**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM**

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

---------------------------------



**FINAL REPORT**

**PART I: TRANSFOMER BASED**

**ENCODER AND GPT MODELS**

*Người hướng dẫn:* **GV Lê Anh Cường**

*Người thực hiện:* **Hà Trung Hiếu – 520H0632**

**Huỳnh Anh Khoa - 520H0465**

**Nguyễn Hoàng Uyên Thư – 520H0686**

*Lớp:* **20H50302**

*Khóa:*  **24**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM**

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

---------------------------------



**FINAL REPORT**

**PART I: TRANSFOMER BASED**

**ENCODER AND GPT MODELS**

*Người hướng dẫn:* **GV Lê Anh Cường**

*Người thực hiện:* **Hà Trung Hiếu – 520H0632**

**Huỳnh Anh Khoa - 520H0465**

**Nguyễn Hoàng Uyên Thư – 520H0686**

*Lớp:* **20H50302**

*Khóa:*  **24**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2**

**LỜI CẢM ƠN**

Để có thể hoàn thành được bài báo cáo cuối kỳ của môn học này, trước tiên chúng em xin trân trọng cảm ơn Thầy Lê Anh Cường, người đã tận tình hướng dẫn và đồng hành cùng chúng em trong suốt quá trình thực hiện bài báo cáo cuối kỳ. Với vốn kiến thức quan trọng chúng em đã tiếp thu được trong cả quá trình học tập không chỉ là nền tảng cho quá trình nghiên cứu bài báo cáo mà còn là bước đệm để chúng em có thể vững bước trong tương lai. Em kính chúc thầy thật nhiều sức khỏe và tiếp tục thành công trong sự nghiệp giảng dạy cao quý.

**LỜI CAM ĐOAN**

Chúng em xin cam đoan Báo cáo cuối kỳ do nhóm em nghiên cứu và thực hiện. Chúng em đã kiểm tra dữ liệu theo quy định hiện hành. Kết quả Báo cáo quá trình là trung thực và không sao chép từ bất kỳ báo cáo của nhóm khác. Các tài liệu được sử dụng trong Báo cáo quá trình có nguồn gốc, xuất xứ rõ ràng.

**(Ký và ghi rõ họ tên)**

**PHẦN NHẬN XÉT VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN**

**Phần nhận xét của Giảng viên**

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

*Tp. Hồ Chí Minh, ngày tháng năm*

*(kí và ghi họ tên)*

**Phần đánh giá của Giảng viên**

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

*Tp. Hồ Chí Minh, ngày tháng năm*

*(kí và ghi họ tên)*

**TABLE OF CONTENTS**

[A. Kiến thức về mô hình Transformers: 1](#_Toc165325655)

[I. Phương pháp Transformer: 1](#_Toc165325656)

[1. Encoder và Decoder trong Transformer: 1](#_Toc165325657)

[2. Các tiến trình self – attention và encoder – decoder attention: 2](#_Toc165325658)

[II. BERT: 3](#_Toc165325659)

[1. Kiến trúc BERT: 4](#_Toc165325660)

[2. Fine – tuning model BERT: 5](#_Toc165325661)

[3. Masked ML – MLM: 6](#_Toc165325662)

[4. Next Sentence Prediction: 8](#_Toc165325663)

[III. GPT – 2: 10](#_Toc165325664)

[1. Kiến trúc GPT – 2 : 10](#_Toc165325665)

[2. Khối Decoder – only: 12](#_Toc165325666)

[B. Tìm hiểu về Continual Pretrain và Fine – tune trên mô hình cụ thể: 16](#_Toc165325667)

[I. Continual Pre – training: 16](#_Toc165325668)

[1. Continual Pre – traning của mô hình ngôn ngữ là gì: 16](#_Toc165325669)

[2. Quá trình thực hiện Continual Pre – training: 16](#_Toc165325670)

[3. Ưu điểm, nhược điểm và cách sử dụng: 17](#_Toc165325671)

[4. Practical examples: 18](#_Toc165325672)

[II. Fine – tuning: 23](#_Toc165325673)

[1. Fine - tuning hay cụ thể Fine - tuning trong mô hình Transformers là gì? 23](#_Toc165325674)

[2. Làm thế nào để fine - tune mô hình transformers: 24](#_Toc165325675)

[3. Tiếp cận mô mô hình để tinh chỉnh: 24](#_Toc165325676)

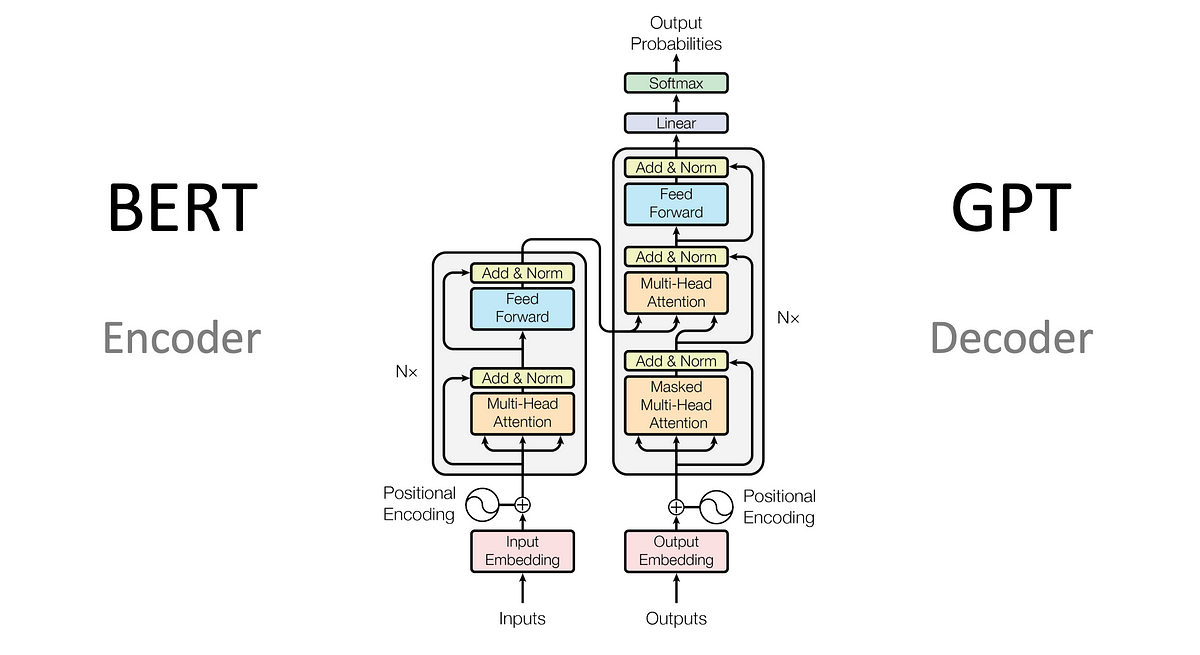
[4. Practical examples: 25](#_Toc165325677)

[**REFERENCES** 31](#_Toc165325678)

A. Kiến thức về mô hình Transformers:

1. Phương pháp Transformer:
2. Encoder và Decoder trong Transformer:

* Trước khi hiểu về BERT chúng ta cùng xem về kỹ thuật transformer. Đây làm một lớp mô hình seq2seq gồm 2 phrase encoder và decoder. Mô hình hoàn toàn không sử dụng các kiến trúc Recurrent Neural Network của RNN mà chỉ sử dụng các layers attention để embedding các từ trong câu. Kiến trúc cụ thể của mô hình như sau:

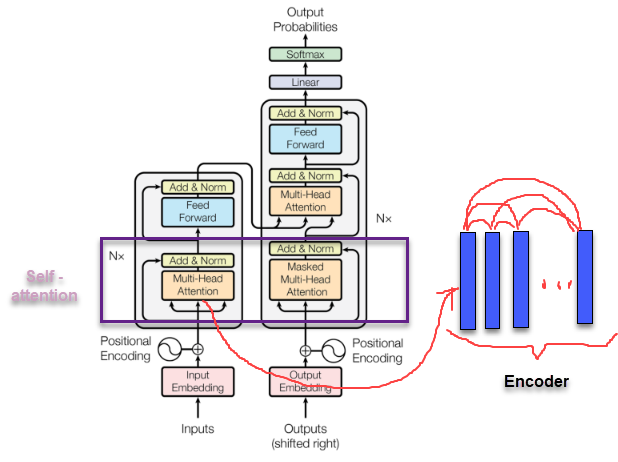


***Hình 1:*** *Sơ đồ kiến trúc transformer kết hợp với attention. Nguồn* [*attention is all you need*](https://arxiv.org/abs/1706.03762)*.*

* Mô hình sẽ bao gồm 2 phase.
  + **Encoder**: Bao gồm 6 layers liên tiếp nhau. Mỗi một layer sẽ bao gồm một sublayer là Multi-Head Attention kết hợp với fully-connected layer như mô tả ở nhánh encoder bên trái của hình vẽ. Kết thúc quá trình encoder ta thu được một vector embedding output cho mỗi từ.
  + **Decoder**: Kiến trúc cũng bao gồm các layers liên tiếp nhau. Mỗi một layer của Decoder cũng có các sub-layers gần tương tự như layer của Encoder nhưng bổ sung thêm sub-layer đầu tiên là Masked Multi-Head Attention có tác dụng loại bỏ các từ trong tương lai khỏi quá trình attention.

1. Các tiến trình self – attention và encoder – decoder attention:

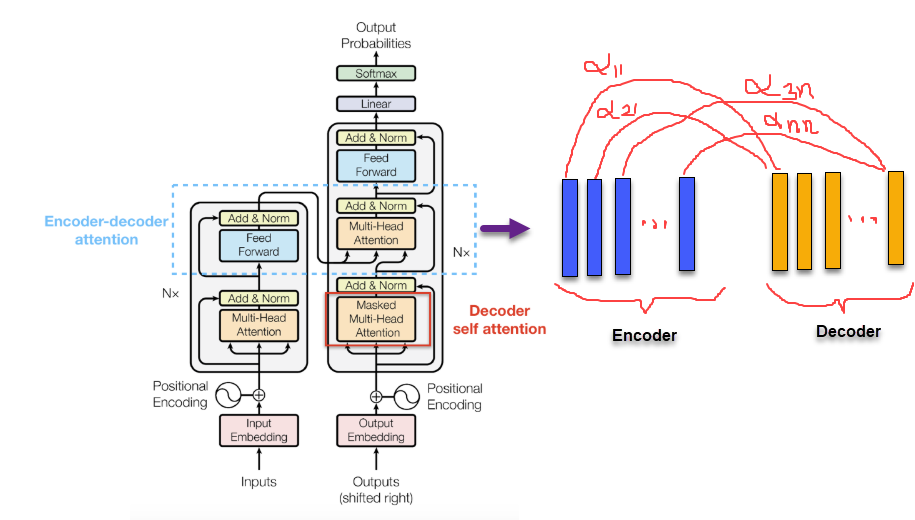
* Trong kiến trúc transformer chúng ta áp dụng 2 dạng attention khác nhau tại từng bước huấn luyện.
  + **self-attention**: Được sử dụng trong cùng một câu input, tại encoder hoặc tại decoder. Đây chính là attention được áp dụng tại các Multi-Head Attention ở đầu vào của cả 2 phase encoder và decoder.



***Hình 2:*** *Sơ đồ vị trí áp dụng self-attention trong kiến trúc transformer.*

Các véc tơ embedding của cùng một chuỗi encoder hoặc decoder tự liên kết với nhau để tính toán attention như hình bên phải.

* + **Encoder-decoder attention**:



***Hình 3:*** *Mô tả hoạt động của encoder – decoder attention*

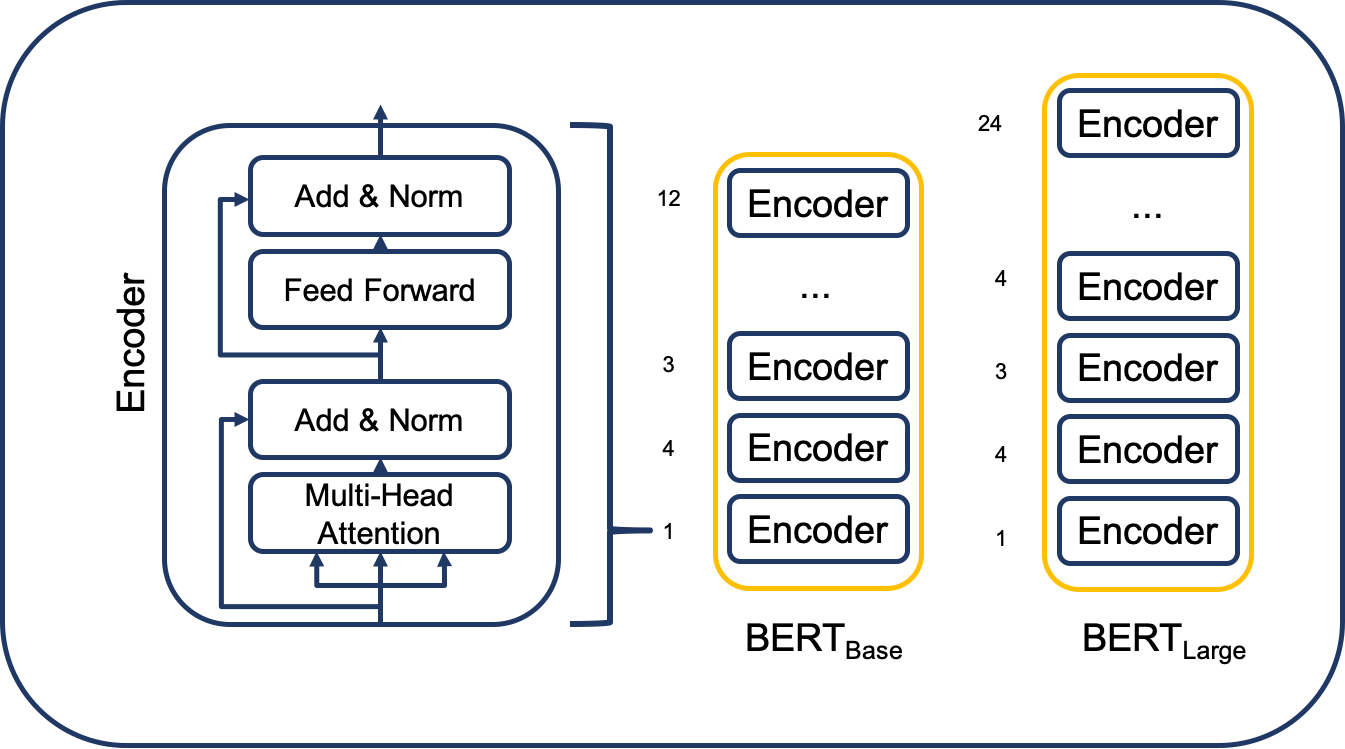
* Bên trái là vị trí áp dụng encoder-decoder attention. Bên phải là cách tính trọng số attention khi kết hợp mỗi véc tơ embedding ở decoder với toàn bộ các véc tơ embedding ở encoder.
* Sở dĩ được gọi là encoder-decoder attention vì đây là kiến trúc attention tương tác giữa các véc tơ embedding của encoder và decoder. véc tơ context được tính toán trên encoder đã được tính tương quan với véc tơ decoder nên sẽ có ý nghĩa giải thích bối cảnh của từ tại vị trí time step decoder tương ứng. Sau khi kết hợp giữa véc tơ context và véc tơ decoder ta sẽ project tiếp qua một fully connected layer để tính phân phối xác suất cho output.
* Mặc dù có kiến trúc chỉ gồm các biến đổi attention nhưng Transformer lại có kết quả rất tốt trong các tác vụ NLP như sentiment analysis và dịch máy.

1. BERT:

[BERT](https://arxiv.org/pdf/1810.04805.pdf) là viết tắt của cụm từ Bidirectional Encoder Representation from Transformer có nghĩa là mô hình biểu diễn từ theo 2 chiều ứng dụng kỹ thuật Transformer. BERT được thiết kế để huấn luyện trước các biểu diễn từ (pre-train word embedding). Điểm đặc biệt ở BERT đó là nó có thể điều hòa cân bằng bối cảnh theo cả 2 chiều trái và phải.

1. Kiến trúc BERT:

* Theo bài viết The Illustrated Transformer, mô hình transformer gốc được tạo thành từ một encoder và một decoder - mỗi phần là một chuỗi các khối transformer. Kiến trúc đó là phù hợp vì mô hình cần giải quyết bài toán dịch máy - một vấn đề mà kiến trúc encoder-decoder đã thành công trong quá khứ.
* Nhiều nghiên cứu sau này đã đưa ra các kiến trúc lược bỏ bộ mã hóa hoặc bộ giải mã, chỉ cần xếp chồng các khối Transformer lên nhau - xếp chúng lên càng cao càng tốt trong thực tế, sau đó đưa vào lượng lớn văn bản huấn luyện và sử dụng một lượng tính toán lớn (hơn hàng trăm nghìn đô la để huấn luyện một số mô hình ngôn ngữ).

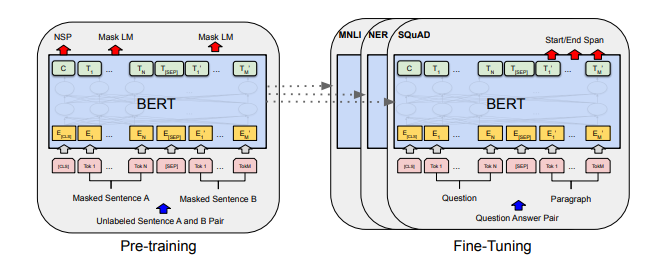


*Hình 2.1. Cấu trúc của BERT chính là xếp chồng các phần encoder của Transformer.*

* Cơ chế attention của Transformer sẽ truyền toàn bộ các từ trong câu văn đồng thời vào mô hình một lúc mà không cần quan tâm đến chiều của câu. Do đó Transformer được xem như là huấn luyện hai chiều (bidirectional) mặc dù trên thực tế chính xác hơn chúng ta có thể nói rằng đó là huấn luyện không chiều (non-directional). Đặc điểm này cho phép BERT học được bối cảnh của từ dựa trên toàn bộ các từ xung quanh nó bao gồm cả từ bên trái và từ bên phải.

1. Fine – tuning model BERT:

* Một điểm đặc biệt ở BERT mà các model embedding trước đây chưa từng có đó là kết quả huấn luyện có thể fine-tuning được. Chúng ta sẽ thêm vào kiến trúc model một output layer để tùy biến theo tác vụ huấn luyện.

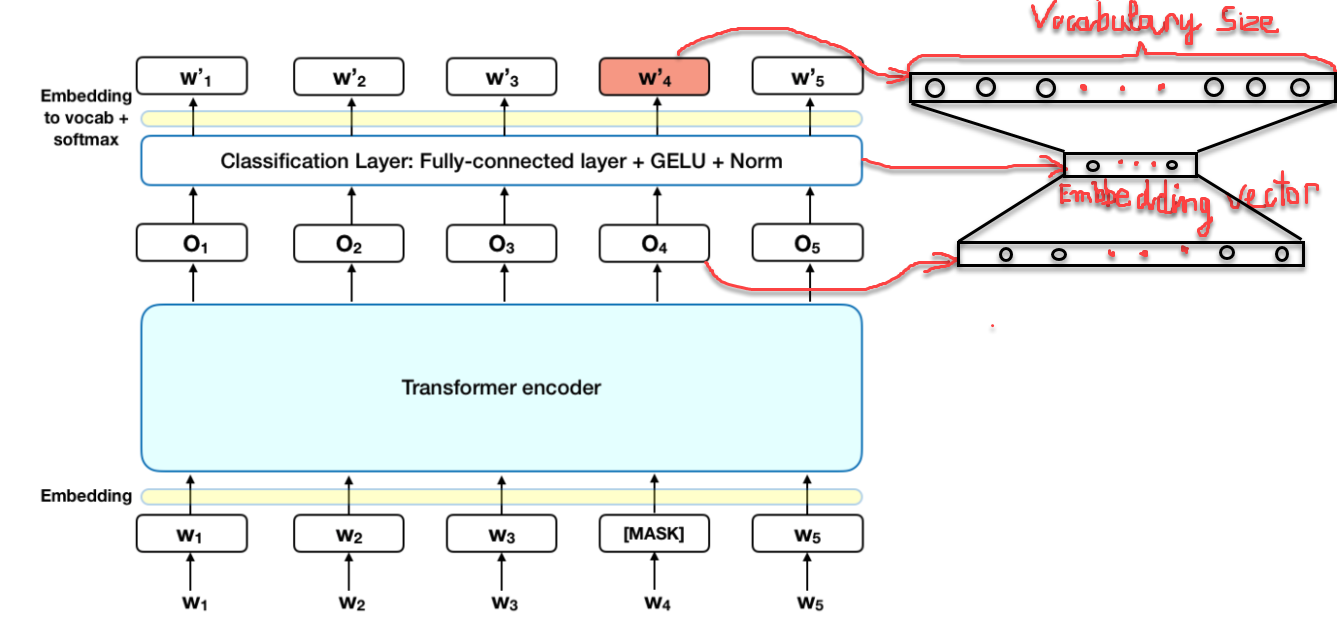


***Hình 4:*** *Toàn bộ tiến trình pre-training và fine-tuning của BERT.*

* Một kiến trúc tương tự được sử dụng cho cả pretrain-model và fine-tuning model. Chúng ta sử dụng cùng một tham số pretrain để khởi tạo mô hình cho các tác vụ down stream khác nhau. Trong suốt quá trình fine-tuning thì toàn bộ các tham số của layers học chuyển giao sẽ được fine-tune. Đối với các tác vụ sử dụng input là một cặp sequence (pair-sequence) ví dụ như question and answering thì ta sẽ thêm token khởi tạo là [CLS] ở đầu câu, token [SEP] ở giữa để ngăn cách 2 câu.
* Tiến trình áp dụng fine-tuning sẽ như sau:
* **Bước 1**: Embedding toàn bộ các token của cặp câu bằng các véc tơ nhúng từ pretrain model. Các token embedding bao gồm cả 2 token là [CLS] và [SEP] để đánh dấu vị trí bắt đầu của câu hỏi và vị trí ngăn cách giữa 2 câu. 2 token này sẽ được dự báo ở output để xác định các phần Start/End Span của câu output.
* **Bước 2**: Các embedding véc tơ sau đó sẽ được truyền vào kiến trúc multi-head attention với nhiều block code (thường là 6, 12 hoặc 24 blocks tùy theo kiến trúc BERT). Ta thu được một véc tơ output ở encoder.
* **Bước 3**: Để dự báo phân phối xác suất cho từng vị trí từ ở decoder, ở mỗi time step chúng ta sẽ truyền vào decoder véc tơ output của encoder và véc tơ embedding input của decoder để tính encoder-decoder attention. Sau đó projection qua liner layer và softmax để thu được phân phối xác suất cho output tương ứng ở time step 𝑡.
* **Bước 4**: Trong kết quả trả ra ở output của transformer ta sẽ cố định kết quả của câu Question sao cho trùng với câu Question ở input. Các vị trí còn lại sẽ là thành phần mở rộng Start/End Span tương ứng với câu trả lời tìm được từ câu input.
* Lưu ý quá trình huấn luyện chúng ta sẽ fine-tune lại toàn bộ các tham số của model BERT đã cut off top linear layer và huấn luyện lại từ đầu các tham số của linear layer mà chúng ta thêm vào kiến trúc model BERT để customize lại phù hợp với bài toán.

1. Masked ML – MLM:

* Masked ML là một tác vụ cho phép chúng ta fine-tuning lại các biểu diễn từ trên các bộ dữ liệu unsupervised-text bất kỳ. Chúng ta có thể áp dụng Masked ML cho những ngôn ngữ khác nhau để tạo ra biểu diễn embedding cho chúng. Các bộ dữ liệu của tiếng anh có kích thước lên tới vài vài trăm tới vài nghìn GB được huấn luyện trên BERT đã tạo ra những kết quả khá ấn tượng.
* Bên dưới là sơ đồ huấn luyện BERT theo tác vụ Masked ML

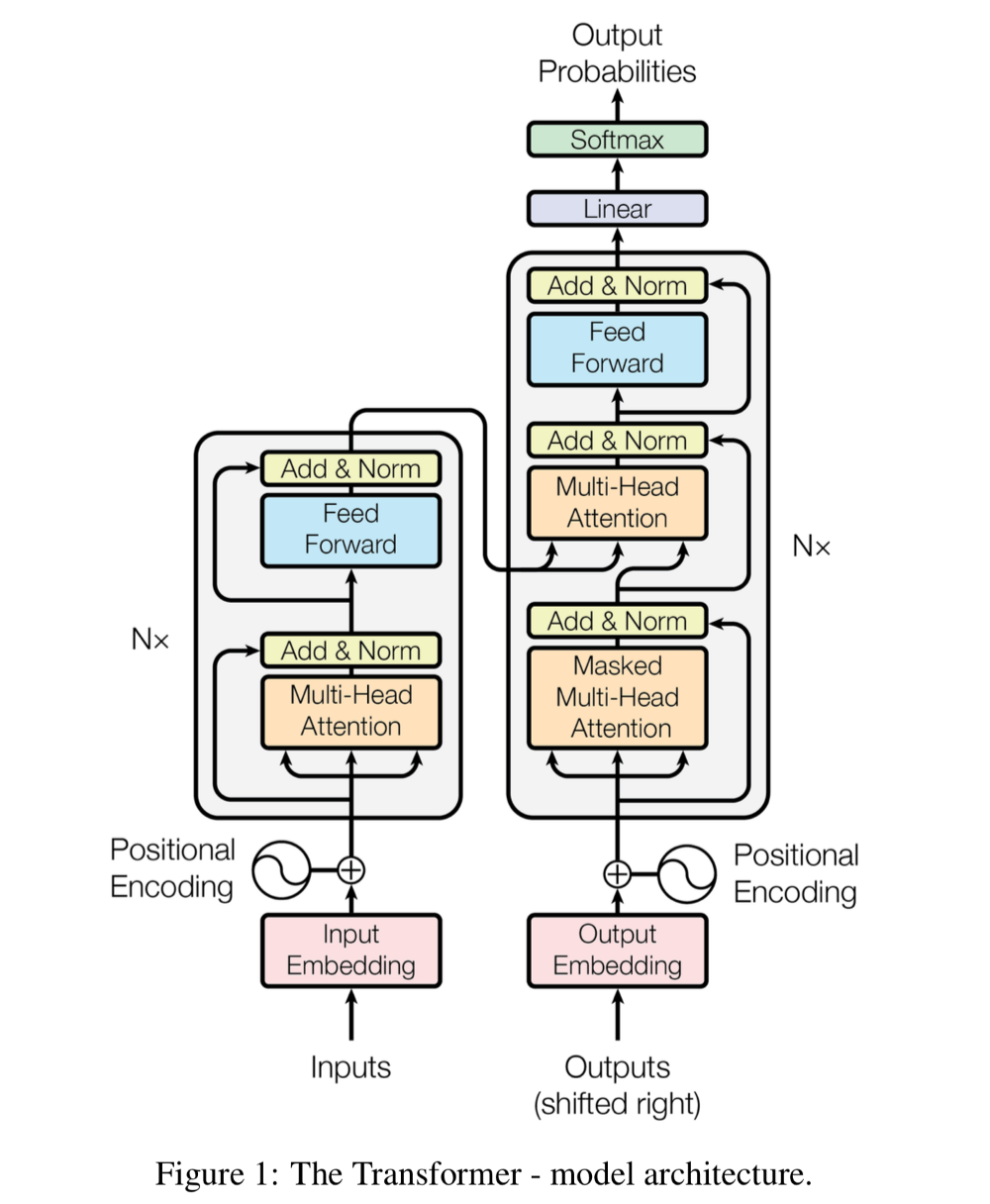


***Hình 5:*** *Sơ đồ kiến trúc BERT cho tá vụ Masked ML.*

* Theo đó:
* Khoảng 15 % các token của câu input được thay thế bởi [MASK] token trước khi truyền vào model đại diện cho những từ bị che dấu (masked). Mô hình sẽ dựa trên các từ không được che (non-masked) dấu xung quanh [MASK] và đồng thời là bối cảnh của [MASK] để dự báo giá trị gốc của từ được che dấu. Số lượng từ được che dấu được lựa chọn là một số ít (15%) để tỷ lệ bối cảnh chiếm nhiều hơn (85%).
* Bản chất của kiến trúc BERT vẫn là một mô hình seq2seq gồm 2 phase encoder giúp embedding các từ input và decoder giúp tìm ra phân phối xác suất của các từ ở output. Kiến trúc Transfomer encoder được giữ lại trong tác vụ Masked ML. Sau khi thực hiện self-attention và feedforward ta sẽ thu được các véc tơ embedding ở output là 𝑂1,𝑂2,…,𝑂5
* Để tính toán phân phối xác suất cho từ output, chúng ta thêm một Fully connected layer ngay sau Transformer Encoder. Hàm softmax có tác dụng tính toán phân phối xác suất. Số lượng units của fully connected layer phải bằng với kích thước của từ điển.
* Cuối cùng ta thu được véc tơ nhúng của mỗi một từ tại vị trí MASK sẽ là embedding véc tơ giảm chiều của véc tơ 𝑂𝑖 sau khi đi qua fully connected layer như mô tả trên hình vẽ bên phải.
* Hàm loss function của BERT sẽ bỏ qua mất mát từ những từ không bị che dấu và chỉ đưa vào mất mát của những từ bị che dấu. Do đó mô hình sẽ hội tụ lâu hơn nhưng đây là đặc tính bù trừ cho sự gia tăng ý thức về bối cảnh. Việc lựa chọn ngẫu nhiên 15% số lượng các từ bị che dấu cũng tạo ra vô số các kịch bản input cho mô hình huấn luyện nên mô hình sẽ cần phải huấn luyện rất lâu mới học được toàn diện các khả năng.

1. Next Sentence Prediction:

* Đây là một bài toán phân loại học có giám sát với 2 nhãn (hay còn gọi là phân loại nhị phân). Input đầu vào của mô hình là một cặp câu (pair-sequence) sao cho 50% câu thứ 2 được lựa chọn là câu tiếp theo của câu thứ nhất và 50% được lựa chọn một cách ngẫu nhiên từ bộ văn bản mà không có mối liên hệ gì với câu thứ nhất. Nhãn của mô hình sẽ tương ứng với IsNext khi cặp câu là liên tiếp hoặc NotNext nếu cặp câu không liên tiếp.
* Cũng tương tự như mô hình Question and Answering, chúng ta cần đánh dấu các vị trí đầu câu thứ nhất bằng token [CLS] và vị trí cuối các câu bằng token [SEP]. Các token này có tác dụng nhận biết các vị trí bắt đầu và kết thúc của từng câu thứ nhất và thứ hai.



***Hình 6:*** *Sơ đồ kiến trúc model BERT cho tác vụ NSP.*

* Thông tin input được preprocessing trước khi đưa vào mô hình huấn luyện bao gồm:
* Ngữ nghĩa của từ (token embeddings): Thông qua các embedding véc tơ cho từng từ. Các véc tơ được khởi tạo từ pretrain model.

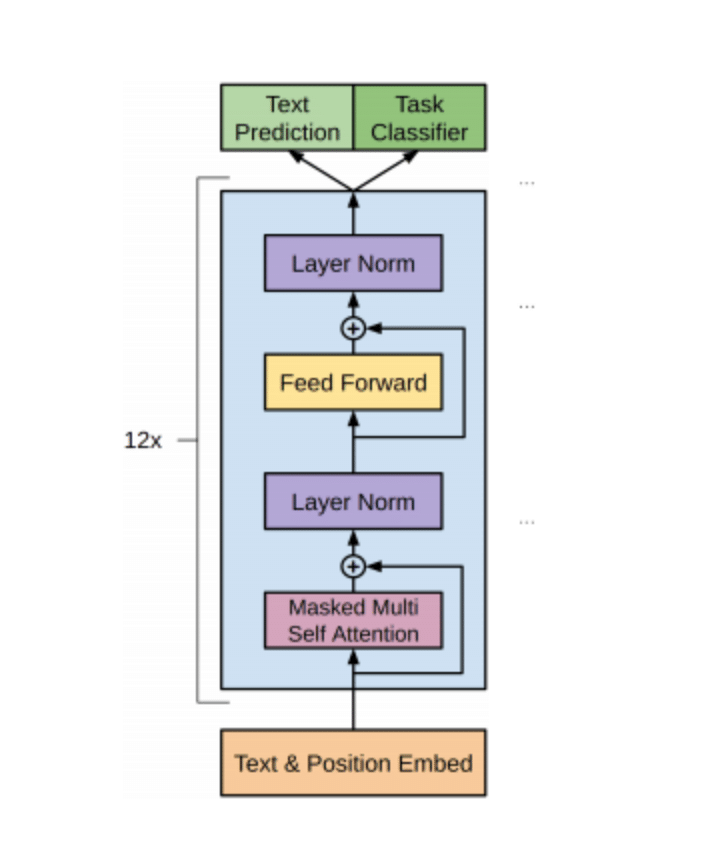
Ngoài embedding biểu diễn từ của các từ trong câu, mô hình còn embedding thêm một số thông tin:

* Loại câu (segment embeddings): Gồm hai véc tơ là 𝐸𝐴 nếu từ thuộc câu thứ nhất và 𝐸𝐵 nếu từ thuộc câu thứ hai.
* Vị trí của từ trong câu (position embedding): là các véc tơ 𝐸0,…,𝐸10. Tương tự như positional embedding trong transformer.

Véc tơ input sẽ bằng tổng của cả ba thành phần embedding theo từ, câu và vị trí.

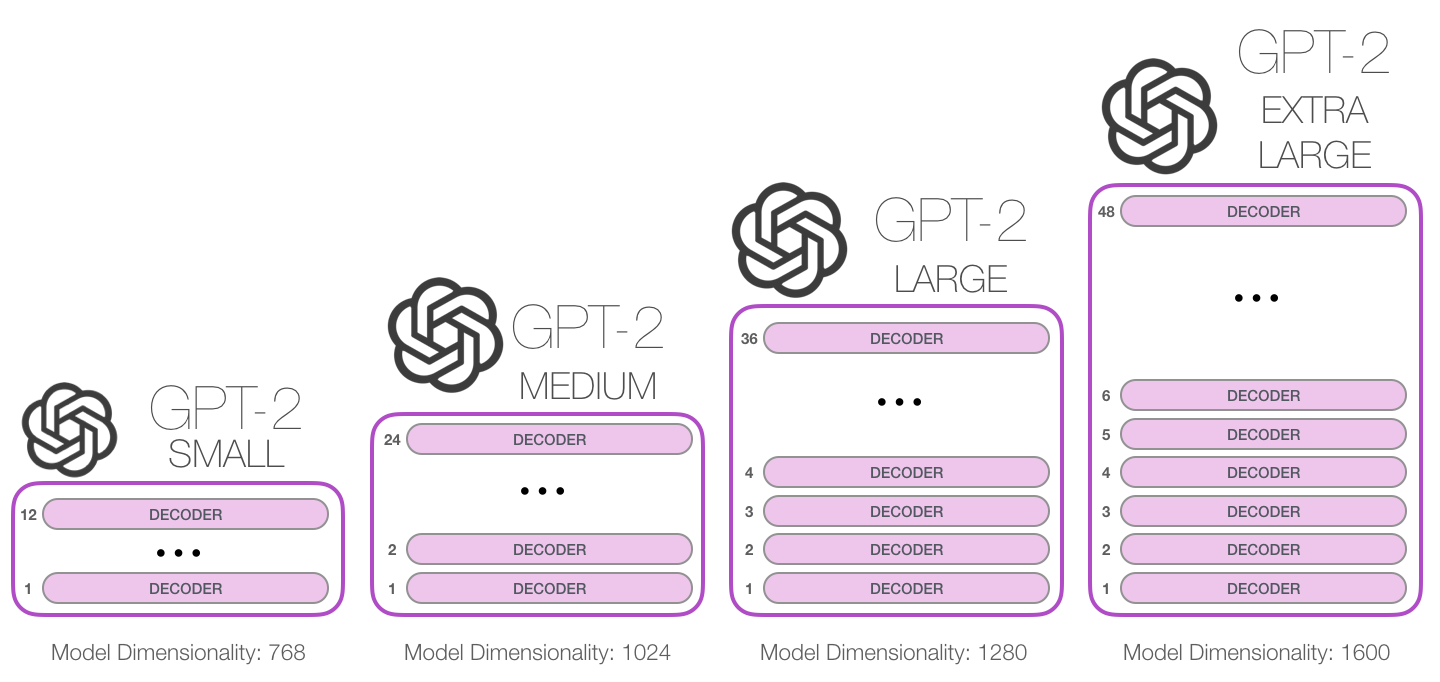
1. GPT – 2:
2. Kiến trúc GPT – 2 :

* BERT được xây dựng dựa trên các khối encoder. Ngược lại, GPT-2 được xây dựng dựa trên các khối decoder.



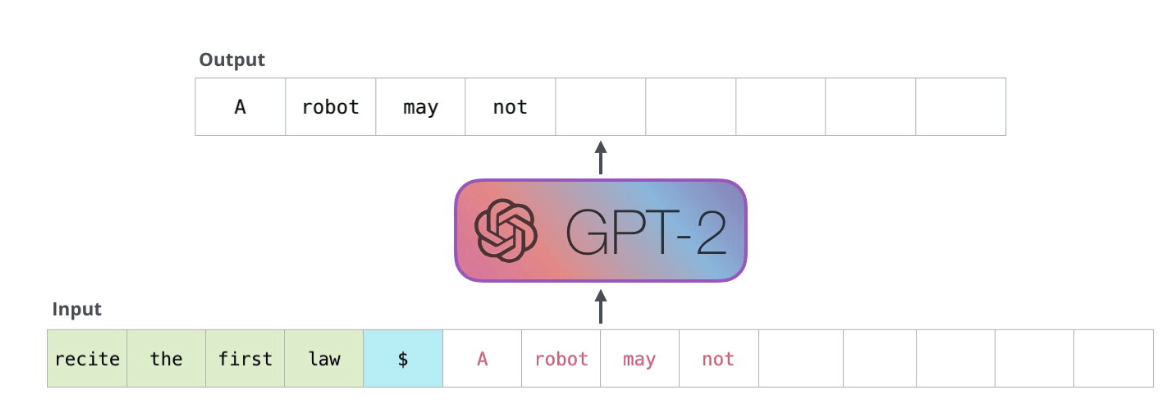
*Hình. Khối decoder - Kiến trúc GPT-2*

* Các khối này sau đó cũng được xếp chồng lên nhau:



*Hình 2. Chúng ta có thể xếp chúng cao bao nhiêu? Hóa ra đó là điểm khác biệt giữa kích thước các GPT2 models*

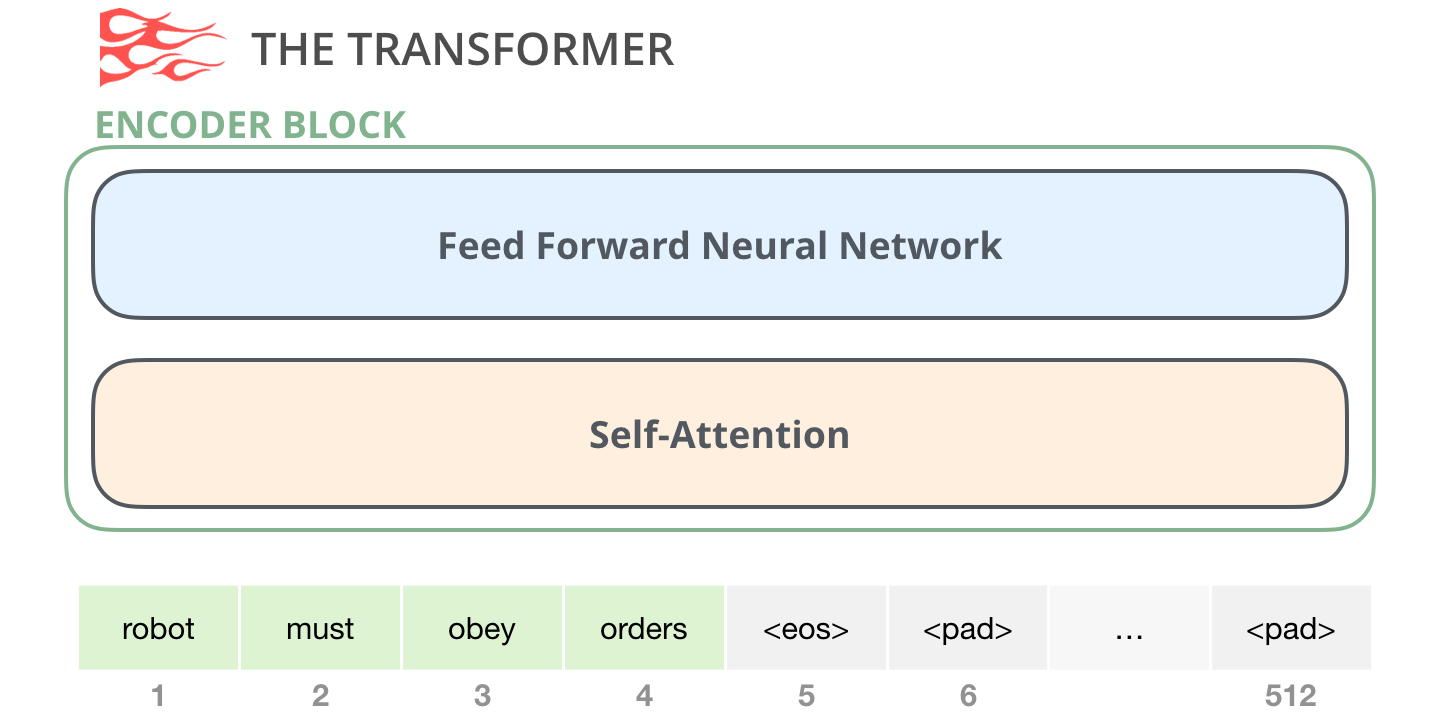
* Một điểm khác biệt chính giữa hai mô hình này là GPT-2 cũng giống như các mô hình ngôn ngữ truyền thống khác, đầu ra là một token mỗi lần.
* Cách mà các mô hình này hoạt động thực sự là sau khi mỗi token được tạo ra, token đó được thêm vào chuỗi các đầu vào. Và chuỗi mới đó trở thành đầu vào cho mô hình trong bước tiếp theo. Đây là một ý tưởng được gọi là "tự hồi quy" (auto-regression). Đây là một trong những ý tưởng làm cho RNN hiệu quả một cách phi lý.



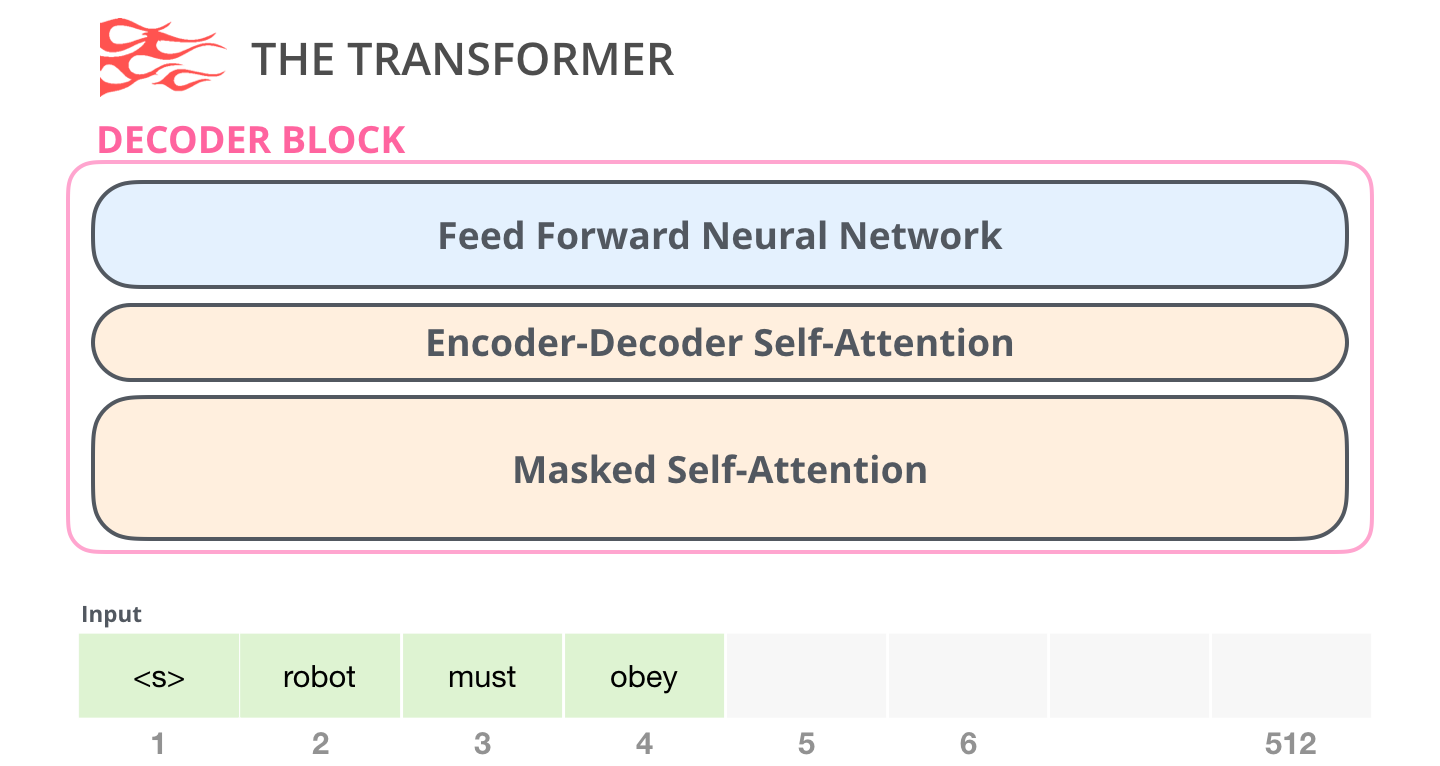
* GPT-2 và một số mô hình sau này như TransformerXL và XLNet cũng có tính tự hồi quy. Trong khi đó, BERT lại không có điều này. Đó là một sự đánh đổi. Trong việc mất sự tự hồi quy, BERT đã có thể tích hợp ngữ cảnh ở cả hai chiều của một từ để đạt được kết quả tốt hơn.

1. Khối Decoder – only:

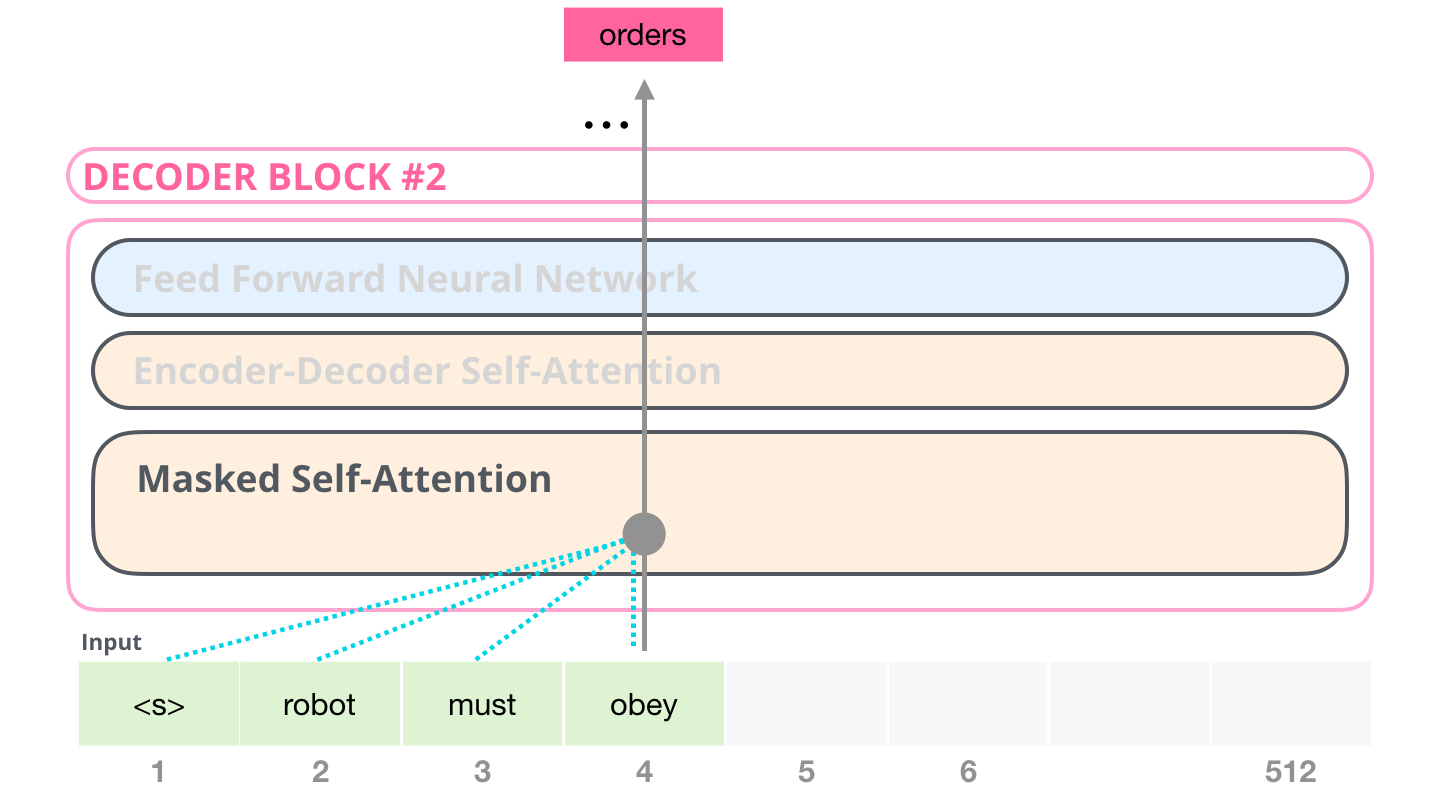
* Bài viết gốc về Transformer giới thiệu hai loại khối Transformer:
* Đầu tiên là khối encoder:



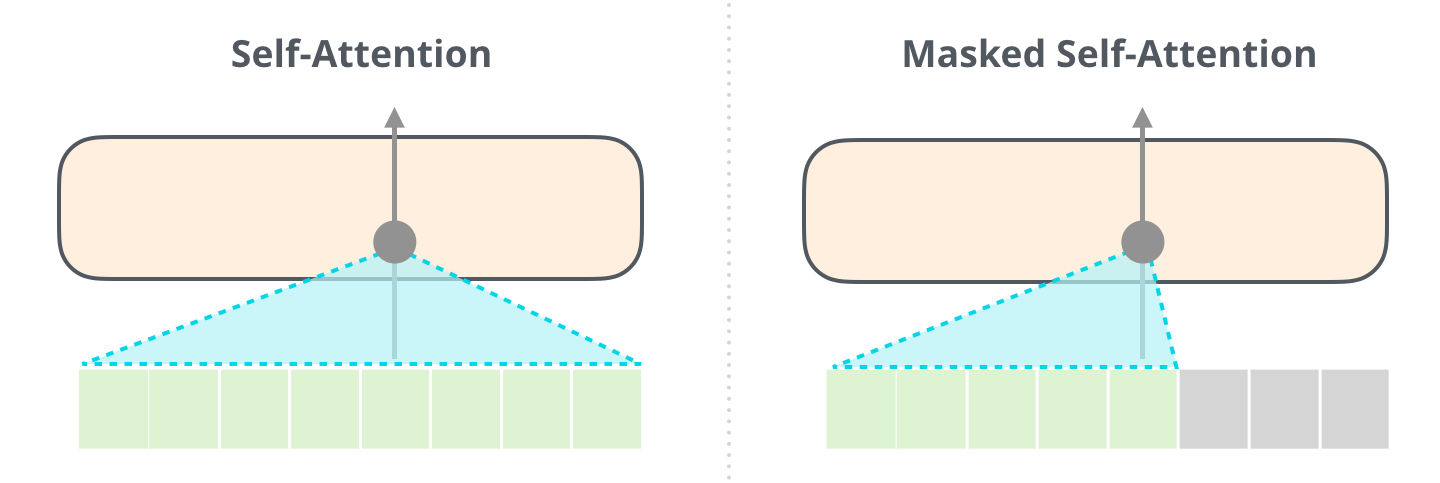
* Thứ hai, khối giải mã là một biến thể nhỏ từ khối mã hóa - có thêm một lớp để cho phép nó chú ý đến các phần cụ thể từ bộ mã hóa:



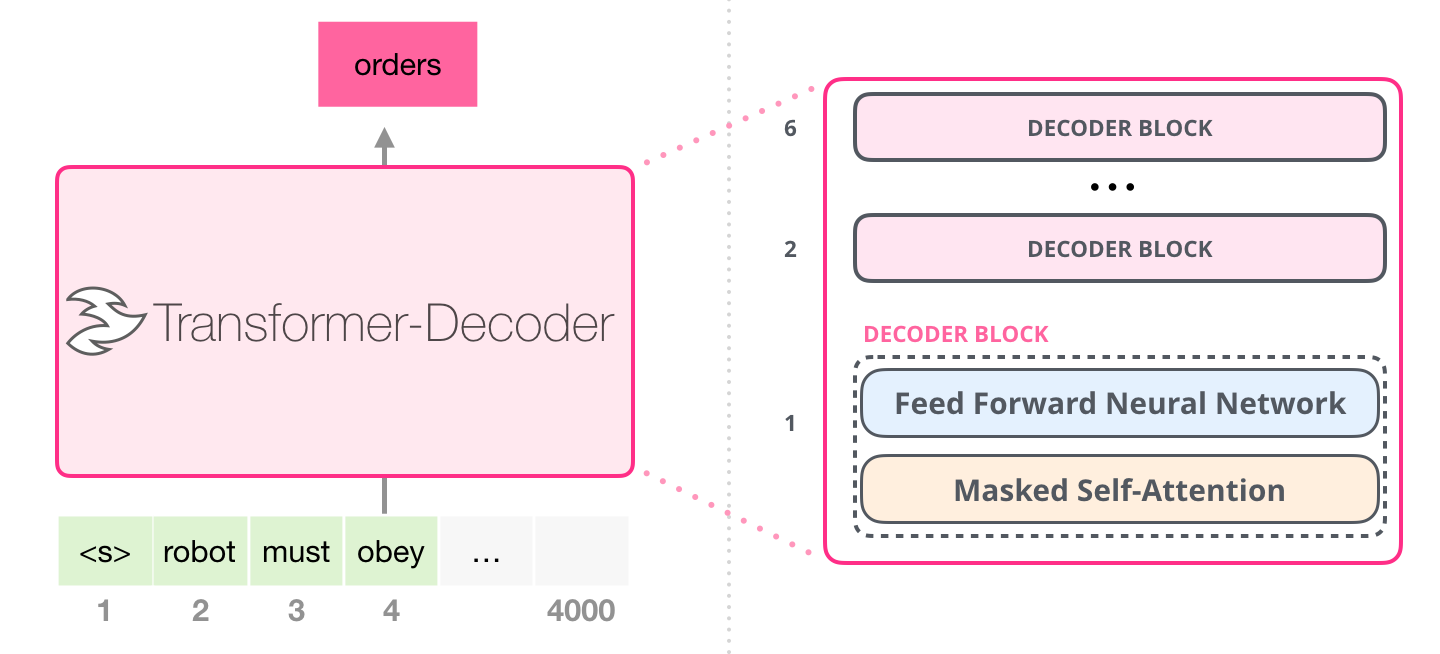
* Một điểm khác biệt chính trong lớp self-attention ở đây là nó ẩn các future tokens - không phải bằng cách thay đổi từ thành [mask] như BERT, mà bằng cách can thiệp vào việc tính toán self-attention để chặn thông tin từ các token ở bên phải của vị trí đang được tính toán.
* Nếu, ví dụ, chúng ta muốn nổi bật đường dẫn của vị trí #4, chúng ta có thể thấy rằng nó chỉ được phép chú ý đến các token hiện tại và trước đó:



* Quan trọng là phải phân biệt rõ giữa self-attention (như BERT sử dụng) và masked self-attention (như GPT-2 sử dụng). Một khối self-attention thông thường cho phép một vị trí nhìn vào các token bên phải của nó. Masked self-attention ngăn chặn điều đó xảy ra:



* Bài báo gốc "Generating Wikipedia by Summarizing Long Sequences" đề xuất một cách khác sử dụng transformer để tạo mô hình ngôn ngữ. Mô hình này loại bỏ bộ mã hóa Transformer. Vì lý do đó, hãy gọi mô hình này là "Transformer-Decoder". Mô hình ngôn ngữ dựa trên transformer này ban đầu được tạo thành từ cách xếp gồm sáu khối giải mã transformer:



* Các khối giải mã là đồng nhất. Tôi đã mở rộng khối đầu tiên để bạn có thể thấy lớp self-attention của nó là biến thể có mask. Lưu ý rằng mô hình giờ đây có thể xử lý đến 4,000 token trong một đoạn nhất định - một nâng cấp lớn từ 512 trong transformer gốc.
* Những khối này rất giống với các khối giải mã gốc, ngoại trừ việc họ loại bỏ lớp self-attention thứ hai. Một kiến trúc tương tự đã được xem xét trong "Character-Level Language Modeling with Deeper Self-Attention" để tạo ra một mô hình ngôn ngữ dự đoán một chữ cái/ký tự mỗi lần.
* Mô hình GPT-2 của OpenAI sử dụng những khối decoder-only này.

# B. Tìm hiểu về Continual Pretrain và Fine – tune trên mô hình cụ thể:

1. Continual Pre – training:
2. Continual Pre – traning của mô hình ngôn ngữ là gì:

CONTINUAL PRE-TRAINING (CPT) là một phương pháp huấn luyện mô hình ngôn ngữ theo cách liên tục và không ngừng cập nhật với dữ liệu mới từ các lĩnh vực hoặc ngữ cảnh khác nhau. Mục tiêu chính của CPT là giữ cho mô hình cập nhật và đáp ứng được với sự thay đổi của dữ liệu đầu vào mà không cần phải tiến hành lại quá trình huấn luyện từ đầu. Thay vì chỉ tiền huấn luyện mô hình một lần trên một tập dữ liệu lớn, CPT tiếp tục huấn luyện mô hình với dữ liệu mới định kỳ, giúp cải thiện hiệu suất và tích lũy kiến thức từ nhiều nguồn khác nhau.

1. Quá trình thực hiện Continual Pre – training:

* Khởi tạo:
* Trước khi bắt đầu quá trình huấn luyện liên tục, một bước khởi tạo được thực hiện để đánh giá độ quan trọng của các đơn vị (như các đầu chú ý và các neuron) đối với kiến thức tổng quát của mô hình.
* Đánh giá độ quan trọng này thường được thực hiện bằng cách sử dụng một phương pháp proxy dựa trên sự kháng cự của mô hình, không cần dùng đến dữ liệu giám sát.
* Huấn luyện liên tục:
* Mô hình tiếp tục được huấn luyện với các tập dữ liệu mới từ các lĩnh vực hoặc ngữ cảnh mới, mà không cần phải huấn luyện lại từ đầu.
* Quá trình huấn luyện này thường bao gồm việc cập nhật các trọng số của mô hình dựa trên hàm mất mát được định nghĩa cho mỗi tác vụ cụ thể.
* Tính toán độ quan trọng của các đơn vị đối với những lĩnh vực hiện tại:
* Sau khi hoàn thành việc huấn luyện với một lĩnh vực mới, độ quan trọng của các đơn vị đối với lĩnh vực này được tính toán dựa trên dữ liệu từ lĩnh vực đó.
* Điều này giúp mô hình tự động thích ứng với kiến thức và ngữ cảnh mới mà không cần sự can thiệp từ phía người dùng.

1. Ưu điểm, nhược điểm và cách sử dụng:
   1. Ưu điểm:

* Tích lũy kiến thức từ nhiều nguồn: CPT cho phép mô hình tích lũy kiến thức từ nhiều nguồn khác nhau, giúp cải thiện hiệu suất và tích hợp tri thức đa ngữ cảnh.
* Tiết kiệm thời gian và tài nguyên: So với việc huấn luyện lại mô hình từ đầu khi có dữ liệu mới, CPT tiết kiệm thời gian và tài nguyên tính toán.
* Tính linh hoạt: CPT cho phép mô hình dễ dàng thích ứng với sự thay đổi của dữ liệu đầu vào mà không cần phải thực hiện lại quá trình huấn luyện từ đầu.
  1. Hạn chế:
* Nguy cơ quên mất: CPT có thể đối mặt với nguy cơ quên mất tri thức đã học từ các lĩnh vực hoặc ngữ cảnh trước đó nếu không được điều chỉnh đúng cách.
* Khó khăn trong việc tính toán độ quan trọng: Tính toán độ quan trọng của các đơn vị đối với từng lĩnh vực có thể đòi hỏi nhiều tài nguyên tính toán và không đơn giản.
  1. Khi nào sử dụng:
* Dữ liệu thay đổi thường xuyên: CPT thích hợp khi dữ liệu đầu vào thay đổi thường xuyên và cần cập nhật mô hình để phản ánh sự thay đổi này.
* Yêu cầu tích lũy kiến thức từ nhiều nguồn: Khi cần một mô hình có khả năng học từ nhiều lĩnh vực và ngữ cảnh khác nhau để cải thiện hiệu suất trên các tác vụ cụ thể.

1. Practical examples:

* Đoạn ví dụ thực hiện quá trình huấn luyện tiếp tục (continual pretraining) cho mô hình ngôn ngữ GPT-2 trên tập dữ liệu PTB (Penn Treebank). Quá trình continual pretraining là một phương pháp huấn luyện liên tục mô hình ngôn ngữ trên các tập dữ liệu mới để cải thiện khả năng hiểu và sản sinh văn bản. Trong đoạn code này, chúng ta sử dụng tập dữ liệu PTB, một tập dữ liệu ngôn ngữ tiếng Anh phổ biến trong lĩnh vực xử lý ngôn ngữ tự nhiên.
* Dữ liệu trong tập PTB được thu thập từ các bài báo, tạp chí và tin tức tiếng Anh. Tập dữ liệu chứa các mẫu văn bản đã được gán nhãn cú pháp (parse tree), điều này giúp cho việc nghiên cứu và phát triển các mô hình ngôn ngữ trở nên dễ dàng hơn. Dữ liệu bao gồm các loại văn bản khác nhau như tiêu đề, đoạn văn, và câu.
* Đầu tiên, chúng ta tải tập dữ liệu PTB và tiến hành chuẩn bị dữ liệu bằng cách sử dụng tokenizer của mô hình GPT-2 để mã hóa các mẫu văn bản. Sau đó, tập dữ liệu được chia thành các phần huấn luyện, validation và test. Quá trình huấn luyện được thực hiện trên một phần nhỏ của tập dữ liệu huấn luyện để tăng tốc quá trình thử nghiệm và phát triển mô hình. Sau khi huấn luyện, chúng ta đánh giá hiệu suất của mô hình bằng cách sử dụng perplexity trên tập dữ liệu validation và test. Đồng thời, chúng ta cũng cung cấp một cách để sử dụng mô hình đã được huấn luyện để sinh văn bản mới từ một chuỗi đầu vào.
  1. Loading dataset và xử lý dữ liệu:
* Đầu tiên là sử dụng thư viện “transformers và datases” để tải thư viện và load dataset về:

**from** **transformers** **import** GPT2Tokenizer, GPT2LMHeadModel, TextDataset, DataCollatorForLanguageModeling, Trainer, TrainingArguments

**from** **datasets** **import** load\_dataset

# Load PTB dataset

dataset = load\_dataset("ptb\_text\_only")

* Sau đó tiến hành tokenizer dữ liệu để ánh xạ mỗi câu trong dữ liệu thành các token và sử dụng GPT2Tokenizer để load tokenizer từ mô hình GPT – 2.

# Tokenize function

**def** **tokenize\_function**(examples):

**return** tokenizer(examples["sentence"])

# Load tokenizer

tokenizer = GPT2Tokenizer.from\_pretrained("gpt2")

# Tokenize the dataset

tokenized\_datasets = dataset.map(tokenize\_function, batched=True)

* Tiếp tục chia dữ liệu thành các tập train, validation, và test. Do đây là bài demo nên sử dụng khoảng 20% của tập train huấn luyện nhằm tiết kiệm thời gian.

# Split dataset into train, validation, and test

train\_dataset = tokenized\_datasets["train"]

validation\_dataset = tokenized\_datasets["validation"]

test\_dataset = tokenized\_datasets["test"]

# Chọn một phần của tập dữ liệu huấn luyện, ví dụ: 20%

percentage = **0.2**

num\_train\_samples = int(len(train\_dataset) \* percentage)

train\_dataset\_subset = train\_dataset.select(range(num\_train\_samples))

**print**(len(train\_dataset\_subset))

* 1. Huấn luyện mô hình:
* Sau khi tiến hành các lớp cần thiết và các thư viện. Tạo hàm ‘load\_data\_collator’ để bắt đầu tạo ra đối tượng từ DataCollatorFor-LanguageModeling để định dạng dữ liệu đầu vào của quá trình train.

**def** **load\_data\_collator**(tokenizer, mlm = False):

data\_collator = DataCollatorForLanguageModeling(

tokenizer=tokenizer,

mlm=mlm,

)

**return** data\_collator

* Tạo hàm train() trong đó gồm các bước như tokenizer để tiền xữ lý dữ liệu đầu vào, các tokenizer, địa chỉ để lưu tokenizer và lưu model sau khi được train, tạo ra cấc thông số traning arguments để huấn luyện.

**def** **train**(dataset,model\_name,

output\_dir,

overwrite\_output\_dir,

per\_device\_train\_batch\_size,

num\_train\_epochs,

save\_steps):

tokenizer = GPT2Tokenizer.from\_pretrained(model\_name)

train\_dataset = dataset

data\_collator = load\_data\_collator(tokenizer)

tokenizer.save\_pretrained(output\_dir)

# Set the padding token to be the end-of-sequence token

tokenizer.pad\_token = tokenizer.eos\_token # đệm tokenizer thành token kết thúc đảm bảo độ dài cho các câu ngắn.

model = GPT2LMHeadModel.from\_pretrained(model\_name) # Tạo mô hình tĩnh GPT 2 với các thông số được train trước đó.

model.save\_pretrained(output\_dir)

training\_args = TrainingArguments( # Đối tượng để lưu các thông số training

output\_dir=output\_dir,

overwrite\_output\_dir=overwrite\_output\_dir,

per\_device\_train\_batch\_size=per\_device\_train\_batch\_size,

num\_train\_epochs=num\_train\_epochs,

)

trainer = Trainer( # Tạo đối tượng trainer để qly quá trình train

model=model, # model huấn luyện

args=training\_args, # thông số huấn luyện

data\_collator=data\_collator, # chuẩn bị dữ liệu huấn luyện

train\_dataset=train\_dataset, # dữ liệu huấn luyện được sd để train

)

trainer.train()

trainer.save\_model()

* Tạo ra các thông số để huấn luyện và bắt đầu huấn luyện:

model\_name = 'gpt2' # Tên model

output\_dir = '/content/drive/MyDrive/continual\_pretrain/model' # Đường dẫn đến thư mục lưu model

overwrite\_output\_dir = False # Thiết lập kh ghi đè, đảm bảo thư mục model trống

per\_device\_train\_batch\_size = **8** # Kích thước huấn luyện cho mỗi thiết bị - mỗi thiết bị sẽ xử lý batch 8

num\_train\_epochs = **3** # Số epoch

save\_steps = **50000** # Lưu trữ trạng thái của mô hình, cách 50000 step sẽ được lưu lại

# Train

train(

dataset=train\_dataset\_subset,

model\_name=model\_name,

output\_dir=output\_dir,

overwrite\_output\_dir=overwrite\_output\_dir,

per\_device\_train\_batch\_size=per\_device\_train\_batch\_size,

num\_train\_epochs=num\_train\_epochs,

save\_steps=save\_steps

)

* 1. Gọi model để kiểm tra và đánh giá:
* Sử dụng model sau khi được train xong và áp dụng hàm generate\_text để sinh ra văn bản mới.

**def** **generate\_text**(model\_path, sequence, max\_length):

model = load\_model(model\_path) # Tải model

tokenizer = load\_tokenizer(model\_path) # Tải tokenizer

ids = tokenizer.encode(f'{sequence}', return\_tensors='pt') # mã hóa các input -> tokenizer

final\_outputs = model.generate(

ids,

do\_sample=True,

max\_length=max\_length,

pad\_token\_id=model.config.eos\_token\_id,

top\_k=**50**,

top\_p=**0.95**,

)

**print**(tokenizer.decode(final\_outputs[**0**], skip\_special\_tokens=True))

model1\_path = "/content/drive/MyDrive/continual\_pretrain/model"

model = load\_model(model1\_path)

input\_text = "Mr Dainel is"

# Tokenize input text

input\_ids = tokenizer(input\_text, return\_tensors="pt").input\_ids

# Generate output

output = model.generate(input\_ids, max\_length=**100**, num\_return\_sequences=**1**, temperature=**0.7**)

# Decode and print output

**print**(tokenizer.decode(output[**0**], skip\_special\_tokens=True))

Ta có thể thấy văn bản sinh ra liên quan đến “promt: Mr Dainiel is” nhưng chỉ một phần nào đó mà không chính xác hoàn toàn.

* Sử dụng perplexity để đánh giá mô hình, kết quả đánh giá perplexity trên tập dữ liệu validation và test cho thấy hiệu suất của mô hình ngôn ngữ GPT-2 trên tập dữ liệu PTB. Perplexity là một độ đo thống kê thường được sử dụng để đánh giá hiệu suất của mô hình ngôn ngữ, và giá trị perplexity càng thấp thì mô hình càng tốt.

# Evaluate perplexity on validation dataset

validation\_loss = trainer.evaluate(eval\_dataset=validation\_dataset)['eval\_loss']

validation\_perplexity = math.exp(validation\_loss)

**print**("Perplexity on validation dataset:", validation\_perplexity)

# Evaluate perplexity on test dataset

test\_loss = trainer.evaluate(eval\_dataset=test\_dataset)['eval\_loss']

test\_perplexity = math.exp(test\_loss)

**print**("Perplexity on test dataset:", test\_perplexity)

* Trong trường hợp này, kết quả perplexity trên tập dữ liệu validation là khoảng 33.54 và trên tập dữ liệu test là khoảng 29.95. Điều này cho thấy mô hình có khả năng dự đoán văn bản trên tập dữ liệu test với mức độ không chắc chắn trung bình là khoảng 30 từ trong tập dữ liệu mà nó chưa từng thấy trước đó.
* Tuy nhiên, để hiểu rõ hơn về hiệu suất của mô hình, cần phải so sánh các giá trị perplexity này với các mô hình khác hoặc với các phiên bản cải thiện của chính mô hình này sau khi được huấn luyện thêm. Đồng thời, cũng cần xem xét các kỹ thuật cải tiến khác nhau để cải thiện hiệu suất của mô hình, chẳng hạn như kiến trúc mô hình, kích thước dữ liệu huấn luyện, hoặc các kỹ thuật tinh chỉnh (fine-tuning) khác.

1. Fine – tuning:

Chắc hẳn khi đã tìm hiểu tới đây mọi người đã quen với khái niệm sau: Transfer learning và Fine-tuning. Transfer learning là một phương pháp linh hoạt cho phép chúng ta tận dụng tri thức đã học được từ một vấn đề để áp dụng vào một vấn đề liên quan khác. Transfer learning có nhiều dạng khác nhau và bạn có thể tìm hiểu thêm trong bài viết này: Tổng hợp Transfer learning. Trong bài viết này, chúng tôi sẽ tập trung vào một dạng phổ biến của transfer learning: Fine-tuning.

* 1. Fine - tuning hay cụ thể Fine - tuning trong mô hình Transformers là gì?
* Fine-tuning là quá trình mà một mô hình đã được huấn luyện trước được tiếp tục huấn luyện trên một nhiệm vụ mới sử dụng dữ liệu cụ thể cho nhiệm vụ đó. Trong ngữ cảnh của các mô hình Transformer, fine-tuning đề cập đến việc sử dụng một mô hình Transformer đã được huấn luyện trước như một điểm khởi đầu cho việc huấn luyện trên một nhiệm vụ mới.
* Ý tưởng đằng sau việc fine-tuning các mô hình Transformer là chúng đã được huấn luyện trên một nguồn dữ liệu văn bản lớn, và do đó đã học được nhiều biểu diễn ngôn ngữ hữu ích. Bằng cách fine-tuning mô hình trên một nhiệm vụ mới, mô hình có thể sử dụng các biểu diễn đã được học trước đó như một điểm khởi đầu tốt, và học thông tin cụ thể cho nhiệm vụ mới từ dữ liệu nhiệm vụ mới đó.
* Quá trình fine-tuning một mô hình Transformer bao gồm việc mở khóa (unfreeze) một số hoặc tất cả các lớp của mô hình đã được huấn luyện trước và huấn luyện chúng trên dữ liệu nhiệm vụ mới bằng cách sử dụng một hàm mất mát cụ thể cho nhiệm vụ đó. Các lớp còn lại có thể được giữ nguyên, bảo toàn các biểu diễn đã học trước đó và ngăn chặn hiện tượng quá mức học trên dữ liệu nhiệm vụ nhỏ.
  1. Làm thế nào để fine - tune mô hình transformers:

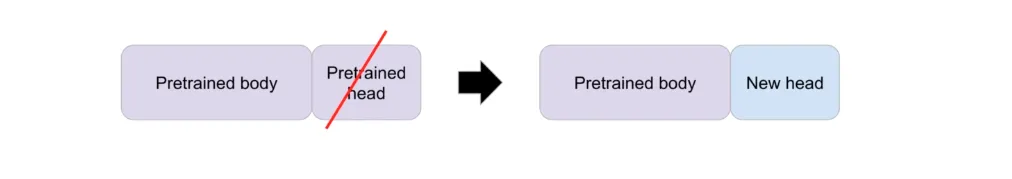
Fine-tuning một mô hình Transformer cho một nhiệm vụ cụ thể thường bao gồm các bước sau:

1. Chuẩn bị dữ liệu cho nhiệm vụ cụ thể.
2. Tách từ dữ liệu.
3. Lựa chọn một mô hình đã được huấn luyện trước.
4. Xác định kiến trúc fine-tuning.
5. Biên dịch mô hình.
6. Huấn luyện mô hình.
7. Đánh giá mô hình.

Bằng cách tuân theo các bước này, bạn có thể fine-tune một mô hình Transformer cho một nhiệm vụ cụ thể, tận dụng các biểu diễn đã được học trước của mô hình để đạt được hiệu suất cao với dữ liệu cụ thể cho nhiệm vụ đó.

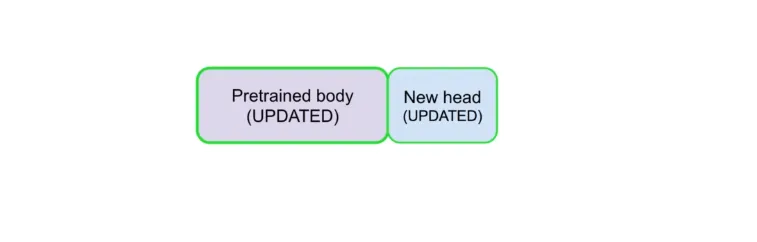
* 1. Tiếp cận mô mô hình để tinh chỉnh:
  2. Cát bỏ lớp cuối và thêm một lớp mới:

Phương pháp này thường được sử dụng khi nhiệm vụ mà mô hình transformer đang được fine-tuning là khác biệt so với nhiệm vụ mà nó được huấn luyện trước. Trong phương pháp này, lớp cuối cùng của mô hình transformer đã được huấn luyện trước được loại bỏ, và một lớp mới được thêm vào để phù hợp với yêu cầu cụ thể của nhiệm vụ đích. Lớp mới sau đó được huấn luyện từ đầu trên dữ liệu của nhiệm vụ đích, trong khi phần còn lại của mô hình được giữ nguyên. Ý tưởng đằng sau phương pháp này là bảo toàn các biểu diễn đã học được của mô hình đã được huấn luyện trước, có thể hữu ích cho nhiệm vụ đích, và chỉ cập nhật lớp cuối cùng để đưa ra dự đoán cho nhiệm vụ mới.



* 1. Fine – tune toàn bộ mọi thứ:

Trong phương pháp này, tất cả các tham số của mô hình transformer đã được huấn luyện trước đều được cập nhật trong quá trình huấn luyện trên dữ liệu của nhiệm vụ đích. Điều này thường được thực hiện khi nhiệm vụ đích tương tự với nhiệm vụ mà mô hình đã được huấn luyện trước, và các biểu diễn đã học được của mô hình đã được huấn luyện trước có thể được fine-tune cho nhiệm vụ đích. Trong quá trình fine-tuning, thường sử dụng một tỷ lệ học nhỏ hơn để tránh làm mất các biểu diễn đã học được từ bước huấn luyện trước. Phương pháp này có thể dẫn đến hiệu suất tốt hơn so với phương pháp trước vì toàn bộ mô hình được tối ưu hóa cho nhiệm vụ đích.



* 1. Practical examples:

Đoạn mã này thực hiện quá trình fine-tuning một mô hình ngôn ngữ GPT-2 trên cuốn sách "Harry Potter and the Sorcerer's Stone" bắt đầu ở 3 chương đầu 1, 2 và 3 nhằm tạo ra một mô hình có khả năng tạo ra văn bản mới phù hợp với ngữ cảnh và phong cách của tác phẩm gốc. Dữ liệu được sử dụng là các đoạn văn bản từ cuốn sách này, được thu thập từ các tệp tin .txt trong một thư mục cụ thể. Sau khi fine-tuning, mô hình có thể được sử dụng để sinh văn bản mới, mở rộng và mở rộng thêm thế giới và câu chuyện của tác phẩm gốc.

* + - * 1. Đọc dữ liệu từ thư mục chỉ định – tạo tập tin huán luyện.
* Tạo tập tinh huấn luyện được load trên drive:

# Sử dụng hàm read\_documents\_from\_directory để đọc tất cả các tập tin .txt trong một thư mục cụ thể

train\_directory = '/content/drive/MyDrive/dataset\_dictionary'

text\_data = read\_documents\_from\_directory(train\_directory)

text\_data = re.sub(r'\n+', '**\n**', text\_data).strip() # Remove excess newline characters

**with** open("/content/drive/MyDrive/dataset\_dictionary/train.txt", "w") **as** f:

f.write(text\_data)

**def** **load\_dataset**(file\_path, tokenizer, block\_size = **128**):

dataset = TextDataset(

tokenizer = tokenizer,

file\_path = file\_path,

block\_size = block\_size,

)

**return** dataset

* Sau khi tiến hành các lớp cần thiết và các thư viện. Tạo hàm ‘load\_data\_collator’ để bắt đầu tạo ra đối tượng từ DataCollatorFor-LanguageModeling để định dạng dữ liệu đầu vào của quá trình train.

**def** **load\_data\_collator**(tokenizer, mlm = False):

data\_collator = DataCollatorForLanguageModeling(

tokenizer=tokenizer,

mlm=mlm,

)

**return** data\_collator

* + - * 1. Huấn luện mô hình:
* Tạo hàm train() trong đó gồm các bước như tokenizer để tiền xữ lý dữ liệu đầu vào, các tokenizer, địa chỉ để lưu tokenizer và lưu model sau khi được train, tạo ra cấc thông số traning arguments để huấn luyện.

**def** **train**(train\_file\_path,model\_name,

output\_dir,

overwrite\_output\_dir,

per\_device\_train\_batch\_size,

num\_train\_epochs,

save\_steps):

tokenizer = GPT2Tokenizer.from\_pretrained(model\_name) # Tạo và load tokenizer từ model

train\_dataset = load\_dataset(train\_file\_path, tokenizer) # Load dataset

data\_collator = load\_data\_collator(tokenizer) # Định đạng và chuẩn bị dữ liệu cho quá trình train

tokenizer.save\_pretrained(output\_dir) # Lưu tokenizer vào output

model = GPT2LMHeadModel.from\_pretrained(model\_name) # Tạo và load model từ model\_name

model.save\_pretrained(output\_dir) # Lưu model vào output

training\_args = TrainingArguments(

output\_dir=output\_dir,

overwrite\_output\_dir=overwrite\_output\_dir,

per\_device\_train\_batch\_size=per\_device\_train\_batch\_size,

num\_train\_epochs=num\_train\_epochs,

)

trainer = Trainer( # Tạo đối tượng trainer để qly quá trình train

model=model, # model huấn luyện

args=training\_args, # Tạo ra các thông số để train mô hình

data\_collator=data\_collator, # Chuẩn bị dữ liệu để huấn luyện

train\_dataset=train\_dataset, # Dữ liệu được train từ tập train dataset

)

trainer.train()

trainer.save\_model()

* Tạo ra các thông số để huấn luyện và bắt đầu huấn luyện:

train\_file\_path = "/content/drive/MyDrive/dataset\_dictionary/train.txt" # Đường đãn đến file train

model\_name = 'gpt2' # Tên model

output\_dir = '/content/drive/MyDrive/dataset\_dictionary/chat\_model' # Đường dẫn đến mục lưu dữ liệu

overwrite\_output\_dir = False # Thiết lập không ghi đè, đảm bảo cho thư mục model trong drive empty

per\_device\_train\_batch\_size = **8** # Kích thước huấn luyện cho mỗi máy - mỗi thiết bị sẽ xử lý 8 batch

num\_train\_epochs = **60.0** # Số epoch

save\_steps = **50000**

* + - * 1. Gọi mô hình và kiểm tra dữ liệu thực tế.
* Sử dụng model sau khi được train xong và áp dụng hàm generate\_text để sinh ra văn bản mới.

**def** **generate\_text**(model\_path, sequence, max\_length):

model = load\_model(model\_path) # Tải model

tokenizer = load\_tokenizer(model\_path) # Tải tokenizer

ids = tokenizer.encode(f'{sequence}', return\_tensors='pt') # mã hóa các input -> tokenizer

final\_outputs = model.generate(

ids,

do\_sample=True,

max\_length=max\_length,

pad\_token\_id=model.config.eos\_token\_id,

top\_k=**50**,

top\_p=**0.95**,

)

**print**(tokenizer.decode(final\_outputs[**0**], skip\_special\_tokens=True))

* Sau khi load model xong chạy các dữ liệu câu hỏi liên quan đến đoạn dataset về chương 1, 2, 3

model1\_path = "/content/drive/MyDrive/dataset\_dictionary/chat\_model"

sequence1 = "[Q] Who is Mr. Dursley?"

max\_len = **50**

generate\_text(model1\_path, sequence1, max\_len)

model1\_path = "/content/drive/MyDrive/dataset\_dictionary/chat\_model"

sequence1 = "[Q] Who is Mr. Dursley?"

max\_len = **50**

generate\_text(model1\_path, sequence1, max\_len)

sequence2 = "Who live in Privet Drive?"

max\_len = **50**

generate\_text(model1\_path, sequence2, max\_len)

sequence3 = "Mrs. Dursley is "

max\_len = **50**

generate\_text(model1\_path, sequence3, max\_len)

Kết quả lần lượt là:

* + Ouput 1.1:

[Q] Who is Mr. Dursley? What did you say?'

'Nothing, nothing'

Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a big,

* + Output 1.2:

[Q] Who is Mr. Dursley? And how is his name?''

Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly

* + Output 2:

Who live in Privet Drive?" cried Professor McGonagall, jumping to her feet and pointing at number four.

"Dumbledore -- you can't. I've been watching them all day. You couldn't find two people who are less

* + Output 3:

Mrs. Dursley is

"a very nice lady," said Mr. Dursley stiffly, though he wasn't sure the cat was really stupid. It might have been hiding in Mrs. Dursley's dresser.

Ta có phân tích tại sao lại có 1.1 và 1.2, 1.1 là câu hỏi khi generate 1 lần tuy nhiên kết quả đầu tiên lại cho ra kết quả với độ tương khớp không cao với đoạn dataset fine tuning trên đó, nhưng sau nhiều lần chạy lại thì kết quả hai đã sinh ra thông tin liên quan đến câu trả lời. Kết luận ví dụ về fine – tuning ở trên chưa được tối ưu và cần cải thiện thêm nhiều để đạt hiệu quả tối đa.

# **REFERENCES**

BERT

1. <https://phamdinhkhanh.github.io/2020/05/23/BERTModel.html>
2. <https://phamdinhkhanh.github.io/2019/06/18/AttentionLayer.html>
3. <https://nlp.seas.harvard.edu/annotated-transformer/>
4. <https://arxiv.org/abs/1810.04805>
5. <https://arxiv.org/abs/1706.03762>
6. <https://github.com/google-research/bert>

GPT-2

1. <https://viblo.asia/p/tu-transformer-den-language-model-bai-1-bat-dau-voi-kien-truc-mo-hinh-transformer-38X4EN1gJN2>
2. <https://cameronrwolfe.substack.com/p/language-models-gpt-and-gpt-2>
3. <https://jalammar.github.io/illustrated-gpt2/>