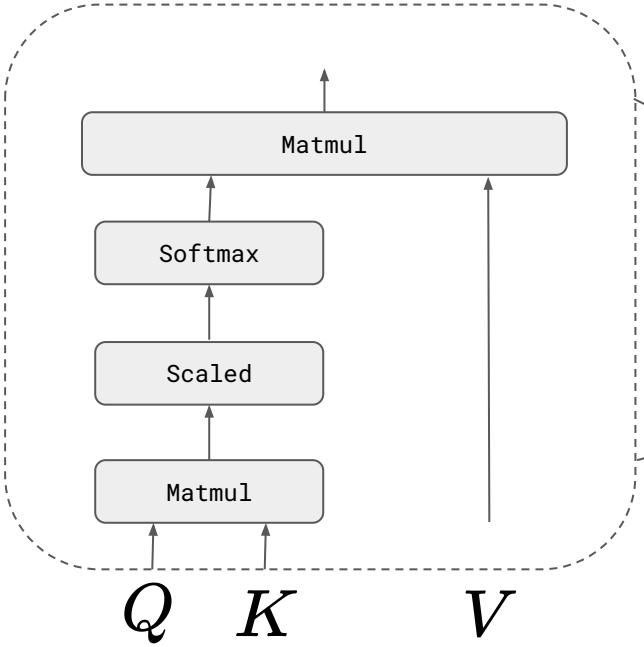
	Tôi	đi	học
Tôi	9.42599405e-01	4.69292610e-02	1.04713335e-02
đi	5.26576432e-04	9.52072524e-01	4.74008998e-02
học	2.02246585e-03	8.15920960e-01	1.82056574e-01

$$\text{Attention(Q, K, V)} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad \text{Attention(Q, K, V)} \in \mathbb{R}^{(\text{length}_q)\times d_v}$$

Tôi	1.21913104	2.01047133	2.88519881	3.93212807
đi	4.95049279	2.0474009	1.00105315	2.95312568
học	4.80985356	2.18205657	1.00404493	2.81996589

Những vector này tương đồng với **context vector** trong những bài trước

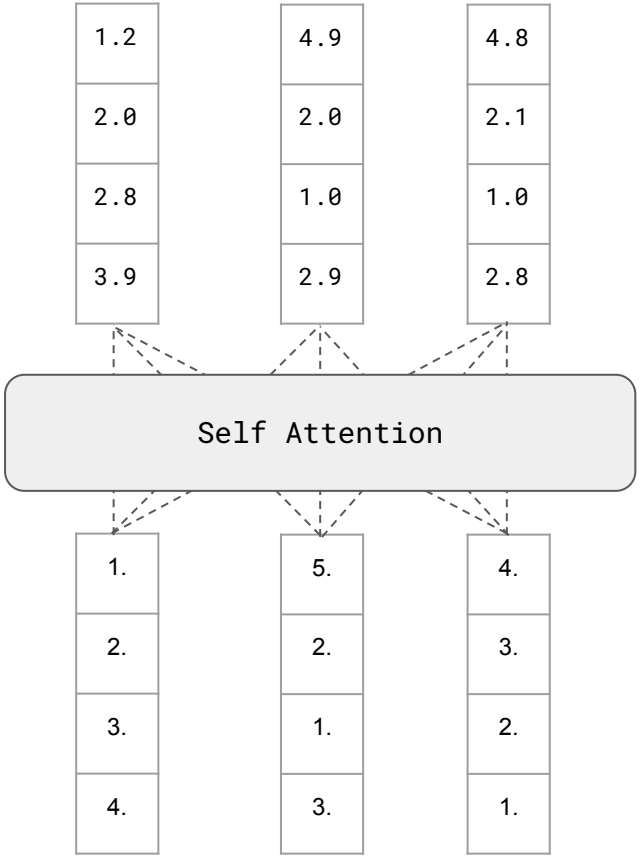
Why Self Attention



Scaled Dot-Product Attention

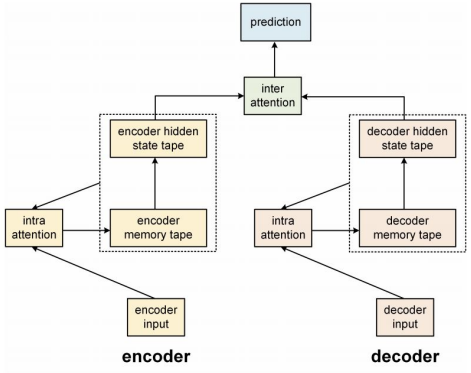
1. Độ phức tạp tính toán mỗi lớp giảm đi
2. Các phép tính toán có thể thực hiện song song
3. Giải quyết vấn đề học long-range dependencies

Self Attention (intra-attention)

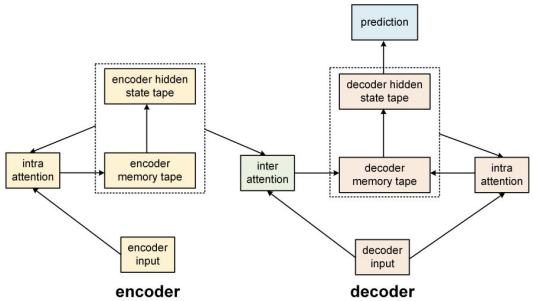


Self Attention áp dụng để học biểu diễn của một câu dựa vào mọi từ với những từ xung quanh nó.

Self Attention trong bài toán Reading comprehension



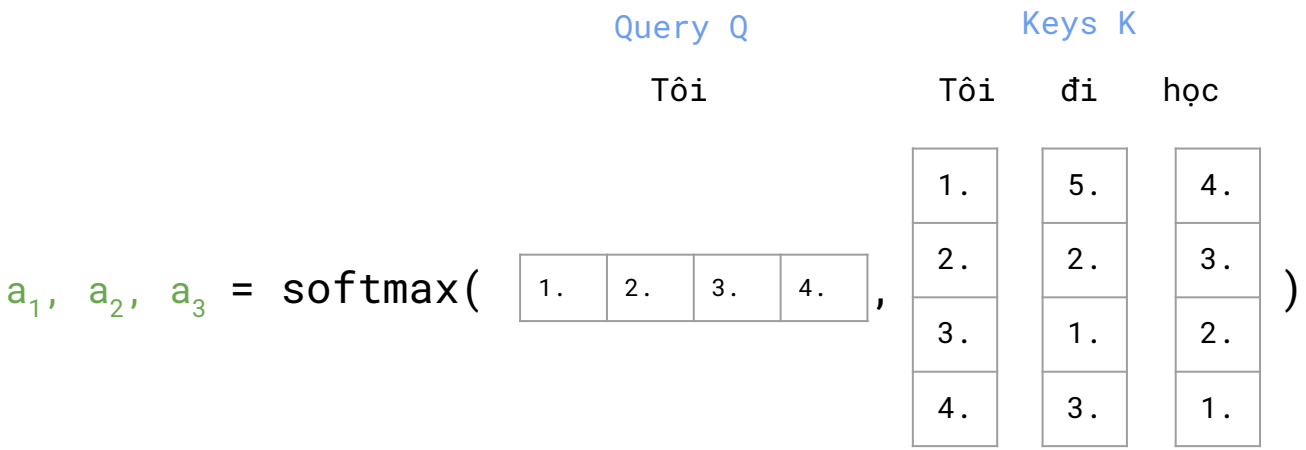
(a) Decoder with shallow attention fusion.

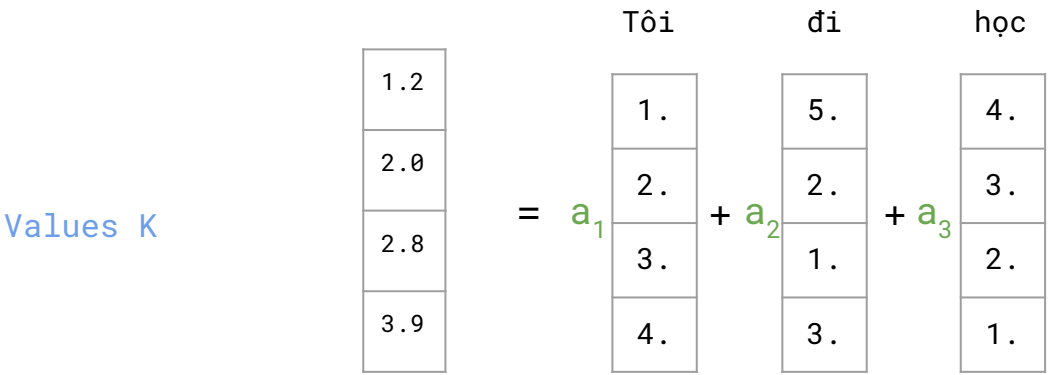


(b) Decoder with deep attention fusion.

https://arxiv.org/pdf/1601.06733.pdf

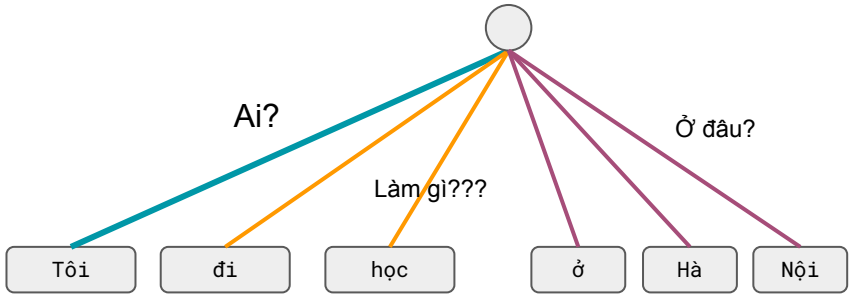
Self Attention





Self Attention Problem

Convolutions

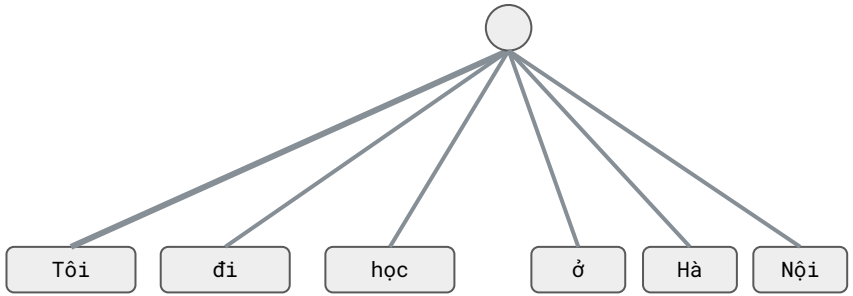


Khi sử dụng Convolution, model bóc tách một số thông tin từ

- embedding của từ “Tôi” để học cách đặt câu hỏi: “Ai?”
- embedding của từ “đi” và từ “học” để đặt câu hỏi: “Làm gì?”
- embedding của từ “ở” và “Hà Nội” để đặt câu hỏi “Ở đâu?”

Self Attention Problem

Self-Attention

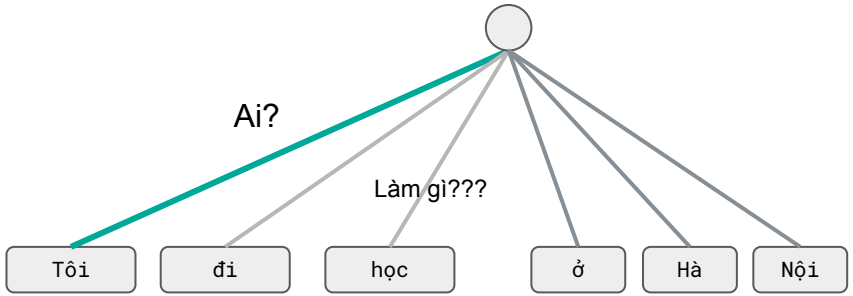


Khi sử dụng nhất duy nhất một Self-Attention, model thực hiện biến đổi tuyến tính trên toàn bộ embedding của các từ, cho nên không có khả năng tương tự như Convolution - bóc tách từng phần nhỏ thông tin tại các khu vực nhất định.

Cách giải quyết???

Self Attention Solution

Attention Head: Tôi

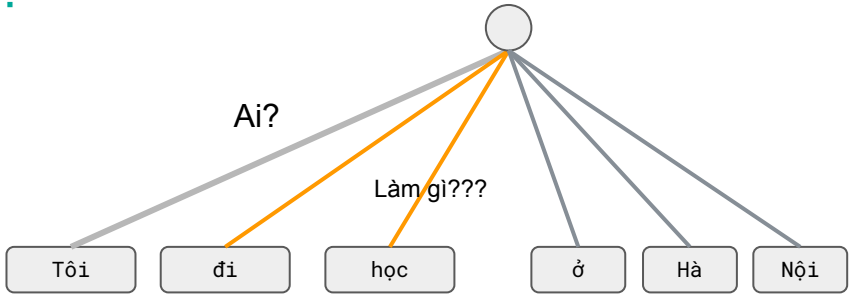


Mô hình sẽ sử dụng nhiều Self-Attention, mỗi attention này sẽ phụ trách học một phần thông tin của câu.

Attention Head này học embedding của từ “Tôi” để học cách đặt câu hỏi: “Ai?”

Self Attention Solution

Attention Head: đi học

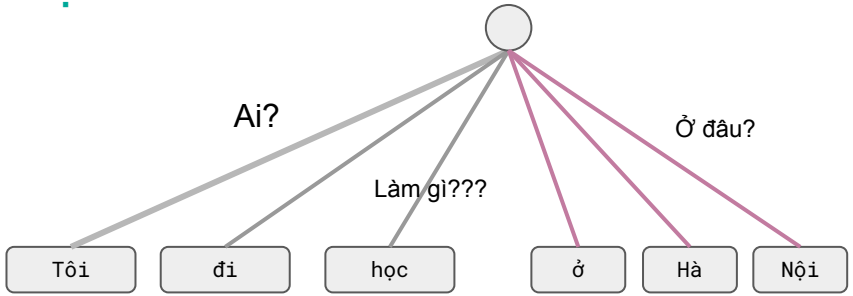


Mô hình sẽ sử dụng nhiều Self-Attention, mỗi attention này sẽ phụ trách học một phần thông tin của câu.

Attention Head này học embedding của từ “đi” và từ “học” để đặt câu hỏi: “Làm gì?”

Self Attention Solution

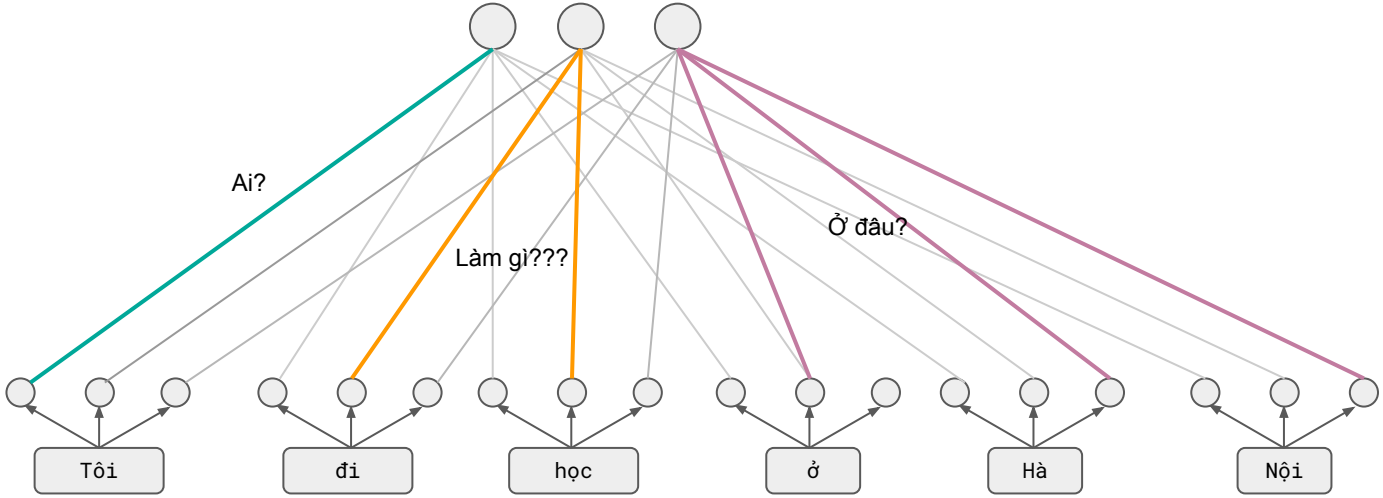
Attention Head: ở Hà Nội



Mô hình sẽ sử dụng nhiều Self-Attention, mỗi attention này sẽ phụ trách học một phần thông tin của câu.

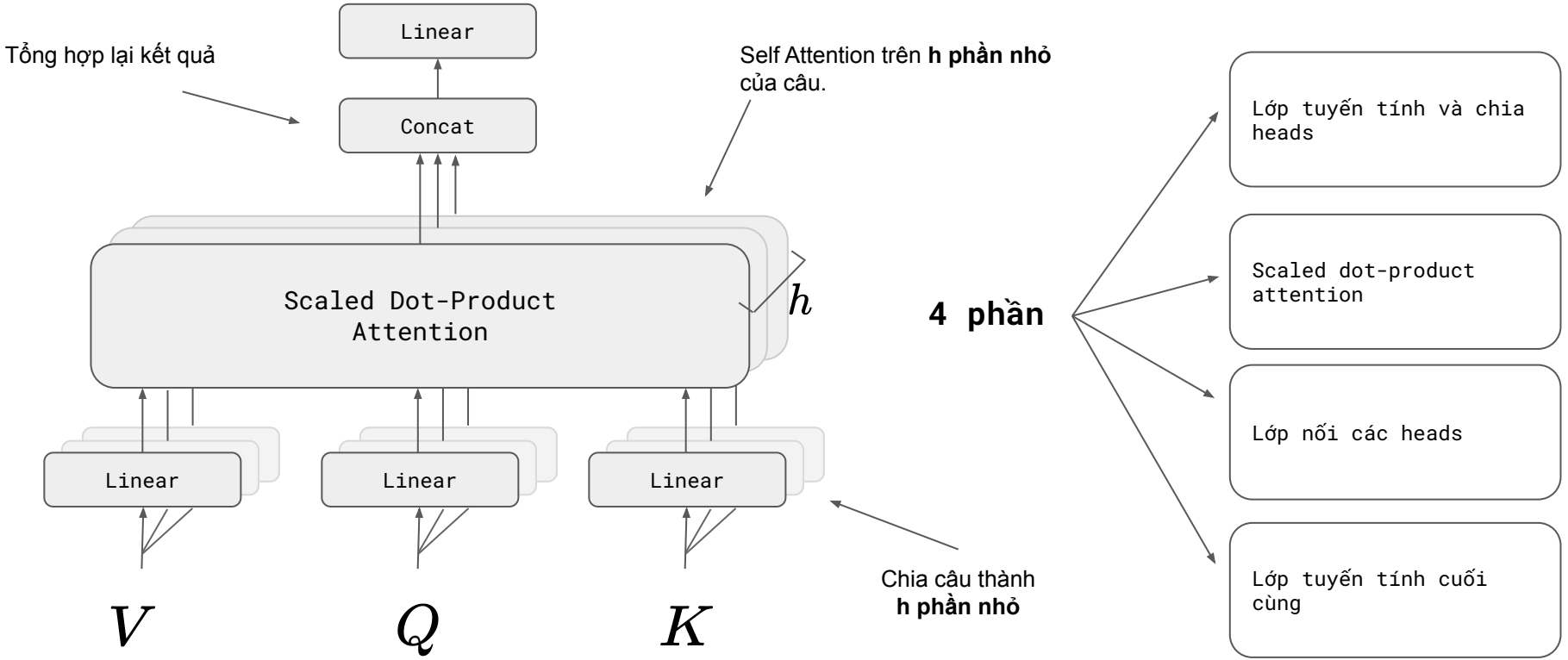
Attention Head này học embedding của từ “ở” và “Hà Nội” để đặt câu hỏi “Ở đâu?”

Multi-Headed Attention

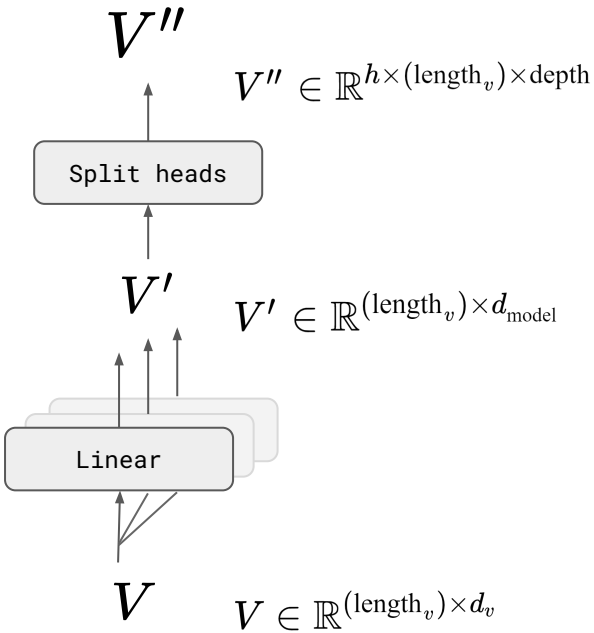


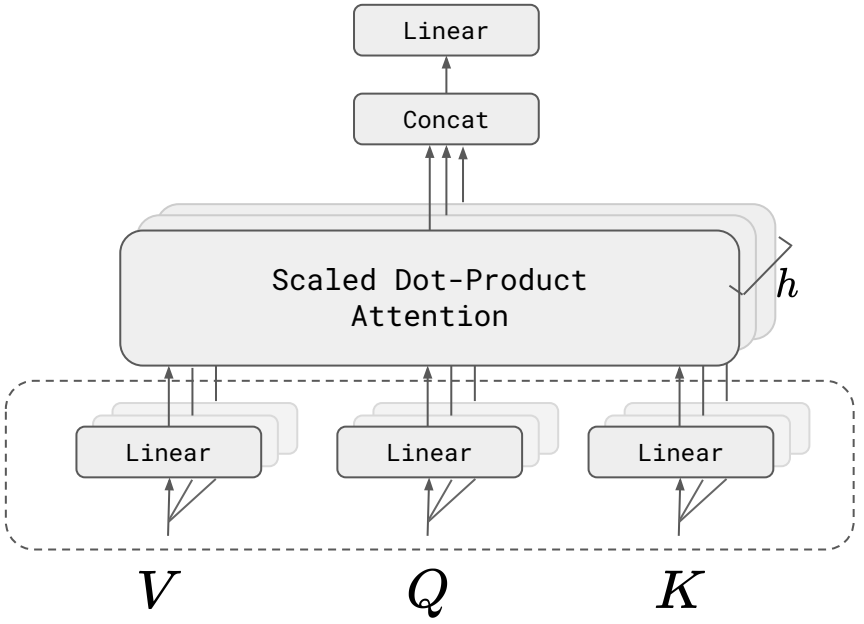
Multi-Headed Attention mô phỏng việc học tương tự như Convolutions,
và đặc biệt là tận dụng sức mạnh tính toán song song.

Multi-Headed Attention



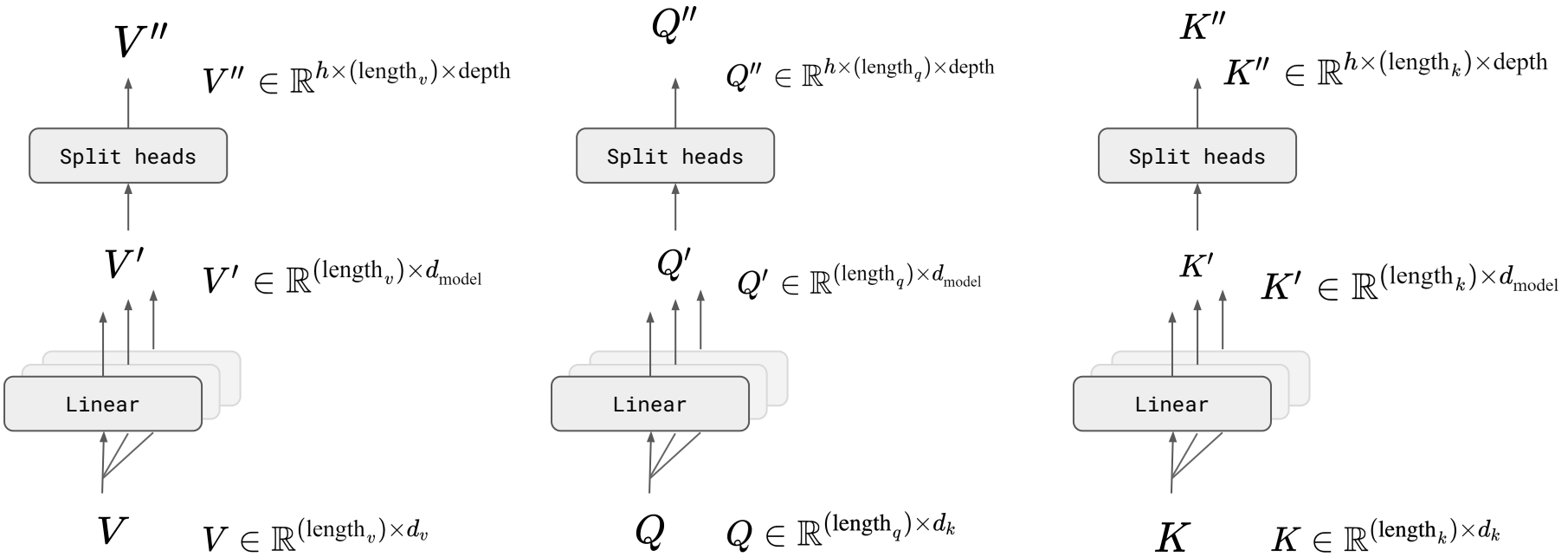
Linear layers and split into heads





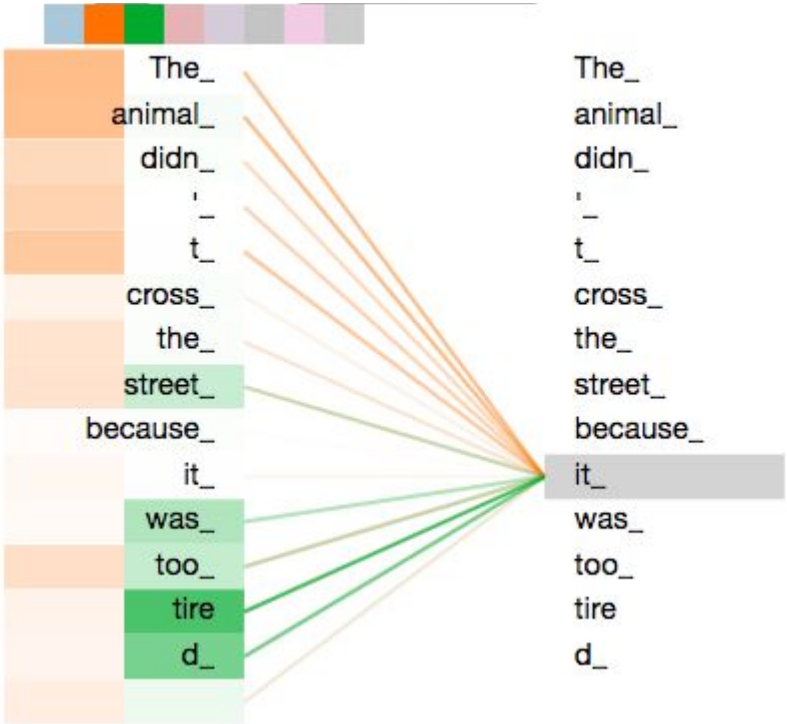
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Linear layers and split into heads



h `number of heads`

Multi-head attention Visualization



Mỗi một Attention Head thực hiện một nhiệm vụ khác nhau để miêu tả từ “it”

- Attention Head đầu tập trung vào “The” và “animal”
- Attention tiếp theo tập trung vào “tired”

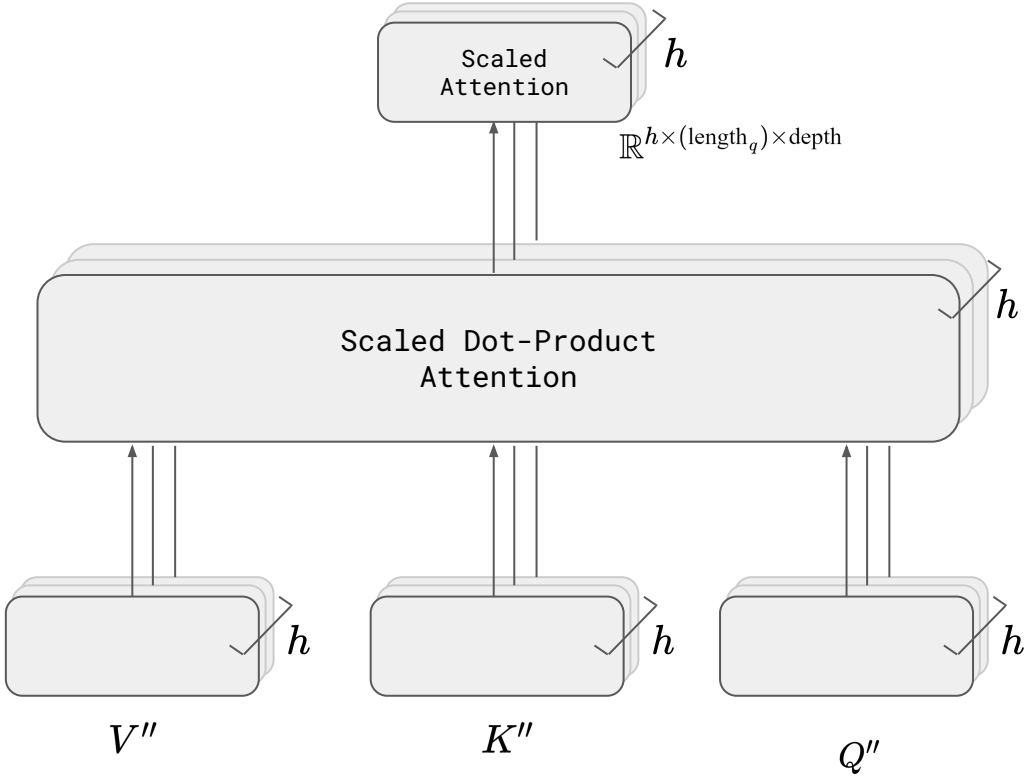
Kết quả thực hiện Attention của các head khác nhau

https://jalammar.github.io/illustrated-transformer

Scaled dot product attention

Attention Weights

$\mathbb{R}^{h \times (\text{length}_q) \times (\text{length}_k)}$

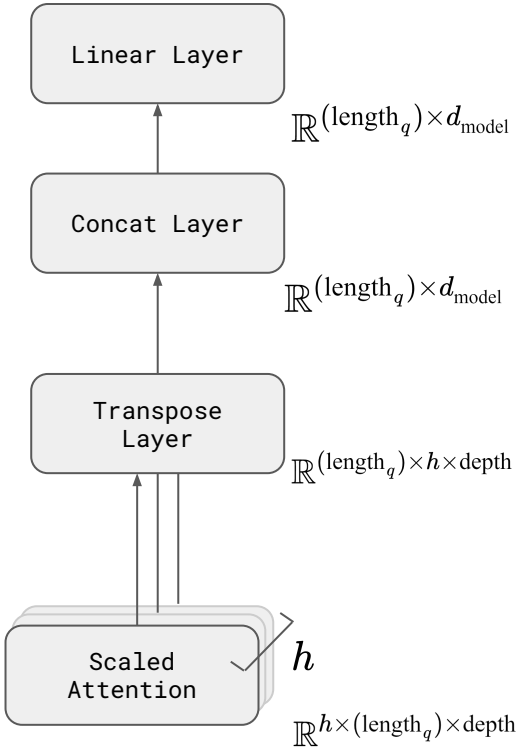


$V'' \in \mathbb{R}^{h \times (\text{length}_v) \times \text{depth}}$

$K'' \in \mathbb{R}^{h \times (\text{length}_k) \times \text{depth}}$

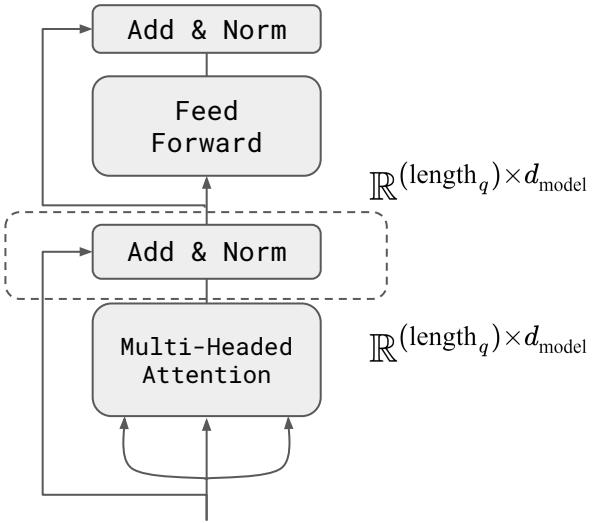
$Q'' \in \mathbb{R}^{h \times (\text{length}_q) \times \text{depth}}$

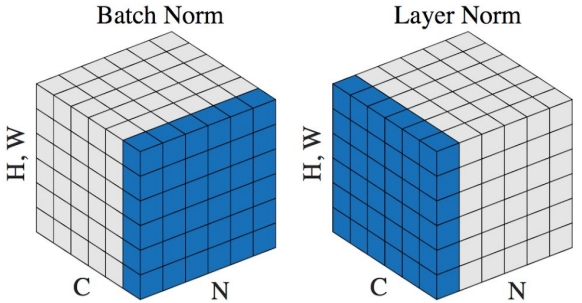
ConCat and Final Linear Layer



Sau khi thực hiện Attention, model nối kết quả Attention trên từng Head và đưa chiều của thông tin về chiều bằng với chiều đầu vào.

Add & Norm Layer





Transformer sử dụng **Layer Norm** để chuẩn hóa dữ liệu nhằm cải thiện khả năng hội tụ của mô hình.

Khác với Batch Normalization, Layer Norm không làm việc trên batch, thay vào đó chuẩn hóa từng mẫu

https://arxiv.org/pdf/1803.08494.pdf

Layer Normalization

Layer Norm với RNN

$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{a}^t - \mu^t) + \mathbf{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^t - \mu^t)^2} \qquad (4)$$

H: số lượng hidden units
a: giá trị hidden
g,b: gain và bias parameters

Khác với Batch Norm, tất cả hidden units trong một lớp sử dụng chung một thông số chuẩn hóa (**normalization term - mu và sigma)**, những data point khác nhau sử dụng thông số chuẩn hóa khác nhau.

Trong **LayerNorm**, Không có sự liên quan độ lớn của minibatch.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

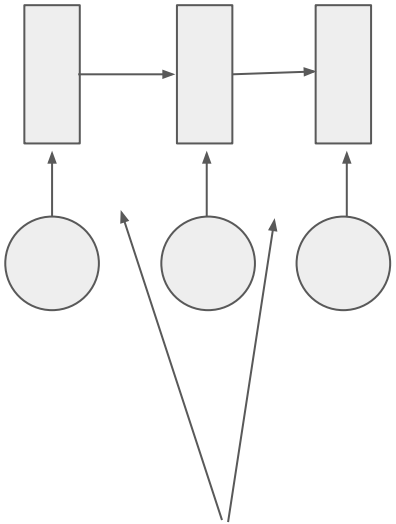
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

https://arxiv.org/pdf/1607.06450v1.pdf

https://arxiv.org/pdf/1502.03167.pdf

Why Layer Normalization



RNN chia sẻ Weight giữa các time step

Batch Norm với RNN

Nếu sử dụng Batch Norm, phải tính toán và lưu trữ **gain và bias parameters** cho từng timestep.

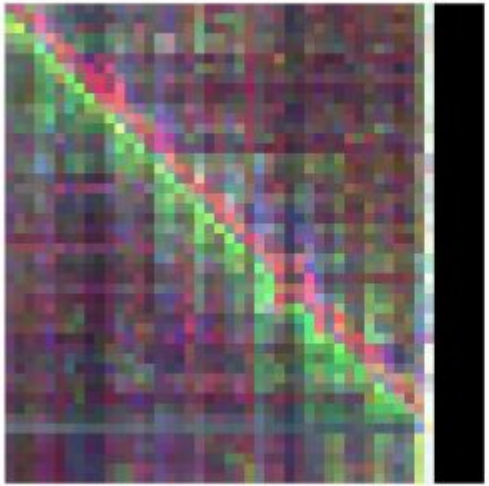
Vấn đề sẽ xảy ra khi câu test **có chiều dài lớn hơn** so với các câu được training.

Layer Norm với RNN

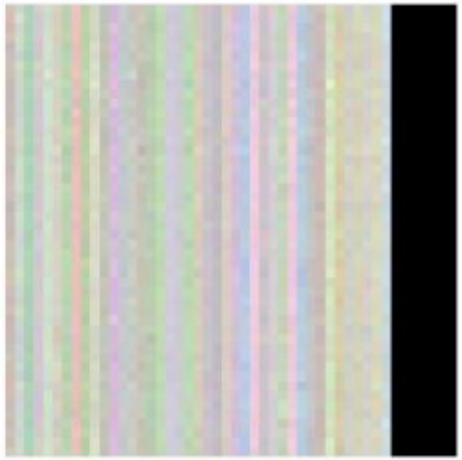
Trái với Batch Norm, Layer Norm chỉ duy trì duy nhất một cặp **gain và bias parameters** cho tất cả timestep.

Residuals Layer

Ngoài việc giúp Gradient Flow tốt hơn - cải thiện việc training, Residual Layer mang theo thông tin vị trí (positional information) từ layer thấp lên layer cao.



With residuals



Without residuals

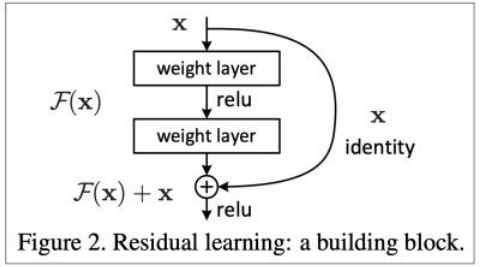
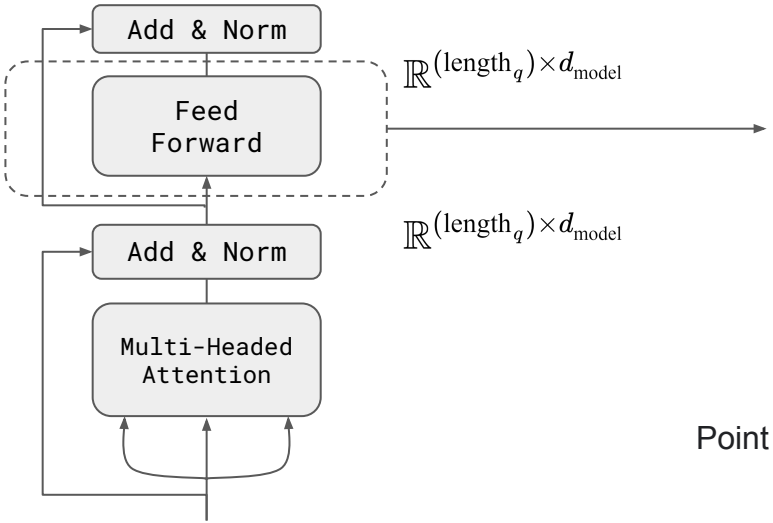


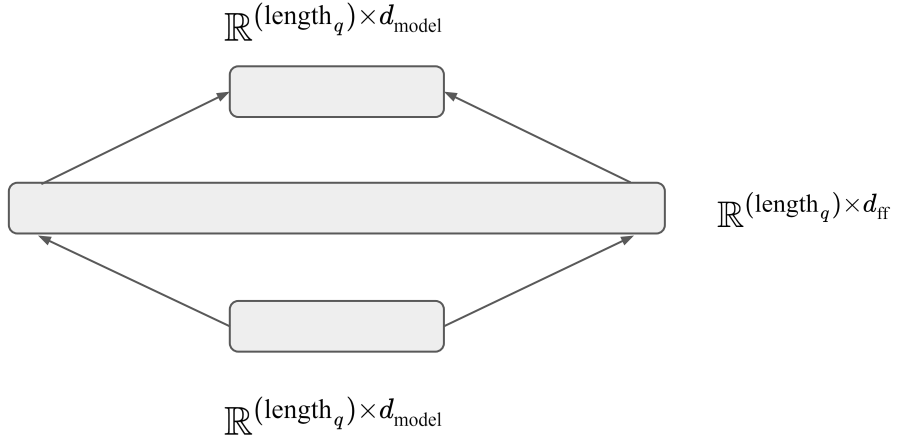
Figure 2. Residual learning: a building block.

Deep Residual Learning for Image Recognition, He et al, 2015

https://arxiv.org/pdf/1512.03385.pdf

Point wise feed forward network





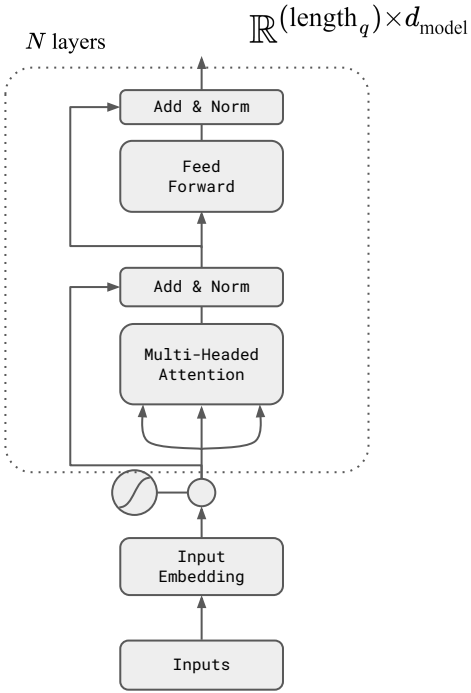
Point wise feed forward network bao gồm 2 lớp kết nối đầy đủ với activation là **relu** ở lớp giữa.

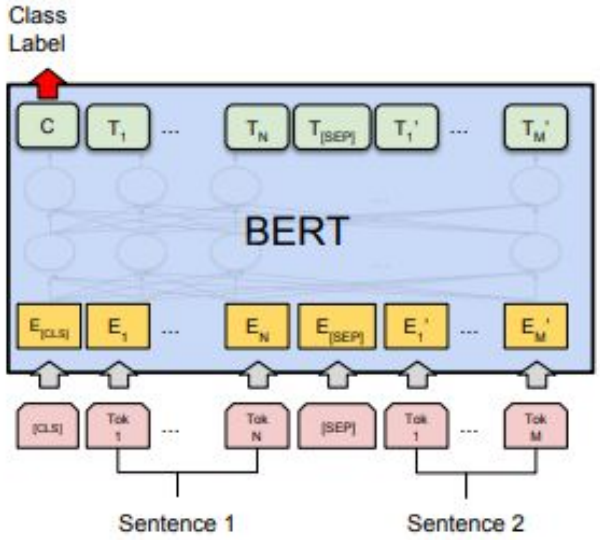
2 lớp Fully Connected này được miêu tả tương tự như **2 lớp Convolution 1D** hoạt động trên vị trí của từng từ

$$d_{ff} = 2048$$

$$d_{\text{model}} = 512$$

Encoder Applications





Bidirectional Encoder Representations from Transformers
Bert là một Language model pre-training

https://arxiv.org/pdf/1810.04805.pdf