

ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN CÔNG NGHỆ TRI THỨC

Generative pre-trained transformer 2

Topic: GPT-2 (Causal Language Modeling) for Text Generation

Môn học: Học Thống Kê

Sinh viên thực hiện:

Lê Ngọc Tường - 20127383
Nguyễn Tư Duy - 20127484
Hồ Quốc Duy - 20127145

Giáo viên hướng dẫn:

Ngô Minh Nhựt
Lê Long Quốc

Ngày 26 tháng 8 năm 2023



Mục lục

1 Giới thiệu	2
2 Transformer	3
2.1 Kiến trúc chung	3
2.2 Encoding	4
2.2.1 Input embedding	4
2.2.2 Self-Attention	6
2.2.3 Position-wise Feed-Forward Networks (FFN)	12
2.2.4 Add-Normalize	12
2.3 Decoding	14
2.3.1 Cơ chế masking	14
2.3.2 Encoder-decoder attention	16
2.4 The final layer: Linear and Softmax	17
3 Generative Pre-trained Transformer 2	18
3.1 Token Embedding và Positional Embedding	20
3.2 Masked Multi Self Attention	20
3.3 Fully Connected Neural Network	24
3.4 Output và Inference	25
3.5 Beyond Language Modeling	26
3.6 Recap	27
4 Fine tune GPT-2	29
4.1 Dataset	29
4.2 Fine-tune model	31
4.2.1 Pre process	31
4.2.2 Training	33
4.3 Evaluation	34
4.3.1 Cross entropy và perplexity	34
4.3.2 Loss function và evaluate function	36
4.3.3 Results	36
5 Web App	38
5.1 Chức năng	38
5.2 Giao diện	38
5.3 Chạy phần mềm	39
Tài liệu	40

1 Giới thiệu

Trong thế giới ngày càng số hóa, khả năng tạo ra văn bản tự động đã trở thành một yếu tố quan trọng trong nhiều lĩnh vực, từ nội dung truyền thông đến tạo ra các bài viết và thậm chí là viết sách. Một trong những tiến bộ quan trọng trong lĩnh vực này là mô hình sinh văn bản GPT (Generative Pre-trained Transformer). Và để bước đầu tìm hiểu điều gì đã tạo nên một mô hình mạnh mẽ như vậy. Nhóm em sẽ thực hiện nghiên cứu, tìm hiểu về cơ chế của attention, transformer và kiến trúc dựa trên transformer đơn giản là GPT-2 phiên bản smallest.

GPT-2 là một mô hình học sâu nổi tiếng, được đào tạo trước trên lượng lớn dữ liệu văn bản. Sự đột phá của GPT-2 nằm trong việc sử dụng kiến trúc Transformer, một mô hình xử lý ngôn ngữ tự nhiên tiên tiến. Kiến trúc này cho phép mô hình hiểu và học được cấu trúc ngôn ngữ phức tạp, cũng như mối quan hệ giữa các từ trong văn bản.



Hình 1: GPT-2 (Generative Pre-trained Transformer 2)

Một khía cạnh quan trọng của GPT-2 là khả năng của nó trong việc thực hiện causal language modeling, tức là dự đoán từ tiếp theo trong một chuỗi dựa trên ngữ cảnh trước đó. Điều này cho phép GPT-2 tạo ra các đoạn văn bản liên quan và tự nhiên, thậm chí là có khả năng tạo ra nội dung có tính sáng tạo.

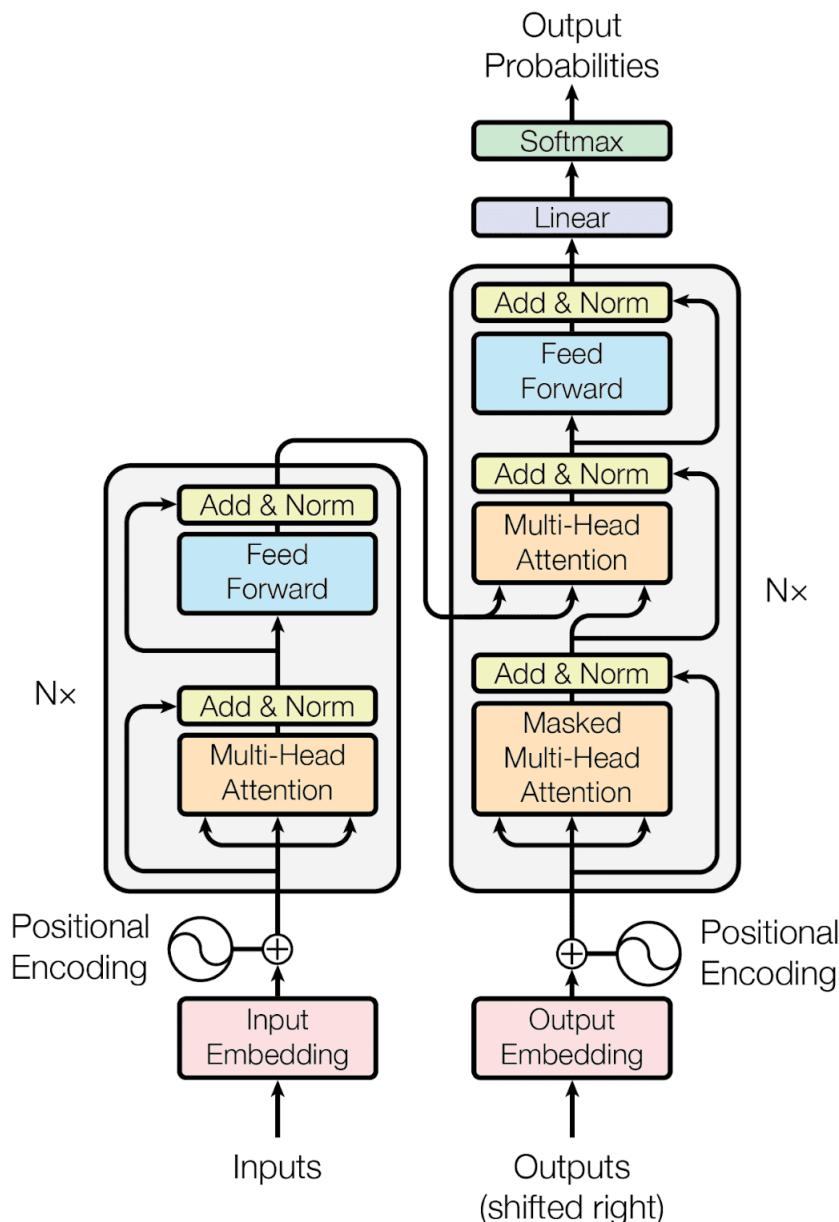
Trong báo cáo này, chúng ta sẽ đi sâu vào kiến trúc Transformer đã thay đổi cách chúng ta xử lý ngôn ngữ tự nhiên như thế nào và làm sao để GPT-2 có thể tạo ra nội dung văn bản đa dạng và hợp ngữ nghĩa. Đồng thời cũng thực hiện demo bằng cách fine-tune trên pre-trained GPT-2 để áp dụng cho tác vụ Causal Language Modeling.

2 Transformer

2.1 Kiến trúc chung

Kiến trúc của transformer [1] gồm 2 phần chính là Encoder (là 1 ngăn xếp gồm 6 khối encoder kiến trúc giống nhau) và Decoder (là 1 ngăn xếp gồm 6 khối decoder giống nhau).

- Mỗi khối encoder có 2 layer chính: self-attention và feed forward.
- Mỗi khối decoder có 3 layer chính: self-attention, encoder-decoder attention và feed forward.



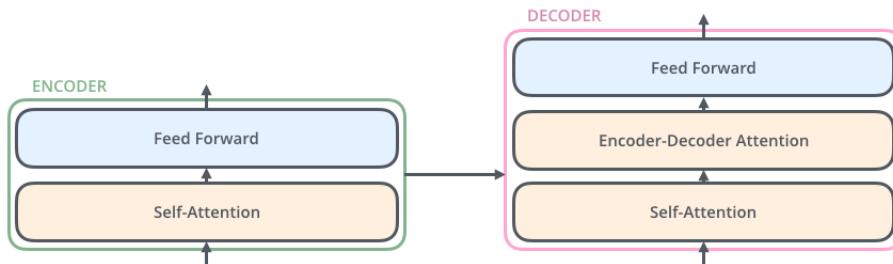
Hình 2: Cấu trúc Encoder-Decoder của kiến trúc Transformer

Điểm cải tiến:

- Tính song song và hiệu suất tính toán:** Kiến trúc Transformer cho phép tính toán đồng thời trên nhiều GPU, giảm thời gian huấn luyện và tăng tốc dự đoán. Điều này phù hợp với yêu cầu tính toán của các ứng dụng thời gian thực.
- Cơ chế Attention và học biểu diễn tốt hơn:** Cơ chế attention cho phép mô hình tập trung vào toàn bộ chuỗi đầu vào, tạo biểu diễn tốt hơn bằng cách kết hợp thông tin từ nhiều vị trí. Điều này giúp mô hình hiểu mối quan hệ giữa các từ và tóm tắt thông tin quan trọng.
- Xử lý chuỗi dài và tương quan:** Transformer xử lý chuỗi dữ liệu dài một cách hiệu quả, tránh vấn đề vanishing gradient. Điều này tạo biểu diễn đáng tin cậy cho các chuỗi phức tạp.
- Tích hợp thông tin vị trí:** Sử dụng mã hóa vị trí giúp mô hình hiểu thứ tự của từ trong chuỗi. Điều này làm việc xử lý ngôn ngữ tự nhiên hiệu quả hơn và phản ánh mối quan hệ giữa các từ.

2.2 Encoding

Encoder bao gồm 6 lớp mã hóa, mỗi lớp bao gồm 2 lớp con. Sub-layer đầu tiên là multi-head self-attention, cho phép các từ tương tác để tạo biểu diễn cải tiến. Sub-layer thứ hai là fully-connected feed-forward layer, tạo sự phức tạp hóa cho các từ. Mỗi sub-layer đều residual connections và normalization để truyền thông tin và ổn định mạng.

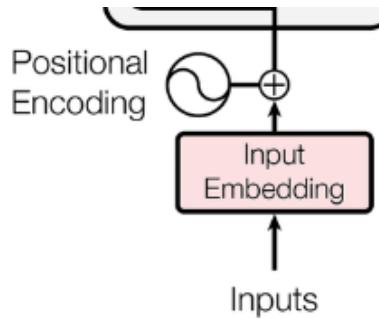


Hình 3: Khối Encoder-Decoder của kiến trúc Transformer

Kết quả của mỗi lớp encoding là biểu diễn mạnh mẽ của chuỗi đầu vào, cung cấp thông tin cho bộ decoder. Encoder đảm nhiệm tiền xử lý và biểu diễn hiệu quả, tạo cơ sở cho việc tạo dự đoán chính xác từ decoder.

2.2.1 Input embedding

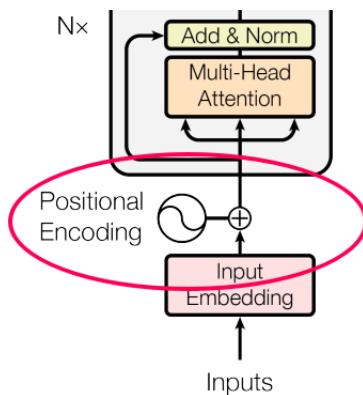
Trong Transformer, khối encoder đầu tiên nhận các vector embeddings từ các từ trong câu, với số chiều là 512. Khối encoder sau nhận đầu vào từ khối phía dưới để cải thiện biểu diễn câu.



Hình 4: Input embedding của kiến trúc Transformer

Việc tạo các vector embeddings này bao gồm sự kết hợp của vocabulary embeddings và positional embeddings.

Word Vocabulary Embedding là biểu diễn vector cho mỗi từ, được tạo ra từ các pre-model như word2vec, glove...



Hình 5: Positional Embedding của kiến trúc Transformer

Positional Embedding biểu diễn thứ tự của từng từ trong chuỗi. Chúng mang thông tin về vị trí và khoảng cách giữa các từ. Nếu không có thông tin vị trí, việc học biểu diễn cho câu sẽ không có sự khác biệt. Vị trí của từng từ rõ ràng mang ý nghĩa quan trọng, và việc thay đổi vị trí của từ cũng có thể thay đổi ý nghĩa của câu. Positional Embedding (PE) được tính bằng công thức:

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

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

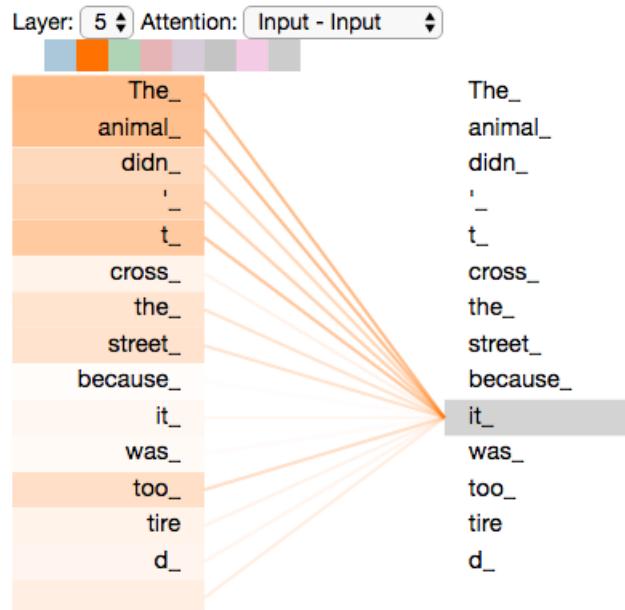
Trong đó d_{model} là số chiều của vector biểu diễn pos là vị trí của từng từ $(0, 1, 2, 3, \dots)$. i là chiều thứ i của vector $i \in 0, 1, 2, \dots, d_{model}$

Giải thích về sin/cosin:

- Học chú ý dựa trên vị trí tương đối dễ dàng: bằng cách sử dụng hàm sin và cosin để mã hóa vị trí, mô hình Transformer có thể dễ dàng học cách chú ý đến sự tương quan giữa các từ ở vị trí khác nhau trong chuỗi. Điều này là do PE_{pos+k} có thể biểu diễn bằng một hàm tuyến tính của PE_{pos} .
- Ngoài suy đến độ dài chuỗi: hàm sin và cosin giúp mô hình ngoại suy đến độ dài chuỗi vượt quá dữ liệu huấn luyện. Sau khi học cách biểu diễn vị trí tương đối, mô hình có khả năng dự đoán và chú ý đến các vị trí trong các chuỗi dài hơn, mà nó chưa từng thấy. Điều này cải thiện khả năng tổng quát hóa của mô hình trong các tình huống thực tế.

2.2.2 Self-Attention

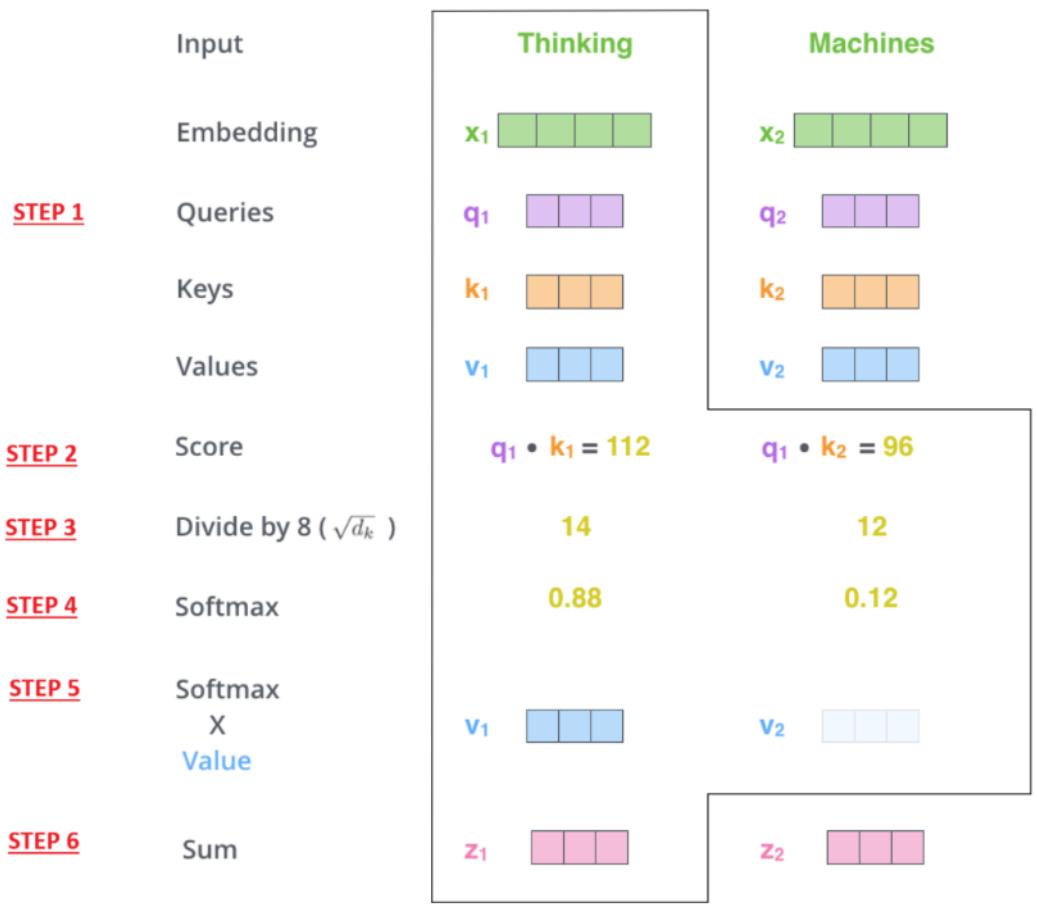
Self-Attention [14] là một khái niệm quan trọng trong kiến trúc Transformer. Đây là cách mà mô hình xử lý thông tin chuỗi bằng cách tập trung vào sự tương tác giữa các thành phần trong chuỗi. Nhờ vậy mà mô hình có khả năng tạo ra biểu diễn phức tạp và sâu hơn về mối quan hệ giữa các thành phần trong dữ liệu. Trong quá trình self-attention, mô hình có thể xem xét thông tin từ các phần khác của cùng một chuỗi, giúp cải thiện biểu diễn của mỗi thành phần riêng lẻ.



Hình 6: Cơ chế self-attention của kiến trúc Transformer

Cụ thể hơn, trong quá trình tự chú ý, mô hình tính toán sự tương quan giữa mỗi thành phần trong chuỗi với tất cả các thành phần khác. Điều này giúp mô hình hiểu được mức độ liên quan và tương tác giữa các thành phần. Khi xử lý một từ cụ thể, ví dụ như từ "it", mô hình có thể dựa vào thông tin từ các từ khác trong câu để tìm hiểu ngữ cảnh và liên kết ý nghĩa.

Quá trình này bao gồm các bước sau:



Hình 7: Chi tiết cơ chế self-attention của kiến trúc Transformer

- **Bước thứ nhất** là quá trình tính toán self-attention bắt đầu bằng việc tạo ra ba vector từ mỗi vector đầu vào của bộ mã hoá (trong trường hợp này, là việc nhúng từng từ). Đối với mỗi từ, ta tạo ra một vector truy vấn (Query), một vector khóa (Key), và một vector giá trị (Value). Những vector này được tạo ra thông qua việc nhân việc nhúng từ với ba ma trận sẽ được huấn luyện trong quá trình đào tạo.
- *Lưu ý rằng những vector mới này có số chiều nhỏ hơn so với vector embedding ban đầu. Số chiều của chúng là 64, trong khi vector embedding và các vector đầu vào/dầu ra của bộ encode có số chiều là 512. Việc giảm số chiều này là một quyết định về kiến trúc, nhằm làm cho tính toán multiheaded attention trở nên ổn định hơn.*
- **Bước thứ hai** là tính toán score. Ví dụ ta cần tính self-attention cho từ "Thinking", thì ta cần tính score cho mỗi từ trong câu input so với từ này. Score xác định mức độ chú ý của một từ so với các từ khác trong câu khi encode một từ ở vị trí cụ thể, được tính bằng cách lấy tích vô hướng của query vector của từ tương ứng đang được tính score.

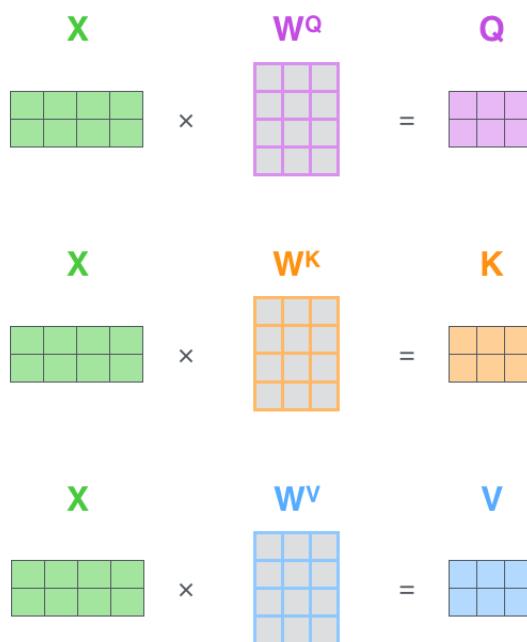
- **Bước thứ ba** là chia điểm số cho 8 (lấy căn bậc hai của số chiều của vector khóa, trong bài báo là 64).
- **Bước thứ tư** là đưa kết quả từ bước ba qua hàm softmax (hàm softmax chuẩn hóa các điểm số để chúng đều là số dương và tổng các điểm số bằng 1). Hàm này xác định mức độ mà mỗi từ sẽ được biểu thị tại vị trí đang xét.
- **Bước thứ năm** là nhân mỗi vector giá trị với softmax score. Ý tưởng ở đây là giữ nguyên giá trị của từ (hoặc các từ) mà chúng ta muốn tập trung vào, và làm giảm tầm quan trọng của các từ không liên quan (bằng cách nhân chúng với các số nhỏ như 0.001, ví dụ).
- **Bước thứ sáu** là tổng hợp các vector giá trị đã được trọng số. Điều này tạo ra đầu ra của lớp self-attention tại vị trí này (cho từ đầu tiên).

Matrix Calculation of Self-Attention

Trong thực tế các phép tính self-attention được thực hiện dưới dạng ma trận để tăng tốc độ xử lí:

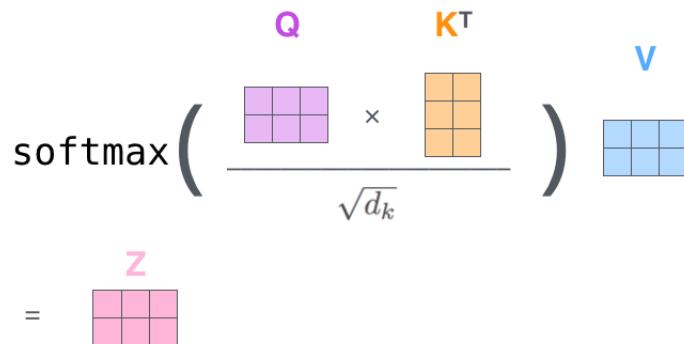
- **Đầu tiên** là tính toán các ma trận Query, Key và Value. Chúng ta thực hiện điều này bằng cách đóng gói embeddings của chúng ta vào một ma trận X, và nhân nó với các ma trận trọng số

$$(W_Q, W_K, W_V)$$



Hình 8: Ma trận trong self-attention của kiến trúc Transformer

- **Sau đó** vì đang tính toán với các ma trận, chúng ta có thể tóm gọn các bước từ hai đến sáu trong một công thức sau để tính toán đầu ra của lớp self-attention.

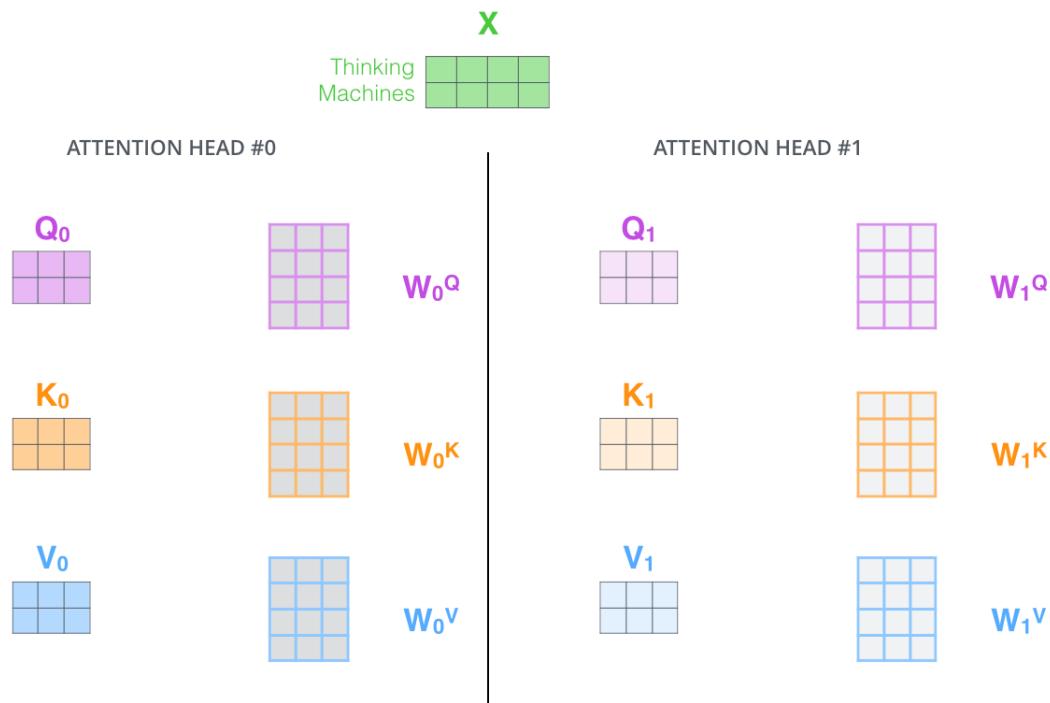


Hình 9: Hàm softmax trong self-attention của kiến trúc Transformer

Multi-Head Attention

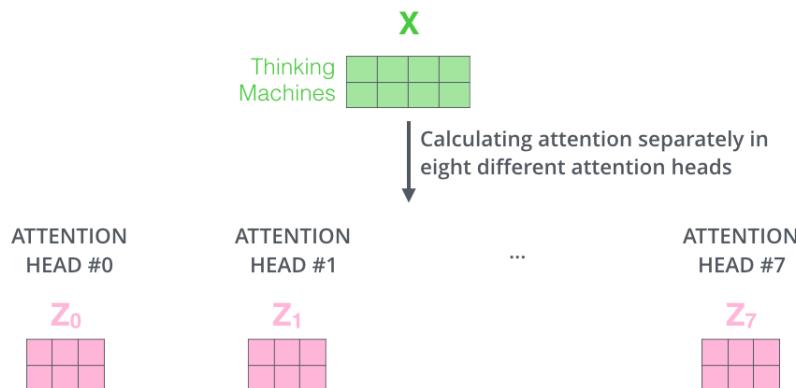
Kiến trúc transformer được thiết kế với 8 lớp self-attention kiến trúc giống hệt nhau nhưng trọng số của 3 ma trận Q , K , V khác nhau. Việc tính toán của 8 layer này được thực hiện song song. Các vector biểu diễn qua mỗi lớp self-attention sẽ được nối lại với nhau sau đó được nhân với một ma trận trọng số W_o để nén thông tin từ 8 vector (8 vector này cùng biểu diễn cho 1 từ) thành một vector duy nhất. Vector này sau đó đi qua một bước gọi là Add - Normalize trước khi đưa vào layer Feed Forward.

Ý nghĩa của cơ chế multi-head này là để tăng thêm phần chắc chắn trong việc quyết định thông tin nào cần khuếch đại, thông tin nào cần bỏ qua. Vì rằng 8 head sẽ cùng vote và đưa ra lựa chọn khách quan, đáng tin cậy hơn 1 head.



Hình 10: Multi-Head Attention của kiến trúc Transformer

Nếu thực hiện phép tính self-attention tương tự như đã trình bày ở trên, nhưng thực hiện 8 lần khác nhau với các ma trận trọng số khác nhau, chúng ta sẽ có 8 ma trận Z khác nhau.



Hình 11: Multi-Head Attention của kiến trúc Transformer

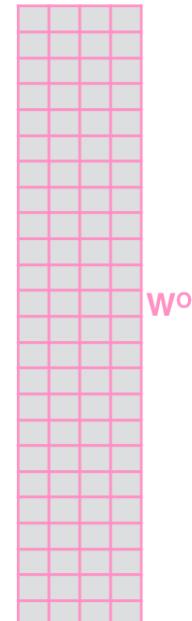
Bây giờ chúng ta sẽ gộp tám ma trận đầu ra ở trên thành một ma trận đơn bằng cách ghép các ma trận lại với nhau và sau đó nhân chúng với một ma trận trọng số bổ sung W_o .

1) Concatenate all the attention heads



2) Multiply with a weight matrix W^o that was trained jointly with the model

\times

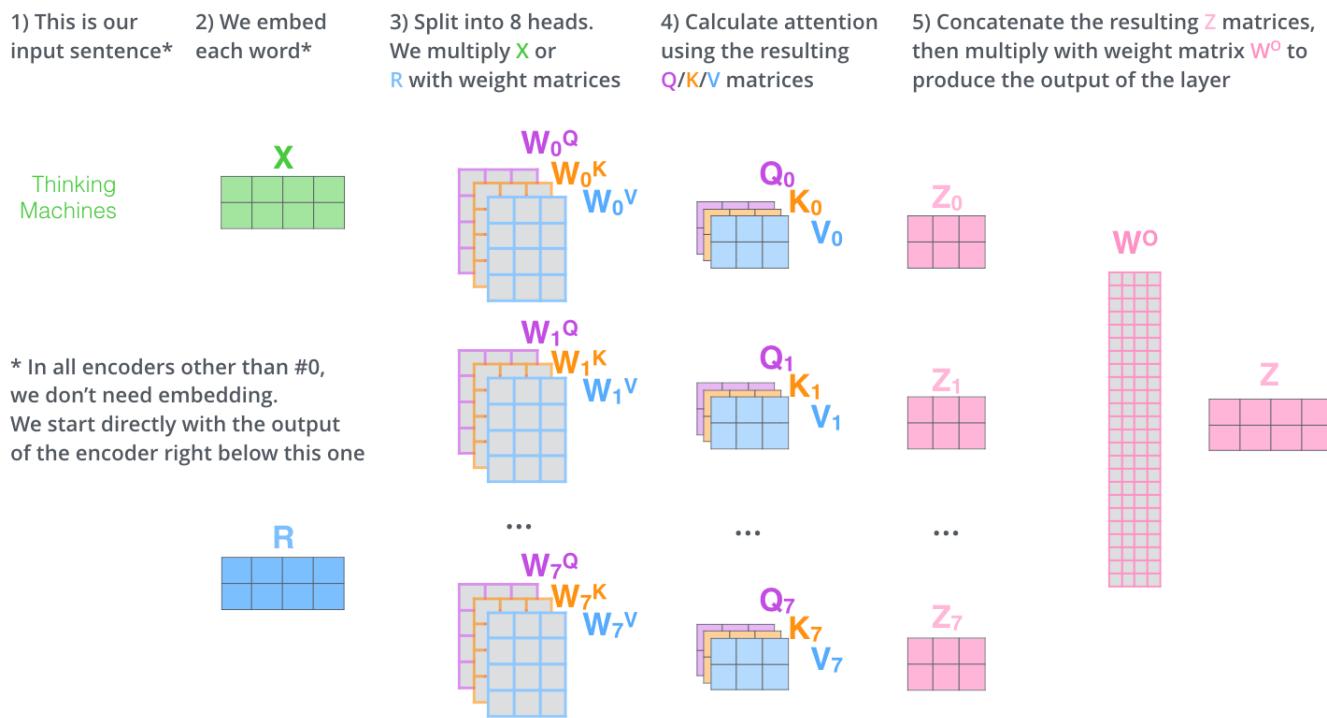


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

$$= \begin{matrix} Z \\ \hline \end{matrix}$$

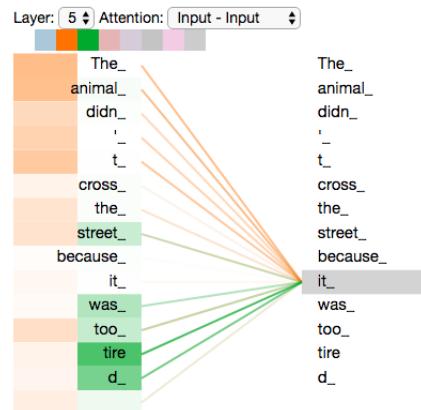
Hình 12: Multi-Head Attention của kiến trúc Transformer

Dây là hình ảnh tổng hợp các bước:



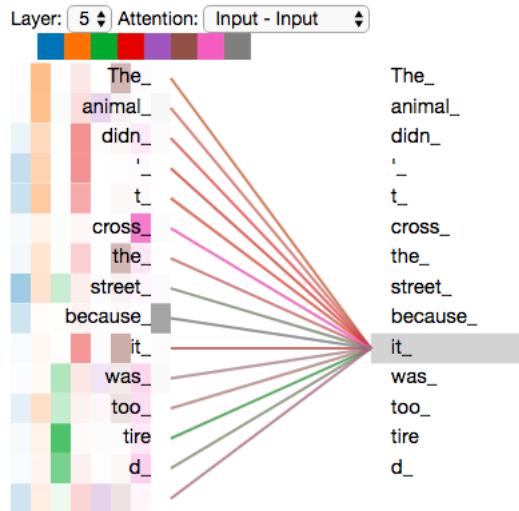
Hình 13: Các bước tính Multi-Head Attention của kiến trúc Transformer

Sau khi đã đề cập đến các attention heads, hãy xem lại ví dụ ở phần trước để thấy các attention heads khác nhau đang tập trung vào đâu khi thực hiện mã hóa từ "it":



Hình 14: Mô phỏng cơ chế Multi-Head Attention của kiến trúc Transformer

Tuy nhiên, nếu chúng ta thêm tất cả các attention heads vào hình ảnh, thì việc diễn giải có thể trở nên khó khăn hơn:



Hình 15: Mô phỏng cơ chế Multi-Head Attention của kiến trúc Transformer

2.2.3 Position-wise Feed-Forward Networks (FFN)

Các vector sau khi đi qua bước Add-Normalize (sẽ được nói ở mục sau) sẽ được gửi tới FFN. Lớp này bao gồm 2 tầng biến đổi thông tin và 1 hàm ReLU (các giá trị < 0 được gán lại = 0) ở giữa. Dropout với tỉ lệ 0.1 cũng được áp dụng ở lần biến đổi thứ nhất sau khi các vector qua hàm ReLU.

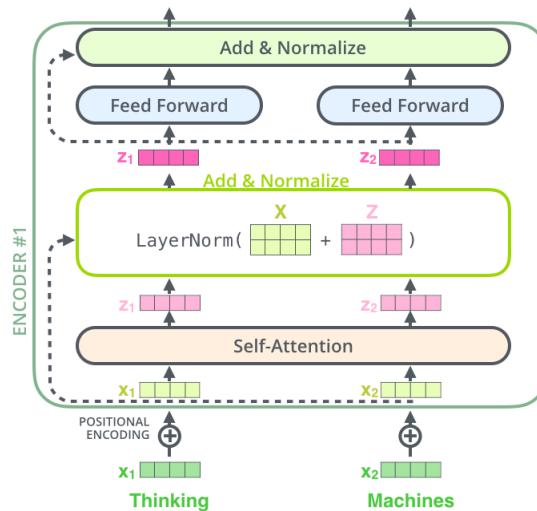
$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Lớp FFN đóng vai trò xử lý đầu ra từ một lớp self-attention giúp khớp theo cách tốt hơn cho đầu vào của lớp self-attention tiếp theo. Sau khi qua lớp FFN các vector cũng phải qua bước Add-Normalize trước khi đi vào khối encoder kế tiếp.

2.2.4 Add-Normalize

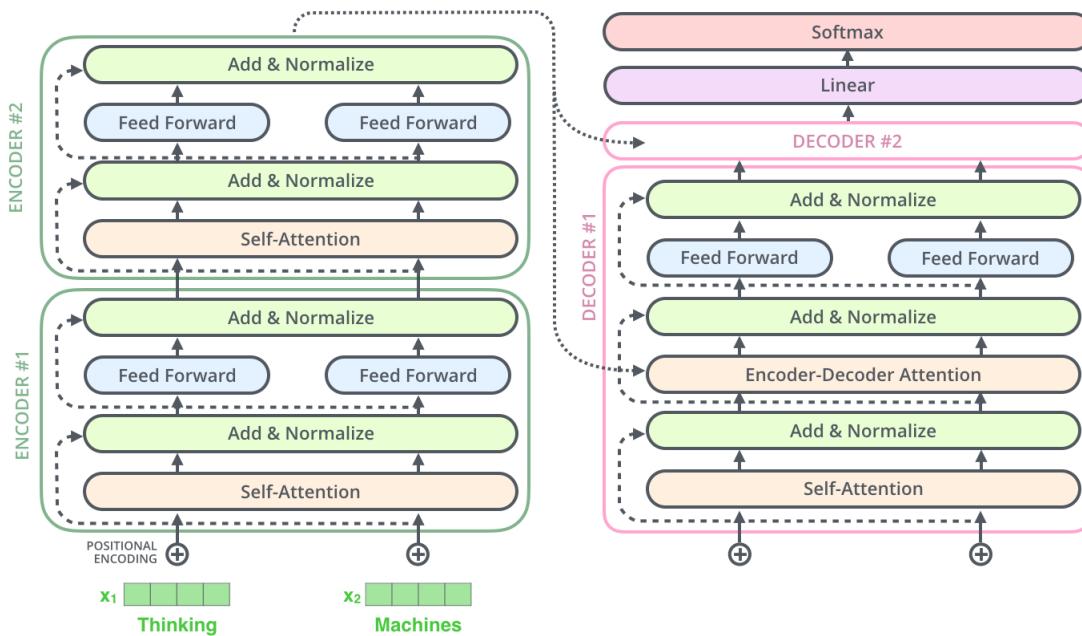
Kết nối residual là việc cộng đầu vào gốc với đầu ra của self-attention hoặc FFN, tạo ra đầu ra cuối cùng. Điều này giúp tránh vấn đề vanishing gradient, tăng cường hiệu suất và ổn định mạng. Trong Transformer, việc sử dụng kết nối residual cho mỗi sub-layer giúp cải thiện quá trình học và tạo ra biểu diễn câu tốt hơn.

Nếu muốn hình dung các vector và hoạt động layer-norm liên quan đến self attention, thì nó sẽ trông như sau:



Hình 16: Kết nối residual của kiến trúc Transformer

Điều này cũng áp dụng cho các sub-layers của bộ decoder. Nếu chúng ta tưởng tượng một Transformer với 2 bộ decoders và 2 bộ decoders xếp chồng lên nhau, thì nó sẽ trông gần như thế này:

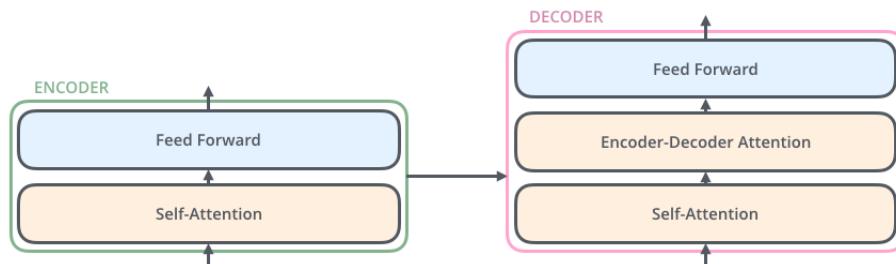


Hình 17: Kết nối residual của kiến trúc Transformer

Dầu ra từ lớp con (multi-head self-attention và feed forward) đi qua bước dropout với tỉ lệ 0.1 và sau đó được cộng thêm vào vector đầu vào ban đầu (trước khi biến đổi). Cuối cùng, kết quả được chuẩn hóa theo một công thức và chuyển vào lớp kế tiếp. Bước này nhằm bổ sung thông tin gốc, nguyên bản và tránh mất mát thông tin quá nhiều sau các biến đổi ở các lớp multi-head self-attention và feed forward.

2.3 Decoding

Cấu trúc chung: cũng giống như thành phần encoding, thành phần decoding trong paper cũng có cấu trúc gồm 6 decoder xếp chồng lên nhau. Một decoder cũng có cấu trúc gần như tương tự với một encoder, khác biệt ở chỗ là decoder sử dụng 2 lớp attention thay vì 1 như encoder.



Hình 18: Cấu trúc cơ bản của encoder và decoder

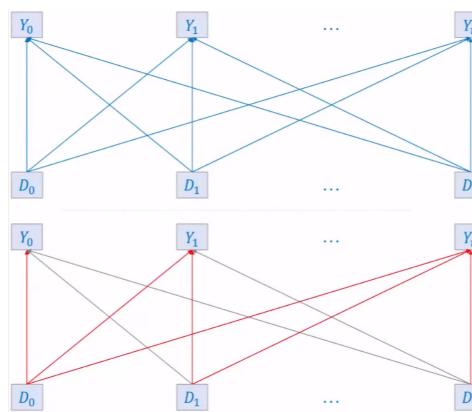
Những điểm khác biệt chính so với encoder:

- Áp dụng cơ chế masking cho các lớp self-attention.
- Encoder-decoder attention.

2.3.1 Cơ chế masking

Tổng quan: các lớp self-attention trong decoder có một chút khác biệt so với trong encoder đó là nó chỉ được phép "nhìn" các vị trí trước đó trong chuỗi đầu ra. Việc này được thực hiện bằng cách che đi các vị trí trong tương lai trước khi đến bước tính *softmax*.

Để dễ hình dung hơn thì hãy xem xét hình [10] bên dưới:



Hình 19: Trên: self-attention. Dưới: masked self-attention

So sánh: ở đây D_0, D_1, \dots, D_n là các từ đã được nhúng thành vector và Y_0, Y_1, \dots, Y_n là các vector từ đã được tổng hợp thông tin. Hình bên trên tương ứng với self-attention bình thường, tức là một từ sẽ tổng hợp thông tin với tất cả các từ còn lại, bao gồm cả chính nó. Còn hình bên dưới tương ứng với self-attention áp dụng masking, lúc này từ D_0 chỉ được tổng hợp thông tin của chính nó, từ D_1 chỉ được tổng hợp của D_0, D_1, \dots

Để cho dễ hiểu thì các vector Y (bình thường) $\rightarrow Y$ (có masking) trong đó α_{ij} là hệ số tương quan giữa từ i và từ j :

- $Y_0 = \alpha_{00}D_0 + \alpha_{01}D_1 + \dots + \alpha_{0n}D_n \rightarrow Y_0 = \alpha_{00}D_0 + 0 * D_1 + \dots + 0 * D_n$
- $Y_1 = \alpha_{10}D_0 + \alpha_{11}D_1 + \dots + \alpha_{1n}D_n \rightarrow Y_1 = \alpha_{10}D_0 + \alpha_{11} * D_1 + \dots + 0 * D_n$
- ...
- $Y_n = \alpha_{n0}D_0 + \alpha_{n1}D_1 + \dots + \alpha_{nn}D_n \rightarrow Y_n = \alpha_{n0}D_0 + \alpha_{n1} * D_1 + \dots + \alpha_{nn} * D_n$

Để có được kết quả như trên thì khi tính Y ta sẽ cộng thêm một ma trận M , công thức sẽ từ dạng bình thường là $Y = softmax(\frac{QK^T}{\sqrt{m}})V$ sang $Y = softmax(\frac{QK^T}{\sqrt{m}} + M)V$, với m là số chiều của vector K .

Ma trận M có dạng như sau:

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Thực hiện: [10]

- Đầu tiên: cũng giống trong encoder, lấy ma trận đầu vào X nhân cho các ma trận trọng số W_Q, W_K, W_V ra được các ma trận Q, K, V .

$$\begin{aligned} W_Q &= \begin{bmatrix} 0.4 & 0.3 \\ -0.1 & -0.1 \\ 0.2 & \end{bmatrix} & Q = XW_Q &= \begin{bmatrix} 1. & 3. & 2. \\ 6. & 2. & 1. \\ 5. & 8. & 4. \\ 7. & 3. & 4. \end{bmatrix} \begin{bmatrix} 0.4 & 0.3 \\ -0.1 & -0.1 \\ 0.2 & \end{bmatrix} = \begin{bmatrix} 0.5 & -0.2 \\ 2.4 & 1.5 \\ 2.0 & 0.3 \\ 3.3 & 1.4 \end{bmatrix} \\ W_K &= \begin{bmatrix} 0.1 & 0.2 \\ -0.3 & -0.4 \\ -0.1 & 0.2 \end{bmatrix} & K = XW_K &= \begin{bmatrix} 1. & 3. & 2. \\ 6. & 2. & 1. \\ 5. & 8. & 4. \\ 7. & 3. & 4. \end{bmatrix} \begin{bmatrix} 0.1 & 0.2 \\ -0.3 & -0.4 \\ -0.1 & 0.2 \end{bmatrix} = \begin{bmatrix} -1.0 & -0.6 \\ -0.1 & 0.6 \\ -2.3 & -1.4 \\ -0.6 & 1.0 \end{bmatrix} \\ W_V &= \begin{bmatrix} -0.2 & 0.1 \\ -0.4 & 0.2 \\ 0.4 & -0.6 \end{bmatrix} & V = XW_V &= \begin{bmatrix} 1. & 3. & 2. \\ 6. & 2. & 1. \\ 5. & 8. & 4. \\ 7. & 3. & 4. \end{bmatrix} \begin{bmatrix} -0.2 & 0.1 \\ -0.4 & 0.2 \\ 0.4 & -0.6 \end{bmatrix} = \begin{bmatrix} -0.6 & -0.5 \\ -1.6 & 0.4 \\ -2.6 & -0.3 \\ -1.0 & -1.1 \end{bmatrix} \\ W_O &= \begin{bmatrix} 0.1 & -0.1 & 0.6 \\ 0.9 & 0.3 & 0.1 \end{bmatrix} \\ X &= \begin{bmatrix} 1. & 3. & 2. \\ 6. & 2. & 1. \\ 5. & 8. & 4. \\ 7. & 3. & 4. \end{bmatrix} & M &= \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned}$$

Hình 20: Ví dụ cho masked self-attention

- Tiếp theo: tính $Y = softmax(\frac{QK^T}{\sqrt{m}} + M)V$. Lý do chọn ma trận M có dạng như trên là bởi vì sau khi tính $\frac{QK^T}{\sqrt{m}} + M$, ta sẽ có ma trận thỏa mãn yêu cầu masking:

$$\frac{QK^T}{\sqrt{m}} + M = \begin{bmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} + \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \alpha_{00} & -\infty & -\infty & -\infty \\ \alpha_{10} & \alpha_{11} & -\infty & -\infty \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & -\infty \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix}$$

$$\begin{aligned}
 A &= Y = \text{softmax} \left(\frac{QK^T}{\sqrt{m}} + M \right) V \\
 &= \text{softmax} \left(\begin{bmatrix} -0.26 & -\infty & -\infty & -\infty \\ -2.3 & 0.46 & -\infty & -\infty \\ -1.54 & -0.01 & -3.54 & -\infty \\ -2.92 & 0.36 & -6.75 & -0.41 \end{bmatrix} \right) \begin{bmatrix} -0.6 & -0.5 \\ -1.6 & 0.4 \\ -2.6 & -0.3 \\ -1.0 & -1.1 \end{bmatrix} \\
 &= \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.06 & 0.94 & 0.0 & 0.0 \\ 0.17 & 0.81 & 0.02 & 0.0 \\ 0.02 & 0.66 & 0.01 & 0.31 \end{bmatrix} \begin{bmatrix} -0.6 & -0.5 \\ -1.6 & 0.4 \\ -2.6 & -0.3 \\ -1.0 & -1.1 \end{bmatrix} = \begin{bmatrix} -0.6 & -0.5 \\ -0.54 & 0.34 \\ -1.45 & 0.22 \\ -1.39 & -0.08 \end{bmatrix} \\
 O &= AW_O = \begin{bmatrix} -0.6 & -0.5 \\ -0.54 & 0.34 \\ -1.45 & 0.22 \\ -1.39 & -0.08 \end{bmatrix} \begin{bmatrix} 0.1 & -0.1 & 0.6 \\ 0.9 & 0.3 & 0.1 \end{bmatrix} = \begin{bmatrix} -0.51 & -0.09 & -0.41 \\ 0.16 & 0.26 & -0.89 \\ 0.06 & 0.21 & -0.85 \\ -0.21 & 0.11 & -0.84 \end{bmatrix}
 \end{aligned}$$

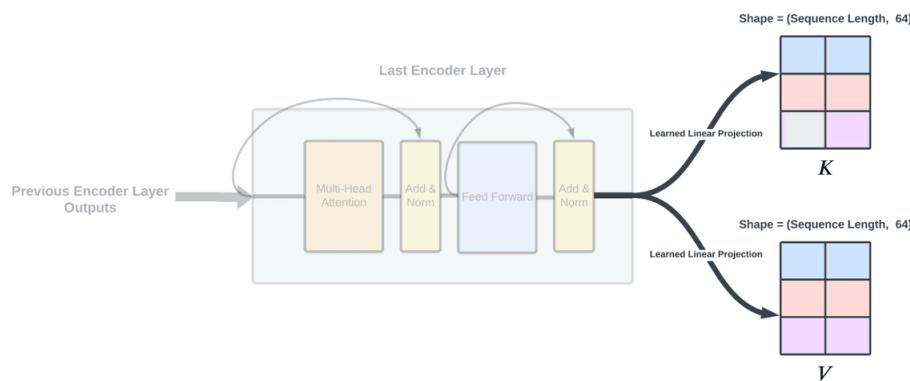
Hình 21: Ví dụ cho masked self-attention

- Sau đó: khi đưa vào hàm softmax, những giá trị $-\infty$ sẽ được chuyển về 0. Những bước cuối cùng cũng sẽ được thực hiện tương tự như encoder.

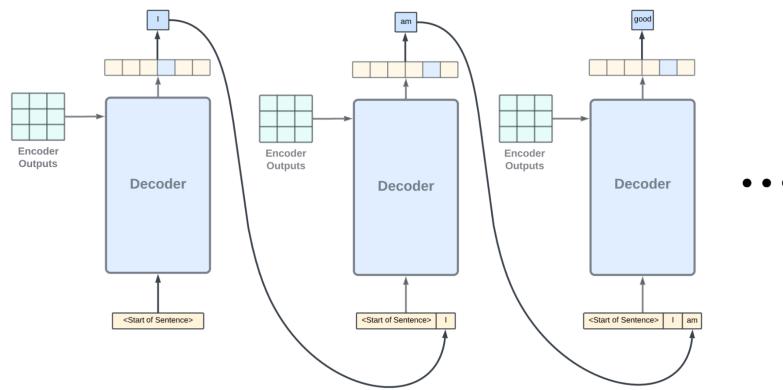
2.3.2 Encoder-decoder attention

Tổng quan: [1] [15] trước đó chúng ta đã biết cách hoạt động của các encoder, bây giờ ta sẽ tìm hiểu cách encoder và decoder hoạt động chung với nhau như thế nào. Encoder-decoder attention (hay còn gọi là cross attention) có nhiệm vụ giúp cho decoder tập trung vào các vị trí phù hợp trong chuỗi đầu vào.

Đầu tiên: các khối encoder xử lý chuỗi đầu vào, sau đó đầu ra của encoder trên cùng sẽ được chuyển thành một tập các vector attention K và V.

Hình 22: Đầu ra của bộ encoder gồm K và V

Tiếp theo: các khối decoder sẽ tính ma trận Q từ đầu vào của chính nó, nhưng ma trận K và V sẽ được lấy từ encoder. Mỗi bước trong decoding sẽ cho ra một phần tử từ chuỗi đầu ra (trong trường hợp này là dịch một câu tiếng Anh).



Hình 23: Quá trình decoding

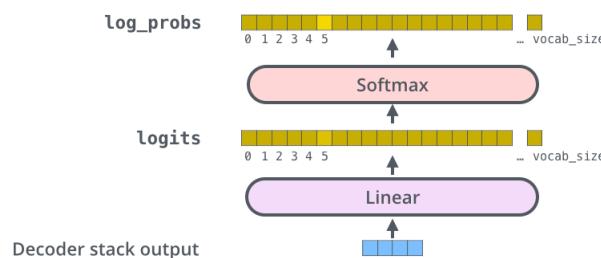
Bước này sẽ được lặp lại cho đến khi một kí tự đặc biệt (<end of sentence>) xuất hiện, báo hiệu rằng chuỗi đầu ra của decoder đã hoàn thành. Trong đó, đầu ra của mỗi bước sẽ được đưa xuống decoder nằm dưới cùng của bước tiếp theo, và các decoder cũng sẽ đẩy các kết quả của chúng cho các decoder ở phía trên giống như bên encoder. Cũng tương tự như đầu vào của encoder, ta cũng sẽ nhúng và thêm thông tin vị trí (positional embedding) cho đầu vào của decoder.

2.4 The final layer: Linear and Softmax

Chức năng: [1] đầu ra của decoder trên cùng là một vector số thực. Nhiệm vụ của lớp Linear và Softmax đó là biến đổi vector này thành phân phối xác suất cho dự đoán các từ tiếp theo.

Lớp Linear là một mạng neural fully connected sẽ chuyển vector đầu ra của bộ decoder thành một vector có kích thước lớn hơn rất nhiều gọi là vector logits. Giả sử mô hình của chúng ta biết 10,000 từ tiếng Anh khác nhau (gọi là "output vocabulary" với vocab_size là 10,000). Như vậy vector logits mà lớp Linear cho ra sẽ có 10,000 phần tử, với mỗi phần tử là điểm số của từ đó.

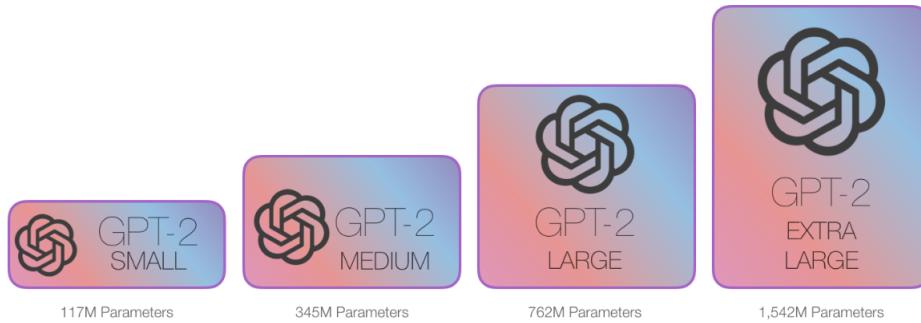
Lớp Softmax sau đó sẽ biến những điểm số đó thành những giá trị xác suất (tất cả là số dương, tổng bằng 1). Phần tử nào có xác suất cao nhất thì từ tương ứng với nó sẽ được mô hình dự đoán cho từ tiếp theo trong chuỗi.



Hình 24: Mô hình dự đoán đầu ra là từ tương ứng với xác suất cao nhất tại vị trí số 5

3 Generative Pre-trained Transformer 2

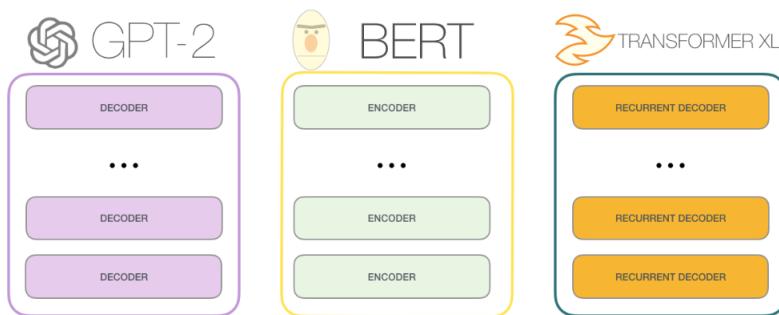
Language Modeling [11]: là tập hợp các kiến thức trước đó về một ngôn ngữ nhất định, các kiến thức này có thể là các kiến thức về từ vựng, ngữ pháp,... Mô hình ngôn ngữ có khả năng xem xét một phần của câu và dự đoán từ tiếp theo, một trong số đó là Generative Pre-trained Transformer 2 (GPT-2).



Hình 25: Tham số của các mô hình GPT-2

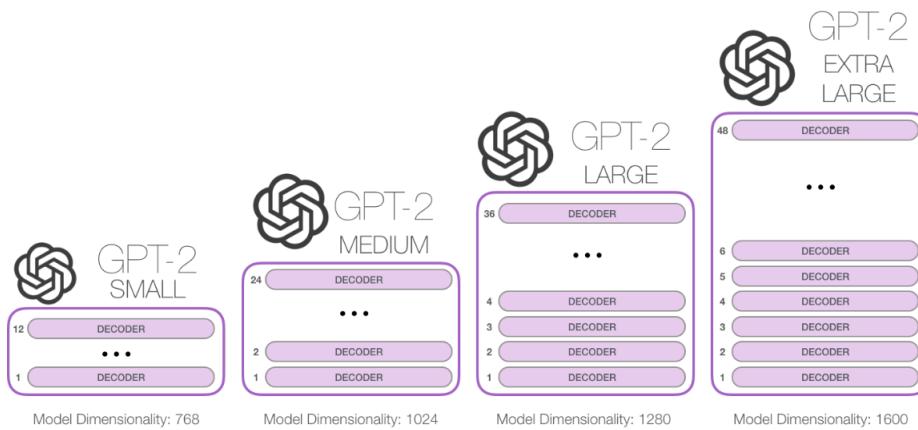
Variant Transformer: như đã biết, kiến trúc Transformer gốc có cả hai khối encoder và decoder và đã thành công trên mô hình dịch máy. Tuy vậy những biến thể dựa trên Transformer chỉ gồm duy nhất encoder hoặc decoder là có thể và dẫn đến sự xuất hiện của nhiều mô hình như: decoder-only (GPT-2 [12] [2], TransformerXL, XLNet) và encoder-only (BERT).

- *Decoder-only:* mô hình chỉ dựa trên duy nhất khối decoder và với cơ chế masking khi đưa vào một chuỗi input hoặc kí tự bắt đầu, mô hình sẽ tạo ra từ mới dựa trên các từ trước đó, từ sau khi được dự đoán này sẽ được gắn vào cuối chuỗi input để tiếp tục quá trình dự đoán cho đến khi gặp kí tự kết thúc. Cả quá trình này được gọi là tự hồi quy.
- *Encoder-only:* mô hình chỉ dựa trên duy nhất khối encoder và vì vậy không có masking, ngược lại mỗi từ trong câu sẽ học được cách chú ý đến tất cả các từ còn lại. Do đó mô hình này sẽ mất đi tự hồi quy, đổi lại nó có được khả năng kết hợp ngữ cảnh từ hai phía.



Hình 26: Transformer-based architectures

Phần tìm hiểu của nhóm em thực hiện dựa trên mô hình **GPT-2** với số chiều vector nhúng là **768**.

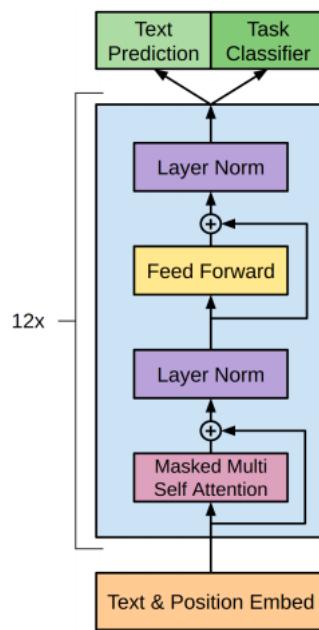


Hình 27: Số chiều và số khối decoder của các mô hình GPT-2

Kiến trúc chung:

- Về cơ bản GPT-2 có kiến trúc gồm 12 khối decoder được nối với nhau. Mỗi khối được train với trọng số độc lập và khối trên nhận input từ output của khối dưới.
- Bên trong mỗi khối decoder gồm các lớp: masked multi self attention, feed forward và norm. Vì không có khối encoder nên mô hình cũng bỏ đi lớp encode-decode attention. GPT-2 được xem như một mô hình pre-trained nên lớp trên cùng sẽ được fine tuning để phù hợp cho những task xác định như: phân loại, tóm tắt, dịch máy,....

Và vì tương đối giống khối decoder dựa trên Transformer nên ở đây sẽ trình bày những điểm chính khác biệt so với phần 1.



Hình 28: GPT-2 architecture

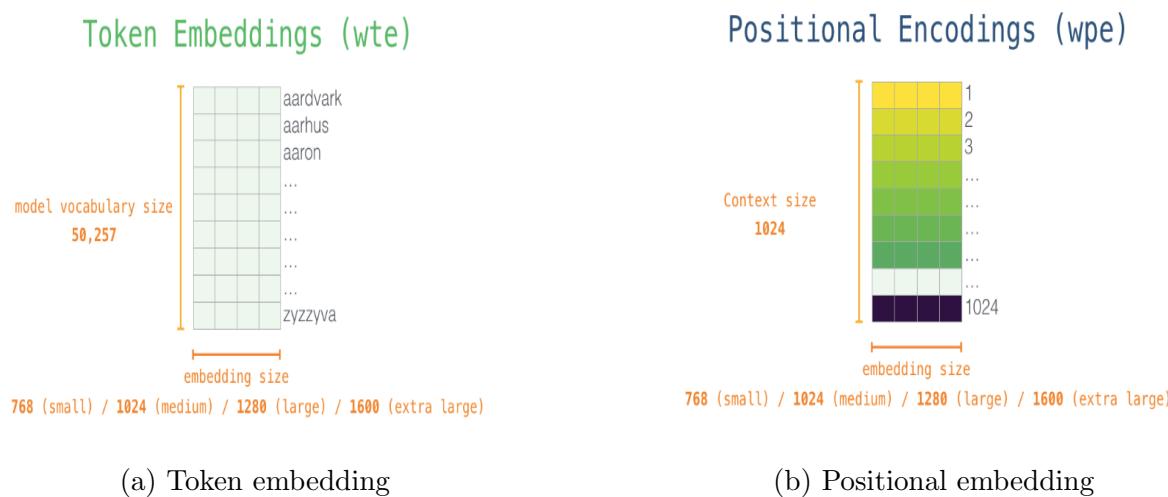
3.1 Token Embedding và Positional Embedding

Chức năng: nhúng các token (sử dụng Byte Pair Encoding) và positional để nhận biết được vị trí các từ trong câu.

Có hai cách để GPT-2 tạo ra văn bản:

- *Tạo mẫu không điều kiện:* chỉ đưa vào kí tự bắt đầu và GPT-2 sẽ tạo ra bất kì văn bản gì.
- *Tạo mẫu có điều kiện:* Dưa ra một chủ đề nhất định và có sử dụng chuyển đổi chuỗi đầu vào cho từng nhiệm vụ xác định.

Thực hiện: GPT-2 nhận đầu vào và thực hiện nhúng thành các token (token ở đây có thể hiểu là một hoặc một phần của từ) và cộng với vị trí cũng đã được nhúng trước khi đi qua lớp attention.



Hình 29: Input embedding

Từ vựng của tập train chứa 50.257 từ, chuỗi input có tối đa 1024 token (cần thêm padding sau cùng nếu chuỗi input chưa đạt đến 1024), vector nhúng có số chiều là 768.

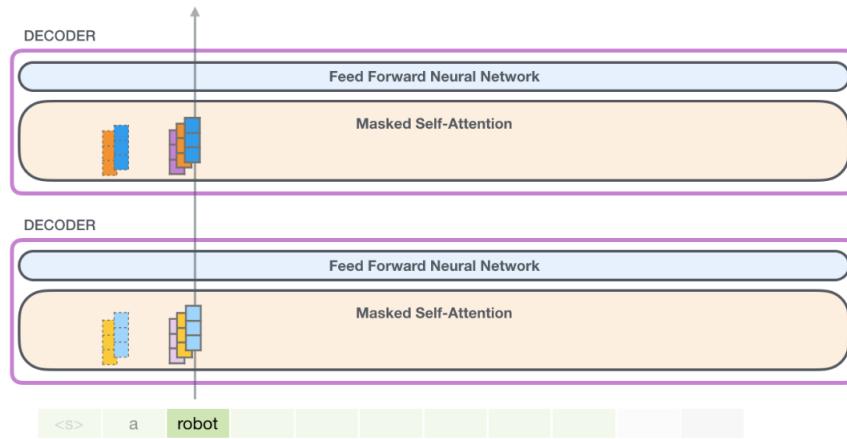
3.2 Masked Multi Self Attention

Chức năng: thực hiện xử lí các token để tìm mối liên hệ với các token trước đó thông qua điểm số, từ đó có thể trả về giá trị đã có sự chú ý đối với các token khác.

Ở lớp này có ba thành phần vector quan trọng:

- *Query (Q):* đại diện cho từ hiện tại, dùng để truy vấn số điểm đến các từ trước đó thông qua Key.
- *Key (K):* đại diện nhãn của token trong segment đang xét, là chìa khóa để tìm kiếm các từ liên quan với Query.
- *Value (V):* đại diện cho giá trị của từ.

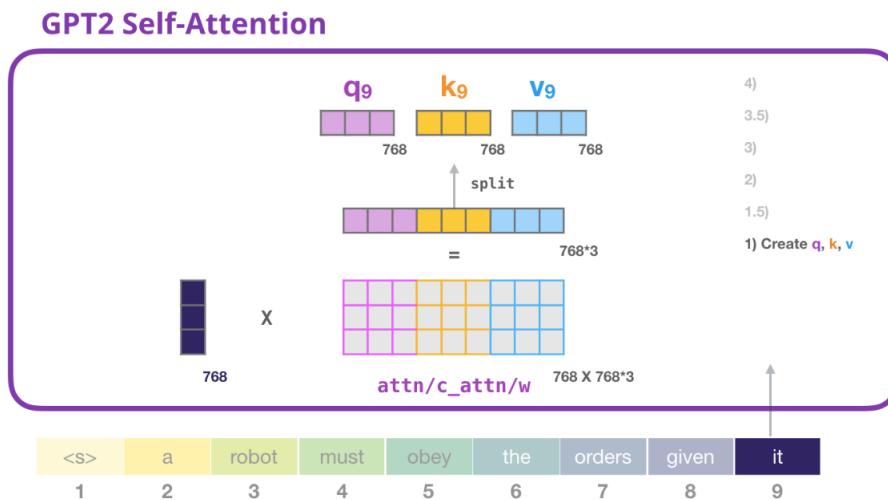
Cơ chế nhớ K và V của các token trước trong quá trình inference và evaluation: với cơ chế này khi tính toán trên token hiện tại, GPT-2 không cần tính toán lại K và V của các token trước đó mà chỉ cần sử dụng lại giá trị đã được lưu trong lớp attention.



Hình 30: Khi tính toán trên từ "robot", attention không cần tính lại K và V của từ "a" mà chỉ thực hiện tính QKV của từ "robot"

Quá trình tính toán self-attention (ví dụ xử lí từ "it" trong câu "<s> a robot must obey the order given it." và giả sử xem mỗi một từ là 1 token).

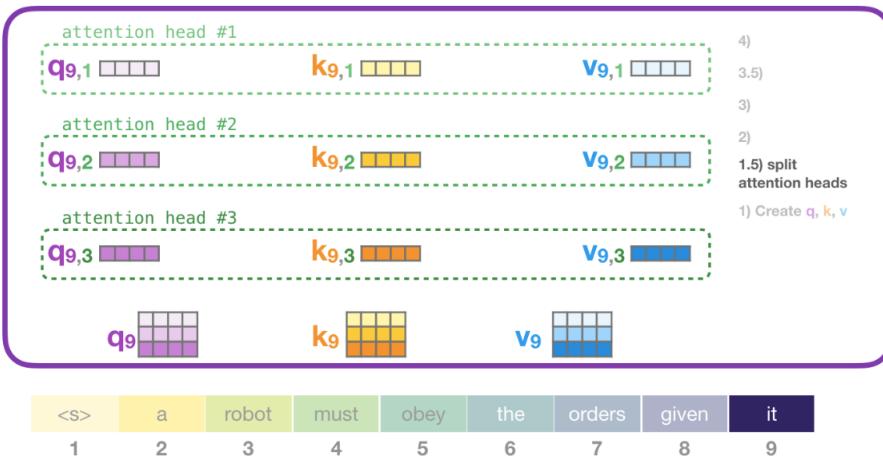
- *Bước 1:* Nhân vector input 768 chiều với ma trận trọng số $768 \times 768 \times 3$, được ma trận 768×3 , kết quả được split ra cho q_9 , k_9 và v_9 .



Hình 31: Step 1: Tạo q, k và v

- Bước 1.5: Tách vector q_9 , k_9 và v_9 (kích cỡ 768) ra 12 head (mỗi head có 64 chiều).

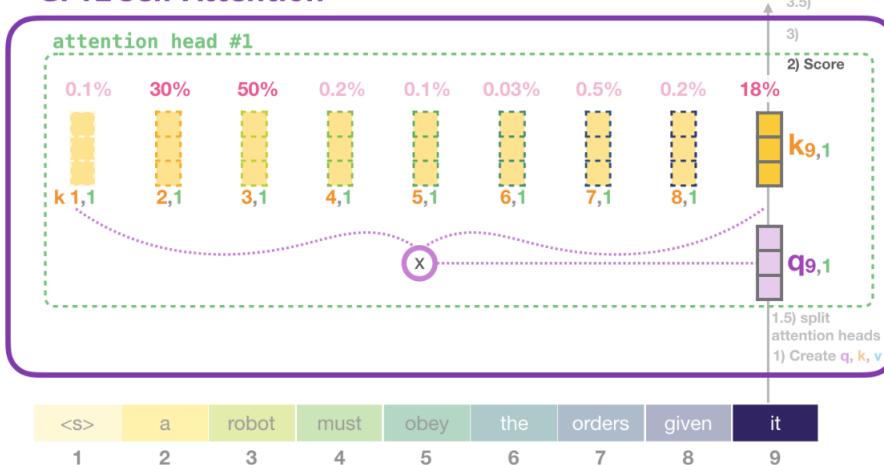
GPT2 Self-Attention



Hình 32: Step 1.5: Tách thành các attention head

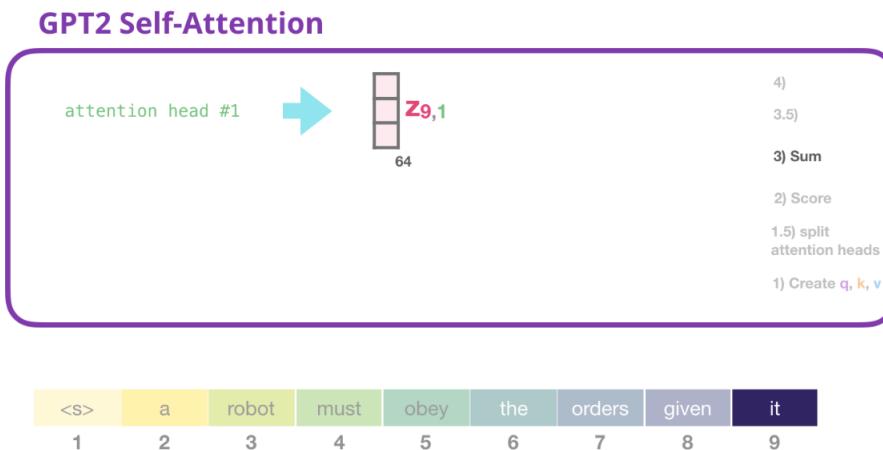
- Bước 2: Tính score cho 8 từ đứng trước từ "it".

GPT2 Self-Attention



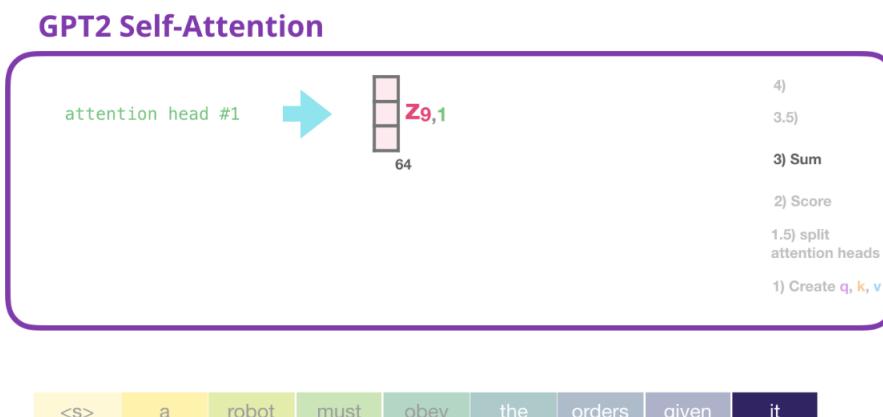
Hình 33: Step 2: Tính score trên attention head #1

- *Bước 3:* Tính tổng score cho từ "it".



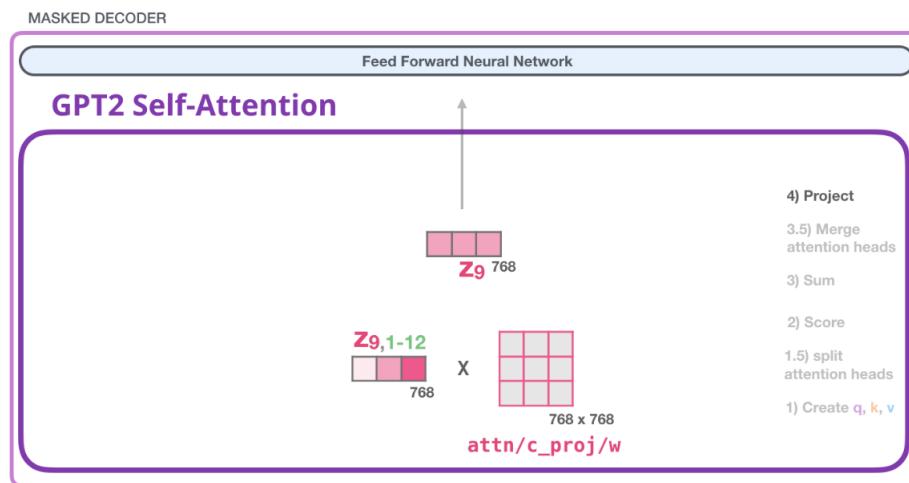
Hình 34: Step 3: Tính tổng score trên attention head #1

- *Bước 3.5:* Gộp các attention head (64 chiều) lại về độ dài cũ.



Hình 35: Step 3.5: Gộp các attention head

- Bước 4:* Chiếu vector sau khi gộp thông qua ma trận 768×768 , được vector output và cũng là input cho lớp feed forward. Mục đích của quá trình này để tìm ảnh xạ tốt nhất cho kết quả self-attention để mạng feed forward có thể xử lý.



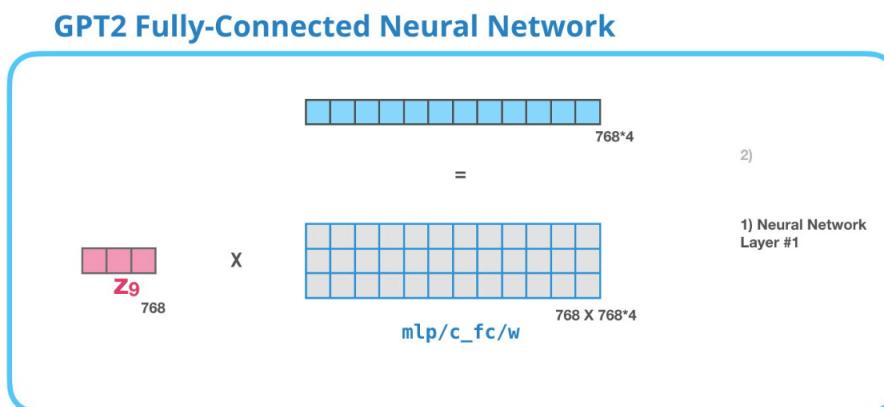
Hình 36: Step 4: Chiếu về 768 chiều

3.3 Fully Connected Neural Network

Chức năng: xử lý đầu ra từ một lớp chú ý để phù hợp hơn với đầu vào cho lớp chú ý tiếp theo, được thực hiện song song cho các token vì không có bất kì sự chú ý nào đến các token khác.

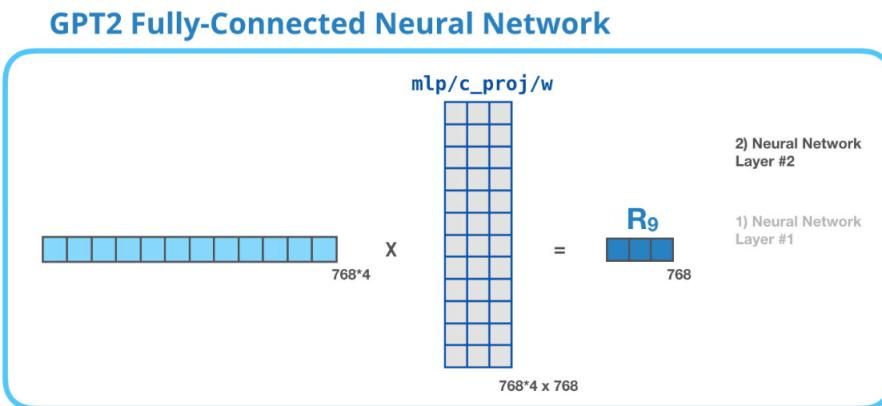
Ở lớp này, có thể được chia ra thành hay lớp con:

- Neural network #1:* Tạo ra $768 \times 4 = 3072$ output là neural bằng cách nhân vector đầu ra của self attention cho ma trận có kích thước $768 \times 768 \times 4$. Với kích cỡ 3072 sẽ cung cấp cho mô hình đủ khả năng biểu diễn để xử lý các nhiệm vụ được giao.



Hình 37: Neural network #1

- *Neural network #2 (projecting)*: Thực hiện chiếu kích cỡ từ 3072 về lại 768 cũng chính là kích cỡ mà input của khối decoder phía trên cần. Quá trình projecting này được thực hiện bằng cách nhân các neural input 768×4 với ma trận $768 \times 4 \times 768$.

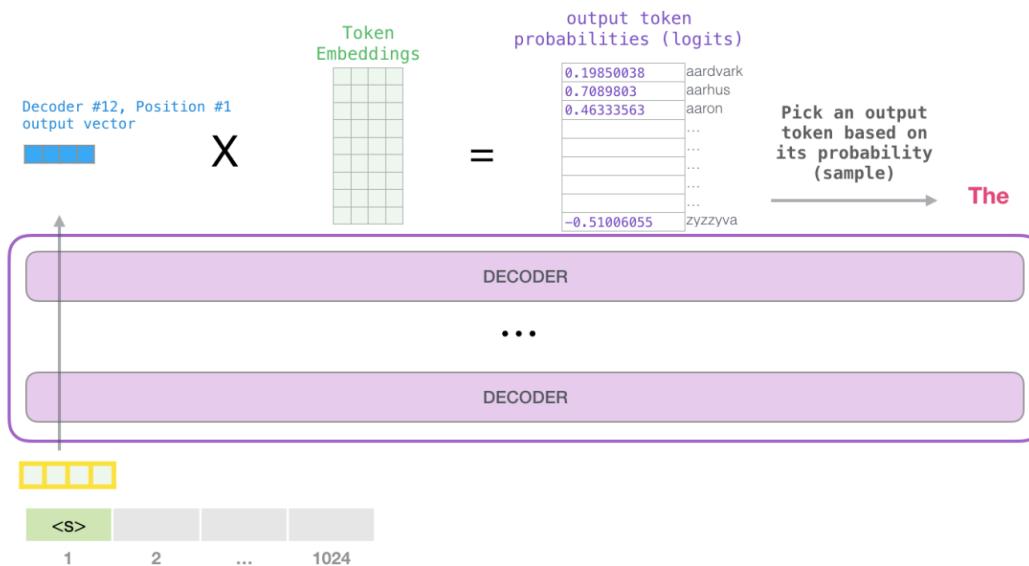


Hình 38: Neural network #2 (projecting)

3.4 Output và Inference

Chức năng: chuyển các vector token về thành điểm số dưới dạng xác suất và dùng các phương pháp suy luận để tạo token kế tiếp.

Thực hiện: bằng cách nhân vector output sau khi qua 12 khối decoder với ma trận token embedding sẽ tạo ra vector xác suất của 50.257 từ, xác suất này được chuẩn hóa về dạng có tổng bằng 1. Từ đó có thể chọn từ kế tiếp với xác suất cao nhất hoặc với phương pháp suy luận cho kết quả tốt hơn là sử dụng top-k=40 (lấy mẫu theo phân phối của 40 từ có xác suất cao nhất).

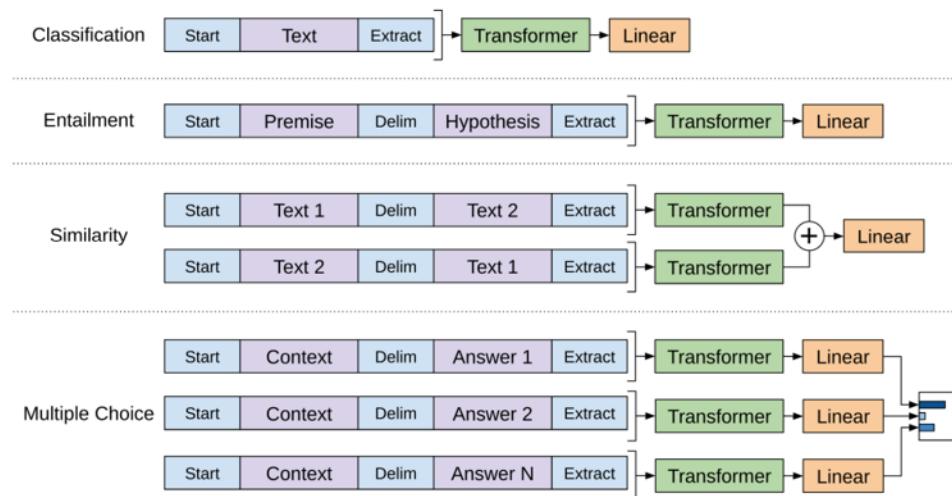


Hình 39: Model output

Tuy vậy, vì GPT-2 bản chất là pre-trained model nên sẽ có nhiều phương pháp tính output và suy luận cho từng tác vụ cụ thể.

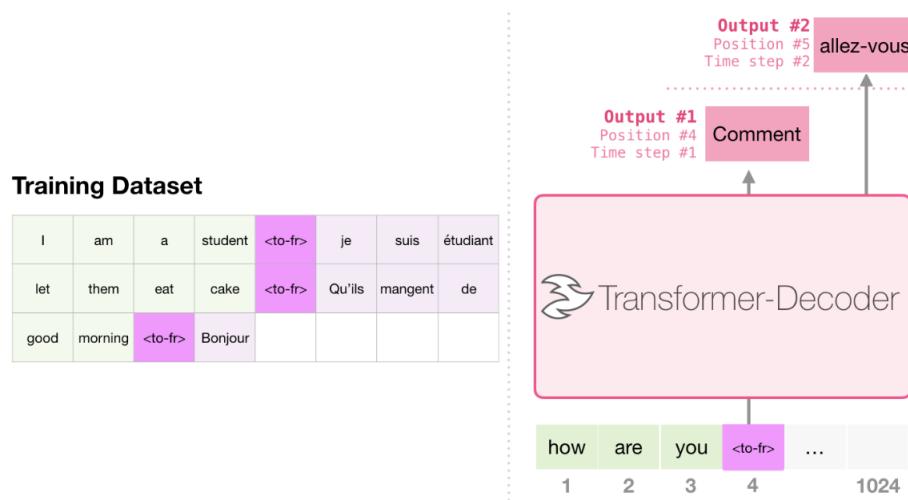
3.5 Beyond Language Modeling

Tiềm năng cho các nhiệm vụ ngoài mô hình ngôn ngữ: là một mô hình pre-trained dựa trên học không giám sát, GPT-2 có khả năng sử dụng cho các tác vụ khác nhau như: dịch máy, tóm tắt, tạo nhạc,... Bằng cách thực hiện chuyển đổi input trước khi vào khối decoder và transfer learning (cụ thể là fine tuning) trên tập dữ liệu có gắn nhãn (học có giám sát) cho các tác vụ cụ thể.



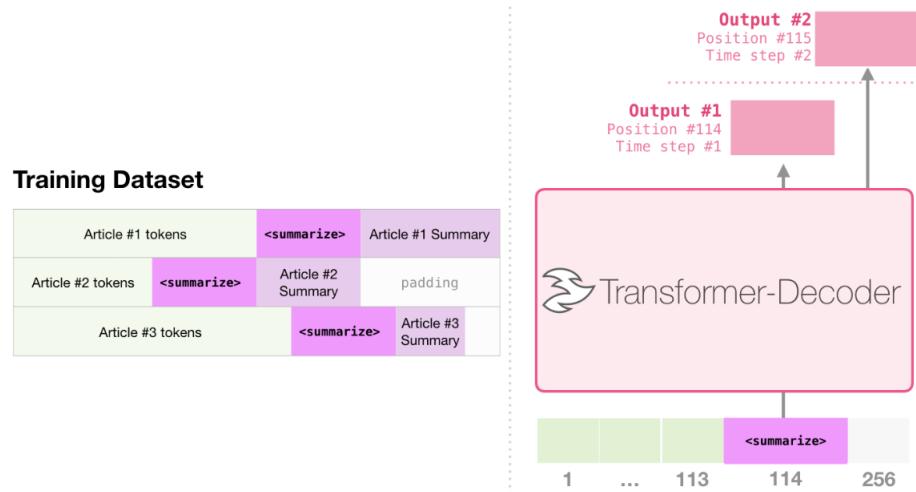
Hình 40: GPT-2 dùng cho các tác vụ cụ thể

Dịch máy: thực hiện chuyển đổi input bằng cách gắn văn bản nguồn với lệnh chỉ ngôn ngữ dịch, phần văn bản sau khi dịch sẽ được GPT-2 tự động tạo ra dựa trên trọng số đã học và mối liên hệ giữa các từ trong câu nguồn.



Hình 41: Machine translation

Tóm tắt: tập dữ liệu lấy từ wiki web, phần mở đầu của các trang wikipedia chính là label (summary), và phần training được đưa vào đầu chuỗi input chính là nội dung nằm dưới summary (article).



Hình 42: Summarization

Tạo nhạc: âm nhạc về cơ bản cũng có các thuộc tính (nốt nhạc, thời gian giữ, lực nhấn,...) vì vậy có thể mã hóa theo các quy tắc và tạo thành một chuỗi, lúc này âm nhạc sẽ giống như văn bản và vì vậy GPT-2 có thể dựa trên sự chú ý để dự đoán các sự kiện tiếp theo sẽ diễn ra.

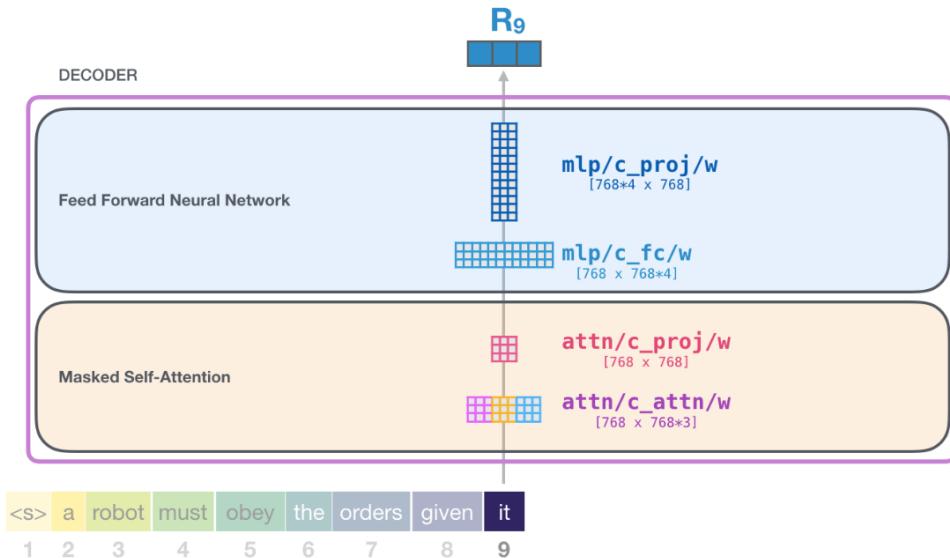
3.6 Recap

Các vấn đề khác:

- Kiến trúc small GPT-2 có khoảng 117 triệu tham số, tính trên các tham số nhúng, tích chập, chuẩn hóa của 12 khối decoder.
- Training mode: quá trình training được thực hiện trên chuỗi input dài, xử lý nhiều token tại cùng một thời điểm (song song) kết hợp với batch size là 512 giúp tăng tốc tính toán.
- Inference and evaluation mode: xử lý một token tại một thời điểm và batch size là 1.

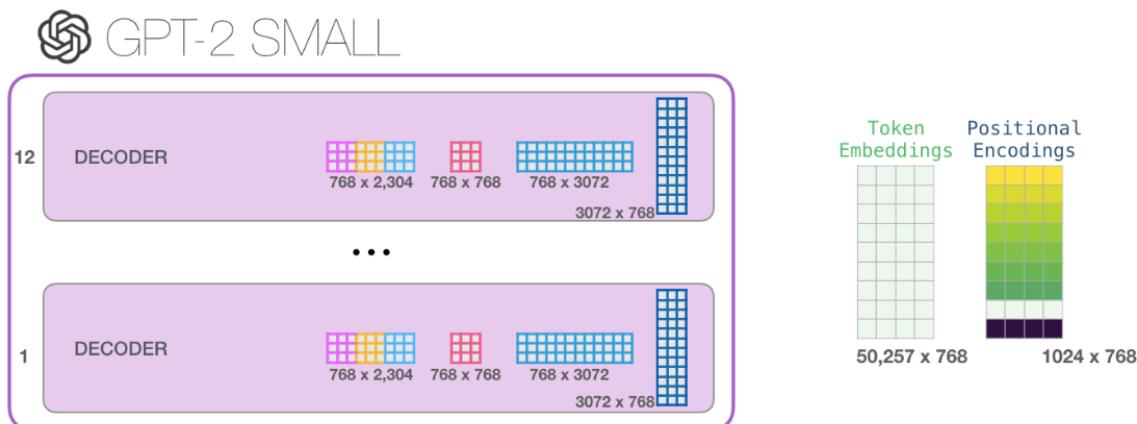
Toàn bộ các trọng số và quá trình của GPT-2 có thể được tóm tắt lại như sau:

- Vector nhúng input khi đi qua các ma trận trọng số của các sub-layer và output.



Hình 43: Các lớp và trọng số trong một khối decoder

- Toàn bộ kiến trúc GPT2 small với token embedding, positional encoding và 12 khối decoder.



Hình 44: Toàn bộ kiến trúc small GPT-2

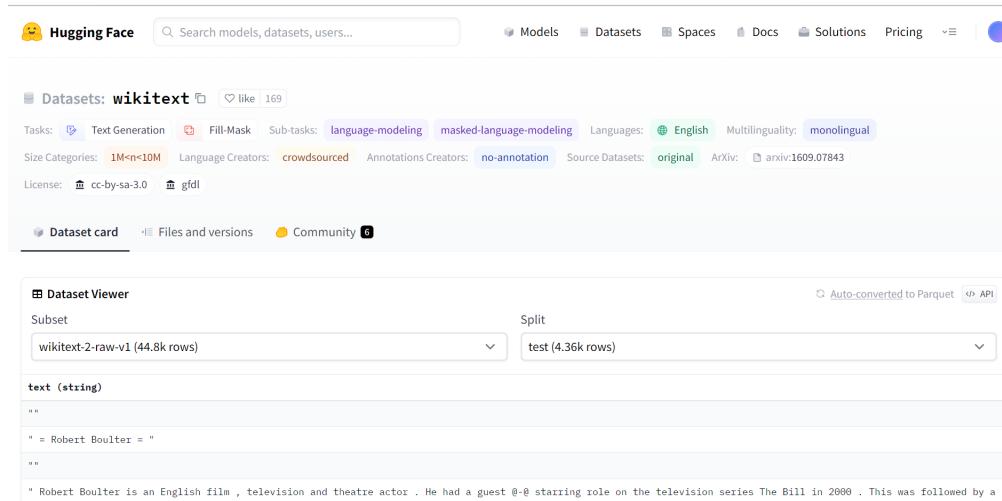
4 Fine tune GPT-2

4.1 Dataset

wikitext là một tập dữ liệu được sử dụng trong lĩnh vực xử lý ngôn ngữ tự nhiên và học máy. Tập dữ liệu này được tạo ra bằng cách trích xuất văn bản từ bài viết Wikipedia và được sử dụng để huấn luyện và đánh giá mô hình xử lý ngôn ngữ tự nhiên, chẳng hạn như mô hình dự đoán từ tiếp theo (language modeling), sinh văn bản tự động (text generation), phân loại văn bản, và nhiều ứng dụng khác.

Tập dữ liệu được chia thành ba phiên bản khác nhau: wikitext-2, wikitext-103 và wikitext-2019. Số sau tên "wikitext" thể hiện số lượng bài viết Wikipedia đã được sử dụng để tạo ra tập dữ liệu. Dưới đây là một số thông tin về các phiên bản chính của tập dữ liệu "wikitext":

- **wikitext-2:** phiên bản nhỏ hơn, bao gồm khoảng 2 triệu từ từ các bài viết Wikipedia. Phiên bản này thường được sử dụng để kiểm tra và huấn luyện nhanh chóng các mô hình xử lý ngôn ngữ tự nhiên.
- **wikitext-103:** phiên bản lớn hơn, bao gồm khoảng 103 triệu từ từ Wikipedia. Đây là tập dữ liệu thường được sử dụng để huấn luyện các mô hình mạnh hơn như các biến thể của mạng nơ-ron biểu diễn ngôn ngữ (RNN) và Transformer.
- **wikitext-2019:** phiên bản gần đây của tập dữ liệu, bao gồm khoảng 2,5 tỷ từ từ Wikipedia và được cập nhật đến năm 2019. Phiên bản này có quy mô lớn và thường được sử dụng để huấn luyện các mô hình mạnh mẽ.



Hình 45: Wikitext-2-raw-v1

Do giới hạn về thời gian và tài nguyên, nên việc train lại gpt-2 sẽ thực hiện trên tập dữ liệu **wikitext-2-raw-v1** và dữ liệu này cũng có sẵn từ [huggingface \[9\]](#) nên ở phần demo nhóm em sẽ **lấy trực tiếp bằng API** (*ngoài ra dữ liệu cục bộ cũng được nộp đính kèm*). Thông tin về tập dữ liệu:

- **Nguồn gốc:** tập dữ liệu "wikitext-2-raw-v1" được tạo ra bằng cách trích xuất nội dung từ các bài viết trên Wikipedia, không qua quá trình tiền xử lý hoặc xử lý bổ sung nào.
- **Số lượng dữ liệu:** Số lượng từ trong tập dữ liệu là khoảng 2 triệu từ (44836 mẫu).
- **Trường dữ liệu:** được biểu diễn bằng một trường dữ liệu duy nhất có tên là "text". Đây là trường dữ liệu chứa các chuỗi văn bản, là nơi mà thông tin ngôn ngữ được lưu trữ. Mỗi mẫu trong tập dữ liệu là một chuỗi văn bản, trong đó định dạng và thứ tự của các từ và ký tự đóng vai trò quan trọng để phân biệt giữa các bài viết và các phần khác nhau của cùng một bài viết.
- **Phân chia tập dữ liệu:** tập dữ liệu được chia thành ba phần chính để sử dụng cho việc huấn luyện, kiểm tra và đánh giá mô hình: train set(36718 mẫu), test set (4358 mẫu) và validation set (3760 mẫu).

```
text (string)
"""
" = Robert Boulter =
"""

" Robert Boulter is an English film , television and theatre actor . He had a guest @-@ starring role on the television series The Bill in 2000 . This was followed by a
starring role in the play Herons written by Simon Stephens , which was performed in 2001 at the Royal Court Theatre . He had a guest role in the television series Judg...
" In 2006 , Boulter starred alongside Whishaw in the play Citizenship written by Mark Ravenhill . He appeared on a 2006 episode of the television series , Doctors ,
followed by a role in the 2007 theatre production of How to Curse directed by Josie Rourke . How to Curse was performed at Bush Theatre in the London Borough of...
"""

" == Career == "
"""

"
```

Hình 46: Cấu trúc dữ liệu

- **Biểu diễn dữ liệu:** dữ liệu được biểu diễn thông qua các thành phần sau:

- **Tiêu đề:** một dòng chứa tiêu đề của bài viết hoặc phần của bài viết, thường được đặt trong dấu "==" (ví dụ: "= Title =").
- **Các phần nhỏ:** các phần con của bài viết, thường được đặt trong các dấu "=" để phân chia (ví dụ: " == Section == " hoặc " == Subsection == ").
- **Nội dung:** các đoạn văn bản chứa nội dung của bài viết hoặc phần của bài viết.

Các phần này thường được tách biệt bằng dòng trống, tức là hai dòng trống được sử dụng để phân chia giữa các phần khác nhau của bài viết.

4.2 Fine-tune model

Causal language modeling: là model dự đoán token kế tiếp trong câu và label là input khi được shift phải. Model sẽ dựa trên các token phía trước (các token phía sau sẽ được masked để tránh chú ý) để dự đoán một token kế tiếp sau đó.

Phần demo code nhóm em thực hiện:

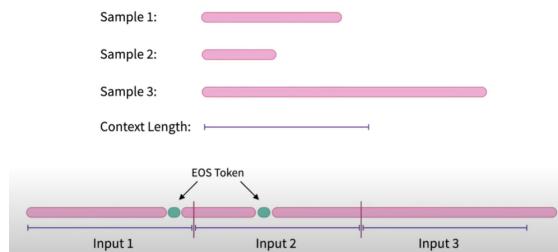
- Lấy pre-trained gpt2 từ thư viện *transformers*.
- Xử lý dữ liệu đầu vào.
- Thiết lập các hyperparameters và vòng lặp training sử dụng thư viện *transformers* và *torch*.
- Re-train gpt2 trên dữ liệu đã xử lý và ứng dụng vào causal language modeling.
- Inference và lưu model lên hub (xem model, chi tiết kết quả và lấy API ở [đây](#) [13] (*ngoài ra thư mục chứa model weights cũng được lưu trên drive*))

4.2.1 Pre process

Cả ba tập dữ liệu train, validation và test đều được xử lý như nhau. Kích thước chuỗi token input có giới hạn là 128 (GPT-2 phiên bản nhỏ nhất có độ dài 1024, trong phần demo có kích thước nhỏ hơn vì giới hạn dung lượng của GPU).

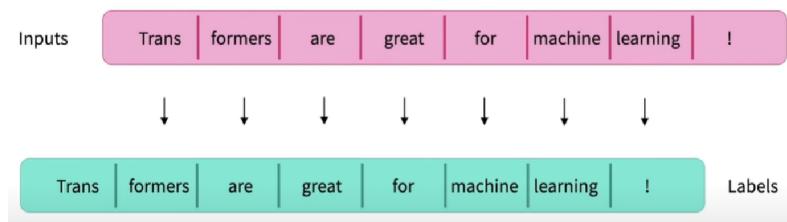
Xử lý input: dữ liệu bao gồm tiêu đề và nội dung của các article và gồm nhiều chuỗi rỗng, một số chuỗi rất ngắn, nhưng cũng có nhiều chuỗi dài vì vậy nếu cắt ở một độ dài nhất định hay bỏ những chuỗi có độ dài ngắn hơn sẽ gây mất mát nhiều thông tin; thay vào đó dữ liệu sẽ được xử lý bằng cách (ý tưởng tiền xử lý tham khảo từ [4] [6]):

- Tải bộ dữ liệu dùng *datasets* API.
- Gọi *tokenizer* từ thư viện để mã hóa tất cả các chuỗi đầu vào, tạo ra output là chuỗi token đã được mã hóa và attention mask (attention mask cho các token bắt đầu, kết thúc và padding sẽ là 0, còn lại là 1; vì khi tính mất mát, hàm loss sẽ không cần quan tâm đến các token này).
- Nối tất cả các chuỗi token theo từng batch là 1000, sau đó cắt phần dư (không chia hết cho 128), chuỗi token mới sau khi cắt (với độ dài $128*x$) sẽ được chia ra thành chuỗi token độ dài 128 để làm input mới cho model. Mất mát thông tin do phần cắt đi phần dư sẽ không nhiều vì với kích thước batch 1000 và mỗi chuỗi sẽ có nhiều token, việc bỏ đi ít hơn 128 token là không đáng kể.



Hình 47: Nối các chuỗi token và cắt bỏ phần dư

- Labels cũng được tạo bằng cách copy chuỗi token input, vì đối với mô hình ngôn ngữ nhân quả label chính là input khi được shift phải.



Hình 48: Label là input khi shift phải

Kết quả sau xử lý: sau quá trình tiền xử lý này số mẫu input (token) sẽ giảm đi khoảng một nửa so với mẫu input dạng chuỗi, vì hầu hết các chuỗi ngắn và chuỗi rỗng sẽ được nối lại. Nhưng vẫn giữ được hầu hết thông tin.

input_ids	attention_mask	input_ids	attention_mask	labels
0	0	0	128	128
1	9	1	128	128
2	0	2	128	128
3	166	3	128	128
4	107	4	128	128
...
36713	160	18661	128	128
36714	173	18662	128	128
36715	194	18663	128	128
36716	71	18664	128	128
36717	0	18665	128	128
36718 rows × 2 columns		18666 rows × 3 columns		

(a) Các chuỗi token có độ dài chênh lệch

(b) Sau khi tiền xử lý, các chuỗi token có cùng độ dài

Hình 49: Nối các chuỗi token và cắt phần dư

4.2.2 Training

Tải model: *gpt2* từ thư viện vào device là GPU (*cuda*), trên task causal language modeling.

Tải dữ liệu vào *DataLoader*: theo batch size đã chỉ định (data loader giúp quá trình training diễn ra theo batch).

Thực hiện định nghĩa các hyperparameters như sau:

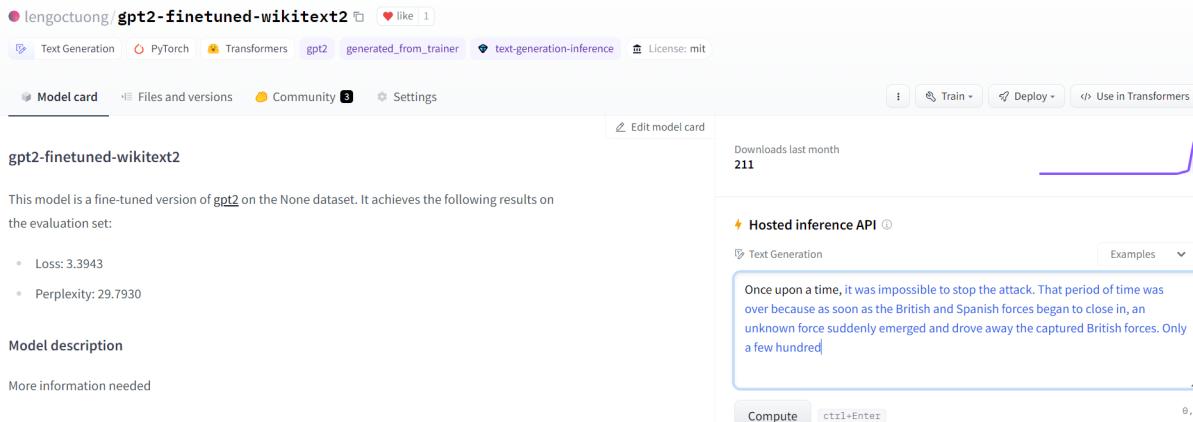
- Kích thước vector input: 128.
- Batch size: 32.
- Số epochs: 3.
- Optimizer: AdamW.
- Learning rate 2e-5 và scheduler là linear (warmup thực hiện tăng tuyến tính lr từ 0 đến 2e-5 trong 60 steps (step có cập nhật trọng số) và thời gian còn lại của các bước huấn luyện sẽ giảm tuyến tính về 0).
- Cứ mỗi 4 lần tính backward (tích lũy gradient) trên 4 batch, sẽ thực hiện cập nhật trọng số 1 lần. Phương pháp này giải quyết được vấn đề batch có kích thước nhỏ do giới hạn GPU.

Thực hiện định nghĩa vòng lặp training: bằng cách lặp qua các epochs và trên từng epoch thực hiện lặp qua các batch của tập validation, với mỗi batch như vậy sẽ thực hiện các bước (ý tưởng thiết lập vòng lặp training tham khảo từ [3]):

- Thực hiện dự đoán trên các batch để lấy vector phân phối xác suất và tính toán loss dựa trên hàm đã định nghĩa.
- Sau mỗi 50 backward step (50 batch) sẽ hiển thị quá trình training gồm:
 - Learning rate thời điểm hiện tại.
 - Số mẫu đã training qua trong mỗi epoch.
 - Số lần đã thực hiện cập nhật trọng số.
 - Metric perplexity của batch tại thời điểm hiển thị quá trình.
- Vì phía trên đã định nghĩa 4 steps backward sẽ cập nhật gradient 1 lần (tức mỗi step đều có tích lũy gradient trên loss) nên cần chia loss cho 4 trước khi backward để sau 4 steps tích lũy gradient sẽ cập nhật đúng trọng số theo loss.
- Để cập nhật trọng số sau 4 training steps, cần thực hiện các bước:
 - Cắt đi các gradient có norm vượt quá 1 để tránh vấn đề exploding gradient.
 - Cập nhật trọng số.
 - Cập nhật learning rate the scheduler đã định nghĩa.
 - Sửa đổi tất cả gradient về 0; vì khi qua các bước backward, gradient sẽ được tích lũy dần dần và nếu cập nhật trọng số dựa trên gradient này sẽ bị sai, vì vậy cần loại bỏ gradient trước lần cập nhật trọng số kế tiếp.

- Sau mỗi 240 backward step (240 batch) sẽ thực hiện định giá, lưu model và tokenizer.
- Đồng thời trong quá trình training các kết quả cross entropy và perplexity trên training và validation cũng được ghi lại.

Lưu model: Sau cùng model và tokenizer được push lên hub, để có thể sử dụng trực tiếp hoặc dùng API lấy về phục vụ tiếp tục train.



Hình 50: Kết quả text generation từ model trên hub

4.3 Evaluation

4.3.1 Cross entropy và perplexity

Trong Causal Language Modelling [8], ta không thể sử dụng các độ đo cho bài toán phân loại (classification) do không có một kết quả chính xác hoàn toàn. Vì vậy, ta sẽ đánh giá dựa trên phân phối của chuỗi sinh ra từ mô hình. Hai độ đo thường được sử dụng đó là cross entropy và perplexity.

Cross entropy là một khái niệm quan trọng trong xử lý ngôn ngữ tự nhiên và các mô hình ngôn ngữ. Trong mô hình ngôn ngữ, cross entropy thường được sử dụng làm hàm mất mát (loss function) để đo lường hiệu suất của mô hình trong việc dự đoán các từ hoặc chuỗi từ tiếp theo trong một văn bản. Đầu tiên, ta sẽ tìm hiểu về likelihood.

Likelihood [7] của một chuỗi được tính bằng tích xác suất diễn ra của một token khi biết các token trước nó. Và vì vậy cần phải tối ưu xác suất này càng lớn càng tốt. Giả sử ta có một chuỗi X đã được tokenized $X = (x_0, x_1, \dots, x_t)$, công thức tính likelihood như sau:

$$P(X) = \prod_{i=0}^t p(x_i | x_{<i})$$

Trong đó:

- x_i là token thứ i (token).
- $x_{<i}$ là những token phía trước i (context).
- t là số token trong một chuỗi.

Hugging Face is a startup based in New York City and Paris
p(word)

Hugging Face is a startup based in New York City and Paris
p(word|context)

Hugging Face is a startup based in New York City and Paris
p(word|context)

Hugging Face is a startup based in New York City and Paris
p(word|context)

Hugging Face is a startup based in New York City and Paris
p(word|context)

Hugging Face is a startup based in New York City and Paris
p(word|context)

Hình 51: Xác suất có điều kiện của từng từ

Với likelihood, ta có thể tính cross entropy bằng công thức sau:

$$CE(X) = -\frac{1}{t} \log P(X) = -\frac{1}{t} \sum_{i=0}^t \log p(x_i | x_{<i})$$

Trong đó:

- $P(X)$ là giá trị likelihood của chuỗi X .
- Cross entropy được định nghĩa dựa trên log-likelihood.
- Dấu trừ được thêm vào để chuyển bài toán thành tối thiểu hóa mất mát (loss).

Giá trị cross entropy càng thấp, tức xác suất từ kế tiếp có khả năng diễn ra dựa trên ngữ cảnh càng cao, và vì vậy mô hình dự đoán càng tốt. Để huấn luyện mô hình ngôn ngữ, ta thường cố gắng tối thiểu hóa giá trị cross entropy bằng cách điều chỉnh các tham số của mô hình để làm cho các dự đoán của mô hình gần với thực tế nhất.

Perplexity [5] (PPL) là một dạng mạnh hơn của cross entropy và thường được sử dụng khi đánh giá hiệu suất của một mô hình ngôn ngữ. Perplexity được định nghĩa là lũy thừa cơ số e của cross entropy:

$$PPL(X) = e^{CE(X)} = e^{-\frac{1}{t} \sum_{i=0}^t \log p(x_i | x_{<i})}$$

Trong đó:

- $CE(X)$ là cross entropy của chuỗi X .

Giá trị perplexity càng thấp, tức là mô hình càng tốt trong việc dự đoán. Nếu mô hình dự đoán chính xác như phân phối xác suất thực tế, perplexity sẽ đạt giá trị tối thiểu là 1.

4.3.2 Loss function và evaluate function

Định nghĩa stop words: sử dụng *nltk*, mã hóa các stop words này thành các stop tokens.

Hàm loss: sử dụng *cross entropy* và có thay đổi về mức độ ưu tiên đối với chuỗi token có chứa nhiều tokens quan trọng (chuỗi chứa nhiều token khác stop tokens) so với các chuỗi token có ít token quan trọng trong cùng 1 batch. Các bước tính toán như sau:

- *Dối với hàm loss thông thường từ thư viện transformers* thực hiện:
 - Tính loss (sử dụng cross entropy) trên từng token được dự đoán trong chuỗi so với label.
 - Thực hiện quá trình tính loss trên với 127 token được dự đoán (không tính dự đoán token bắt đầu), loss trên từng chuỗi cũng chính là trung bình cross entropy của 127 token này.
 - Vì vì mỗi batch có kích thước 32 chuỗi token, nên cần tiếp tục tính trung bình loss của các chuỗi này. Do tính trung bình trên 32 chuỗi token nên mọi chuỗi được dự đoán đều có trọng số loss như nhau.
- *Hàm loss của nhóm em thực hiện* cũng bắt đầu với 2 bước như của thư viện. Và bước thứ 3 thực hiện tính trung bình có trọng số trên các chuỗi. Trọng số được tính bằng số token khác stop tokens và cộng thêm 1 (vì nếu chuỗi có nhiều token khác stop tokens sẽ ưu tiên cao hơn, cũng như để phòng trường hợp chuỗi chỉ toàn stop tokens sẽ cho trọng số bằng 0 vì vậy cần cộng thêm 1 vào trọng số).
- *Cuối cùng* hàm này trả về 2 loss có và không có trọng số: đối với loss có trọng số sẽ dùng để cập nhật model, loss không có trọng số sẽ được lưu lại để so với kết quả evaluate trên tập validation và test.

VD: Giả sử xem mỗi từ là 1 token thì chuỗi "I am a student" (trọng số là 2, có từ "student" khác stop words) sẽ ưu tiên thấp hơn chuỗi "A beautiful girl" (trọng số là 3). Và khi mô hình dự đoán sai chuỗi thứ 2 sẽ tạo loss lớn hơn.

Hàm evaluate: thực hiện định giá bằng loss có sẵn khi model tạo ra text (không có trọng số) và định giá trên từng batch của tập dữ liệu validation hoặc test. Tính trung bình loss của tất cả các batch, trả về kết quả gồm loss và perplexity (là dạng exponent của loss).

4.3.3 Results

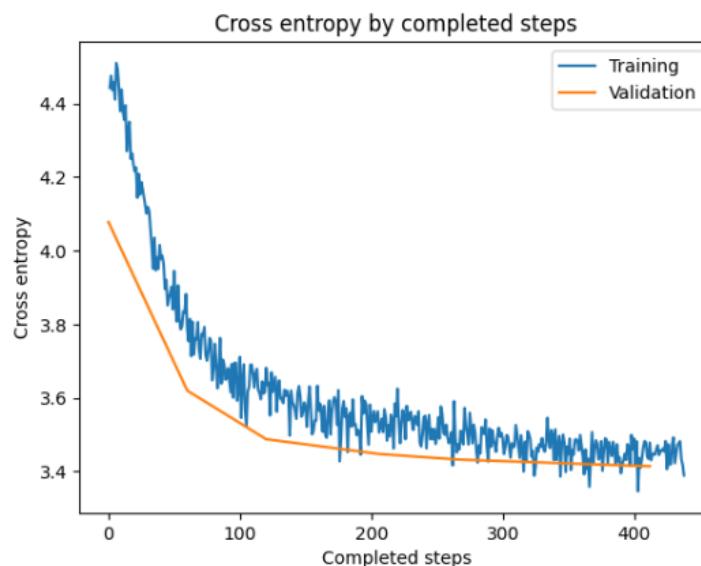
Định giá: được thực hiện trước và trong quá trình training. Kiểm tra trên tập test thực hiện sau khi training:

- *Trước training:* trên tập validation có $\text{crossentropy} \approx 4.0780$ và $\text{perplexity} \approx 59.0288$.
- *Sau training:* trên tập test có $\text{crossentropy} \approx 3.3943$ và $\text{perplexity} \approx 29.7930$, chúng tỏ rằng mô hình đã học khác tốt tập dữ liệu training.

Để hiểu rõ hơn, hãy xem sự cải thiện của model trong **quá trình training trên tập validation**. Theo các tham số thiết lập mô hình sẽ thực hiện 2 lần định giá trên mỗi epoch và ở các step (backward) 240 và 480.

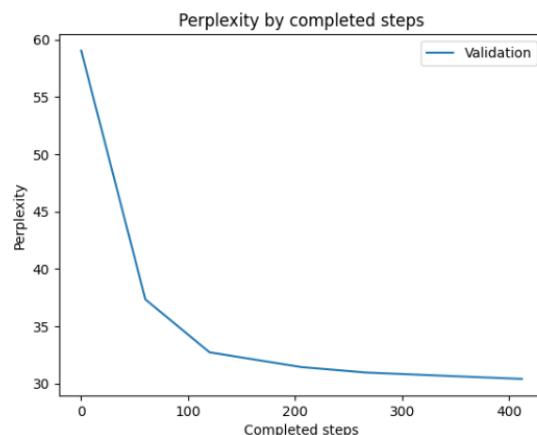
Epoch	Step	Cross Entropy	Perplexity
1.0	240	3.6200	37.3293
1.0	480	3.4878	32.7127
2.0	240	3.4476	31.4245
2.0	480	3.4325	30.9543
3.0	240	3.4216	30.6186
3.0	480	3.4142	30.3937

Và bên dưới là biểu đồ của **Cross Entropy** trên training (cập nhật theo step có thay đổi trọng số) và validation (tính 2 lần mỗi epoch), trực hoành là step có thay đổi trọng số. Có thể thấy đối với training thực hiện cập nhật theo batch nên sẽ có sự biến động trong từng epoch tuy vậy về dài hạn Cross Entropy đã giảm. Kết quả cho ra tương tự với tập validation nhưng mượt mát ít hơn.



Hình 52: Cross Entropy on Training and Validation

Và kết quả trên **Perplexity** giảm từ khoảng 60 đến 30 là khá tốt.



Hình 53: Perplexity on Validation

5 Web App

5.1 Chức năng

Chức năng chính của phần mềm là sử dụng mô hình GPT-2 đã được huấn luyện để sinh ra chuỗi từ chuỗi đầu vào mà người dùng nhập.

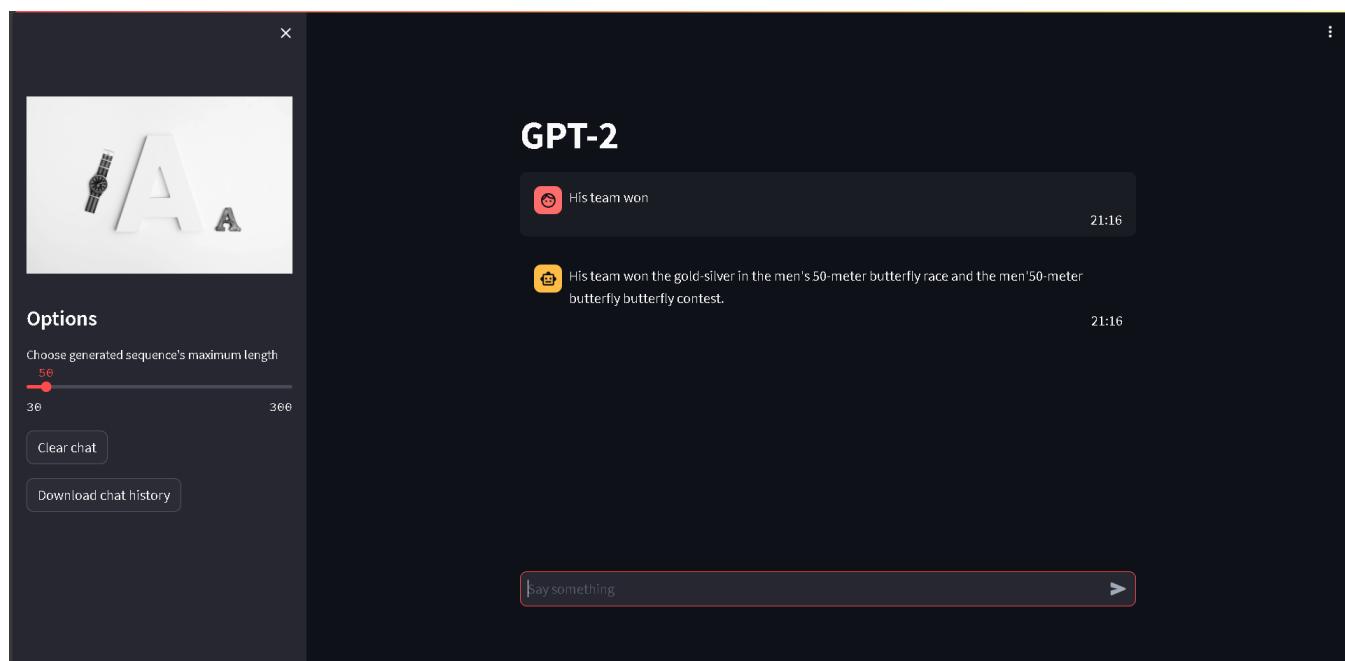
Các **chức năng phụ**:

- Điều chỉnh độ dài tối đa (max_length) của chuỗi sinh ra từ mô hình.
- Xóa các đoạn trò chuyện.
- Tải về lịch sử trò chuyện.

5.2 Giao diện

Mô tả giao diện:

- Có một ô chat input cho người dùng nhập văn bản. Lịch sử cuộc trò chuyện được hiển thị dưới dạng cuộc trò chuyện giữa người dùng và trợ lý ảo.
- Nút "Clear chat" để xóa toàn bộ cuộc trò chuyện.
- Thanh trượt "Choose generated sequence's maximum length" để chọn độ dài tối đa cho đoạn văn bản được tạo ra bởi GPT-2.
- Nút "Download chat history" để tải về lịch sử cuộc trò chuyện dưới dạng tệp văn bản.



Hình 54: Giao diện chính của phần mềm

5.3 Chạy phần mềm

Web app chủ yếu sử dụng thư viện *streamlit* và *transformers* để tạo ra một giao diện chat với GPT-2 (dùng pipeline để tải từ model [13] đã lưu trên hub).

Cách chạy phần mềm:

- Đảm bảo đã cài đặt các thư viện cần thiết bằng cách chạy lệnh: pip install [tên thư viện].
- Mở dòng lệnh hoặc terminal và di chuyển đến thư mục chứa file code của phần mềm.
- Chạy phần mềm bằng lệnh: streamlit run [tên_file].py.
- Phần mềm sẽ được chạy local ở port 8501 (<http://localhost:8501>).

Tài liệu

- [1] Jay Alammar. [The Illustrated Transformer](#). 2018.
- [2] Jay Alammar. [The Illustrated GPT-2 \(Visualizing Transformer Language Models\)](#). 2019.
- [3] HuggingFace NLP Course. [Training a causal language model from scratch](#). 2021.
- [4] HuggingFace. [Fine-tune a language model](#). 2021.
- [5] HuggingFace. [Perplexity of fixed-length models](#). 2021.
- [6] HuggingFace. [Causal language modeling](#). 2021.
- [7] HuggingFace. [What is perplexity?](#) 2021.
- [8] HuggingFace. [Tasks: Causal Language Modeling](#). 2021.
- [9] HuggingFace. [wikitext dataset](#). 2022.
- [10] Trang học Machine Learning. [Transformer Decoder](#). 2023.
- [11] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. [Improving Language Understanding by Generative Pre-Training](#). 2018.
- [12] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. [Language Models are Unsupervised Multitask Learners](#). 2019.
- [13] Le Ngoc Tuong, Nguyen Tu Duy, and Ho Quoc Duy. [lengoctuong/gpt2-finetuned-wikitext2](#). 2023.
- [14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. [Attention Is All You Need](#). In *Advances in Neural Information Processing Systems, (NIPS)*, 2017.
- [15] Zian (Andy) Wang. [Visualizing and Explaining Transformer Models From the Ground Up](#). 2023.