**Phần 3 - Về transformer và ViT**

**Notes:**

* *Về format cấu trúc soạn như sau:*
  + *Lý thuyết…*
  + *Bộ code mẫu/ ví dụ …*
  + *Ứng dụng (nếu có)...*
* *Mems làm nhớ note tên để mn dễ contact*

[**I. Nội dung 2**](#_eangervw4etq)

[**II. Phần soạn chi tiết nội dung 2**](#_vhgil1udi0w)

[1: Cách chia ảnh thành ô lưới nxn pixel - Hoàng Ngọc Tiến. 2](#_yixc8zxot818)

[2: Positional embedding là gì - Hoàng Ngọc Tiến. 3](#_ok0evnze305n)

[3: Attention là gì? [Lê Dương Khoa] 4](#_vse6n2gqr4t8)

[4: Nguyên tắc tính toán của self-attention? [Lê Dương Khoa] 4](#_jr6b3duee8nm)

[5: Nguyên tắc tính toán của transformer? [Nguyễn Đỗ Quốc Duy] 9](#_964cksoanuw3)

[5.1. Khối Input Embedding: 9](#_mklwdnmwrmto)

[5.2 Khối Positional Encoding: 9](#_q5uexaae90xc)

[5.3. Khối Encoder: 9](#_vdouozdfpntn)

[5.3.1 Khối Multi-head Attention: 9](#_tndjzmdp87fm)

[5.3.2 Khối Attention head: 10](#_5dxzjc5nlssa)

[5.3.3. Khối add & Norm: 10](#_ortcqcur9h2l)

[5.3.4. Khối FeedForward 11](#_hzd54ztxwoav)

[5.3.5. Khối Output Embedding: 11](#_ocdp2gjuy2o0)

[5.4. Khối Decoder: 11](#_g1mwedfbbh66)

[Khối Masked Multi-Head Attention: 11](#_rtogjsclqnus)

[Khối Encoder-Decoder Multi-Head Attention: 12](#_hc21ha4n9r4p)

### 

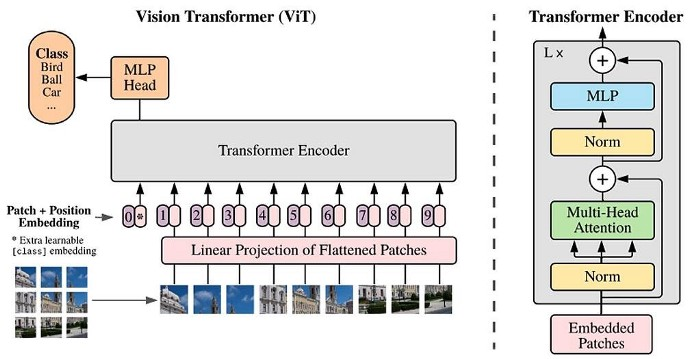
### I. Nội dung

Về transformer và ViT

* + (1) Cách chia ảnh thành ô lưới nxn pixels?
  + (2) Positional Embedding là gì?
  + (3) Attention là gì và bằng cách nào?
  + (4) Nguyên tắc tính toán của từng khối trong transformer
  + (5) Số tham số cần học?

### II. Phần soạn chi tiết nội dung

#### **1: Cách chia ảnh thành ô lưới nxn pixel - Hoàng Ngọc Tiến.**



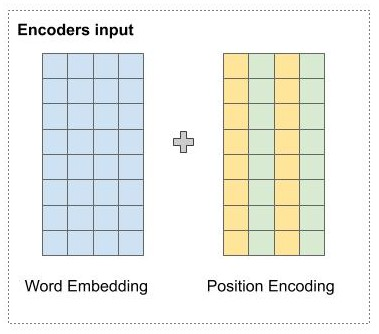
Mỗi bức hình sẽ được chia ra làm nxn phần mỗi phần như vậy gọi là một **patch**. Mỗi mô hình ViT sẽ có một thông số gọi là patch size tức là size của mỗi patch. Ví dụ hình ảnh đầu vào có kích thước 224x224, patch\_size là 32x32 => mỗi hình sẽ được chia ra làm (224/32)\*(224/32) = 49 patch hay token.

Mỗi patch sẽ tương ứng với một token tương tự như một token của một từ trong mô hình NLP.

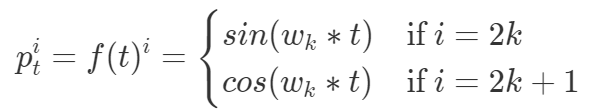
Các patch sẽ được đưa một tầng projection theo thứ tự từ trái sang phải từ trên xuống dưới, sau đó cộng với position embedding rồi đưa vào transformer encoder.

#### **2: Positional embedding là gì - Hoàng Ngọc Tiến.**

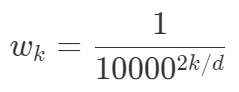
Vị trí của các từ được mã hóa bằng một vector có kích thước bằng word embedding và được cộng trực tiếp vào word embedding.



Cụ thể, tại vị trí chẵn, tác giả sử dụng hàm sin, và với vị trí lẻ tác giả sử dụng hàm cos để tính giá trị tại chiều đó.

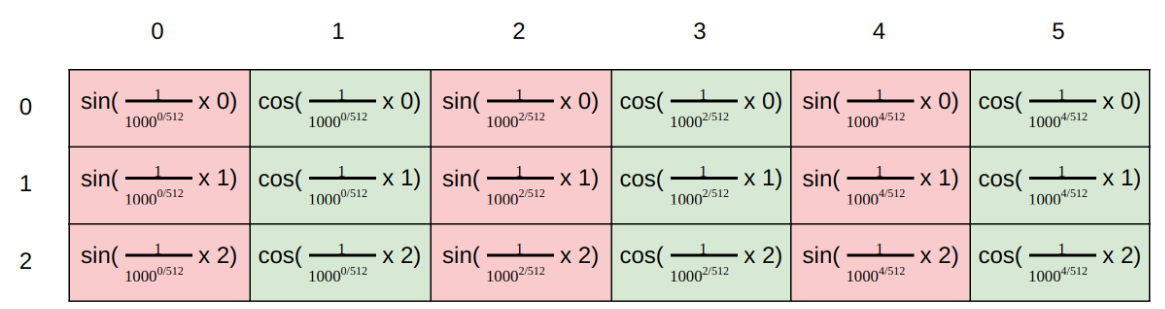


Trong đó:



* i là giá trị của chiều thứ i trong vector embedding.
* t là token thứ t trong đầu vào

Trong hình dưới này, mình minh họa cho cách tính position encoding của tác giả. Giả sử chúng ta có word embedding có 6 chiều, thì position encoding cũng có tương ứng là 6 chiều. Mỗi dòng tương ứng với một từ. Giá trị của các vector tại mỗi vị trí được tính toán theo công thức ở hình dưới.



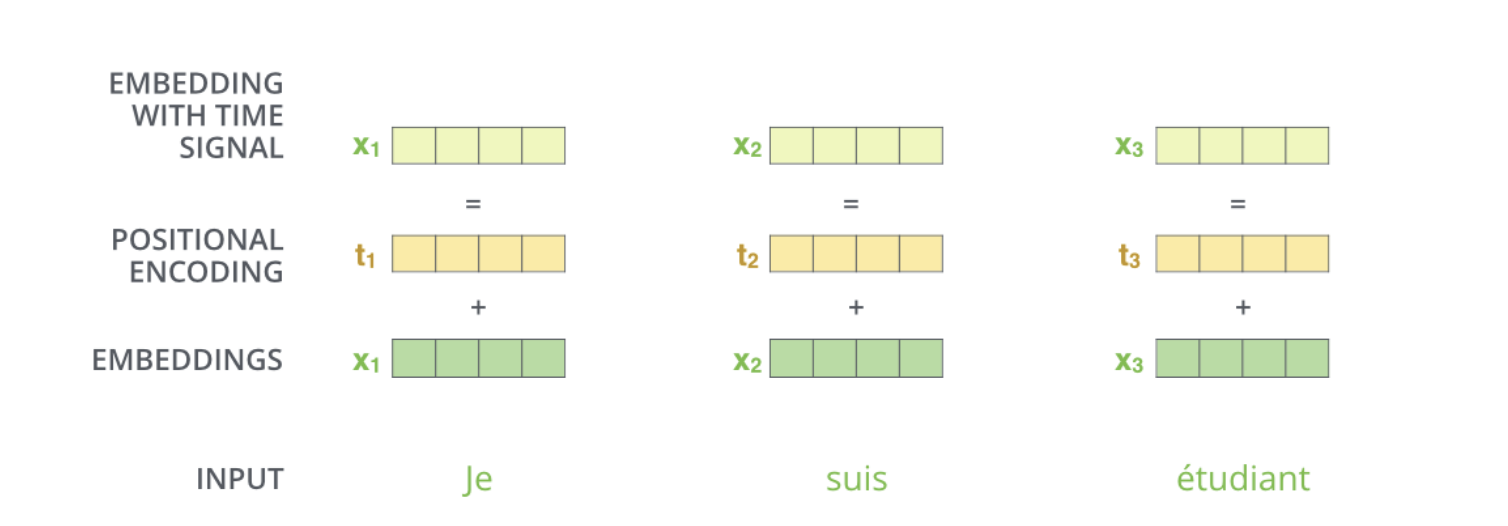
#### 

#### **3: Attention là gì? [Lê Dương Khoa]**

So với RNN, việc tính toán tuần tự từng bước sẽ không hiệu quả bằng việc tính toán song song tất cả các phần khác nhau của đầu vào cùng một lúc. Self-attention và attention là những cơ chế cho phép các mô hình Transformer tính toán nhiều phần khác nhau của đầu vào và đầu ra cùng một lúc.

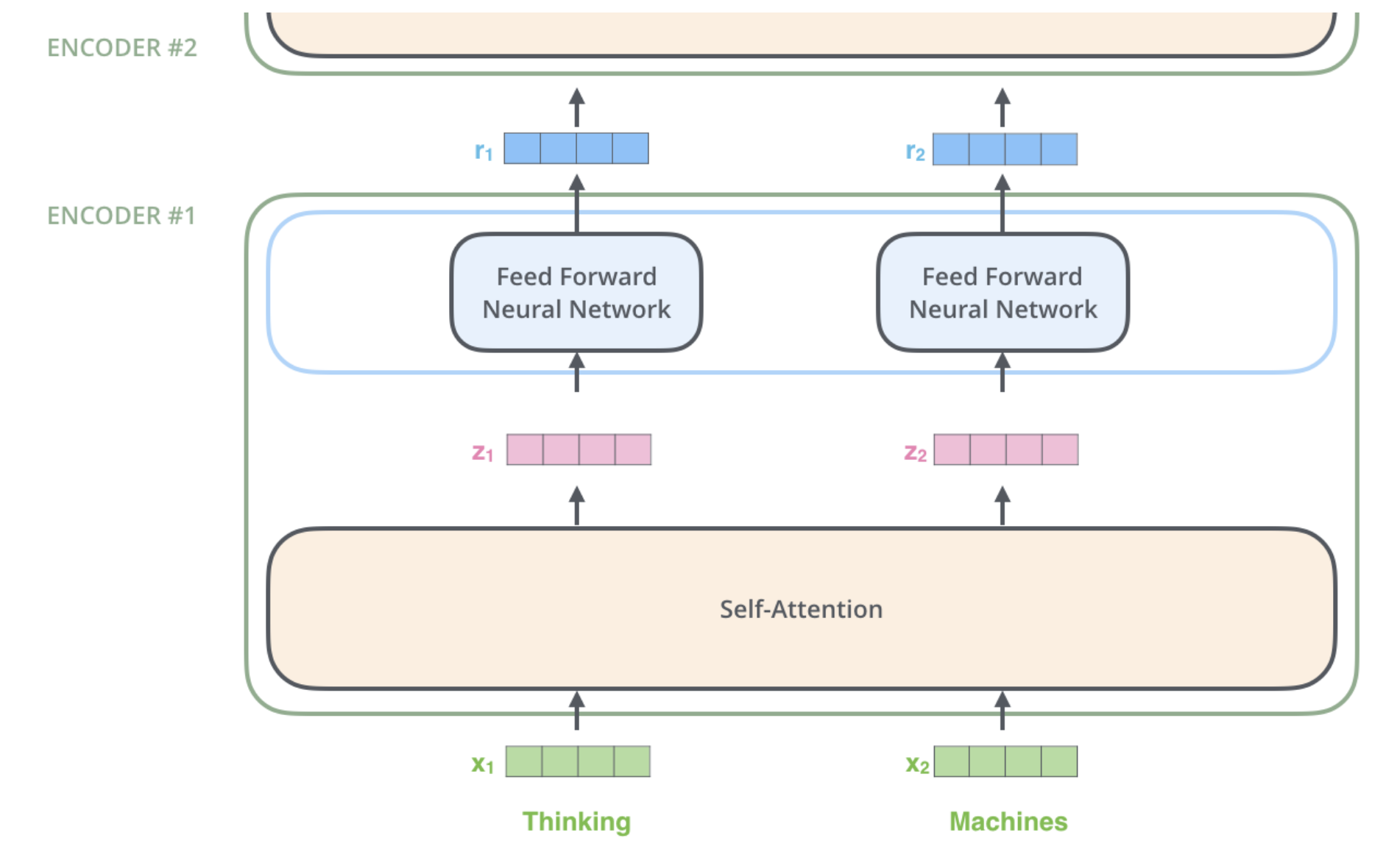
#### **4: Nguyên tắc tính toán của self-attention? [Lê Dương Khoa]**

Sau khi thực hiện xong 2 bước Input Embedding (Word Embedding) và Positional Encoding. Mô hình sẽ chuyển qua bước self-attention. Giả sử với đầu vào input (đầu vào) như hình dưới. Mô hình sẽ cộng Input Embedding và Positional Encoding, kết quả sẽ là input cho bước tiếp theo, self-attention.



Hình dưới là kiến trúc model Transformer.

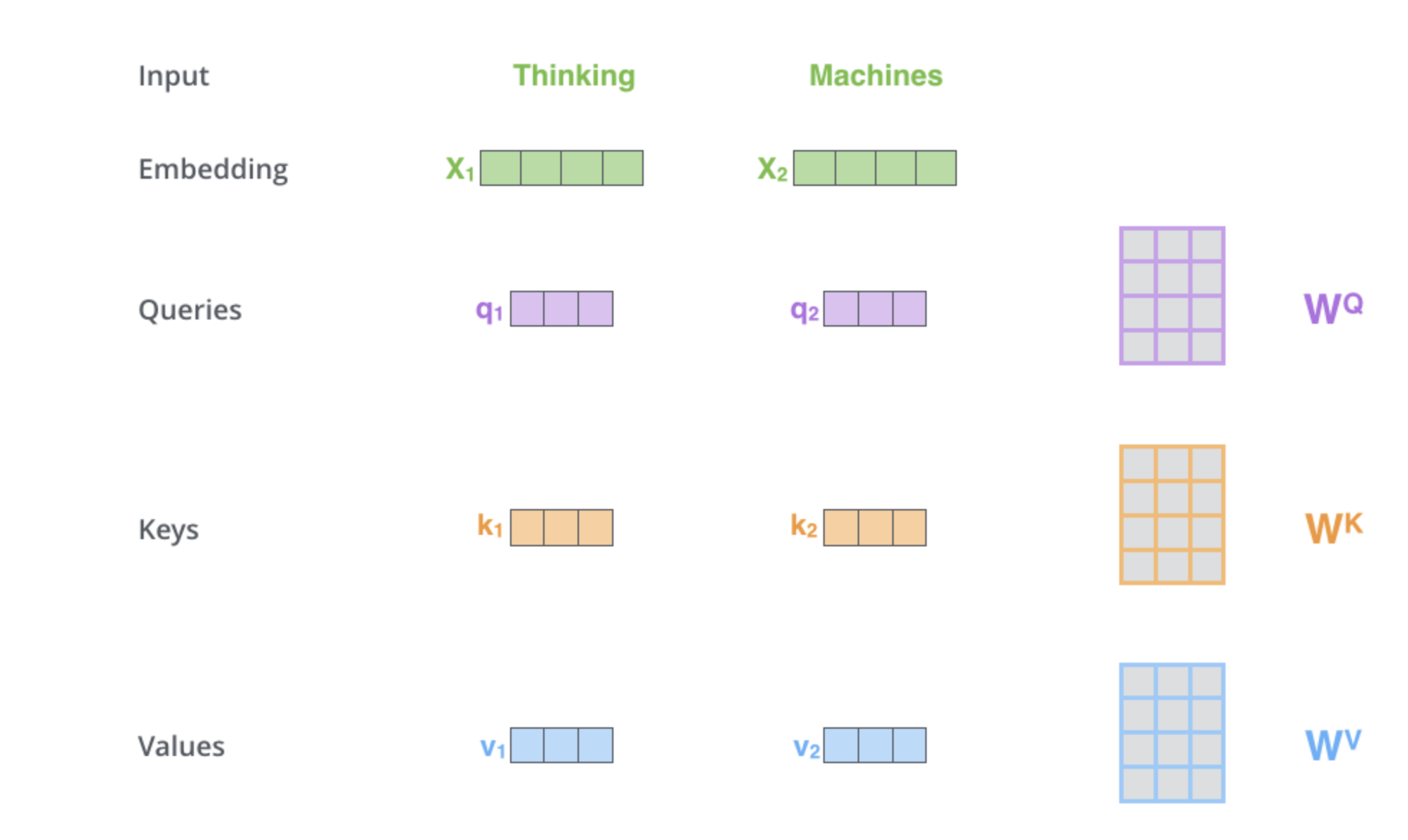




Chi tiết về self-attention:

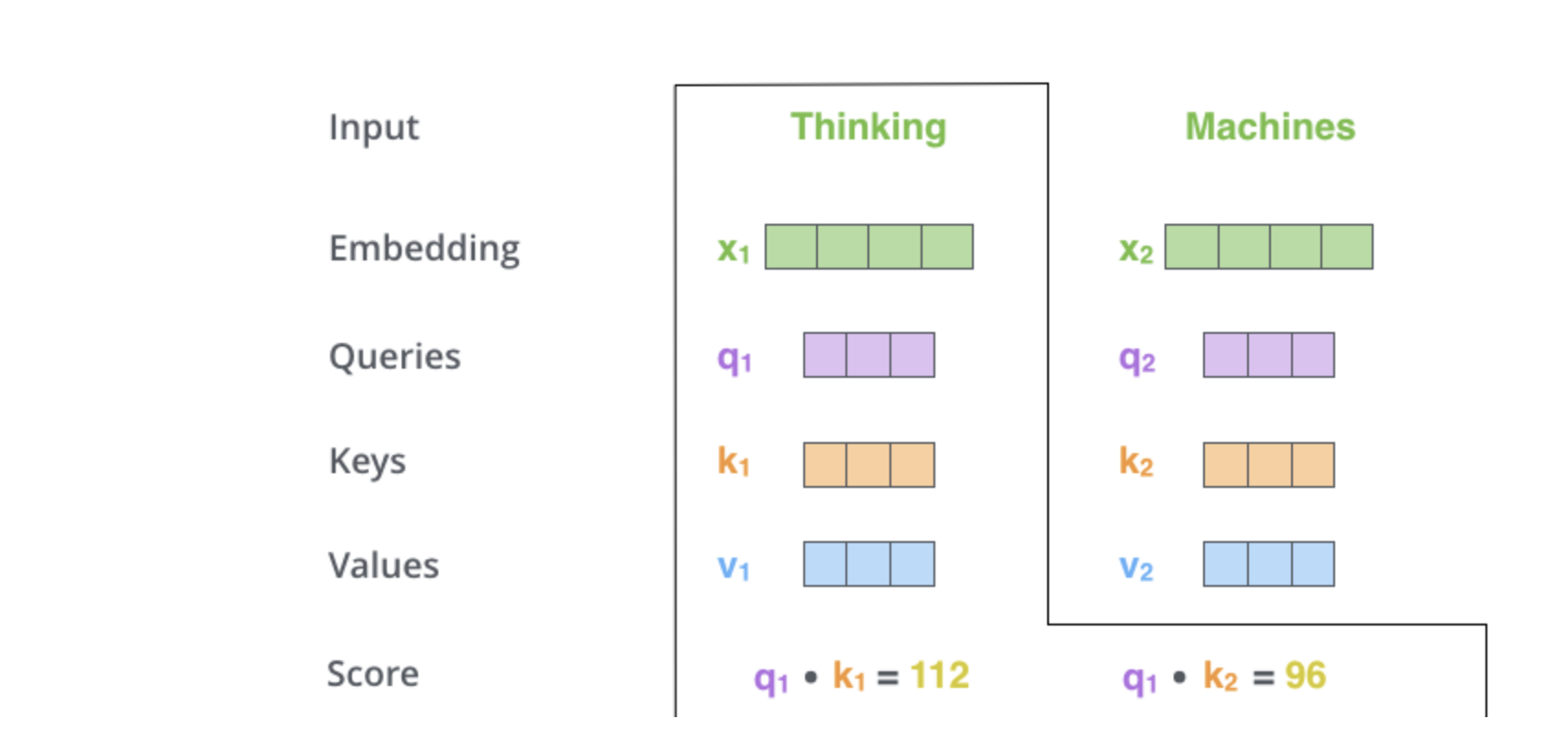
Với mỗi từ được input vào self-attention, ta sẽ tạo ra 3 vector Query vector, Key vector và Value vector. Những vector này được tạo ra bằng cách nhân đầu vào (Input) X1, X2 với 3 ma trận tương ứng WQ, WK, WV.

Chẳng hạn, q1 = X1 x WQ, k1 = X1 x WK. Các ma trận WQ, WK, WV sẽ được cập nhật trong quá trình training nhằm tối thiểu hàm mất mát (loss function).



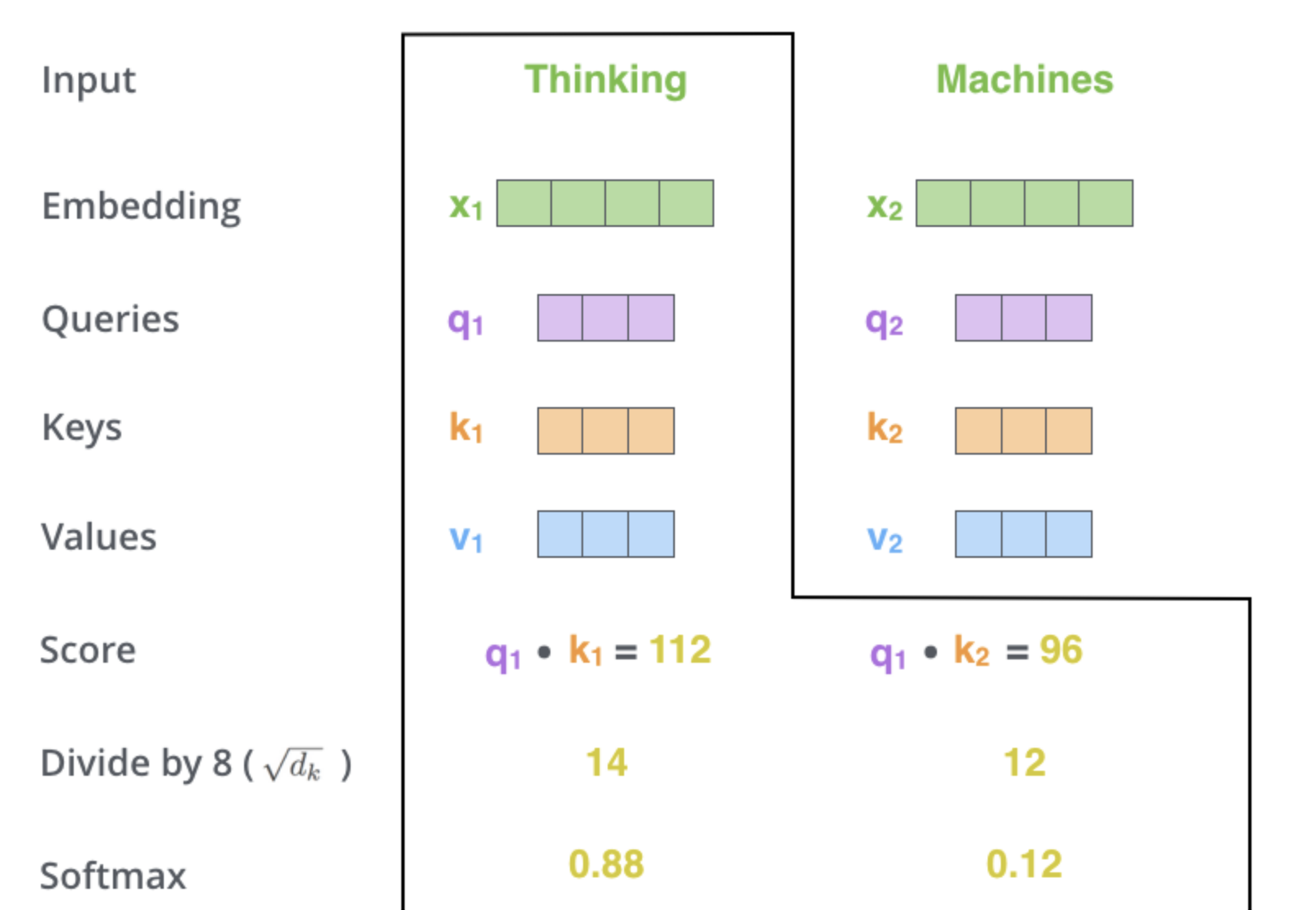
Bước tiếp theo trong việc tính toán self-attention là **tính điểm**. Giả sử chúng ta đang tính self-attention cho từ đầu tiên trong ví dụ này, “**Thinking**”. Chúng ta cần chấm điểm từng từ của câu đầu vào so với từ này. Điểm số xác định mức độ tập trung vào các phần khác của câu đầu vào khi chúng ta mã hóa một từ ở một vị trí nhất định.

Điểm sẽ được tính bằng cách nhân tích vô hướng của các **query vector** và **key vector** của từ mà chúng ta đang xét tới.

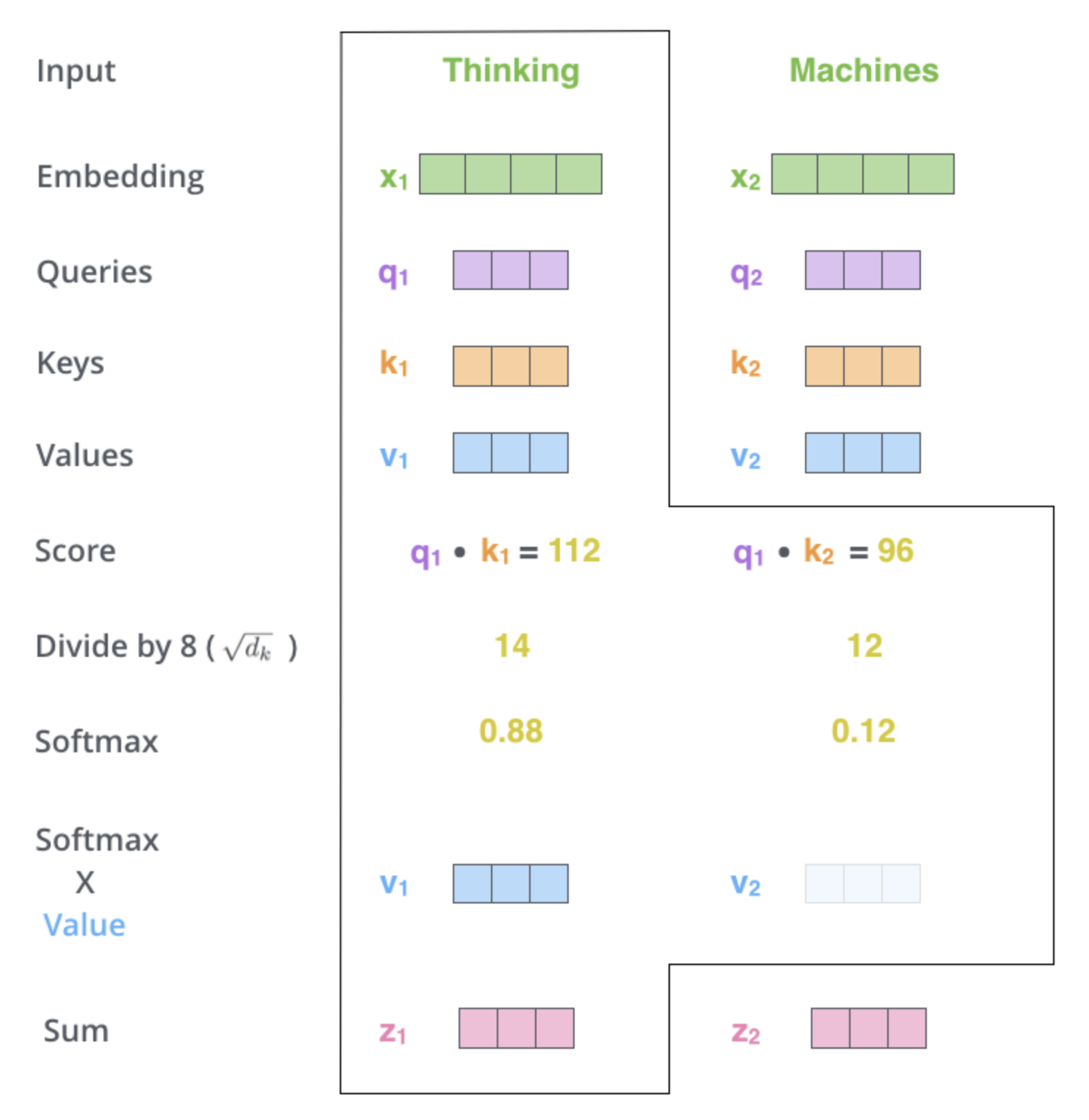


Sau khi có kết quả **q1.k1và q1.k2**, chúng ta sẽ chia kết quả cho 8 (mn có thể đọc paper để hiểu rõ hơn tại sao chia 8) . Sau đó kết quả sẽ được đưa qua hàm Softmax.

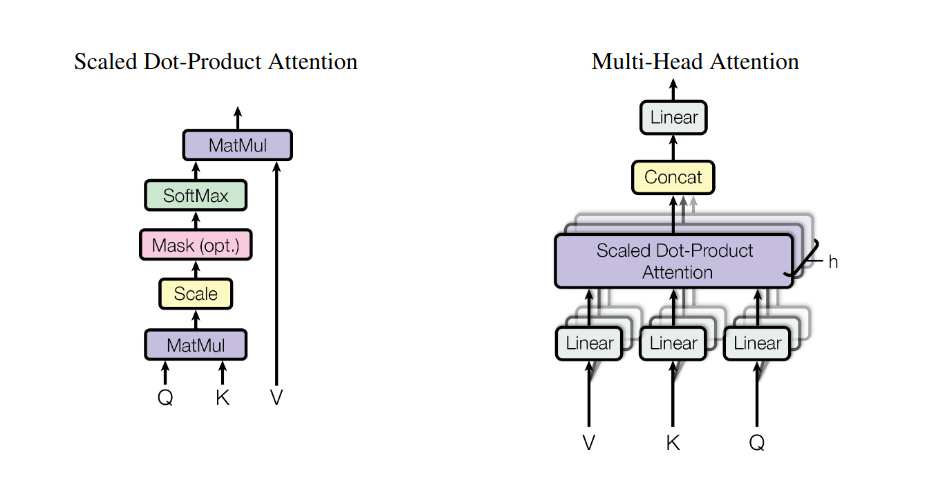
Điểm softmax này xác định mức độ mỗi từ sẽ được thể hiện ở vị trí này. Rõ ràng từ ở vị trí này sẽ có điểm softmax cao nhất, nhưng đôi khi việc chú ý đến một từ khác có liên quan đến từ hiện tại sẽ hữu ích hơn.



Bước tiếp là nhân từng value vector với điểm softmax (để chuẩn bị tính tổng). Ở đây ta muốn giữ nguyên giá trị của các từ mà chúng ta muốn tập trung vào và loại bỏ các từ không liên quan (ví dụ: bằng cách nhân chúng với các số nhỏ như 0,001).



#### **5: Nguyên tắc tính toán của transformer? [Nguyễn Đỗ Quốc Duy]**



##### **5.1. Khối Input Embedding:**

* Input: Một câu có N từ.
* Output: ma trận kích thước N x d
  + N là số từ trong câu
  + d là số chiều
* Cách hoạt động:
  + mỗi từ trong câu sẽ được convert thành một one-hot vector (vector có số chiều là kích thước của từ điển và có 1 giá trị 1, các giá trị còn lại bằng 0).
  + Ma trận one-hot vector được nhân với ma trận embedding để convert mỗi từ thành biểu diễn vector của nó.
  + Có thể hiểu khối Input Embedding theo cách khác là một dictionary map giữa từ và vector.

##### **5.2 Khối Positional Encoding:**

* Output: ma trận kích thước N x d

##### **5.3. Khối Encoder:**

* Input: ma trận kích thước N x d là kết quả phép cộng output hai khối Positional Encoding và Input Embedding.
* Output: ma trận kích thước N x d
* Cách hoạt động: mỗi khối encoder gồm 4 khối bên trong: Multi-head Attention, Add & Norm, Feed Forward, Add & Norm.

###### **5.3.1 Khối Multi-head Attention:**

* Input: ma trận kích thước N x d là kết quả phép cộng output hai khối Positional Encoding và Input Embedding.
* Output: ma trận kích thước N x d
* Cách hoạt động: Gồm h khối Attention head, kết quả của h khối (ma trận kích thước N x dv) được concat lại với nhau để cho ra kết quả có kích thước N x d.

###### **5.3.2 Khối Attention head:**

* Input: ma trận kích thước N x d là kết quả phép cộng output hai khối Positional Encoding và Input Embedding.
* Output: ma trận kích thước N x dv
* Cách hoạt động:
  + ma trận Input lần lượt được nhân với ba ma trận **Wk** (d x dk), **Wq** (d x dk), **Wv** (d x dv) để tạo thành ba ma trận **key** (N x dk), **query** (N x dk), **value** (N x dv). Trong đó **dk = dv = d/h**
  + Ba ma trận key, query, value đưa qua **cơ chế attention** tạo thành ma trận kích thước N x dv
  + Ma trận này đi qua một lớp Fully connected trước khi tạo thành output

###### **5.3.3. Khối add & Norm:**

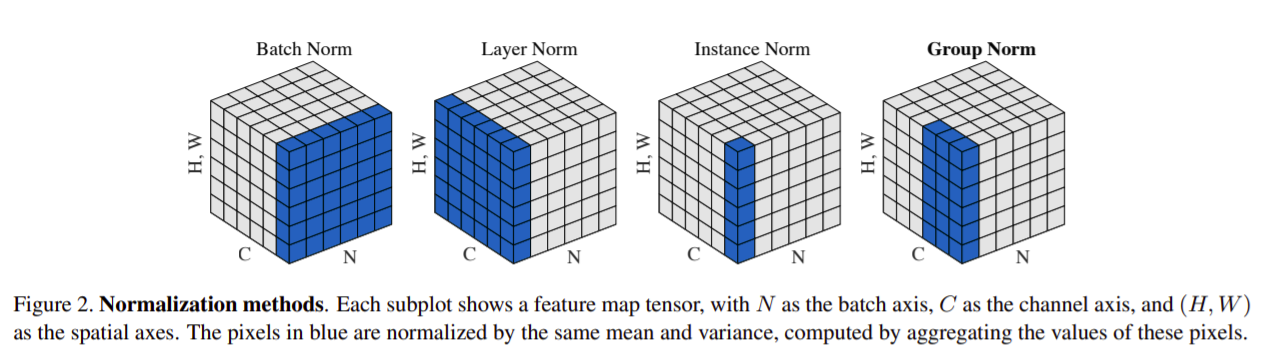
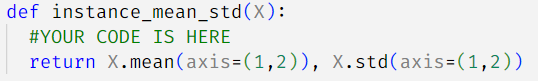
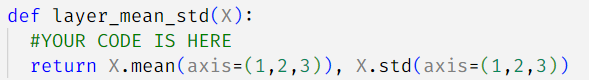
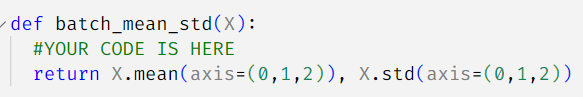
Gồm hai khối nhỏ là Add và Norm

**Khối Add:**

* + Input: hai ma trận cùng kích thước
  + Output: ma trận cùng kích thước với input và có giá trị từng phần tử bằng tổng các phần tử cùng vị trí trong input.

**Khối Norm:**

* + Input: ma trận
  + Output: ma trận cùng kích thước với input
  + Cách hoạt động: normalize dựa theo công thức bên dưới, trong đó x là vector (hoặc ma trận), trong hình thì x có kích thước C x (H, W) (bỏ chiều N = chiều batch ra). gamma và beta là hai tham số học được.

###### **5.3.4. Khối FeedForward**

* Input: ma trận kích thước N x d từ khối Add & Norm đầu tiên của khối encoder
* Output: ma trận kích thước N x d
* Cách hoạt động: là hai lớp Fully connected thông thường
  + Lớp thứ nhất: có relu activation, số output node = dff = 2048
  + Lớp thứ hai: không có activation, số output node = d

###### **5.3.5. Khối Output Embedding:**

* Tương tự như khối Input Embedding, tuy nhiên dùng để embed các từ của output

##### **5.4. Khối Decoder:**

* Input: Ma trận có kích thước N x d từ khối Output Embedding, Ma trận kích thước N x d từ khối Encoder cuối cùng.
* Output: ma trận kích thước N x d
* Cách hoạt động: mỗi khối decoder gồm 6 khối chồng lại với nhau: Masked Multi-Head Attention, Add & Norm, Encoder - Decoder Multi-Head Attention, Add & Norm, Feed Forward, Add & Norm

###### **Khối Masked Multi-Head Attention:**

* Input: ma trận kích thước N x d là kết quả phép cộng output hai khối Positional Encoding và Output Embedding.
* Output: ma trận kích thước N x d
* Cách hoạt động: Gồm h khối Attention head, kết quả của h khối được concat lại với nhau để cho ra kết quả tương tự như khối Multi-Head Attention thông thường. Tuy nhiên trong quá trình attention, ma trận Q\*K^T được mask các giá trị ở trên đường chéo chính để các từ không phải chú ý đến các từ trong tương lai.
  + Ví dụ:
    - Giả sử ta cần dịch một câu “Tôi là con mèo” sang câu tiếng anh “I am a cat”. Dãy output hiện tại được đưa vào khối Decoder là “<Start> I am a”. Output dự định là từ “cat”.
    - Lúc này từ <Start> chỉ được attention với chính nó, từ “I” chỉ được attention với chính nó và ký tự <Start>, từ “am” chỉ được attention với chính nó, từ “I” và từ “<Start>”. Do đó ma trận Q \* K^T biểu diễn mức độ tương tự của các từ này phải được “mask” ở các giá trị trên đường chéo chính:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **<Start>** | **I** | **am** | **a** |
| <Start> | 1.1 | -inf | -inf | -inf |
| I | 0.4 | 1.2 | -inf | -inf |
| am | 0.3 | 0.2 | 0.9 | -inf |
| a | 0.2 | 0.6 | 0.7 | 1.3 |

* + - (Các giá trị được mask là -inf để khi qua hàm softmax trở thành giá trị 0)

###### **Khối Encoder-Decoder Multi-Head Attention:**

* Input: Ma trận có kích thước N x d từ khối Add & Norm đầu tiên trong Decoder, Ma trận kích thước N x d **từ khối Encoder cuối cùng.**
* Output: Ma trận có kích thước N x d
* Cách hoạt động: tương tự khối Multi-Head Attention thông thường, chỉ khác Input. Ở đây **key** và **value** được lấy từ ma trận N x d của khối Encoder cuối cùng, **query** được lấy từ kết quả khối Add & Norm.

