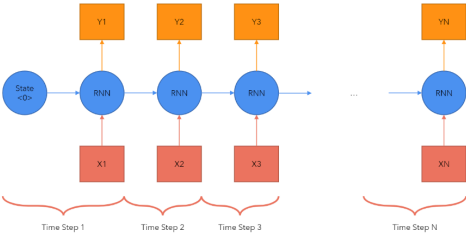


Personal Note - Week 6 - Transformer

1.Sơ lược

Before	Transformer
các model thường sử dụng RNN cho encoder-decoder, chỉ áp dụng attention cho phần cross attention.	Transformer loại bỏ hoàn toàn RNN trong việc mô hình hóa chuỗi và thay thế bằng self-attention.
thường bị hạn chế vì phải chờ đợi kết quả từ trạng thái trước đó trước khi tính toán trạng thái tiếp theo. Recurrent Neural Networks (RNN)	tốc độ xử lý nhanh hơn (bao gồm các phép nhân ma trận).



Attention is All You Need (Vaswani et al. 2017)

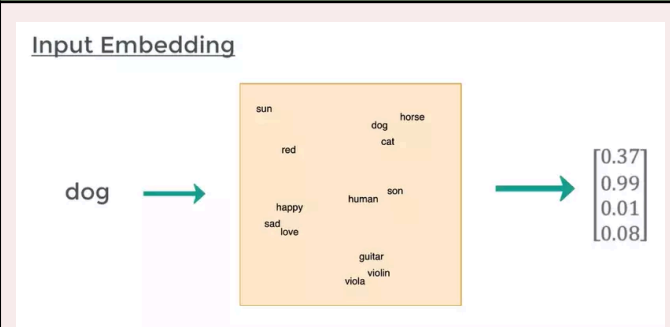
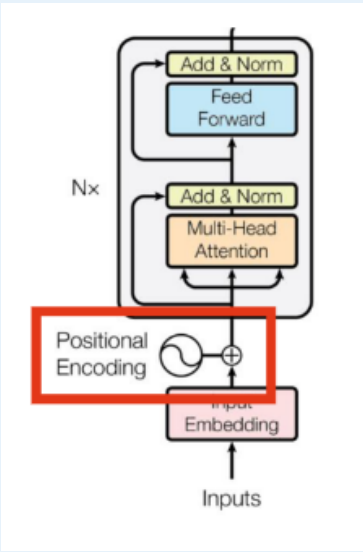
<https://arxiv.org/pdf/1706.03762v1>

2. Types of Transformer

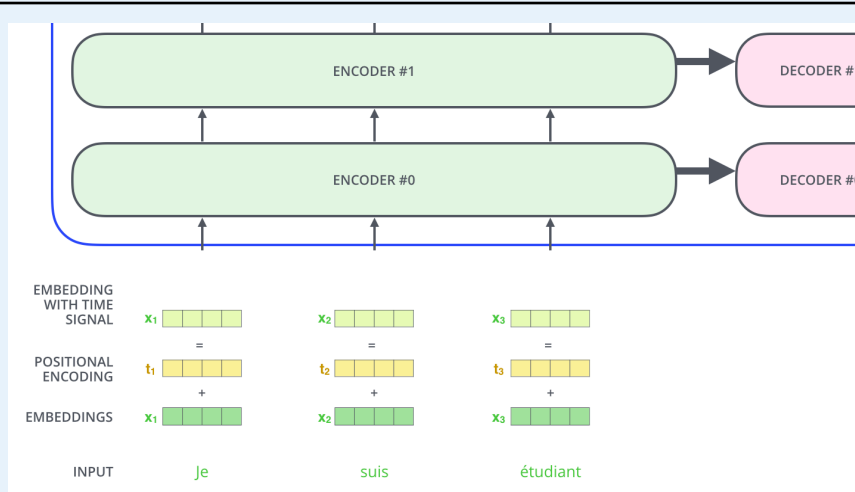
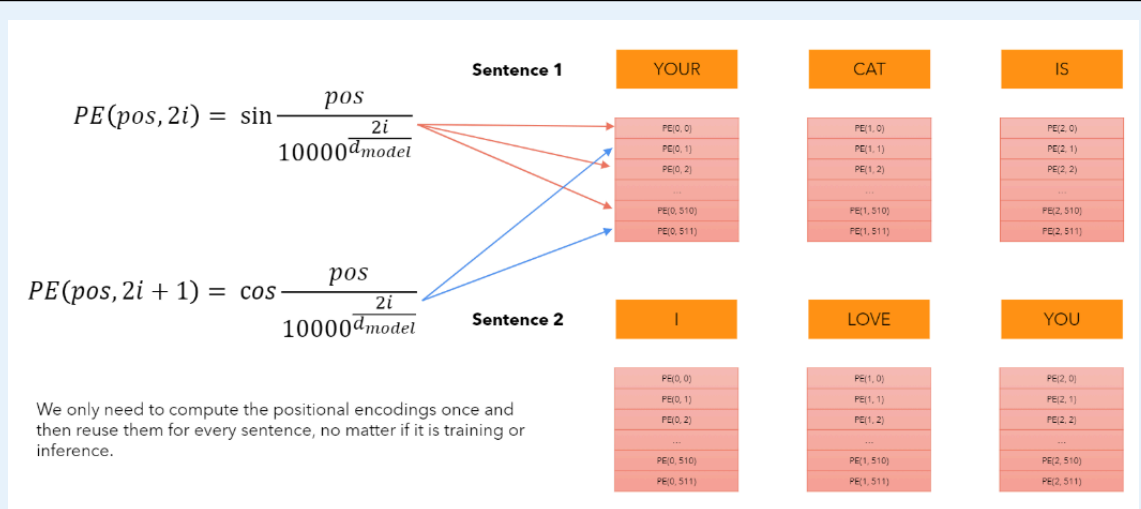
encoder-decoder	decoder-only
<p>T5, BART</p> <p>Encoder-Decoder Model (e.g. T5, MBART)</p>	<p>GPT, LLaMA</p> <p>Decoder Only Model (e.g. GPT, LLaMa)</p>
Thường được sử dụng cho các tasks có input-output rõ ràng:	ứng dụng: + chatbot

<ul style="list-style-type: none"> + tóm tắt văn bản (text summarization) + dịch máy (translation) 	=> việc xác định input-output phức tạp hơn.
	đễ dàng điều chỉnh kích thước của các lớp hoặc số lượng lớp mà không làm tăng quá nhiều số lượng tham số.

3. Core Transformer Concepts

Encoder																							
<div>Input embedding</div>	<p>biểu diễn text dưới dạng vector, (input embedding). => đảm bảo các từ gần nghĩa có vector gần giống nhau.</p>	<div>Input Embedding</div> <div></div> <div><table><tr><td>Original sentence (tokens)</td><td>YOUR</td><td>CAT</td><td>IS</td><td>A</td><td>LOVELY</td><td>CAT</td></tr><tr><td>Input IDs (position in the vocabulary)</td><td>105</td><td>6587</td><td>5475</td><td>3578</td><td>65</td><td>6587</td></tr><tr><td>Embedding (vector of size 512)</td><td>[952.207, 5450.840, 1853.448, ..., 2671.529]</td><td>[171.411, 3276.350, 9192.819, ..., 8390.473]</td><td>[621.659, 1304.051, 0.565, ..., 4596.025]</td><td>[776.562, 5567.288, 58.942, ..., 5119.949]</td><td>[6422.693, 6315.080, 9358.778, ..., 2141.081]</td><td>[171.411, 3276.350, 9192.819, ..., 3633.421]</td></tr></table><p>We define $d_{model} = 512$, which represents the size of the embedding vector of each word</p></div>	Original sentence (tokens)	YOUR	CAT	IS	A	LOVELY	CAT	Input IDs (position in the vocabulary)	105	6587	5475	3578	65	6587	Embedding (vector of size 512)	[952.207, 5450.840, 1853.448, ..., 2671.529]	[171.411, 3276.350, 9192.819, ..., 8390.473]	[621.659, 1304.051, 0.565, ..., 4596.025]	[776.562, 5567.288, 58.942, ..., 5119.949]	[6422.693, 6315.080, 9358.778, ..., 2141.081]	[171.411, 3276.350, 9192.819, ..., 3633.421]
Original sentence (tokens)	YOUR	CAT	IS	A	LOVELY	CAT																	
Input IDs (position in the vocabulary)	105	6587	5475	3578	65	6587																	
Embedding (vector of size 512)	[952.207, 5450.840, 1853.448, ..., 2671.529]	[171.411, 3276.350, 9192.819, ..., 8390.473]	[621.659, 1304.051, 0.565, ..., 4596.025]	[776.562, 5567.288, 58.942, ..., 5119.949]	[6422.693, 6315.080, 9358.778, ..., 2141.081]	[171.411, 3276.350, 9192.819, ..., 3633.421]																	
<div>Positional Encoding</div> <div></div>	<p>1. Word embeddings: biểu diễn ngữ nghĩa của một từ. 2. Tuy nhiên cùng một từ ở vị trí khác nhau của câu lại mang ý nghĩa khác nhau. 3. Mô hình Transformer hoàn toàn dựa vào cơ chế attention, nếu chỉ sử dụng embedding mà không có thông tin về vị trí, sẽ không thể phân biệt được giữa các từ giống nhau.</p> <p>=> Positional Encoding để inject thêm thông tin về vị trí của một từ. => Mục đích: giải quyết vấn đề phân biệt giữa các từ giống nhau trong ngữ cảnh khác nhau.</p>	<div>$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$</div> <div><ul style="list-style-type: none">- pos là vị trí của từ trong câu.- PE là giá trị phần tử thứ i trong embeddings có độ dài d_{model}.<p>=> cộng PE vector và Embedding vector:</p></div> <div><div><div>$\begin{bmatrix} 0.37 \\ 0.99 \\ 0.01 \\ 0.08 \end{bmatrix}$<p>Embedding of "Dog"</p></div><div><div>Positional Encoding</div><div>$\begin{bmatrix} 0.42 \\ 0.84 \\ 0.12 \\ 0.81 \end{bmatrix}$<p>Embedding of Dog (with context info)</p></div></div><div>$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$</div></div></div>																					

Original sentence	YOUR	CAT	IS	A	LOVELY	CAT
Embedding (vector of size 512)	932.207 5450.840 1853.448 ... 1.458 2671.529	171.411 3276.350 9192.819 ... 3633.421 8390.473	621.859 1304.051 0.565 ... 7679.805 4506.025	776.562 5567.289 55.942 ... 2716.194 5119.999	6422.893 6315.000 9350.770 ... 2141.081 735.147	171.411 3276.350 9192.819 ... 3633.421 8390.473
Position Embedding (vector of size 512). Only computed once and reused for every sentence during training and inference.	1664.068 8080.133 2620.399 ... 9356.405 3120.159	1281.453 7902.890 912.970 ... 3521.102 1659.217 7018.420
Encoder Input (vector of size 512)	1035.479 11256.483 11813.218 ... 13019.826 11510.632	1452.869 11175.24 10105.789 ... 5292.438 15409.093



Self-Attention

- "hiểu" được sự liên quan giữa các từ trong một câu.

- vector embedding từ:

- + xem xét ngữ cảnh gần để học các vector ngữ pháp.
- + hoặc xem ngữ cảnh xa để học các vector nghĩa.
- + => các phần khác nhau của ngữ cảnh có thể

hữu ích cho các mục đích khác nhau.

Self-Attention allows the model to relate words to each other.

In this simple case we consider the sequence length $\text{seq} = 6$ and $\mathbf{d}_{\text{model}} = \mathbf{d}_k = 512$.

The matrices \mathbf{Q} , \mathbf{K} and \mathbf{V} are just the input sentence.

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

$\text{softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{512}}\right) =$

	YOUR	CAT	IS	A	LOVELY	CAT	Σ
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	1
CAT	0.124	0.278	0.201	0.128	0.154	0.115	1
IS	0.147	0.132	0.262	0.097	0.218	0.145	1
A	0.210	0.128	0.206	0.212	0.119	0.125	1
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	1
CAT	0.195	0.114	0.203	0.103	0.157	0.229	1

* all values are random.

* for simplicity I considered only one head, which makes $\mathbf{d}_{\text{model}} = \mathbf{d}_k$.

(6, 6)

Compute Self-attention:

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

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

\times

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

$=$

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

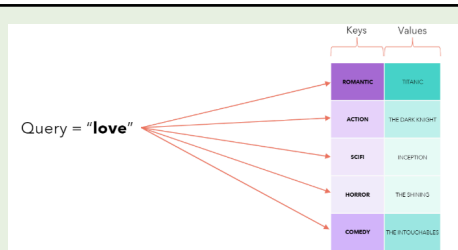
(6, 6)

Đầu vào của Multi-head Attention có 3 mũi tên, tương ứng 3 vectors:

- **Querys (Q)**
- **Keys (K)**
- **Values (V).**

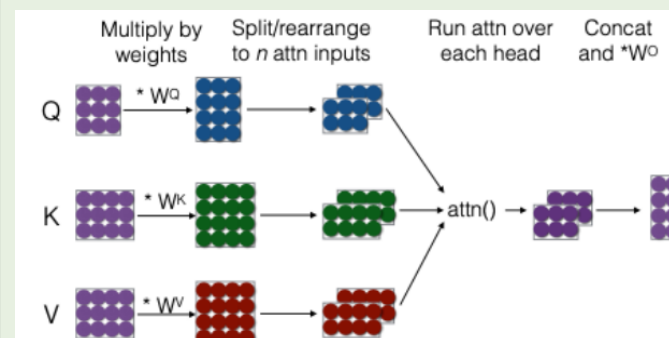
=> Tính vector **attention Z** cho một từ theo công thức:

$$\mathbf{Z} = \text{softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^T}{\sqrt{\text{Dimension of vector Q, K or V}}}\right) \cdot \mathbf{V}$$



Concept:

- Bước 1: nhân các vector đầu vào với các ma trận trọng số.
- Bước 2: tách và sắp xếp các vector thành nhiều đầu vào attention.
- Bước 3: Khi chạy attention trên mỗi đầu, tính toán vector attention bằng cách sử dụng các vector truy vấn và khóa, sau đó nhân các vector giá trị với vector attention để có được kết quả cuối cùng.
- Bước 4: thực hiện phép nhân ma trận cuối cùng.



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_n)$$

$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

code example:

- thực hiện các phép chiếu tuyến tính cho tất cả các đầu,
- định hình lại để có H đầu,
- áp dụng attention cho tất cả các đầu,
- nối lại trước khi áp dụng một lớp tuyến tính cuối cùng.

```
def forward(self, query, key, value, mask=None):
    nbatches = query.size(0)
    # 1) Do all the linear projections
    query = self.W_q(query)
    key = self.W_k(key)
    value = self.W_v(value)
    # 2) Reshape to get h heads
    query = query.view(nbatches, -1, self.heads, self.d_k).transpose(1, 2)
    key = key.view(nbatches, -1, self.heads, self.d_k).transpose(1, 2)
    value = value.view(nbatches, -1, self.heads, self.d_k).transpose(1, 2)
    # 3) Apply attention on all the projected vectors in batch.
    x, self.attn = attention(query, key, value)
    # 4) "Concat" using a view and apply a final linear.
    x = (
        x.transpose(1, 2)
        .contiguous()
        .view(nbatches, -1, self.h * self.d_k)
    )
```

```
)
return self.W_o(x)
```

Multi-headed attention

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

$$MultiHead(Q, K, V) = \text{Concat}(\text{head}_1 \dots \text{head}_h)W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

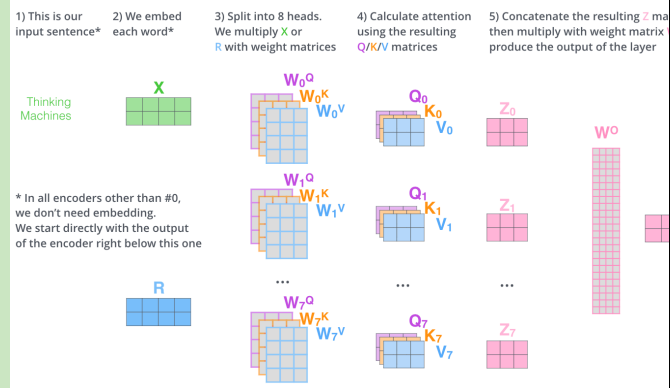
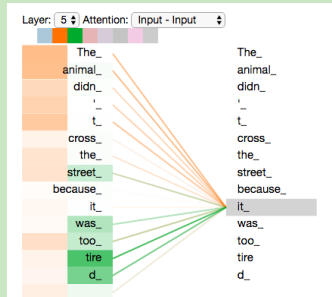
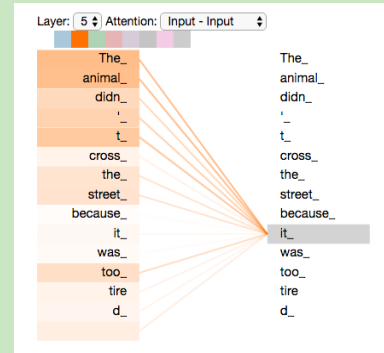
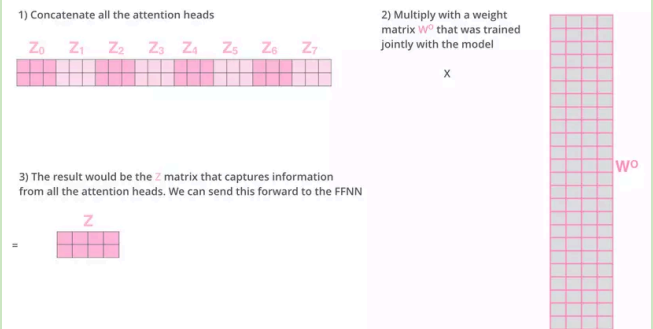
Vấn đề của Self-attention là attention của một từ sẽ luôn "chú ý" vào chính nó.

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

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

Để có sự tương tác giữa các từ KHÁC NHAU trong câu => Multi-head attention.

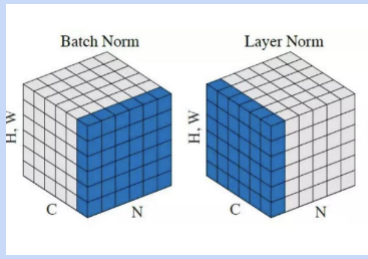
- Mỗi "head" sẽ cho ra một ma trận attention riêng => concat các ma trận này
- Sau đó nhân với ma trận trọng số W^O để ra một ma trận attention duy nhất (weighted sum).

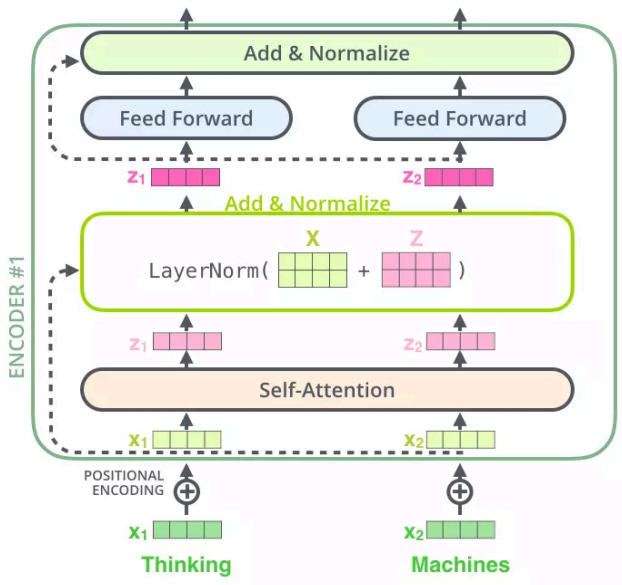
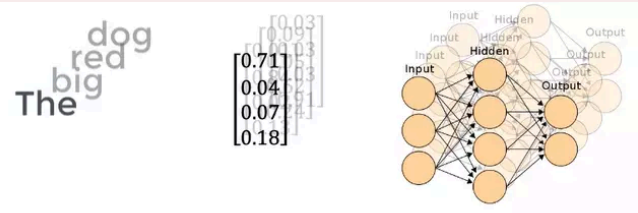


Layer Normalization

Layer Normalization (Ba et al. 2016)

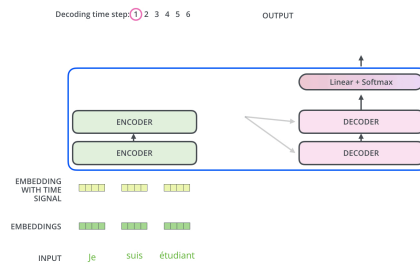
-Bước 1: tính toán trung bình của các vector bằng cách cộng tất cả các giá trị lại và chia

	<p>Khi chúng ta chạy các mô hình Transformer với số lượng lớp lớn, chẳng hạn như 8, 12 hoặc 16 lớp, một vấn đề thường gặp là hiện tượng triệt tiêu tiêu gradient. Để khắc phục điều này, Layer Normalization được sử dụng.</p> <p>-Layer nomination: chuẩn hóa các đầu ra để chúng nằm trong một khoảng giá trị nhất định, từ đó giảm thiểu sự biến thiên trong quy mô.</p> <p>Layer Normalization (Ba et al. 2016)</p>	<p>cho số lượng phần tử trong vector.</p> <p>-Bước 2: tính độ lệch chuẩn của vector bằng cách lấy tổng bình phương của các giá trị trừ đi trung bình.</p> <p>-Bước 3: lấy căn bậc 2 của tổng đó.</p> <p>Kết quả là các giá trị trong vector sẽ được chuẩn hóa để có trung bình bằng 0 và độ lệch chuẩn bằng 1.</p> $\mu(x) = \frac{1}{n} \sum_{i=1}^n x_i$ $\sigma(x) = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2}$ $LayerNorm(x; g, b) = \frac{g}{\sigma(x)} \odot (x - \mu(x)) + b$ <p>with g: gain, b: bias</p> <p>=></p> <p>Sau khi chuẩn hóa: thêm một độ lệch (bias) và nhân với một hệ số khuếch đại (gain). (sau khi chuẩn hóa, các giá trị sẽ được dịch chuyển ra khỏi khoảng giá trị chuẩn)</p> <p>Ví dụ, nếu chúng ta có một vector mới X1 và một vector khác X2, Layer Normalization sẽ chuẩn hóa X2 và sau đó dịch chuyển nó lên một vị trí mới trong không gian. Nhờ vậy, tất cả các vector sẽ nằm trong một phần không gian nhất quán, với vị trí và độ phân tán được xác định bởi bias và gain.</p> <p>=> mang lại lợi ích cho sự ổn định trong quá trình huấn luyện, vì mỗi lần chúng ta tiêu thụ đầu ra từ một lớp đã được chuẩn hóa, chúng ta sẽ nhận được một kết quả có thể dự đoán được.</p>
<p>So sánh Layer normalization và Batch normalization</p> 	<p>Batch Normalization chuẩn hóa trên toàn bộ batch, tức là tất cả các phần tử trong batch. (thay đổi các thống kê dựa trên các phần tử khác trong batch)</p>	<p>Layer Normalization chỉ phụ thuộc vào từng trường hợp cụ thể, do đó không cần lo lắng về các batch khác nhau. Mỗi đầu vào và đầu ra đều nhất quán, bất kể các phần tử khác trong batch.</p>

<p>Residual Connections (kết nối phần dư)</p>	<p>$Residual(x, f) = f(x) + x$</p> <ul style="list-style-type: none"> - mỗi sub-layer đều là một residual block. - skip connections trong Transformers cho phép thông tin đi qua sub-layer trực tiếp. - Thông tin này (x) được cộng với attention (z) của nó và thực hiện Layer Normalization. 	<p>Khi có kết nối phần dư, mô hình sẽ giảm thiểu việc chú ý đến chính nó, mà thay vào đó sẽ tập trung vào các thông tin xung quanh để hiểu rõ hơn về những gì cần thêm vào để tạo ngữ cảnh.</p> 
<p>Feed-forward layer</p>	<ul style="list-style-type: none"> - trích xuất các đặc trưng kết hợp từ đầu ra đã được chú ý. <p>Sau khi được Normalize, các vectors z được đưa qua mạng fully connected trước khi đẩy qua Decoder.</p> <p>Các vectors này không phụ thuộc vào nhau nên ta có thể tận dụng được tính toán song song cho cả câu.</p>	<p>$FFN(x; W_1, b_1, W_2, b_2) = f(xW_1 + b_1)W_2 + b_2$</p> 
<p>Decoder</p>		
<p>Masked Multi-head Attention</p>	<p>Khi Decoder dịch đến từ thứ i, phần sau của câu sẽ bị che lại (masked) và Decoder chỉ được phép "nhìn" thấy phần nó đã dịch trước đó.</p>	<p>Self Attention</p> <p>Le → Le gros chien rouge gros → Le gros chien rouge chien → Le gros chien rouge rouge → Le gros chien rouge</p>

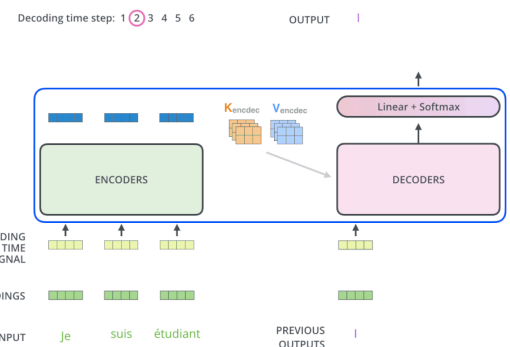
SS Decoding & Encoding

Encoding



Decoding

- Decoder decode từng từ một và input của Decoder bị masked.
- Sau khi masked input đưa qua sub-layer #1 của Decoder, nó sẽ không nhân với 3 ma trận trọng số để tạo ra Q, K, V nữa mà chỉ nhân với 1 ma trận trọng số WQ.
- K và V được lấy từ Encoder cùng với Q từ Masked multi-head attention đưa vào sub-layer #2 và #3 tương tự như Encoder.
- Cuối cùng, các vector được đẩy vào lớp Linear (là 1 mạng Fully Connected) theo sau bởi Softmax để cho ra xác suất của từ tiếp theo.



references:

1. <https://drive.google.com/file/d/1y8YxaJwjnhdpYeLWOTq27PIEInoOLHj/view>
2. https://github.com/hkproj/transformer-from-scratch-notes/blob/main/Diagrams_V2.pdf
3. https://www.youtube.com/watch?v=GTda3VKWUe8&list=PLMm4sOMuA2QI5x_0KINT3LuKDio-ByboB&index=34
4. <https://www.datacamp.com/tutorial/how-transformers-work>
5. <https://200lab.io/blog/transformer-cong-nghe-dang-sau-chatgpt-va-bard/>
6. https://d2l.ai/vn.com/chapter_attention-mechanisms/index_vn.html?ref=200lab.io

Project: Implement an attention mechanism for text summarization and report the results.

Reference: [VNDS Dataset | Papers With Code](#)

Dataset: [vietnews/data at master · ThanhChinhBK/vietnews \(github.com\)](#)

(2) ViT5: Pretrained Text-to-Text Transformer for Vietnamese Language Generation | [hieu tran - Academia.edu](#)