TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO HỌC KỲ I NĂM HỌC 2022 – 2023

MÔN: **HỌC MÁY**

**DỰ ÁN CUỐI KÌ**

*Người hướng dẫn*: **PGS. LÊ ANH CƯỜNG**

*Người thực hiện*: **TRẦN TRUNG DŨNG – 520K0339**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO HỌC KỲ I NĂM HỌC 2022 – 2023

MÔN: **HỌC MÁY**

**DỰ ÁN CUỐI KÌ**

*Người hướng dẫn*: **PGS. LÊ ANH CƯỜNG**

*Người thực hiện*: **TRẦN TRUNG DŨNG – 520K0339**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Đầu tiên, em xin gửi lời cảm ơn chân thành đến Thầy Lê Anh Cường. Thầy là người hỗ trợ và hướng dẫn tận tình cho chúng em trong quá trình nghiên cứu, tìm hiểu để hoàn thành bài báo cáo môn Học Máy

Tiếp theo, chúng em xin gửi lời cảm ơn đến khoa Công Nghệ Thông Tin trường Đại học Tôn Đức Thắng vì đã tạo điều kiện cho chúng em được học tập và nghiên cứu môn học này. Khoa đã luôn sẵn sàng chia sẻ các kiến thức bổ ích cũng như kinh nghiệm tham khảo tài liệu, giúp ích không chỉ cho việc thực hiện và hoàn thành đề tài nghiên cứu mà còn giúp ích cho việc học tập và rèn luyện trong quá trình thực hành tại trường Đại học Tôn Đức Thắng nói chung.

Cuối cùng, trong quá tình hoàn tất đề tài của mình nhờ vào sự hướng dẫn, giúp đỡ và những kiến thức học hỏi được từ quý thầy cô giảng viên. Do giới hạn về mặt kiến thức và khả năng lý luận nên chúng em còn nhiều thiếu sót, kính mong sự chỉ dẫn và đóng góp của quý thầy cô giảng viên để bài nghiên cứu của chúng em được hoàn thiện hơn. Hơn nữa, nhờ những góp ý từ thầy cô và bạn bè, chúng em sẽ hoàn thành tốt hơn ở các bài nghiên cứu khác sau này. Chúng em mong quý thầy cô giảng viên và bạn bè sẽ luôn quan tâm và hỗ trợ chúng em và chúc quý thầy cô giảng viên và các bạn luôn tràn đầy sức khỏe.

CHÚNG EM XIN CHÂN THÀNH CẢM ƠN!

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của tôi và được sự hướng dẫn của thầy Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày tháng năm*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Trần Trung Dũng*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Table of Contents**

[**LÝ THUYẾT VỀ GRADIENT DESCENT** 7](#_Toc131087915)

[**1. Giới thiệu:** 7](#_Toc131087916)

[**2. Ví dụ về cách thức hoạt động của thuật toán Gradient Descent:** 8](#_Toc131087917)

[**3. Các thuật toán tối ưu Gradient Descent:** 8](#_Toc131087918)

[3.1 Momentum: 8](#_Toc131087919)

[3.2 Nesterov accelerated gradient (NAG): 10](#_Toc131087920)

[**STOCHASTIC GRADIENT DESCENT** 11](#_Toc131087921)

[**1. Giới thiệu:** 11](#_Toc131087922)

[**2. Một ví dụ:** 12](#_Toc131087923)

[**LÝ THUYẾT TRANSFORMER** 13](#_Toc131087924)

[**1. Giới thiệu:** 13](#_Toc131087925)

[**2. Cơ chế Attention và Self-attention:** 14](#_Toc131087926)

[2.1 Attention: 14](#_Toc131087927)

[2.2. Seft-Attention: 16](#_Toc131087928)

[**3. Các cấu trúc của Transformer:** 19](#_Toc131087929)

[**3.1 Cấu trúc Encoder:** 19](#_Toc131087930)

[**3.2 Cấu trúc Decoder:** 24](#_Toc131087931)

[**3.3 Cấu trúc kết hợp Transformer Encoder – Decoder:** 27](#_Toc131087932)

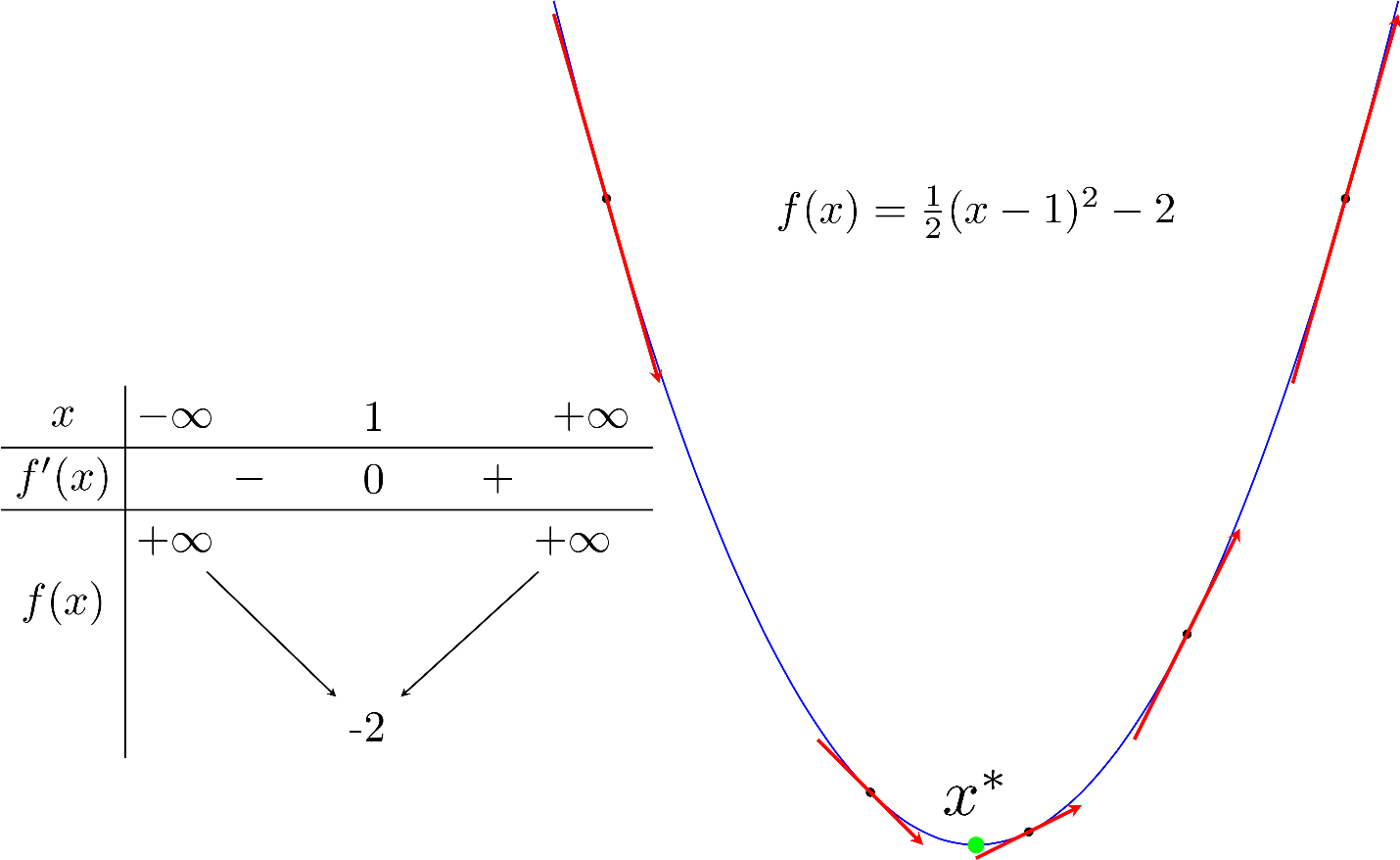
[**TÀI LIỆU THAM KHẢO** 33](#_Toc131087933)

# **LÝ THUYẾT VỀ GRADIENT DESCENT**

## **1. Giới thiệu:**

Gradient Descent là một kỹ thuật tối hưu hóa cơ bản, thường được thấy ở phần lớn các ứng dụng của các mô hình học máy SOTA (state-of-the-art machine learning models)

Gradient Descent thường được dùng để tìm giá trị cực tiểu của một **hàm số mục tiêu**. Trong kiến thức toán phổ thông chúng ta đã biết, muốn tìm cực trị một hàm số chúng ta sẽ giải phương trình đạo hàm của hàm số .



Tuy nhiên trong thực tế, những **hàm số mục tiêu** của các mô hình học máy thường rất phức tạp, khiến cho việc giải phương trình đạo hàm của hàm số bằng 0 gần như là bất khả thi. Và Gradient Descent chính là lời giải cho những trường hợp này, khi nó cách thức để tìm những điểm cực tiểu lân cận (local minimum) một cách xấp xỉ theo một vòng lặp, cho đến khi ta xác định được một con số xấp xỉ với giá trị cực tiểu của của **hàm số mục tiêu** dựa vào việc so sánh với các giá trị liền kề (chẳng hạn ), điều này cho thấy độ dốc (slope) của hàm số đã đạt một ngưỡng khá nhỏ và có thể chấp nhận được.

## **2. Ví dụ về cách thức hoạt động của thuật toán Gradient Descent:**

Cho một hàm số , với. Để tìm được khiến cho hàm số này đạt giá trị cực tiểu, thuật toán Gradient Descent sẽ thực hiện các bước sau:

Bước 1: Chọn một giá trị khởi đầu ngẫu nhiên.

Bước 2: Chọn một số lần lặp tối đa T.

Bước 3: Chọn một giá trị **learning rate** *η* Bước 4: Thực hiện vòng lặp các bước sau với số lần lặp tối đa T:

a. Thực hiện phép tính:

b. Cập nhật

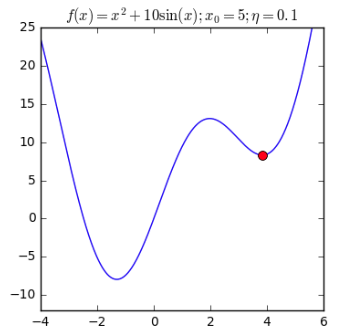
Ở đây chính là Gradient của được cho bởi công thức:

## **3. Các thuật toán tối ưu Gradient Descent:**

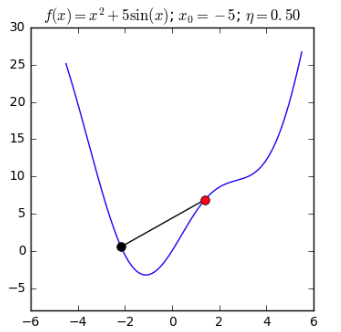
### 3.1 Momentum:

Khi sử dụng thuật toán Gradient Descent, ta thường sẽ bắt gặp một số vấn đề như:

a. Bị mắc lại ở một điểm cực tiểu lân cận không mong muốn và không thể đạt được giá trị khiến hàm số đạt cực tiểu toàn cục (global minimum).



b. Bước nhảy quá lớn dẫn đến kết quả bị vượt quá điểm cực tiểu toàn cục mong muốn, điều này là hệ quả của việc chọn một **learning rate** lớn hơn mức cần thiết.



Để khắc phục những vấn đề này, một **momentum**  sẽ được thêm vào trong bước tính toán để cân bằng **learning rate** trong suốt quá trình thực hiện thuật toán với số lần lặp tối đa cho trước qua công thức:

tại thời điểm có thể được hiểu là vận tốc thay đổi của giá trị từ thời điểm đến thời điểm , và được tính bằng công thức:

Trong đó thường được chọn là một giá trị khoảng 0.9, là độ dốc của đồ thị tại thời điểm trước đó (coi ).

**Ưu điểm :**

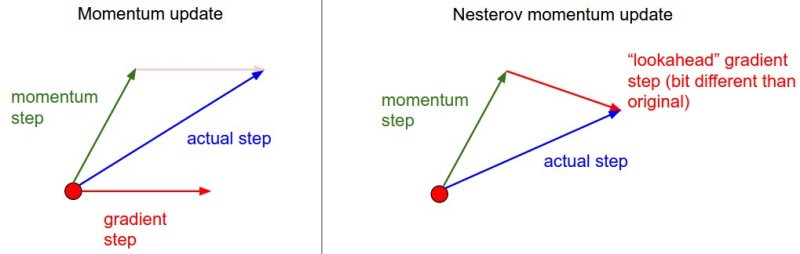
Momentum giải quyết được vấn đề: Gradient Descent không tiến được tới điểm global minimum mà chỉ dừng lại ở local minimum.

**Nhược điểm :**

Tuy momentum giúp ta vượt qua được những cực tiểu lân cận không mong muốn để tiến tới điểm cực tiểu toàn cục, tuy nhiên khi tới gần đích, giá trị của vẫn mất khá nhiều thời gian giao động qua lại trước khi dừng hẳn do vẫn còn lượng momentum chưa được triệt tiêu hoàn toàn.

### 3.2 Nesterov accelerated gradient (NAG):

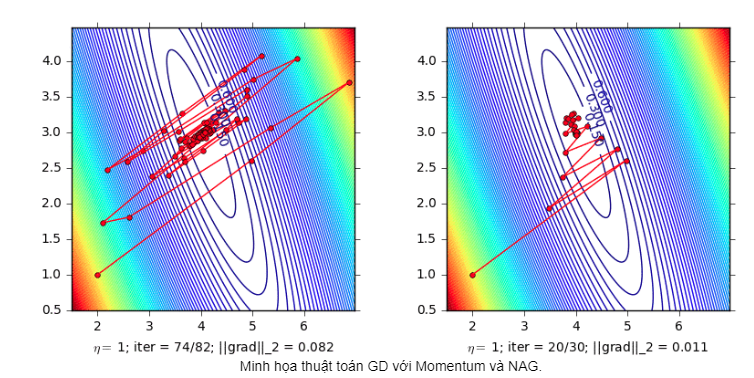
Thuật toán **NAG** được sinh ra để bù đắp phần khuyết điểm của **Momentum**, cho phép ta xác định được hướng di chuyển của giá trị trong bước kế tiếp bằng cách thay vì sử dụng gradient của vị trí hiện tại, **NAG** sử dụng gradient của vị trí tiếp theo để tính toán giá trị thực tế của và vị trí của nó trên đồ thị của **hàm số mục tiêu.** Dưới dây là hình minh họa sự khác biệt giữa **Momentum** và **NAG:**



Với **momentum** thông thường: lượng thay đổi là tổng của hai vector: **momentum vector** và **gradient** ở thời điểm **hiện tại**.

Với **Nesterove momentum**: lượng thay đổi là tổng của hai vector: **momentum vector** và **gradient** ở thời điểm được xấp xỉ là điểm **tiếp theo**

Vận tốc thay đổi của thời điểm đến thời điểm lúc này sẽ được tính bằng công thức:



# **STOCHASTIC GRADIENT DESCENT**

## **1. Giới thiệu:**

**Stochastic gradient descent (SGD)** là một biến thể của thuật toán **Gradient Descent,** là một thuật toán phù hợp với những bài toán machine learning sử dụng dữ liệu lớn (như deep learning) và các bài toán yêu cầu mô hình thay đổi liên tục (online learning) hơn hẳn **Gradient Descent,** khi thuật toán yêu cầu ít vòng lặp (hay gọi là **epoch**) hơn để cho ra kết quả. Ở **GD** thông thường, mỗi epoch thuật toán sẽ cập nhật giá trị của 1 lần, trong khi đó ở **SGD**, mỗi epoch thuật toán này sẽ cập nhật giá trị của lần, với là số điểm dữ liệu.

Điểm đặc biệt ở **SGD** là sau mỗi epoch, ta cần xáo trộn thứ tự của các dữ liệu để đảm bảo tính ngẫu nhiên, thứ sẽ ảnh hưởng lên hiệu năng của thuật toán một cách đáng kể.

Công thức cập nhật của **SGD**:

Với là hàm mất mát của chỉ **1** cặp điểm dữ liệu (input, label).

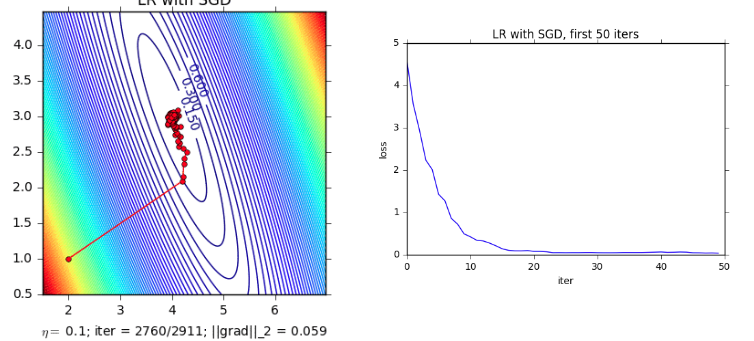
Ngoài ra, ta hoàn toàn có thể áp dụng các thuật toán tối ưu hóa **GD** như **Momentum, NAG**,…vào **SGD.**

## **2. Một ví dụ:**

Cho bài toán **Linear Regression**, với , **hàm mất mát** tại một điểm dữ liệu được tính bằng:

Ta có được **đạo hàm** theo tương ứng:

Hình ảnh mô tả bài toán này khi áp dụng thuật toán **SGD:**



Hình bên trái mô tả đường đi của nghiệm. Chúng ta thấy rằng đường đi khá là **zigzag**, không như khi sử dụng **GD**. Điều này xảy ra vì một điểm dữ liệu không thể đại diện cho toàn bộ dữ liệu. Tuy nhiên, chúng ta cũng thấy rằng thuật toán **hội tụ** khá nhanh đến vùng lân cận của nghiệm. Với 1000 điểm dữ liệu, **SGD** chỉ cần gần **3 epoche** (2911 tương ứng với 2911 lần cập nhật, mỗi lần lấy 1 điểm). Nếu so với con số **49 epoche** như kết quả tốt nhất có được bằng **GD**, thì kết quả này vượt trội hơn rất nhiều.

Hình bên phải mô tả hàm mất mát cho toàn bộ dữ liệu sau khi chỉ sử dụng 50 điểm dữ liệu đầu tiên. Mặc dù hàm mất mát biến thiên khá nhiều, tốc độ hội tụ vẫn là rất nhanh, tạo nên ưu điểm của **SGD.**

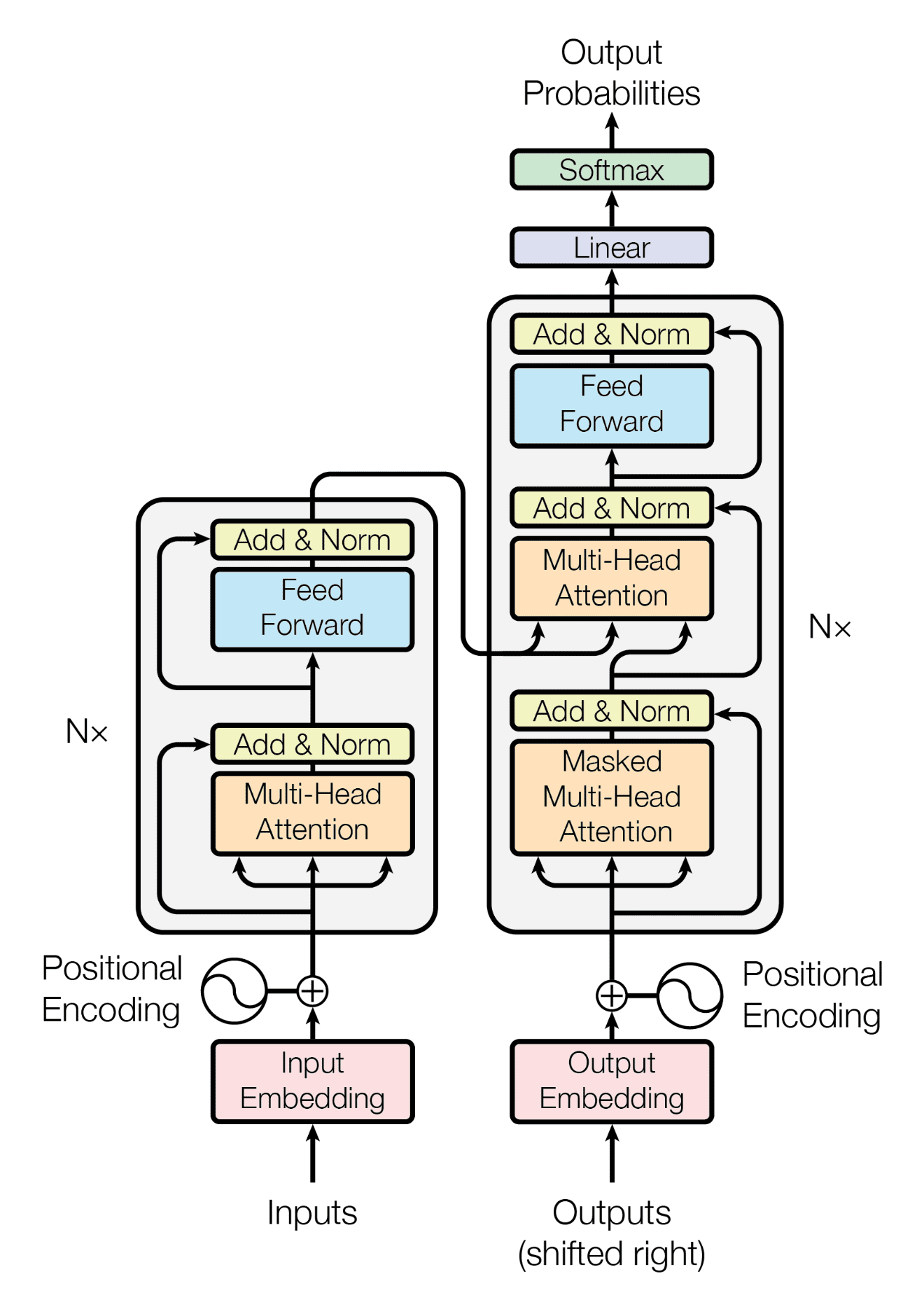
# **LÝ THUYẾT TRANSFORMER**

## **1. Giới thiệu:**

**Transformer** **model** là một **mạng neural** có khả năng học đồng thời **ngữ cảnh** và **ngữ nghĩa** của dữ liệu **ngôn ngữ tự nhiên** bằng cách theo dõi **mối quan hệ** trong các bộ dữ liệu tuần tự như các **từ** trong một **câu**,..

**Transformer** **model** sử dụng một tập hợp các kỹ thuật **toán học** tiên tiến gọi là cơ chế **Attention** hay **Self-attention**, để có thể nhận diện được sự **liên quan** và **phụ thuộc** giữa các phần tử dữ liệu kể cả khi chúng ở những vị trí cách **xa** nhau.

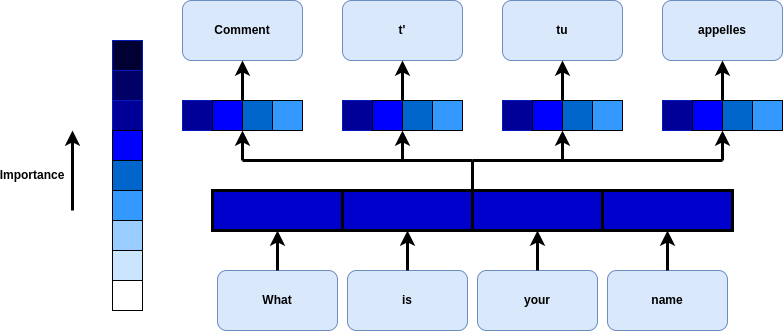
Lần đầu tiên được mô tả trong một bài báo năm 2017 của **Google**, **Transformer** là một trong những loại mô hình **mới nhất** và **mạnh mẽ nhất** được phát minh cho đến nay. Mô hình này đang thúc đẩy một làn sóng tiến bộ trong **học máy**, còn được biết đến với cái tên **Transformer AI**.



## **2. Cơ chế Attention và Self-attention:**

### 2.1 Attention:

Cơ chế **Attention** được giới thiệu để tăng cường **hiệu năng** của những mô hình học máy dạng **Encoder-Decoder** cho những bài toán **xử lý ngôn ngữ tự nhiên** nói chung, và trong bài toán **Machine Translation** nói riêng. Ý tưởng chính của cơ chế này là cho phép **Decoder** tận dụng tối đa **mối liên hệ** giữa các phần khác nhau của dữ liệu tuần tự (sequential data) một cách linh hoạt, bằng những **tổ hợp có trọng số** của toàn bộ các vector đầu vào đã được mã hóa, cho phép nhận diện được **mức độ liên quan** đến nhau giữa các vector này.



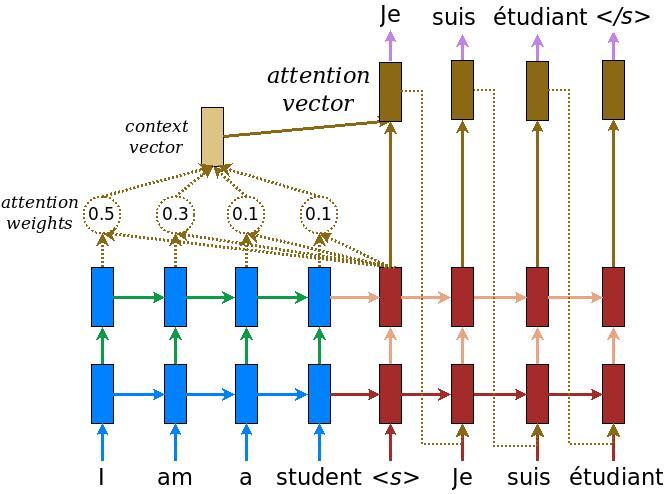
Dạng tổng quát của cơ chế **Attention** sử dụng 3 thành phần chính: ***Queries*** (), ***keys*** () và ***values*** () và thực hiện các tính toán như sau:

1. Mỗi **vector truy vấn (query vector)**, , được khớp với database chứa các **khóa (keys)** để tính **giá trị score (score value)** dưới dạng **tích vô hướng** **(dot product)** của truy vấn cụ thể đang được xem xét với mỗi **vector khóa (key vector)** :

2. Kế đến các **giá trị score** được truyền qua một hàm để tạo ra các **trọng số**:

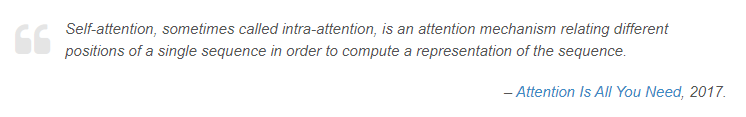
3. Sau cùng, **Attention tổng quát** được tính bằng tổng trọng số của các **vector giá trị (value vectors)** , khi mà mỗi **vector giá trị** được bắt cặp với một **khóa** tương ứng:

Lấy ví dụ cụ thể đối với bài toán **Machine translation**, mỗi từ trong một câu đầu vào sẽ được gán cho các **vector truy vấn**, **khóa** và **giá trị** riêng của nó. Các vector này được tạo bằng cách **nhân** biểu diễn của bộ mã hóa từ đang được xem xét với 3 **ma trận trọng số** khác nhau được tạo ra trong quá trình training.



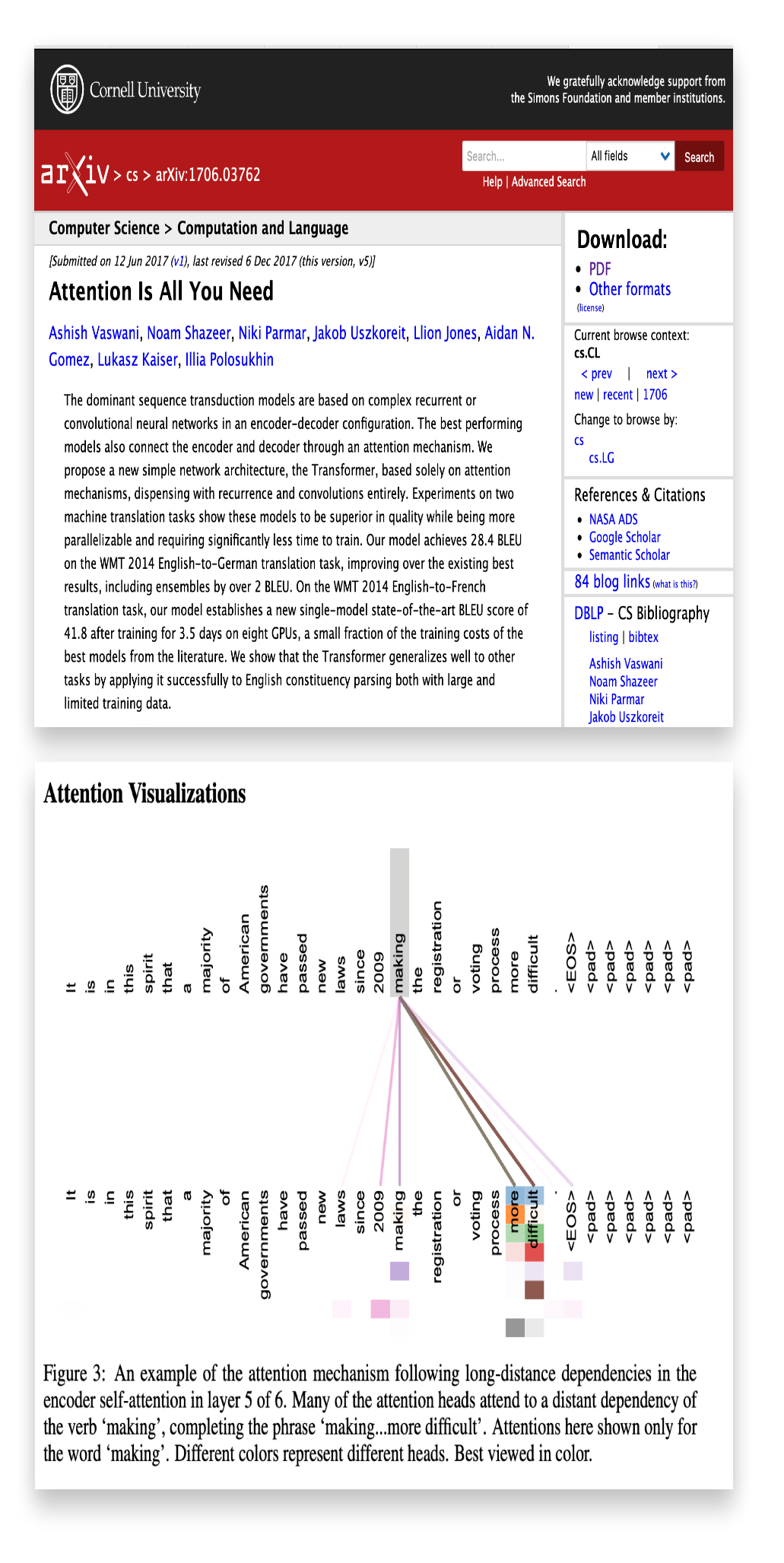
Về bản chất, khi cơ chế **Attention tổng quát** được trình bày với một chuỗi các từ, nó sẽ lấy **vector truy vấn** được gán cho một số từ cụ thể trong chuỗi và **chấm điểm** nó theo từng khóa trong cơ sở dữ liệu. Từ đó, **Attention** nắm bắt cách mà các từ đang được xem xét **liên quan** đến các từ khác trong chuỗi. Sau đó, **Attention** sẽ **chia** tỷ lệ các giá trị theo **trọng số chú ý** (được tính toán từ **điểm số**) để duy trì **sự tập trung** vào những từ có liên quan đến truy vấn, qua đó tạo ra một **attention output** cho từ đang được xem xét.

### 2.2. Seft-Attention:



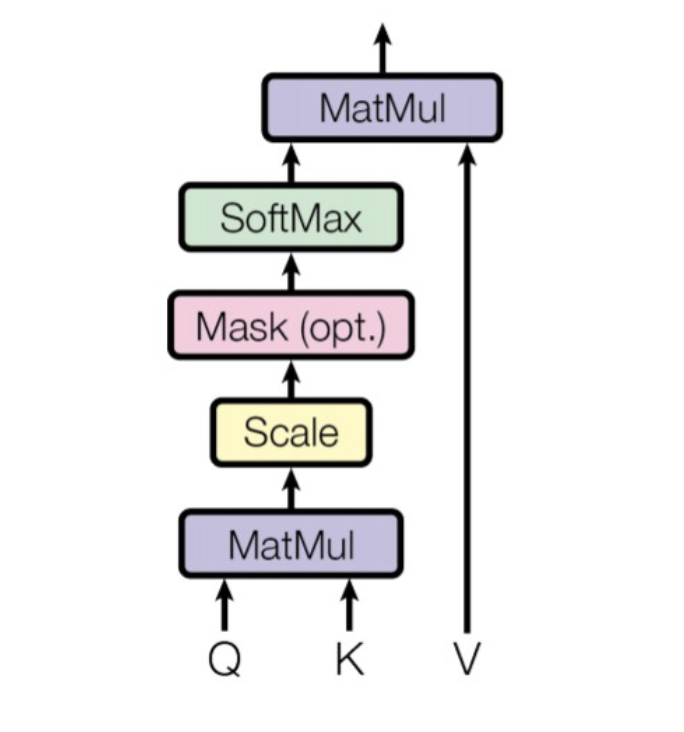
Được giới thiệu trong bài báo nổi tiếng “**Attention is All You Need**” vào năm 2017, **self-attention** đã và đang trở thành nền tảng của phần lớn các mô hình deep learning **SOTA,** đặc biệt là trong lĩnh vực **Xử lý ngôn ngữ tự nhiên (NLP)**.

**Self-attention** có thể được hiểu là một cơ chế **tăng cường lượng nội dung thông tin** của quá trình embed bằng cách bổ sung thông tin về **ngữ cảnh** của dữ liệu đầu vào. Hay nói cách khác, self-attention cho phép mô hình **cân nhắc** về mức độ **quan trọng** của các yếu tố khác nhau trong chuỗi **đầu vào** và tự động điều chỉnh **mức độ ảnh hưởng** của chúng đối với **đầu ra**. Điều này đặc biệt quan trọng đối với các tác vụ **xử lý ngôn ngữ**, trong đó **ý nghĩa** của một từ có thể thay đổi dựa trên **ngữ cảnh** của từ đó trong một câu hoặc một tài liệu.



Trong hình trên có thể thấy được rằng cơ chế nhận thấy được rất nhiều “chú ý” đối với từ “**making**” và nhiều nhất là ở cụm “**more difficult**”.

Có nhiều biến thể của **Self-attention** được tạo ra nhằm tối ưu hóa cơ chế này, tuy nhiên đa số những **bài báo khoa học** tập trung vào phiên bản **đầu tiên**, **Scaled-dot product** **attention**, bởi kết quả thu được của nó là vượt trội về **độ chính xác**.



**Scaled-dot product attention** vẫn sẽ sử dụng 3 thành phần chính là **, ,** và **.** Ở **Encoder**, chúng đều sẽ mang cùng 1 chuỗi đầu vào đã được **embed** và tăng cường bởi thông tin về vị trí (**positional information**). Tương tự ở phía **Decoder**, các truy vấn, khóa và gia trị được đưa vào khối **Attention đầu tiên** thể hiện chuỗi đầu ra tương tự sau khi chuỗi này cũng được **embed** và tăng cường thêm **thông tin về vị trí**. Kế đến là khối **attention thứ 2** của **Decoder** nhận đầu ra của **Encoder** dưới dạng các khóa và giá trị, bên cạnh các đầu ra đã được chuẩn hóa của khối **Attention đầu tiên** dưới dạng các truy vấn. **Chiều** của các vector truy vấn và vector khóa được kí hiệu là , còn **chiều** của các vector giá trị sẽ được kí hiệu là

Khối **Scaled-dot product** sau đó nhận các truy vấn, khóa và giá trị này làm **đầu vào** và thực hiện tính **tích vô hướng** của các vector truy vấn với các vector khóa. Kết quả của phép toán sau đó được chia với **căn bậc 2** của để tính được các điểm số attention (**Attention scores**). Những điểm số này tiếp tục được đi qua một hàm , thu được một tập hợp các trọng số attention (**Attention weights**). Sau cùng, các trọng số attention sẽ được sử dụng để xác định **tỷ lệ** của các giá trị thông qua **phép nhân trọng số**. Toàn bộ quá trình sẽ được khái quát lại thông qua công thức sau:

Với:

Ta có được:

Phiên bản **Scaled-dot product Attention** này được tác giả Vaswni của bài báo “**Attention is all you need**” giải thích là khá tương đồng với phiên bản **Multiplicative Attention** của **Minh Thắng Lương et al**. (2015), ngoại trừ một **cải tiến** với việc bổ sung **hệ số tỷ lệ** . Hệ số này được giới thiệu với mục đích nhằm chống lại **tác động** của việc kết quả của các **tích vô hướng** tăng **quá cao** bởi độ lớn của **chiều** , khi mà **tác động** của hàm sẽ trả về gradient **quá nhỏ** dẫn đến vấn đề kinh điển **vanishing gradient**. Hệ số tỷ lệ có công dụng **kéo thấp** kết quả sinh ra của quá trình tính **tích vô hướng**, thứ sẽ ngăn chặn vấn đề này.

## **3. Các cấu trúc của Transformer:**

### **3.1 Cấu trúc Encoder:**

Khối Encoder của Transformer bao gồm N tầng, với mỗi tầng gồm 2 tầng con (sub-layer) chính:

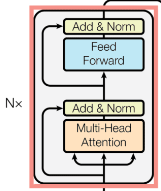
- Tầng con thứ nhất bao gồm 1 cơ chế Multi-head attention có nhiệm vụ nhận các truy vấn, khóa và giá trị làm input.

- Tầng con thứ 2 bao gồm một mạng fully-connected feed-forward.

Nối tiếp ngay sau mỗi tầng con này là khối chuẩn hóa tầng, trong đó input và output của các tầng con được cung cấp (thông qua kết nối residual). Output của mỗi bước chuẩn hóa tầng được tính như sau:

LayerNorm(sub\_layer\_input + sub\_layer\_output)

Để phục vụ cho điều này, Vaswani et al. đã thiết kế tất cả các tầng con và tầng embedding trong mô hình để tạo ra output có số chiều = 512.



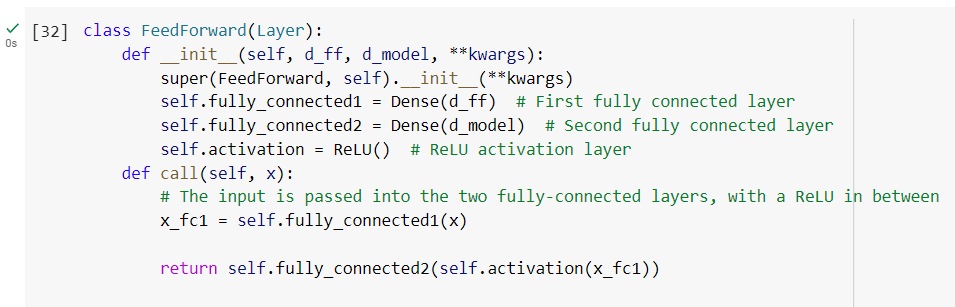
Kế đến, hãy cùng xem phần cài đặt của Transformer Encoder:

Đầu tiên, import các thư viện cần thiết cho việc cài đặt:

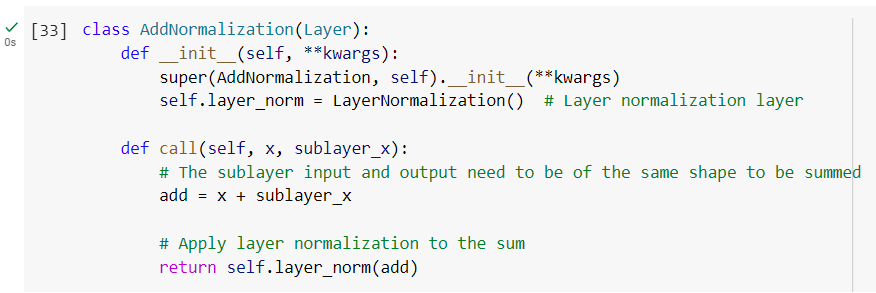
****

Tiếp theo, bắt đầu cài đặt với việc tạo ra 2 class cho các tầng Feed Forward và Add & Norm.

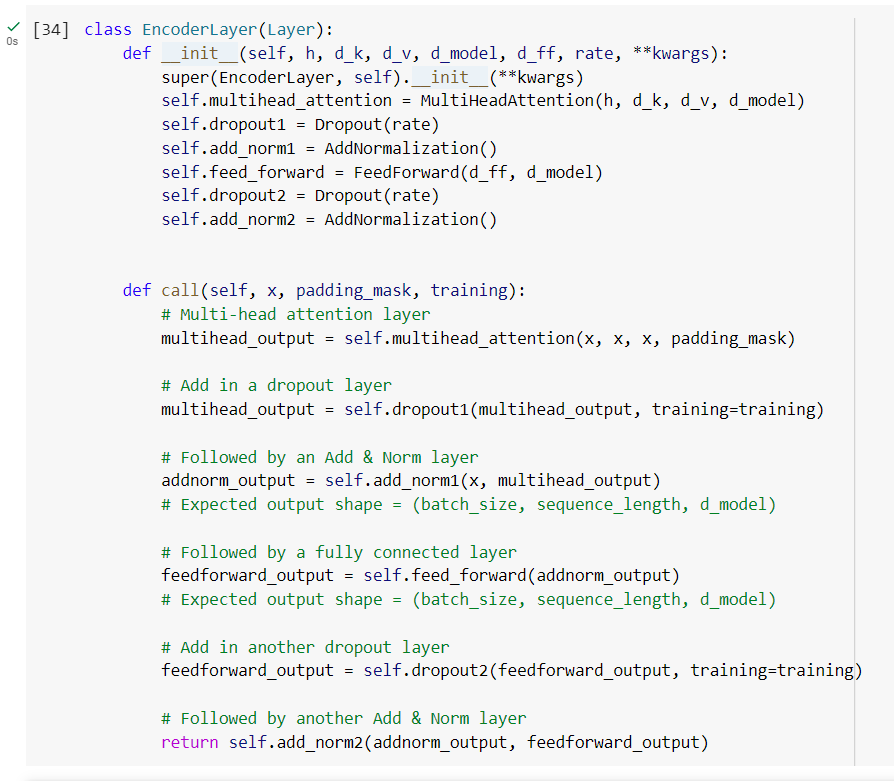
Theo Vaswani et al, mạng fully-connected feed-forward gồm có 2 bộ biến đổi tuyến tính (linear transformation) với 1 tầng activation ReLU ở giữa. Bộ biến đổi tuyến tính đầu tiên tạo ra output có số chiều , và bộ biến đổi tuyến tính thứ 2 tạo ra output có số chiều . Từ đây, ta có thể cài đặt class FeedForward như sau:



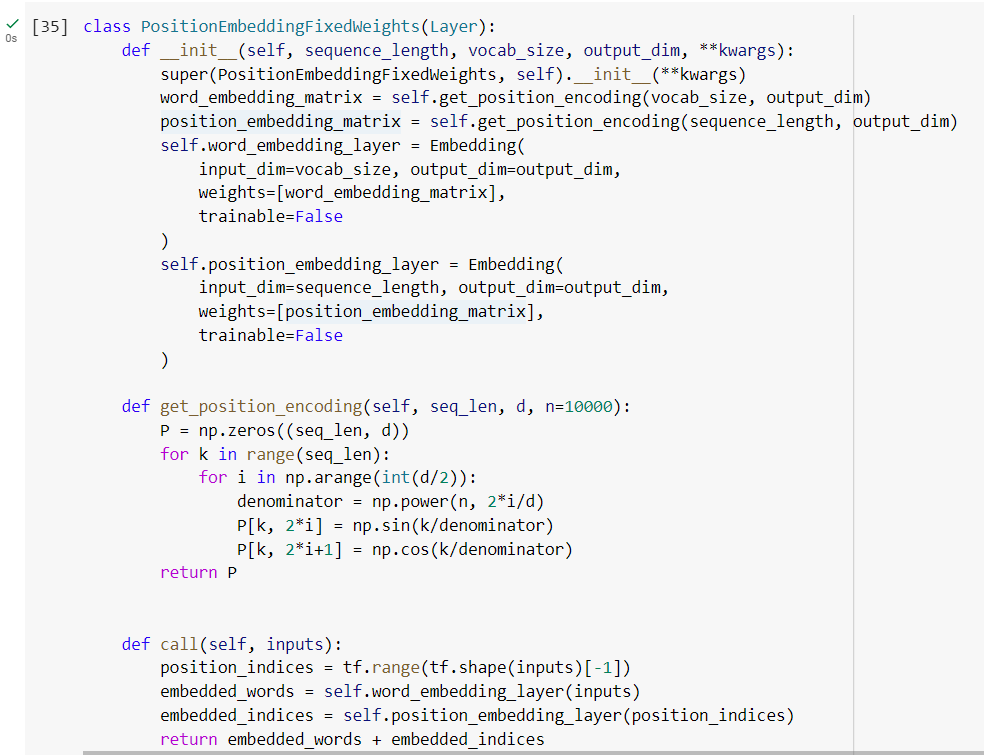
Bước tiếp theo là cài đặt class AddNormalization:



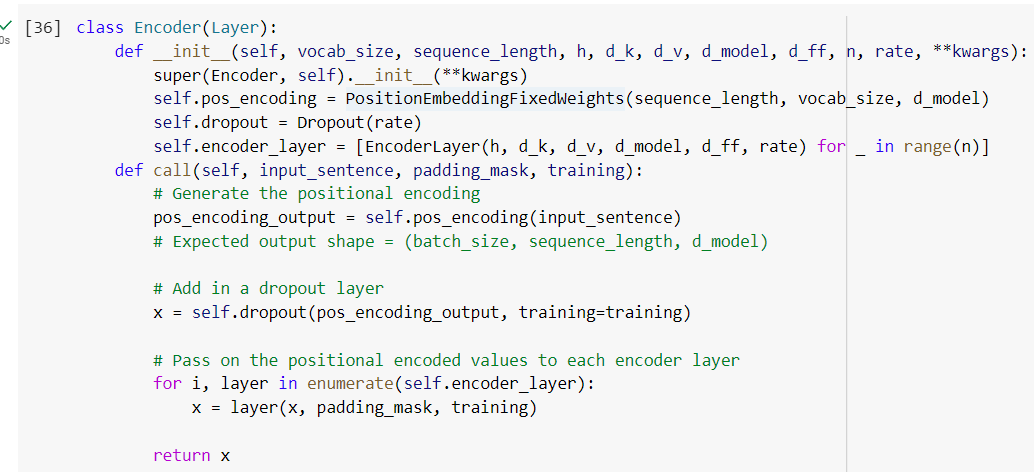
Kế đến là cài đặt tầng Encoder, thứ sau đó sẽ được Encoder của Transformer nhân bản lên N lần, đặt tên là EncoderLayer:



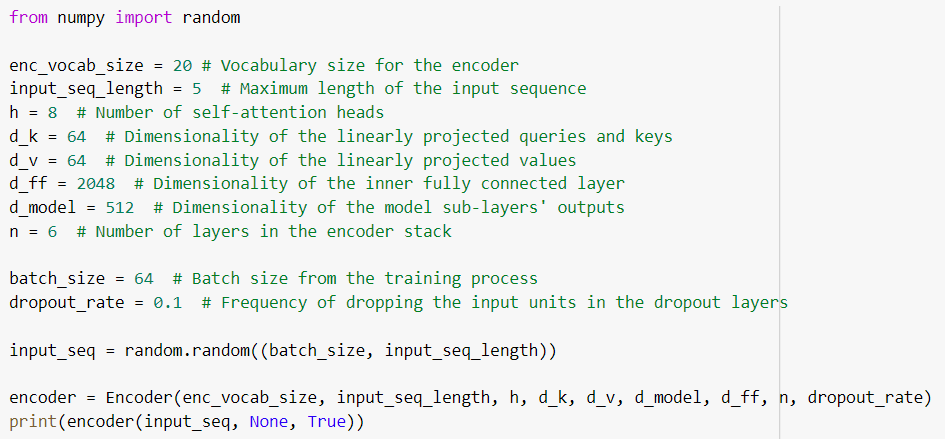
Ở bước cuối cùng, ta sẽ cần 1 tầng embedding vị trí (Positional Embedding) với các trọng số được cố định. Vì vậy, tiếp đến sẽ cần cài đặt một class PositionEmbeddingFixedWeights:



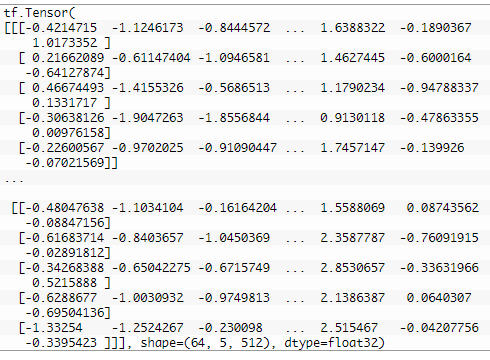
Và bước cuối cùng của việc cài đặt là tạo ra 1 class cho Transformer Encoder:



Ta sẽ cần 1 ví dụ để test cấu trúc vừa cài đặt (các tham số tham khảo từ tài liệu “Attention is all you need” của Vaswani et al.)



Output:



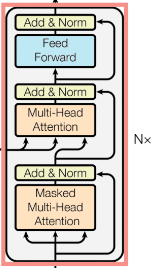
### **3.2 Cấu trúc Decoder:**

Tương tự như ở cấu trúc Encoder của Transformer, Transfomer Decoder cũng gồm N các tầng tương tự nhau. Bên cạnh đó, Transformer Decoder sẽ có thêm một khối Multi-head attention với tổng cộng 3 tầng con chính:

- Tầng con đầu tiên bao gồm 1 cơ chế Multi-head attention nhận các truy vấn, khóa và giá trị làm input.

- Tầng con thứ 2 chứa 1 cơ chế Multi-head attention thứ 2.

- Tầng con thứ 3 gồm một mạng fully-connected feed-forward.

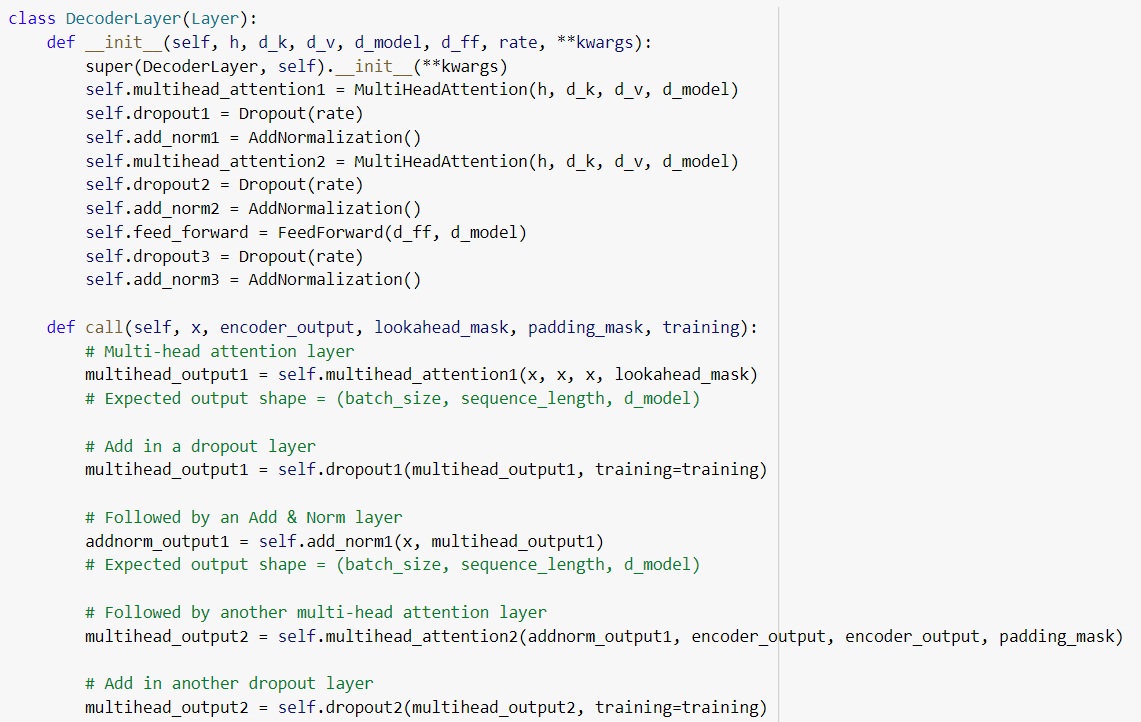


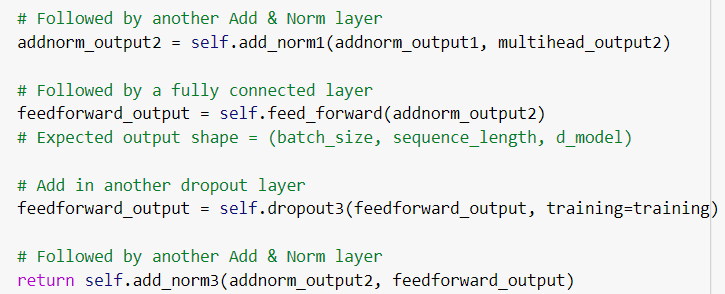
Đồng thời, theo sau mỗi tầng con này cũng là khối chuẩn hóa tầng (giống với Transformer Encoder).

Ở phía Decoder, các truy vấn, khóa và giá trị được đưa vào khối Multi-head attention đầu tiên đại diện cho cùng 1 chuỗi đầu vào. Nhưng khác với Encoder là chuỗi kết quả được embed và bổ sung thông tin vị trí trước khi được cung cấp cho Decoder. Mặt khác, khối Multi-head attention thứ 2 nhận output của Encoder ở dạng khóa - giá trị, và nhận ouput được chuẩn hóa của khối attention đầu tiên dưới dạng các truy vấn. Ở cả 2 trường hợp này, chiều của các truy vấn và khóa giữ nguyên bằng giá trị của , còn chiều của các giá trị sẽ được giữ nguyên bằng .

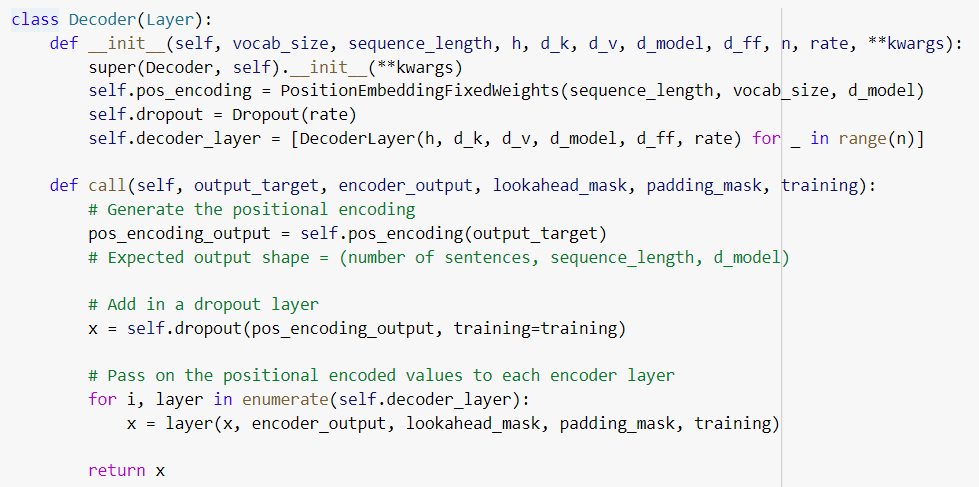
Kế tiếp, sẽ là phần cài đặt cho Transformer Decoder.

Ở phần cài đặt Encoder, ta đã import đầy đủ các extension cần thiết, vì vậy bây giờ sẽ ngay lập tức đi vào cài đặt class DecoderLayer, thứ sẽ được nhân bản lên N lần ở bước sau:

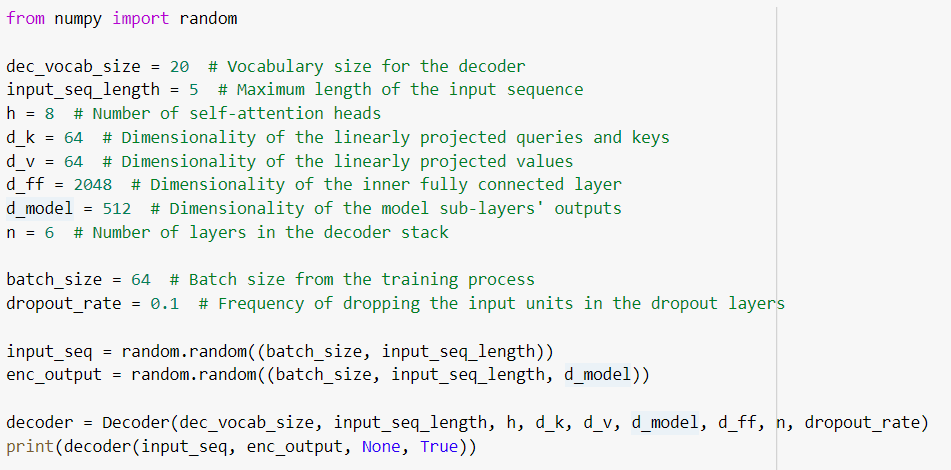




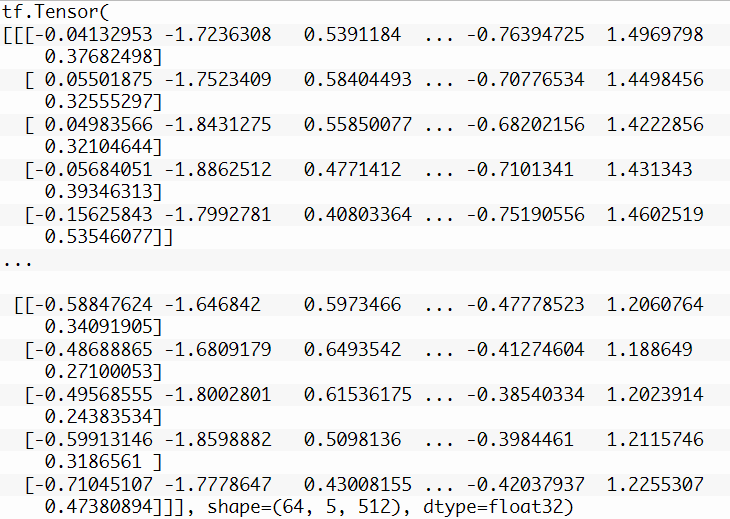
Kế tiếp, ta sẽ cài đặt class Decoder cho toàn bộ khối Transformer Decoder:



Sau cùng sẽ là 1 ví dụ để xem thử cách thức hoạt động của khối Transformer Decoder đã cài đặt:



Output:

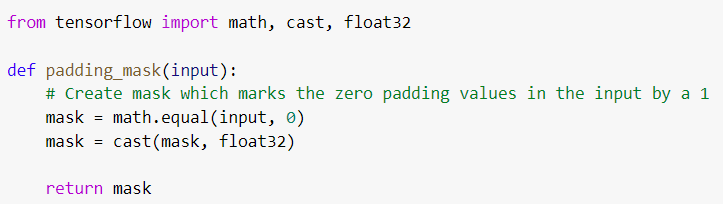


### **3.3 Cấu trúc kết hợp Transformer Encoder – Decoder:**

Ở 2 cấu trúc của phần trên, ta có thể thấy được tầm quan trọng của việc masking các giá trị input trước khi đưa chúng vào Encoder và Decoder.

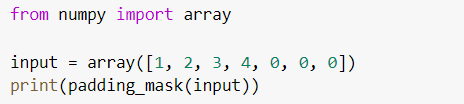
Khi thực hiện huấn luyện một mô hình Transformer, các chuỗi input được đưa vào Encoder và Decoder sẽ được zero-padded trước tiên thành các chuỗi có độ dài cụ thể. Tầm quan trọng của việc có một padding mask là đảm bảo rằng các giá trị zero này không được xử lý cùng với các giá trị input thực bởi cả Encoder và Decoder.

Vậy, ở bước đầu tiên của việc kết hợp 2 cấu trúc Encoder và Decoder của mô hình Transformer sẽ là cài đặt một hàm padding\_mask để tạo ra padding mask cho cả Encoder và Decoder:



Khi nhận được 1 input, hàm này sẽ tạo ra một tensor đánh dấu bằng một giá trị one ở bất cứ nơi nào input chứa giá giá trị zero.

Từ đó, nếu ta cho vào hàm trên 1 input dưới dạng 1 array như sau:

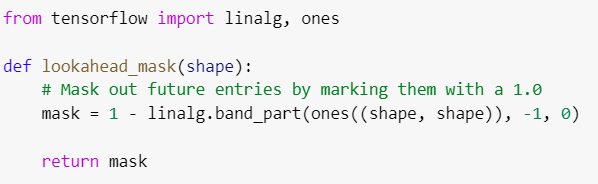


Thì giá trị trả về của hàm này sẽ là một tensor có dạng:



Bước kế tiếp, ta cần một Look-Ahead mask để ngăn chặn Decoder tham gia vào các từ tiếp theo, sao cho dự đoán của một từ cụ thể chỉ phụ thuộc vào output đã biết của các từ xuất hiện trước nó.

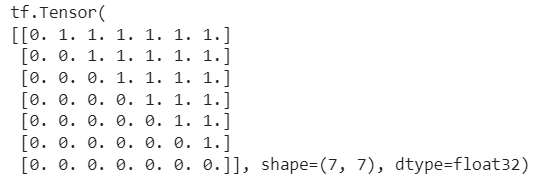
Dưới đây là phần cài đặt của hàm lookahead\_mask:



Để sử dụng hàm này, ta cần cung cấp cho nó độ dài input của Decoder. Cùng xem thử 1 ví dụ:

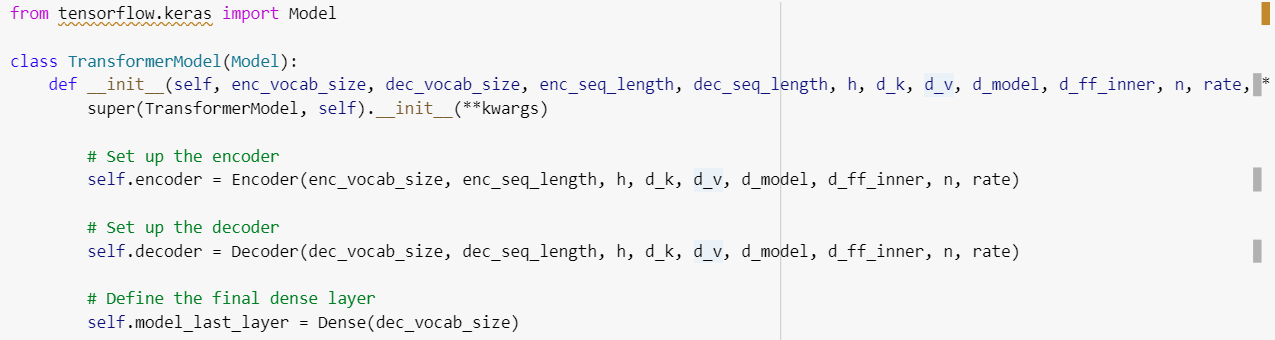


Ouput thu được của hàm sẽ là:



Có thể thấy, ở đây những giá trị one mask hết những mục không đáng quan tâm. Bằng cách này, dự đoán của bất kì từ nào cũng sẽ chỉ phụ thuộc vào những từ đằng trước nó.

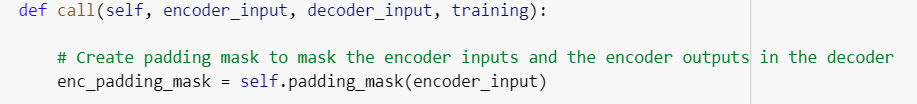
Ngay sau bước này, ta có thể bắt đầu thực hiện việc cài đặt model Transformer bao gồm khối Encoder và Decoder đã được cài đặt từ các phần trước như sau:



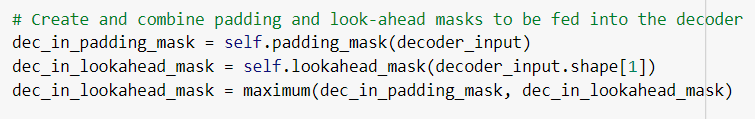
Bước đầu tiên của việc cài đặt class TransformerModel là khởi tạo các instance của class Encoder và class Decoder và gán output của chúng với các biến encoder và decoder. Ta cũng cần phải thêm vào 1 tầng Dense ở cuối để tạo ra final\_output, như trong kiến trúc Transformer của Vaswani et al. (2017).

Kế đó sẽ là method call() như thường lệ để cung cấp các input có liên quan vào Encoder và Decoder.

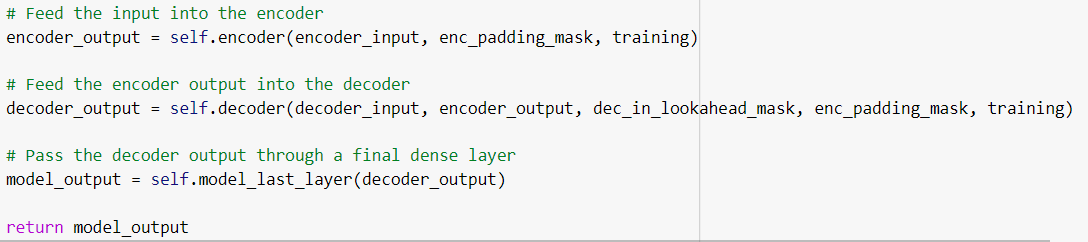
Một padding mask sẽ được tạo ra đầu tiên ở method call() nhằm mask các input và output của Encoder khi chúng được đưa vào khối self-attention thứ 2 của decoder:



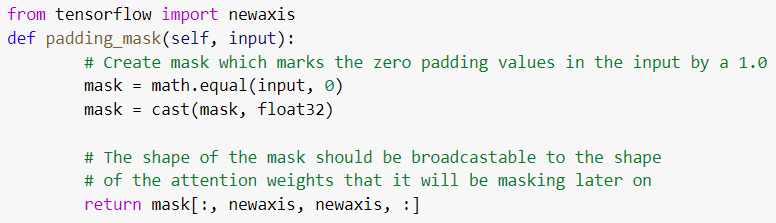
Và tiếp theo đó, 1 padding mask và 1 look-ahead mask sẽ được khởi tạo để mask input của Decoder. Ta có thể tổng hợp chúng với hàm tính toán maximum:



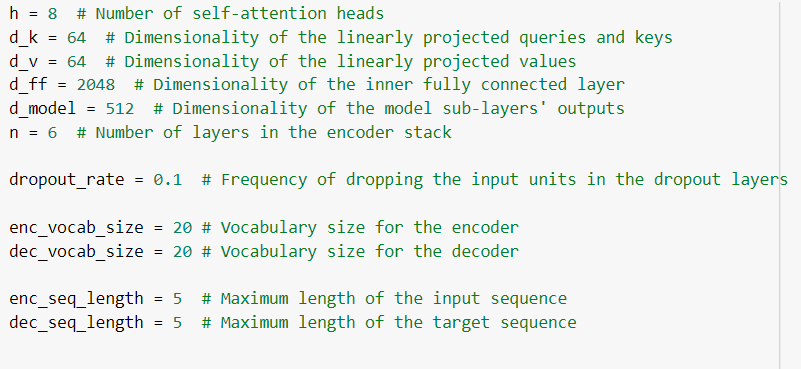
Tiếp theo, những input liên quan sẽ được đưa vào Encoder và Decoder, và output của mô hình Transformer sẽ được sinh ra bằng cách đưa output của Decoder vào một tầng Dense cuối cùng:

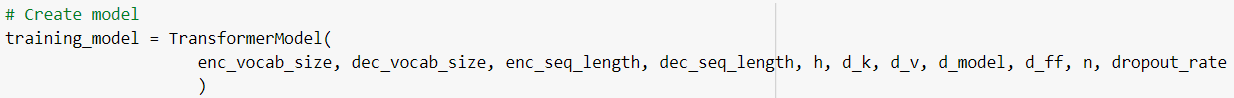


Lưu ý là ở class TransformerModel ta sẽ cần tinh chỉnh 1 chút ở ouput của hàm padding\_mask. Kích thước của nó được tùy chỉnh cho phù hợp với kích thước của tensor trọng số attention mà hàm này sẽ mask khi ta huấn luyện mô hình Transformer:



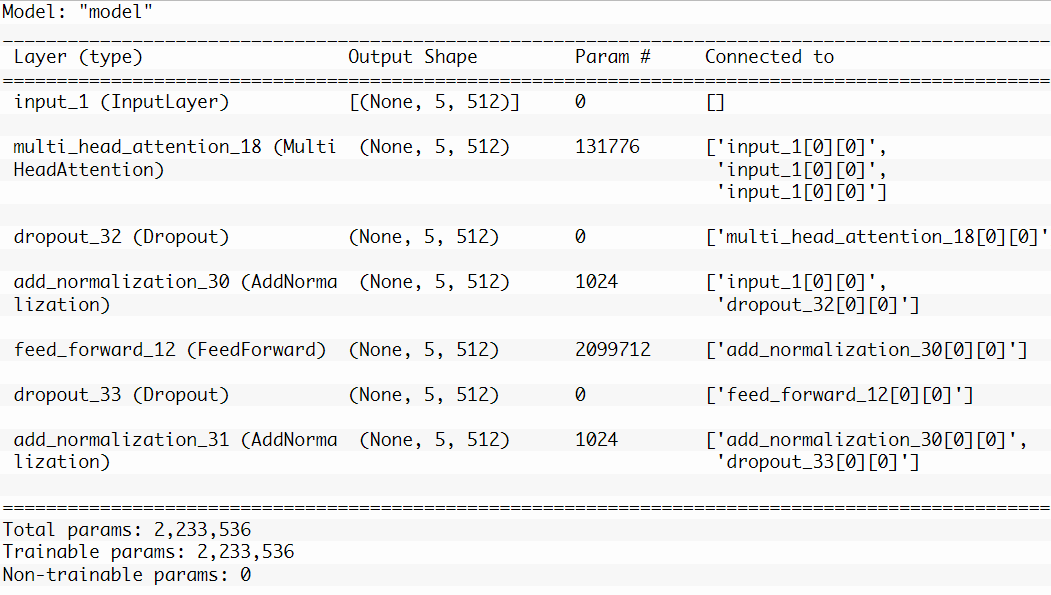
Sau đây sẽ là phần test mô hình Transformer đã cài đặt:

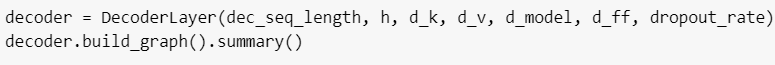




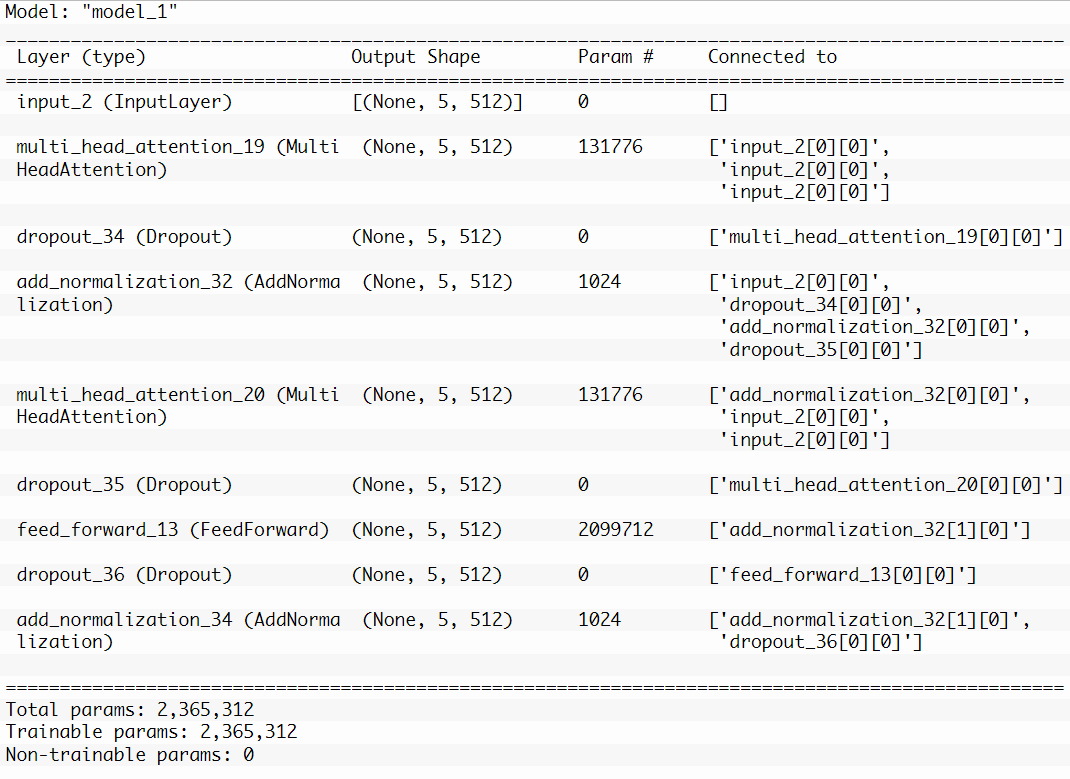


Kết quả của hàm summary cho Encoder:





Kết quả của hàm summary cho Decoder:



# **TÀI LIỆU THAM KHẢO**

[1]. **Stackabuse.com**, *“Gradient Descent in Python: Implementation and Theory”*, ngày 21 tháng 7 năm 2022.

[2]. **Machinelearningcoban.com**, *“Gradient Descent (phần 1 và phần 2)”*, ngày 16 tháng 1 năm 2017.

[3]. **Ashish Vaswani et al**, *“Attention is all you need”*, chỉnh sửa lần cuối ngày 6 tháng 12 năm 2017.

[4]. **Stefania Christina, machinelearningmastery.com**, *“The Transformer Attention Mechanism”*, ngày 15 tháng 9 năm 2022.

[5]. **Georgios Nanos, Baeldung.com**, *“Differences Between Luong Attention and Bahdanau Attention”*, chỉnh sửa lần cuối ngày 20 tháng 3 năm 2023.