TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KÌ

MÔN HỌC SÂU

ĐỒ ÁN CUỐI KÌ MÔN HỌC SÂU

Người hướng dẫn: **GV. LÊ ANH CƯỜNG**

Người thực hiện: **TRẦN THỊ VẸN – 52100674**

NGUYỄN THANH TÚ – 52100349

VÕ LUYỆN - 52100911

Lớp: **21050301**

Khoá: **25**

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

BÁO CÁO CUỐI KÌ

MÔN HỌC SÂU

ĐỒ ÁN CUỐI KÌ MÔN HỌC SÂU

Người hướng dẫn: **GV. LÊ ANH CƯỜNG**

Người thực hiện: **TRẦN THỊ VẸN – 52100674**

NGUYỄN THANH TÚ – 52100349

VÕ LUYỆN - 52100911

Lớp: **21050301**

Khoá: **25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Trong suốt quá trình học tập và rèn luyện, chúng em đã nhận được rất nhiều sự giúp đỡ tận tình, sự quan tâm, chăm sóc của thầy Lê Anh Cường. Ngoài ra, chúng em còn được thầy truyền đạt những kiến thức về xử lý ảnh hay ho và thú vị, thầy cô còn giúp sinh viên có được nhiều niềm vui trong việc học và cảm thấy thoải mái, … Chúng em xin chân thành cảm ơn các thầy cô rất nhiều trong suốt quá trình học tập này!

Bởi lượng kiến thức của chúng em còn hạn hẹp và gặp nhiều vấn đề trong quá trình học nên báo cáo này sẽ còn nhiều thiếu sót và cần được học hỏi thêm. Chúng em rất mong em sẽ nhận được sự góp ý của quý thầy cô về bài báo cáo này để chúng em rút kinh nghiệm trong những môn học sắp tới. Cuối cùng, chúng em xin chân thành cảm ơn quý thầy cô.

TP Hồ Chí Minh, ngày 20 tháng 04 năm 2024

Sinh viên:

Trần Thị Vẹn – 52100674

Nguyễn Thanh Tú – 52100349

Võ Luyện - 52100911

BÁO CÁO CUỐI KÌ ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

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

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

TP. Hồ Chí Minh, ngày 20 tháng 04 năm 2024

Tác giả

(ký tên và ghi rõ họ tên)

Trần Thị Vẹn

Nguyễn Thanh Tú

Võ Luyện

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

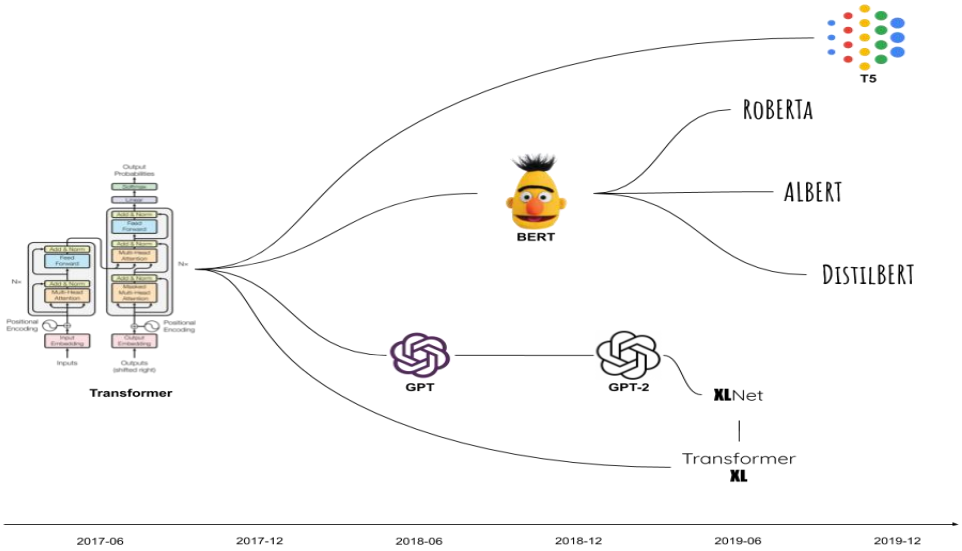
TÓM TẮT

MỤC LỤC

DANH MỤC HÌNH ẢNH, BẢNG BIỂU VÀ ĐỒ THỊ

Danh mục hình ảnh

CHƯƠNG 1 - TRANSFORMER BASE ENCODER



Hình 1.1 Sơ đồ phát triển

GPT là viết tắt của "Generative Pre-trained Transformer", được phát triển bởi OpenAI để tạo ra văn bản giống con người từ các đầu vào cho trước. Nó sử dụng một mô hình ngôn ngữ đã được đào tạo trước trên các tập dữ liệu lớn về văn bản để tạo ra các đầu ra có tính thực tế dựa trên các yêu cầu từ người dùng. Một ưu điểm của GPT so với các mô hình học sâu khác là khả năng tạo ra các chuỗi văn bản dài mà không làm giảm độ chính xác hoặc sự mạch lạc. Ngoài ra, nó cũng có thể được sử dụng cho nhiều nhiệm vụ, bao gồm dịch và tóm tắt.

BERT, viết tắt của Bidirectional Encoder Representations from Transformers, được phát triển bởi nhóm Google AI Language và được công bố mã nguồn mở vào năm 2018. Khác với GPT, mà chỉ xử lý đầu vào từ trái sang phải như cách con người đọc từ, BERT xử lý đầu vào cả từ trái sang phải và từ phải sang trái để hiểu ngữ cảnh của các văn bản cho trước. Hơn nữa, BERT cũng đã được chứng minh là vượt trội hơn so với các mô hình NLP truyền thống như LSTM trên các nhiệm vụ liên quan đến hiểu ngôn ngữ tự nhiên.

Tuy nhiên, có một sự khác biệt bổ sung trong cách BERT và GPT được đào tạo:

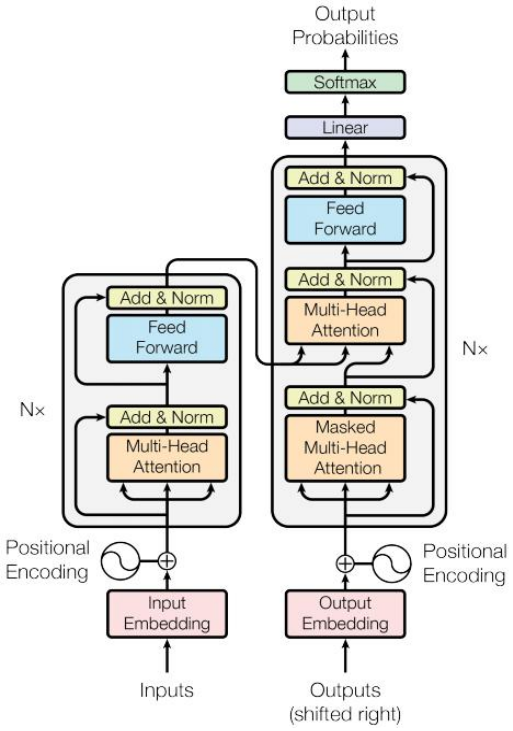
BERT là một bộ mã hóa transformer, có nghĩa là, cho mỗi vị trí trong đầu vào, đầu ra tại cùng một vị trí là cùng một token (hoặc token [MASK] cho các token đã được che). Các mô hình chỉ có một ngăn xếp bộ mã hóa như BERT tạo ra tất cả các đầu ra của mình cùng một lúc.

GPT là một bộ giải mã transformer tự sinh, có nghĩa là mỗi token được dự đoán và điều kiện trên token trước đó. Chúng ta không cần một bộ mã hóa, vì các token trước đó được nhận bởi bộ giải mã chính nó. Điều này khiến cho các mô hình này rất tốt trong các nhiệm vụ như sinh ngôn ngữ, nhưng không tốt trong phân loại. Những mô hình này có thể được đào tạo với các tập dữ liệu văn bản lớn không có nhãn từ sách hoặc bài viết trên web.

Điều đặc biệt về các mô hình transformer là cơ chế chú ý, cho phép các mô hình này hiểu sâu hơn về ngữ cảnh của các từ.

1.1 BERT

Transformer: Attention is all you need cùng với cơ chế Attention [Vaswani et al., 2017) đã cách mạng hoá việc xử lý các dữ liệu dạng sequence (chuỗi thời gian) như time series, text, music.... Bằng cách khai thác khả năng tính toán song song tương tự như các mô hình CNNs, đồng thời mã hóa vị trí của từng phần tử trong chuỗi, các tokens sẽ Attend (quan sát) lẫn nhau và cố gắng "hiểu" nhau trong ngữ cảnh của các tokens xung quanh thay vì phải tính toán hồi quy.



Hình 1.2 Transformer

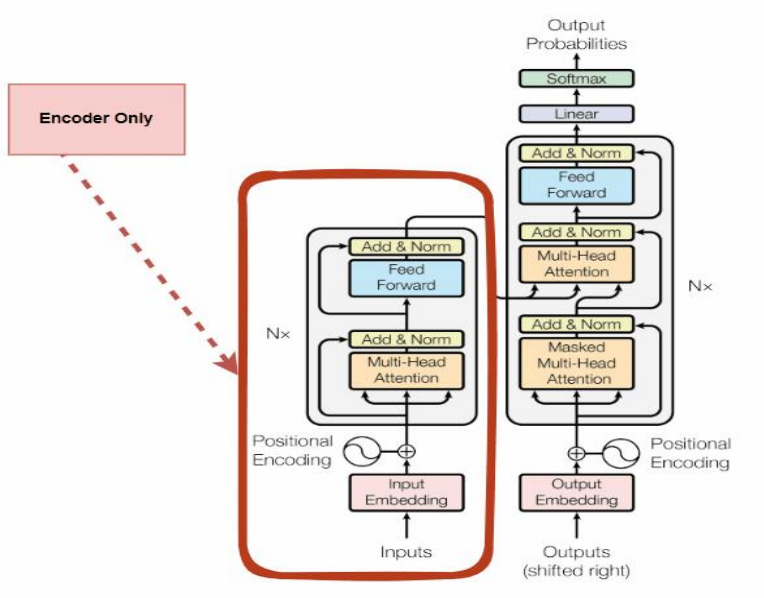
Mô hình sẽ bao gồm 2 phase.

- Encoder: Bao gồm 6 layers liên tiếp nhau. Mỗi một layer sẽ bao gồm một sub-layer 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.

Cơ chế Attention đã khắc phục được các nhược điểm lớn của các mô hình dạng hồi quy như RNN, LSTM.... Cụ thể:

• Các mạng CNN có thể dễ dàng được thực hiện song song ở một tầng nhưng không có khả năng nắm bắt các phụ thuộc chuỗi có độ dài biến thiên.

Các mạng RNN có khả năng nắm bắt các thông tin cách xa nhau trong chuỗi có độ dài biến thiên, nhưng không thể thực hiện song song trong một chuỗi.



Hình 1.3 Encoder only Transformer

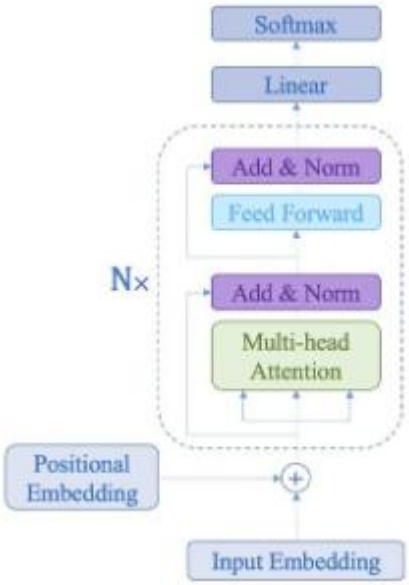
Trasformer cùng cơ chế Attention đã kết hợp các ưu điểm của CNN và RNN, từ đó giúp cho quá trình huấn luyện nhanh hơn và kết quả đạt được cũng tốt hơn.

BERT: Bidirectional Encoder Representations from Transformers

- **BERT 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.**
- 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 mô hình 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.

Kiến trúc của mô hình BERT là một kiến trúc đa tầng gồm nhiều lớp Bidirectional Transformer encoder. Đúng như tên gọi BERT về bản chất chính là khối Encoder của kiến trúc Transformer

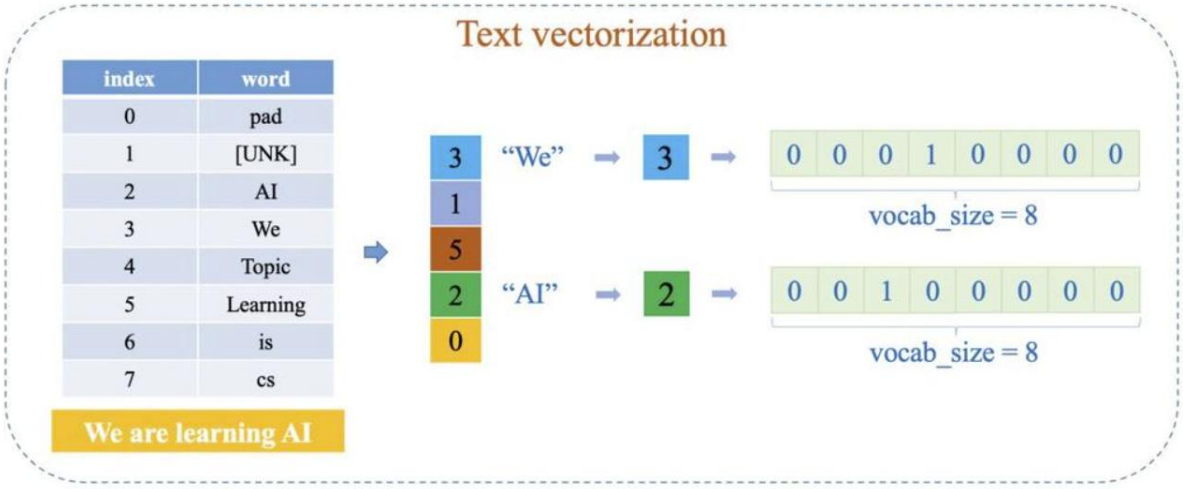


Hình 1.4 Kiến trúc BERT

Các bộ phận cấu thành:

1.1.1 Embedding Layer

Trong biểu diễn one-hot của một ngôn ngữ, mỗi từ có một phần tử vector có độ lớn bằng kích thước của tập từ vựng. Đối với tập từ vựng 8 từ, vector biểu diễn cho mỗi từ cũng sẽ có kích thước là 8.

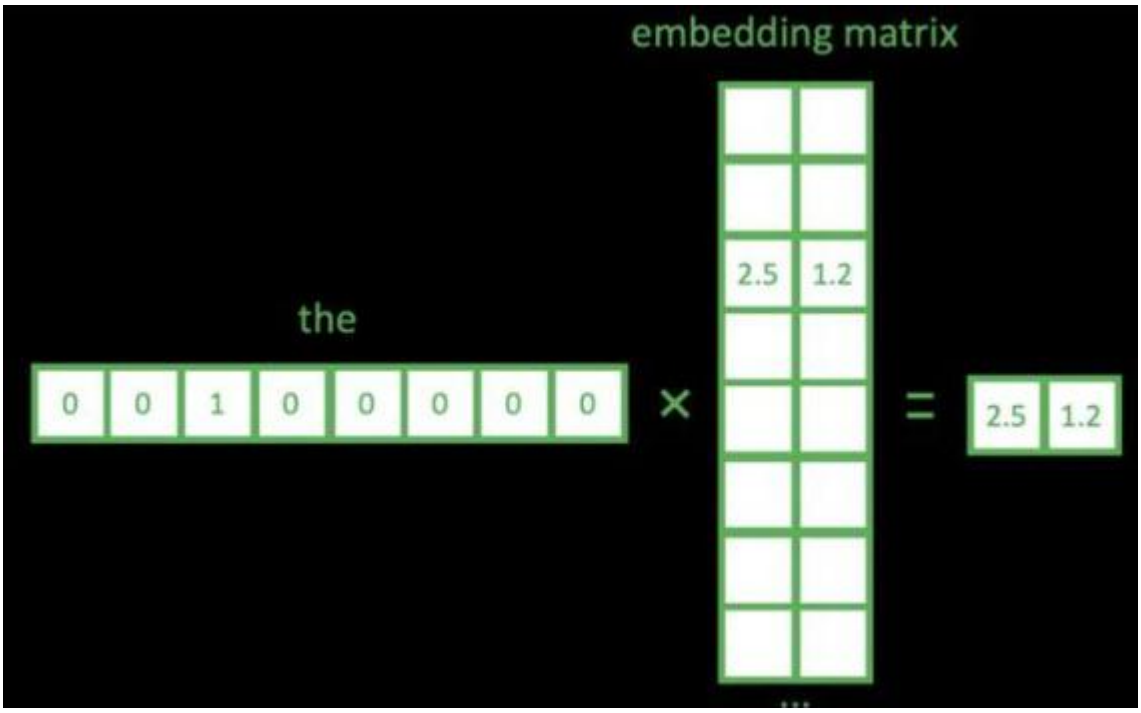


Hình 1.5 Text vectorization

Khi tập từ vựng lớn (VD: 500k từ, vector biểu diễn cho mỗi từ cũng sẽ có kích thước là 500k. Những vector này khiến cho việc tính toán trở nên kém hiệu quả.

Trong lớp Embedding, những vector này sẽ được ánh xạ xuống một vùng không gian có số chiều thấp hơn. Điều này giúp công việc tính toán nhẹ nhàng hơn tuy nhiên số chiều được ánh xạ càng nhỏ thì thông tin bị thất thoát càng nhiều.

Thông qua phép nhân ma trận, một ma trận ánh xạ có thể chuyển tập vector one-hot ban đầu thành mọi kích thước mong muốn. Thông qua huấn luyện, mô hình có thể học cách biểu diễn ngôn ngữ một cách tốt nhất có thể (những từ có liên quan được nhóm gần với nhau).



Hình 1.6 Positional Encoding

1.1.2 Positional Encoding

Sự xuất hiện của kiến trúc Attention giúp những mô hình tương tự với Transformer loại bỏ hoàn toàn cấu trúc hồi quy có trong những mô hình RNNs. Những token đầu vào được truyền đồng thời vào mô hình, chính vì vậy bản thân mô hình không có khái niệm về vị trí/ thứ tự giữa các token.

Để khắc phục vấn đề này, chúng ta cần truyền thêm thông tin về vị trí của các token thông qua Positional Encoding. Có nhiều phương pháp khác nhau để khởi tạo Positional Encoding, ở đây chúng ta sẽ giới thiệu hàm Sinusoid theo như bản gốc của bài báo Transformer.

Có thể hiểu một cách đơn giản, hàm Sinusoid giúp xê dịch các vector embeding của các tokens một cách độc nhất. Phương trình của hàm này như sau:

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

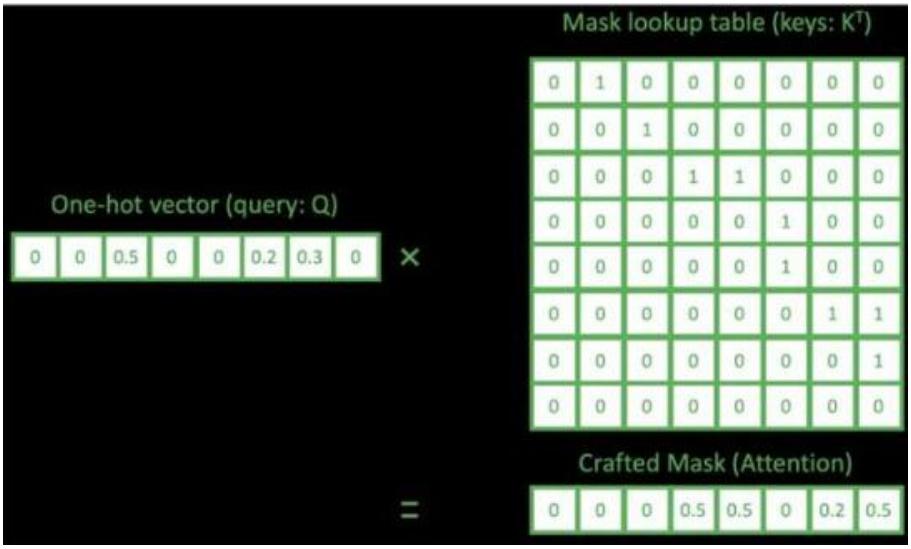
1.1.3 Attention Block

1.1.3.1 Attention

Attention là phép nhân ma trận Cơ chế Attention là trái tim của toàn bộ mô hình Transformer và các biến thể của Trans- former. Phương trình toán học của cơ chế này được mô tả bởi công thức sau:

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

Thông qua phép nhân ma trận Q * K ^ T mô hình có thể chọn lọc những tokens nào cần tập trung và những tokens nào ít quan trọng hơn. Ví dụ sau đây:



Khi chúng ta nhân 1 vector được lấy từ ma trận Q với ma trận K ^ T chúng ta sẽ được kết quả là tổng của ba hàng 3,6,7 cùng với các trọng số tương ứng 0.5,0.2, 0.3.

Chúng ta có thể diễn giải kết quả của phép nhân này như sau: Query Q bảo mô hình tạo ra một Key mask mới từ tổng của 50% vector thứ 3, 20% vector thứ 6, và 30% vector thứ 7. Trên thực tế ma trận Q và ma trận K ^ T đều được học từ mô hình, kết quả của phép nhân này cho cúng ta một bộ những Key masks. Chính những Key masks này sẽ cho mô hình biết được cần tập trung (Attention) vào những Value: V nào.

1.1.3.2 Hàm Softmax

Sau phép nhân ma trận Q * K ^ T kết quả trả về thường không tuân theo một phân phối xác suất (Kết quả của hình minh hoạ phía trên có tổng lớn hơn 1). Việc này làm cho mô hình không ổn định. Hàm Softmax có tác dụng kéo những giá trị về một phân phối xác suất có tổng bằng 1.

1.1.3.3 Scaling

Hàm Softmax rất nhạy cảm với những tập giá trị có phương sai lớn. Trong một số trường hợp, sử dụng hàm softmax không đúng cách sẽ dẫn đến mô hình học chậm, hay thậm chí dừng việc học. Việc chia Q * K ^ T cho sqrt(d_{k}) với d_{k} là số chiều (dimension) của vector kết quả giúp giảm phương sai của các giá trị về gần bằng 1 (Phân phối Standard Normal)

1.1.4 Mutihead-Attention

Có 2 ưu điểm chính khi sử dụng Mutihead-Attention:

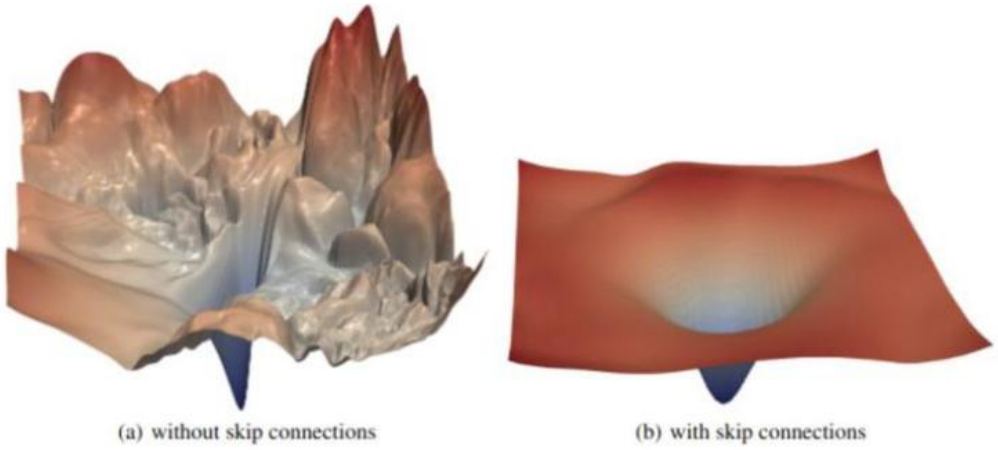
- Thứ nhất là hàm Softmax thường kéo sự tập trung về một hoặc hai giá trị. Bằng cách dùng nhiều heads mô hình có thể nhìn thấy được mối tương quan giữa nhiều tokens hơn.
- Mỗi một head sẽ có ba bộ ma trận Q, K, V khác nhau và trong quá trình huấn luyện các ma trận này sẽ học cách mô tả những mối tương quan khác (VD: từ học sinh' vừa liên quan đến 'người cũng vừa liên quan đến trường học')

1.1.5 Skip connection - Normalization

1.1.5.1 Skip connection

Skip connection giúp cho dòng chảy đạo hàm trong mô hình được trơn tru hơn. Nếu như có bất kỳ vấn đề nào xảy ra trong quá trình tính toán đạo hàm lan truyền ngược ở trục mô hình chính, Skip connection giống như một con đường tắt nhằm đảm bảo mô hình vẫn có thể tiếp tục huấn luyện hiệu quả. Điều này càng đúng hơn đối với những mô hình 'sâu' như Transformer: có nhiều khối Encoder, Decoder chồng lên nhau, nếu có bất kỳ khối nào hoạt động không hiệu quả Skip connection sẽ giúp đảm bảo

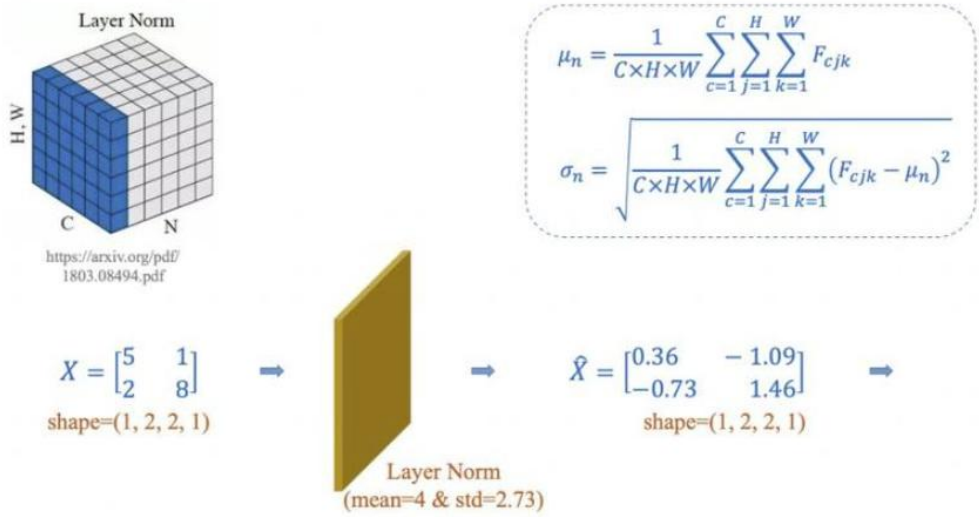
dòng chảy đạo hàm được ổn định đồng thời giữ lại được những thông tin gốc để truyền vào những khối phía sau.



Hình 1.7 Tác dụng của Skip connections

1.1.5.2 Layer Norm

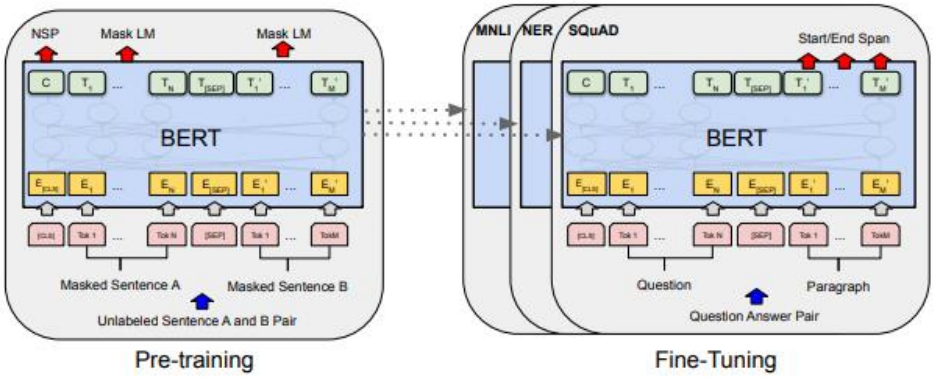
Các hàm như Softmax, ReLU rất nhạy cảm với sự phân bố dữ liệu. Mục đích của Layer Norm là chuyển dịch dữ liệu về phân phối chuẩn - Standard Normal (trung bình = 0, độ lệch chuẩn = 1, việc này giúp cho mô hình hoạt động ổn định hơn.



Hình 1.8 Layer Norm

1.2 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 1.9 Pre-training và fine-tuning của Bert

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:

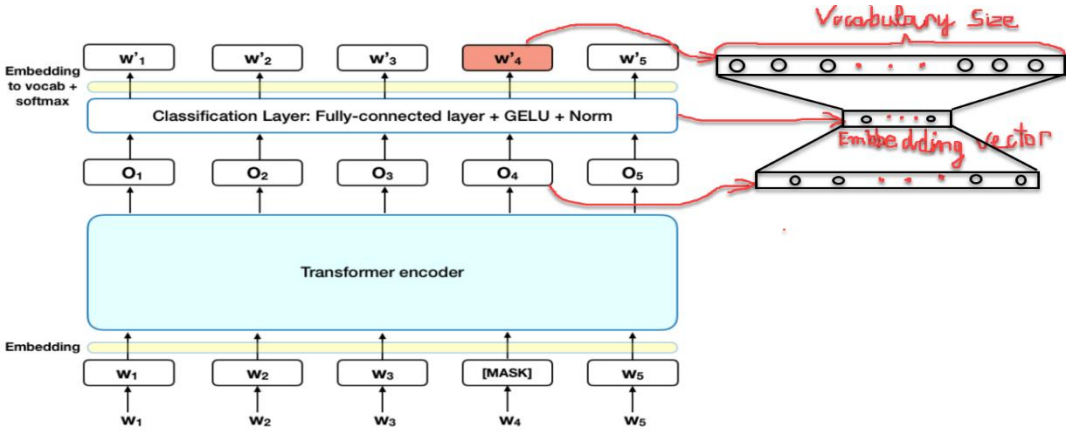
- Bước 1: Embedding toàn bộ các token của cặp câu bằng các véc tơ nhúng từ pretrain model. Các token embedding bao gồm cả 2 token là [CLS] và [SEP] để đánh dấu vị trí bắt đầu của câu hỏi và vị trí ngăn cách giữa 2 câu. 2 token này sẽ được dự báo ở output để xác định các phần Start/End Spand của câu output.
- 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.
- 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.

1.3 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.



Hình 1.10 Sơ đồ kiến trúc BERT cho tá vụ Masked ML

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 feed forward ta sẽ thu được các véc tơ embedding ở output là $O_1,O_2,\ldots O_5$.
- Để tính toán phân phối xác suất cho từ output, chúng ta thêm một Fully connect 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 O_i 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.

1.4 Next Sentence Prediction (NSP)

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.

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 E_A là nếu từ thuộc câu thứ nhất và E_B nếu từ thuộc câu thứ hai.
- Vị trí của từ trong câu (position embedding): là các véc tơ $E_0,\ldots,E_{10}$. 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.

Các kiến trúc model BERT

Hiện tại có nhiều phiên bản khác nhau của model BERT. Các phiên bản đều dựa trên việc thay đổi kiến trúc của Transformer tập trung ở 3 tham số:

L: số lượng các block sub-layers trong transformer,

H: kích thước của embedding véc tơ (hay còn gọi là hidden size),

A: Số lượng head trong multi-head layer, mỗi một head sẽ thực hiện một self-attention.

Tên gọi của 2 kiến trúc bao gồm:

$\mathbf{BERT_{BASE}}(L = 12, H = 768, A = 12)$: Tổng tham số 110 triệu.

$\mathbf{BERT_{LARGE}}(L = 24, H = 1024, A = 16)$: Tổng tham số 340 triệu.

Như vậy ở kiến trúc BERT Large chúng ta tăng gấp đôi số layer, tăng kích thước hidden size của embedding véc tơ gấp 1.33 lần và tăng số lượng head trong multi-head layer gấp 1.33 lần.

1.5 RoBERTa

RoBERTa được giới thiệu bởi Facebook là một phiên bản được huấn luyện lại của BERT với một phương pháp huấn luyện tốt hơn với dữ liệu được tăng gấp 10 lần.

Để tăng cường quá trình huấn luyện, RoBERTa không sử dụng cơ chế dự đoán câu kế tiếp (NSP) từ BERT mà sử dụng kỹ thuật mặt nạ động (dynamic masking), theo đó các token mặt nạ sẽ bị thay đổi trong quá trình huấn luyện. Sử dụng kích thước batch lớn hơn cho thấy hiệu quả tốt hơn khi huấn luyện.

RoBERTa xây dựng trên cơ sở của BERT bằng cách điều chỉnh các siêu tham số chính. Nó loại bỏ mục tiêu tiền huấn luyện cho câu tiếp theo và điều chỉnh quá trình huấn luyện với các lô nhỏ và tỷ lệ học lớn hơn.

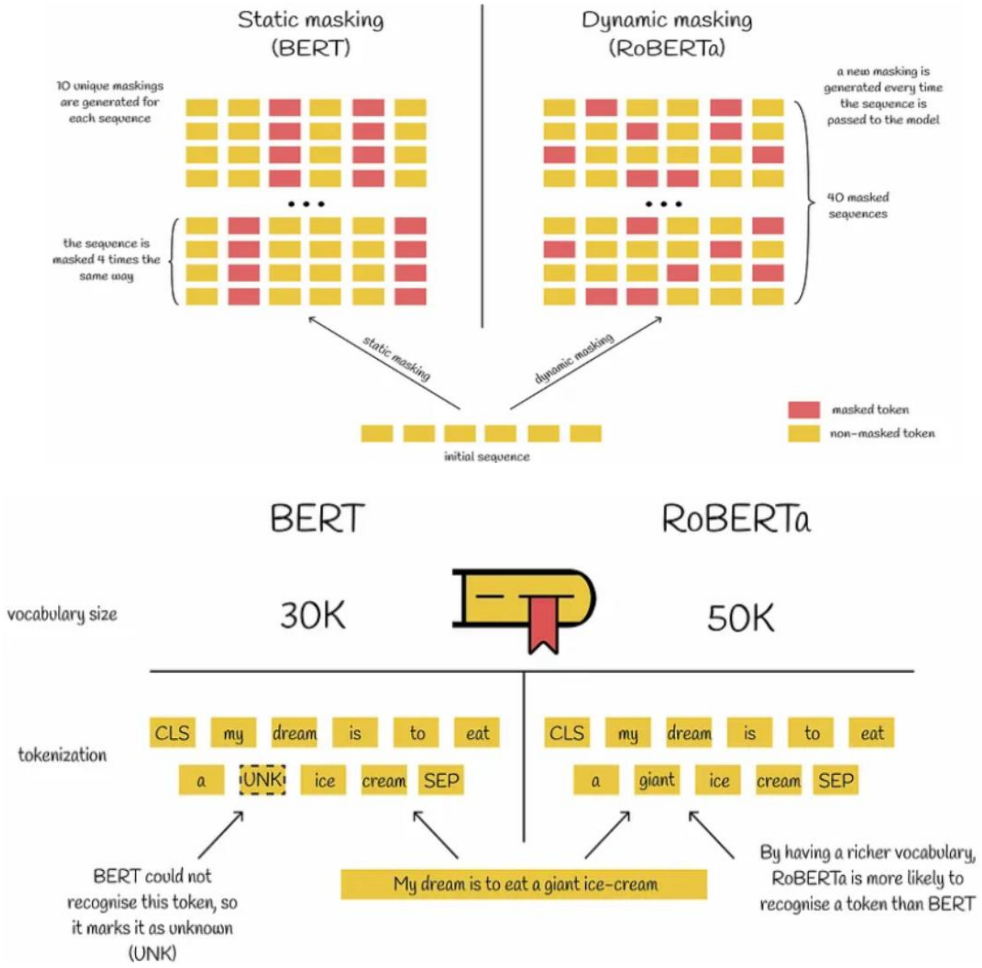
Một điều quan trọng nữa, RoBERTa sử dụng 160GB văn bản để huấn luyện. Trong đó, 16GB là sách và Wikipedia tiếng Anh được sử dụng trong huấn luyện BERT. Phần còn lại bao gồm CommonCrawl News dataset (63 triệu bản tin, 76 GB), ngữ liệu văn bản Web (38 GB) và Common Crawl Stories (31 GB). Mô hình này được huấn luyện với GPU của Tesala 1024 V100 trong một ngày.

Dưới đây là một số điểm khác biệt chính giữa RoBERTa và BERT:

- Đào tạo tối ưu: RoBERTa áp dụng các tối ưu hóa về siêu tham số và quá trình đào tạo so với BERT. Nó điều chỉnh các siêu tham số quan trọng như kích thước lô, tỷ lệ học, số lượng epoch và thời gian đào tạo để tạo ra một mô hình hiệu quả hơn.
- Loại bỏ mục tiêu tiền huấn luyện cho câu tiếp theo: BERT sử dụng mục tiêu tiền huấn luyện cho câu tiếp theo (Next Sentence Prediction - NSP) để cải thiện hiểu biết ngôn ngữ. Tuy nhiên, RoBERTa loại bỏ mục tiêu này và tập trung vào việc huấn luyện mô hình theo cách tối ưu hóa hơn.

- Kích thước dữ liệu huấn luyện: RoBERTa được đào tạo trên các tập dữ liệu lớn hơn so với BERT, giúp cải thiện hiệu suất của mô hình.
- Hiệu suất: RoBERTa đã được chứng minh là vượt trội hơn so với BERT trên nhiều nhiệm vụ NLP tiêu chuẩn như GLUE (General Language Understanding Evaluation) và SQuAD (Stanford Question Answering Dataset).

Tóm lại, RoBERTa là một biến thể tinh chỉnh của BERT, được tối ưu hóa để cải thiện hiệu suất và độ chính xác của mô hình trong các nhiệm vụ xử lý ngôn ngữ tự nhiên.



Hình 1.11 BERT và RoBERTa khác nhau

CHƯƠNG 2 - TRANSFORMER BASE GPT MODEL

2.1 LLaMa 1 là gì?

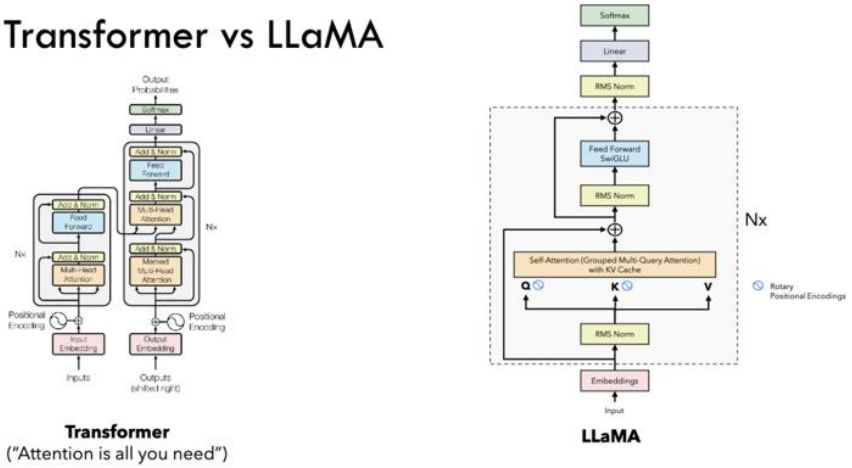
LLama 1 (LaMDA Language Model for Applications) là một mô hình ngôn ngữ lớn (LLM) được phát triển bởi Facebook AI Research, ra mắt vào tháng 1 năm 2023. Nó được xây dựng dựa trên kiến trúc Transformer và được huấn luyện trên một tập dữ liệu khổng lồ gồm văn bản và mã, bao gồm sách, bài báo, mã nguồn và các dạng văn bản khác.

LLaMa 1 là một mô hình ngôn ngữ lớn với 137B tham số, lớn hơn nhiều so với các mô hình ngôn ngữ khác như GPT-3 (175B tham số). Kích thước lớn hơn cho phép LLaMa 1 học hỏi nhiều thông tin hơn và xử lý ngôn ngữ một cách phức tạp hơn.

LLaMa 1 được huấn luyện trên một tập dữ liệu đa dạng gồm văn bản và mã, bao gồm sách, bài báo, mã nguồn và các dạng văn bản khác. Điều này giúp LLaMa 1 có khả năng xử lý nhiều loại văn bản khác nhau và thích ứng với nhiều ngữ cảnh.

LLaMa 1 sử dụng kiến trúc Transformer, một kiến trúc mạng nơ-ron tiên tiến được chứng minh là hiệu quả trong các nhiệm vụ xử lý ngôn ngữ tự nhiên.

LLama là một mô hình ngôn ngữ lớn (LLM) được xây dựng dựa trên kiến trúc Transformer, nhưng nó có một số điểm khác biệt so với các mô hình Transformer bình thường. Dưới đây là mô tả chi tiết về kiến trúc của LLaMa:



Hình 2.1 So sánh về kiến trúc của Transformer thông thường và LlaMA

Những cải tiến của một mô hình LLaMA so với kiến trúc Transformer gốc.

- Chuẩn hóa trước (Pre-normalization) [GPT3]:
  - Thay vì chuẩn hóa đầu ra của mỗi lớp con (sub-layer) trong Transformer, mô hình này chuẩn hóa đầu vào.
  - Điều này được cho là cải thiện độ ổn định của quá trình huấn luyện.
  - Hàm chuẩn hóa được sử dụng là RMSNorm.
- Hàm kích hoạt SwiGLU [PaLM]:
  - Thay thế hàm kích hoạt ReLU mặc định bằng SwiGLU.
  - Mục đích là cải thiện hiệu năng của mô hình.
- Rotary Embeddings [GPTNeo]:
  - Loại bỏ positional embeddings.
  - Thay vào đó, sử dụng mã hóa vị trí quay (RoPE).

Chi tiết về các siêu tham số (hyper-parameters) được cung cấp trong bảng sau:

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

Model sizes, architectures, and optimization hyper-parameters.

2.2 Kiến trúc của LLaMa

Chúng ta sẽ tìm hiểu về các kiến trúc cải tiến của mô hình LLaMa so với khung mô hình Transformers thông thường như RMS Norm, Rotary Positional Embedding, KV - Cache, Group Multi-Query Attention (GQA), SwiGLU Activation function:

2.2.1 RMS Norm

Một giải thích nổi tiếng về sự thành công của LayerNorm là thuộc tính không đổi về re-centering và re-scaling. Thuộc tính trước cho phép mô hình không nhạy cảm với nhiễu dịch chuyển trên cả đầu vào và trọng số, và thuộc tính sau giữ cho các biểu

diễn đầu ra nguyên vẹn khi cả đầu vào và trọng số đều được tỷ lệ ngẫu nhiên. Giả định rằng sự bất biến về re-scaling là nguyên nhân của sự thành công của LayerNorm, thay vì sự không đổi về re-centering.

Đề xuất RMS Norm chỉ tập trung vào tính bất biến về tỷ lệ lại và điều chỉnh tổng các đầu vào đơn giản theo thống kê trung bình bình phương căn (RMS):

RMS Norm (Root Mean Square Normalization) là một kỹ thuật chuẩn hóa được sử dụng trong mạng nơ-ron nhằm mục tiêu làm cho đầu ra của mỗi lớp mạng có phân phối trung bình bằng 0 và phương sai bằng 1. Kỹ thuật này giúp cải thiện độ ổn định của quá trình huấn luyện và tăng hiệu suất của mô hình.

RMS Norm đơn giản hóa LayerNorm bằng cách hoàn toàn loại bỏ thống kê trung bình trong công thức :

$$\mu = \frac{1}{n}\sum_{i=1}^{n} a_i, \quad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - \mu)^2}.$$

mà không cần phải hy sinh sự bất biến mà việc chuẩn hóa trung bình mang lại. Khi giá trị trung bình của tổng các đầu vào là không, RMS Norm chính xác bằng với LayerNorm. Mặc dù RMS Norm không căn giữa tổng các đầu vào như trong LayerNorm và thuộc tính này không phải là điều cơ bản cho sự thành công của LayerNorm cho nên RMS Norm cũng hiệu quả tương đương hoặc hơn.

Công thức toán học cho RMS Norm:

$$\bar{a}_i = \frac{a_i}{\text{RMS}(\mathbf{a})} g_i, \quad \text{where } \text{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}.$$

Ưu điểm của RMS Norm với LayerNorm:

- Ít yêu cầu tính toán hơn LayerNorm: không cần phải tính toán giá trị trung bình
- Hoạt động tốt trong thực tế: vì chúng ta cần tính re-scaling để đạt được hiệu quả như LayerNorm không cần tới re-centering và điều này tốt cho LLaMA.

2.2.2 Rotary Positional Embedding

Trong Transformer models, positional encoding là một kỹ thuật dùng để bổ sung thông tin về thứ tự của các từ trong một câu vào vector biểu diễn của chúng. Transformer models dựa trên kiến trúc self-attention, học mối quan hệ giữa các từ, nhưng chúng không tự học được thứ tự của các từ trong câu. Positional encoding giải quyết vấn đề này bằng cách cung cấp cho mô hình thông tin về vị trí tương đối hoặc tuyệt đối của mỗi từ.

Ví dụ: các cụm từ như "con chó đuổi con lợn " và "con lợn đuổi con chó", mặc dù có ý nghĩa khác nhau nhưng được xử lý không thể phân biệt được vì chúng được coi là một tập hợp các token không có thứ tự. Để duy trì thông tin trình tự và do đó có ý nghĩa, các phần embedding vị trí được tích hợp vào mô hình.

Các loại Positional Encoding:

2.2.2.1 Absolute Positional Encoding

Trong ngữ cảnh của một câu, giả sử chúng ta có một từ embedding đại diện cho một từ. Để mã hóa vị trí của nó, chúng tôi sử dụng một vectơ khác có chiều giống hệt nhau, trong đó mỗi vectơ biểu thị duy nhất một vị trí trong câu.

Ví dụ: một vectơ cụ thể được chỉ định cho từ thứ hai trong câu. Như vậy, mỗi vị trí câu có một vectơ riêng biệt. Sau đó, đầu vào cho lớp Transformer được hình thành bằng cách tính tổng từ embedding với vị trí embedding tương ứng của nó.

Chủ yếu có hai phương pháp để tạo ra các phần embedding này:

- Học từ dữ liệu: Ở đây, các vectơ vị trí được học trong quá trình huấn luyện, giống như các tham số mô hình khác. Chúng ta học một vectơ duy nhất cho mỗi vị trí, chẳng hạn từ 1 đến 512. Tuy nhiên, điều này đưa ra một hạn chế độ dài chuỗi tối đa bị giới hạn. Nếu mô hình chỉ học đến vị trí 512 thì nó không thể biểu diễn các chuỗi dài hơn thế.
- Hàm hình sin: Phương pháp này liên quan đến việc xây dựng các phần embedding duy nhất cho từng vị trí bằng cách sử dụng hàm hình sin. Mặc dù các chi tiết phức tạp của cấu trúc này rất phức tạp nhưng về cơ bản nó cung cấp

khả năng embedding vị trí duy nhất cho mọi vị trí trong một chuỗi. Các nghiên cứu thực nghiệm đã chỉ ra rằng việc học từ dữ liệu và sử dụng các hàm hình sin mang lại hiệu suất tương đương trong các mô hình trong thế giới thực.

$$p_{i,j} = \begin{cases} \sin\left(\frac{i}{10000^{\frac{j}{d_{emb\_dim}}}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{\frac{j-1}{d_{emb\_dim}}}}\right) & \text{if } j \text{ is odd} \end{cases}$$

Trong đó i là vị trí của từ trong câu, j là vị trí ô trong vecto p.

Mặc dù được sử dụng rộng rãi nhưng việc embedding vị trí tuyệt đối không phải là không có nhược điểm:

- Độ dài chuỗi giới hạn: Như đã đề cập, nếu một mô hình học vectơ vị trí đến một điểm nhất định, thì nó không thể biểu thị các vị trí vượt quá giới hạn đó.
- Tính độc lập của các phần embedding theo vị trí: Mỗi phần embedding theo vị trí đều độc lập với các phần embedding khác. Điều này có nghĩa là theo quan điểm của mô hình, sự khác biệt giữa vị trí 1 và 2 giống như giữa vị trí 2 và 500. Tuy nhiên, về mặt trực giác, vị trí 1 và 2 phải có liên quan chặt chẽ hơn so với vị trí 500, vốn ở xa hơn đáng kể. Việc thiếu vị trí tương đối này có thể cản trở khả năng hiểu các sắc thái của cấu trúc ngôn ngữ của mô hình.

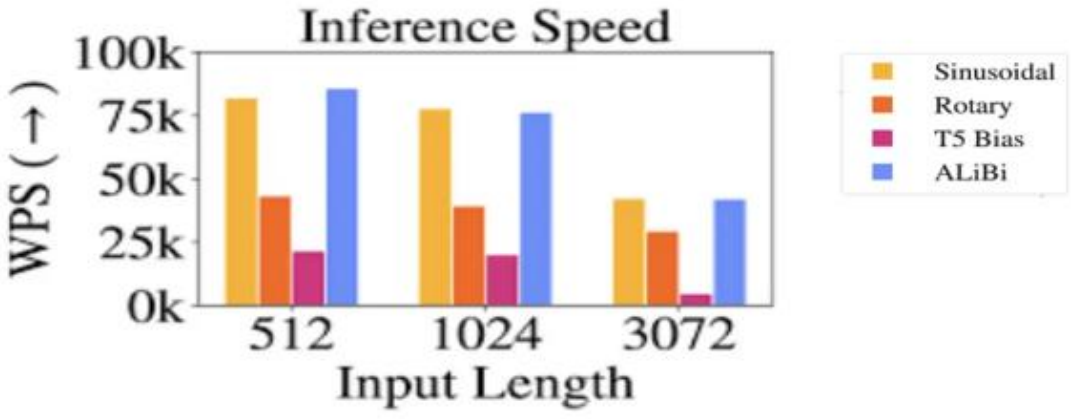
2.2.2.2 Relative Positional Encoding

Thay vì tập trung vào vị trí tuyệt đối của token trong câu, phần embedding vị trí tương đối tập trung vào khoảng cách giữa các cặp token. Phương pháp này không thêm trực tiếp vectơ vị trí vào vectơ từ. Thay vào đó, nó thay đổi cơ chế attention để kết hợp thông tin vị trí tương đối.

Nghiên cứu điển hình: Mô hình T5

Một mô hình nổi bật sử dụng các phần embedding vị trí tương đối là T5 (Biến áp chuyển văn bản thành văn bản). T5 giới thiệu một cách xử lý thông tin vị trí theo sắc thái:

- Độ lệch cho độ lệch vị trí: T5 sử dụng độ lệch, số dấu phẩy động, để biểu thị từng độ lệch vị trí có thể có. Ví dụ: độ lệch B1 có thể biểu thị khoảng cách tương đối giữa hai token bất kỳ cách nhau một vị trí, bất kể vị trí tuyệt đối của chúng trong câu.
- Tích hợp trong lớp tự attention: Ma trận độ lệch vị trí tương đối này được thêm vào tích của ma trận truy vấn và ma trận chính trong lớp tự attention. Điều này đảm bảo rằng các token ở cùng khoảng cách tương đối luôn được biểu thị bằng cùng một độ lệch, bất kể vị trí của chúng trong chuỗi.
- Khả năng mở rộng: Một lợi thế đáng kể của phương pháp này là khả năng mở rộng của nó. Nó có thể mở rộng thành các chuỗi dài tùy ý, một lợi ích rõ ràng so với việc embedding vị trí tuyệt đối.
- Bất chấp sự hấp dẫn về mặt lý thuyết của chúng, việc embedding vị trí tương đối đặt ra thách thức thực tế nhất định.
- Vấn đề về hiệu suất: Điểm chuẩn so sánh khả năng embedding tương đối của T5 với các loại khác đã cho thấy rằng chúng có thể chậm hơn, đặc biệt đối với các chuỗi dài hơn. Điều này chủ yếu là do bước tính toán bổ sung trong lớp tự attention, trong đó ma trận vị trí được thêm vào ma trận khóa truy vấn.



Hình 2.2 Khả năng embedding tương đối của T5 với các loại khác

Độ phức tạp trong việc sử dụng bộ nhớ đệm khóa-giá trị: Vì mỗi token bổ sung sẽ thay đổi cách embedding cho mọi token khác, điều này làm phức tạp việc sử dụng hiệu quả bộ nhớ đệm khóa-giá trị trong Transformers. Đối với những người chưa quen, bộ nhớ đệm khóa-giá trị rất quan trọng trong việc nâng cao hiệu quả và tốc độ trong các mô hình Transformer.

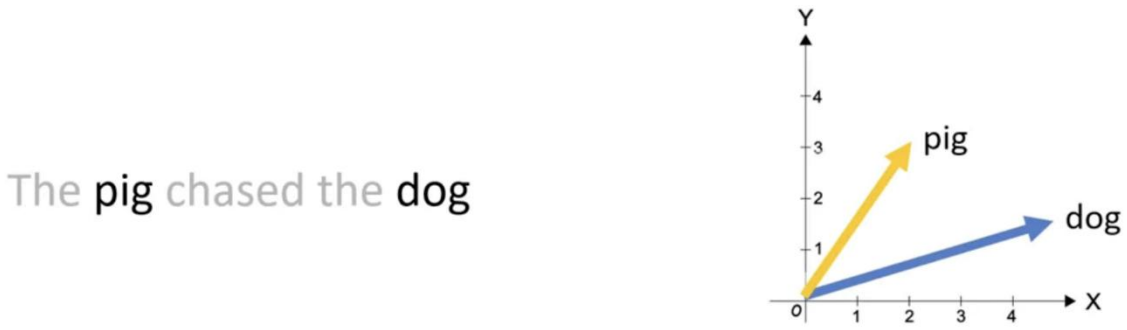
Do sự phức tạp về kỹ thuật này, việc embedding tương đối chưa được áp dụng rộng rãi, đặc biệt là trong các mô hình ngôn ngữ lớn hơn.

2.2.2.3 Rotary Positional Embeddings (RoPE)

RoPE đại diện cho một cách tiếp cận mới trong việc mã hóa thông tin vị trí. Các phương pháp truyền thống, dù tuyệt đối hay tương đối, đều có những hạn chế. Việc embedding vị trí tuyệt đối chỉ định một vectơ duy nhất cho mỗi vị trí, mặc dù đơn giản nhưng không có quy mô tốt và không nắm bắt được các vị trí tương đối một cách hiệu quả. Mặt khác, các phần embedding tương đối tập trung vào khoảng cách giữa các token, nâng cao sự hiểu biết của mô hình về các mối quan hệ token nhưng lại làm phức tạp kiến trúc mô hình.

RoPE khéo léo kết hợp thế mạnh của cả hai. Nó mã hóa thông tin vị trí theo cách cho phép mô hình hiểu cả vị trí tuyệt đối của token và khoảng cách tương đối của chúng. Điều này đạt được thông qua cơ chế quay, trong đó mỗi vị trí trong chuỗi được thể hiện bằng một phép quay trong không gian nhúng. Sự tinh tế của RoPE nằm ở tính đơn giản và hiệu quả, cho phép các mô hình nắm bắt tốt hơn các sắc thái của cú pháp và ngữ nghĩa ngôn ngữ.

Cơ chế của Rotary Positional Embeddings



Hình 2.3 Ví về cơ chế của Rotary Positional Embeddings

RoPE giới thiệu một khái niệm mới.Thay vì thêm một vectơ vị trí, nó áp dụng phép quay cho vectơ từ. Hãy tưởng tượng một vectơ từ hai chiều cho “dog”. Để mã hóa vị trí của nó trong câu, RoPE quay vectơ này. Góc quay (θ) tỷ lệ thuận với vị trí của từ trong câu.

Ví dụ, vectơ được quay một góc θ cho vị trí đầu tiên, 2θ cho vị trí thứ hai, v.v. Cách tiếp cận này có một số lợi ích:

- Tính ổn định của vectơ: Việc thêm token vào cuối câu không ảnh hưởng đến vectơ cho các từ ở đầu, tạo điều kiện cho bộ nhớ đệm hiệu quả.
- Bảo toàn vị trí tương đối: Nếu hai từ, chẳng hạn như “lợn” và “chó”, duy trì cùng một khoảng cách tương đối trong các ngữ cảnh khác nhau, thì vectơ của chúng sẽ được quay một lượng như nhau. Điều này đảm bảo rằng góc và do đó tích số chấm giữa các vectơ này không đổi.

RoPE độc đáo ở chỗ nó mã hóa vị trí tuyệt đối m token thông qua ma trận xoay R_(θ,m) và cũng kết hợp sự phụ thuộc vị trí tương đối rõ ràng trong công thức self-attention . Ý tưởng là embedding vị trí của token theo trình tự bằng cách xoay các truy vấn và khóa, với một góc xoay khác nhau ở mỗi vị trí.

Bằng cách xoay từng truy vấn/khóa tùy thuộc vào vị trí của chúng trong chuỗi, chúng tôi ngày càng giảm tích số chấm (do mức độ sai lệch ngày càng tăng) tùy thuộc vào khoảng cách giữa các token với nhau.

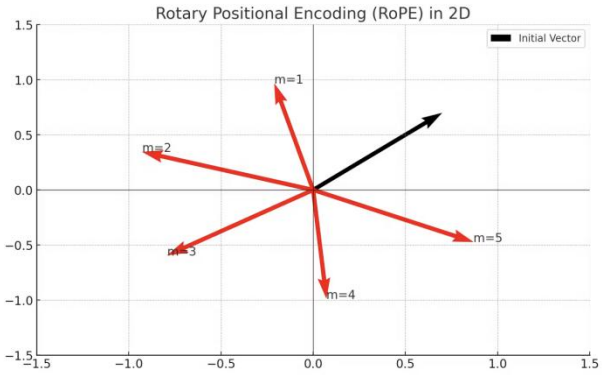
Công thức toán học trong 2D:

Đối với vectơ truy vấn 2D, $q_m = \begin{pmatrix} q_m^{(1)} \\ q_m^{(2)} \end{pmatrix}$ tại vị trí m ma trận xoay $R_{\theta,m}$ là ma trận 2x2 được xây dựng dưới dạng :

$$R_{\Theta,m} = \begin{pmatrix} cos\ m\theta & -sin\ m\theta \\ sin\ m\theta & cos\ m\theta \end{pmatrix}$$

với θ ϵ R là một hằng số được cho trước khác 0.

Chúng ta có thể sử dụng ma trận này để quay một vectơ q, để thu được vectơ quay mới q'. Dưới đây là hình ảnh trực quan về phép quay được áp dụng cho vectơ 2D q. Vectơ ban đầu được hiển thị bằng màu đen và vectơ màu đỏ là phiên bản quay của vectơ ban đầu.



Hình 2.4 Hình ảnh trực quan về phép quay được áp dụng cho vectơ 2D

Công thức toán học tổng quát:

Để khái quát hóa cho bất kỳ chiều chẵn nào d ≥2, chúng tôi chia d không gian chiều vào d/2 không gian con và áp dụng các phép quay riêng lẻ.

Hãy xem một ví dụ đơn giản với d=4, ma trận quay được định nghĩa là:

$$R^d_{\Theta,m} = \begin{pmatrix} cos\ m\theta_1 & -sin\ m\theta_1 & 0 & 0 \\ sin\ m\theta_1 & cos\ m\theta_1 & 0 & 0 \\ 0 & 0 & cos\ m\theta_2 & -sin\ m\theta_2 \\ 0 & 0 & sin\ m\theta_2 & cos\ m\theta_2 \end{pmatrix}$$

với $\theta = \{\theta_i = 1000^{-2i/4}, i \in [1,2]\}$ là các tham số được xác định trước và i là chỉ số cho $\frac{d}{2}$ không gian con tương ứng:

$$q_m = \begin{pmatrix} q_m^{(1)} \\ q_m^{(2)} \\ q_m^{(3)} \\ q_m^{(4)} \end{pmatrix}$$

Sub-space 1

Sub-space 2

Dạng tổng quát của các tham số là:

$$\theta = \{\theta_i = 1000^{-2(i-1)/d}, i \in [1, \ldots, d/2]\}$$

Vì ma trận $R_{\theta,m}^d$ quá thưa thớt nên các hiệu quả để tính toán $R_{\theta,m}^d$. q^m là:

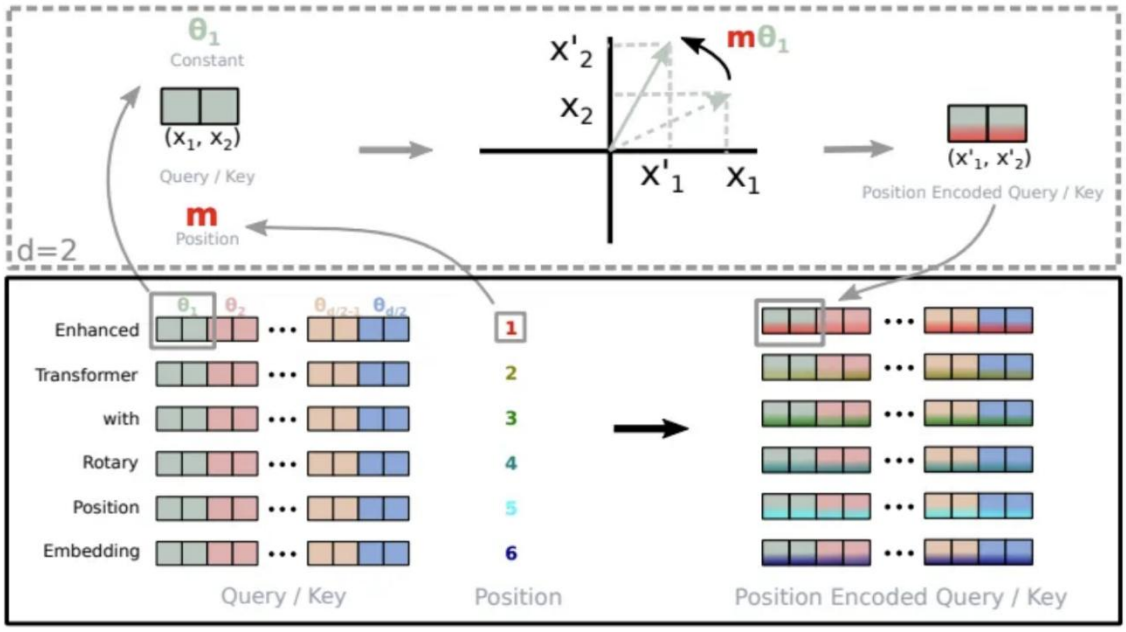
$$\boldsymbol{R}_{\Theta,m}^d \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix}$$

Trong LLaMA, phép xoay được áp dụng cho cả query và key trước khi tính dot product trong cơ chế attention. Đối với querty q_m ở vị trí m và key k_(n)ở vị trí n công thức trở thành:

$$q_m^T k_n = (R_{\theta,m}^d q_m)^T (R_{\theta,n}^d k_n) = q_m^T R_{\theta,n-m}^d k_n,$$

với $R_{\theta,n-m} = (R_{\theta,m}^d)^T . R_{\theta,n}^d$

Cách triển khai Rotary Positional Embeddings:



Hình 2.5 Cách triển khai Rotary Positional Embeddings

Rotary Positional Embeddings chỉ được áp dụng cho query và key vectors . Việc RoPE được áp dụng sau khi vectơ q và k đã được nhân với ma trận W trong self-attention trong khi transformer thông thường thì được áp dụng trước đó.

2.2.3 KV - Cache

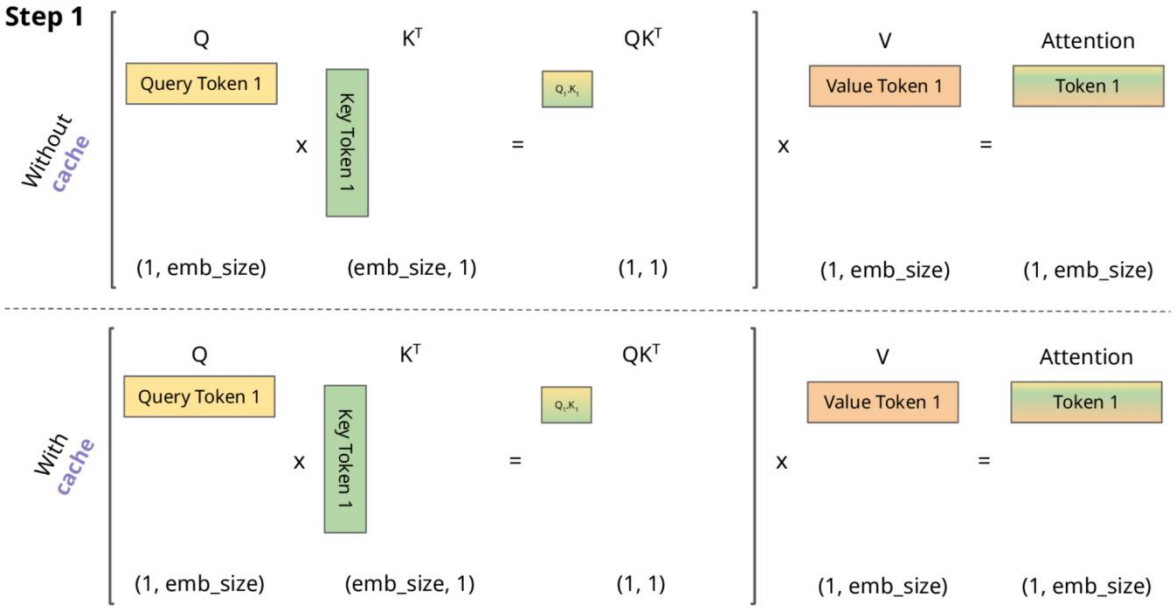
KV Cache (Key-Value Cache) là một kỹ thuật được sử dụng để tăng tốc quá trình suy luận trong các mô hình học máy, đặc biệt là trong các mô hình tự hồi quy như GPT và Llama. Trong các mô hình này, việc tạo từng token là một phương pháp phổ biến nhưng có thể tốn kém về mặt tính toán vì nó lặp lại một số phép tính nhất định ở mỗi bước . Để giải quyết vấn đề này, KV Cache sẽ phát huy tác dụng. Nó liên quan đến việc lưu trữ Key và Value trước đó vào cache , vì vậy chúng tôi không cần tính toán lại chúng cho mỗi token mới. Điều này làm giảm đáng kể kích thước của ma trận được sử dụng trong tính toán, giúp phép nhân ma trận nhanh hơn. Sự đánh đổi duy nhất là bộ nhớ đệm KV yêu cầu nhiều bộ nhớ GPU hơn (hoặc bộ nhớ CPU nếu không sử dụng GPU) để lưu trữ các trạng thái key và Value này.

Cách thức hoạt động:

- Cập nhật cache: Khi xử lý một chuỗi văn bản, LLaMa sẽ tính toán các giá trị biểu diễn cho từng từ trong câu. Sau đó, nó sẽ lưu trữ các cặp khóa-giá trị tương ứng với vị trí bắt đầu của mỗi từ và giá trị biểu diễn của nó trong KV Cache.
- Truy cập cache: Khi cần truy cập lại giá trị biểu diễn của một từ, LLaMa sẽ kiểm tra KV Cache trước tiên. Nếu cặp khóa-giá trị tương ứng được tìm thấy trong cache, LLaMa sẽ sử dụng giá trị được lưu trữ thay vì tính toán lại.
- Cập nhật cache định kỳ: Để đảm bảo tính chính xác và hiệu quả, KV Cache được cập nhật định kỳ trong quá trình xử lý văn bản. Việc cập nhật này giúp đảm bảo rằng cache luôn chứa thông tin mới nhất về các từ và giá trị biểu diễn của chúng.

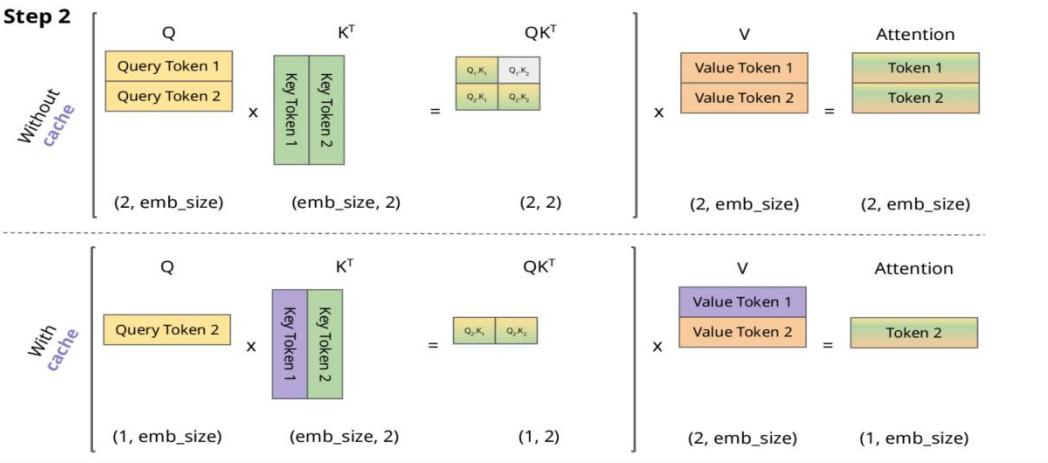
Ví dụ về Attention có KV Cache và không có KV Cache:

Bước 1:



Hình 2.6 Ví dụ về Attention có KV Cache và không có KV Cache bước 1

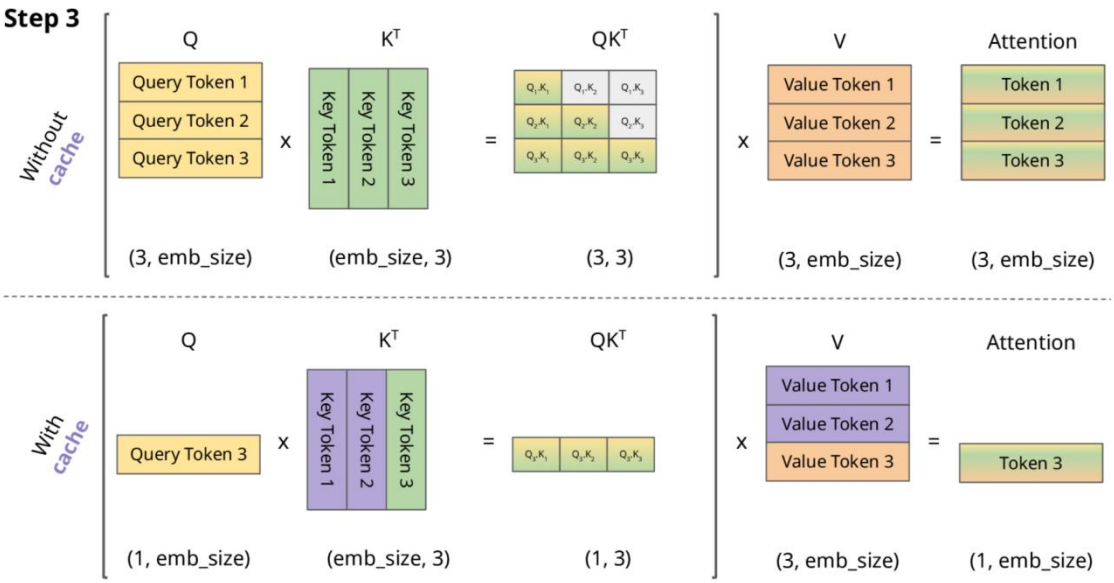
Bước 2:



Hình 2.7 Ví dụ về Attention có KV Cache và không có KV Cache bước 2

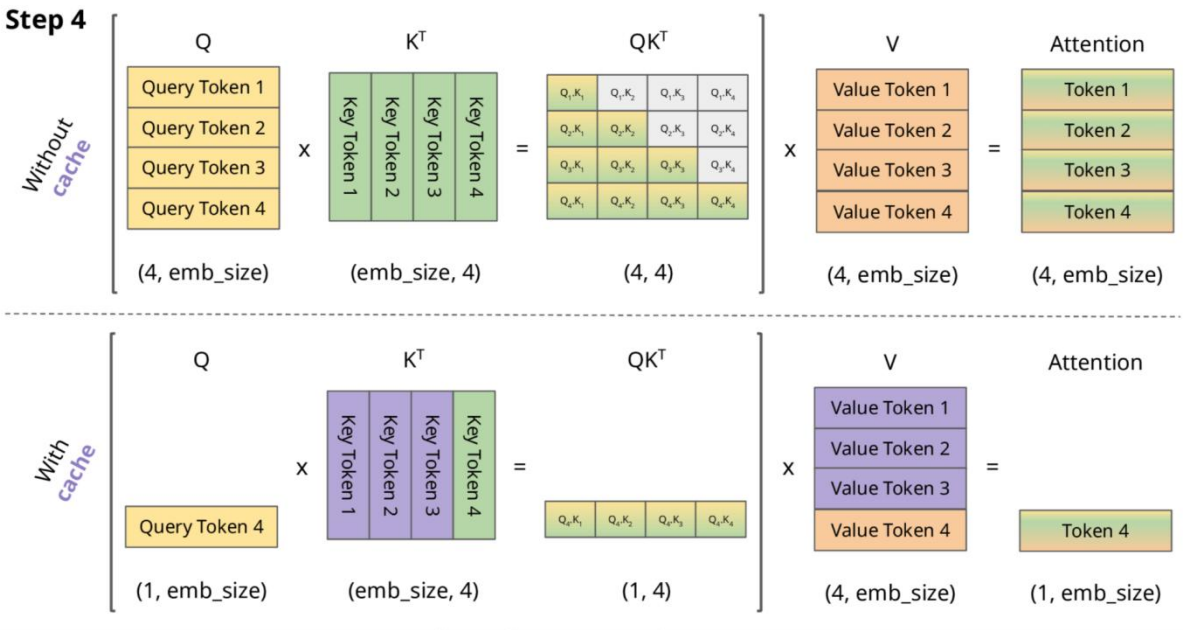
Chúng ta không thêm Query Token 2 vào trong Q mà thay thế Query Token 1 bằng Query Token 2 và ở Attention cũng như vậy. Các Key Token và Value Token được lưu và cache. Các bước tiếp theo cũng thực hiện như vậy.

Bước 3:



Hình 2.8 Ví dụ về Attention có KV Cache và không có KV Cache bước 3

Bước 4:



Hình 2.9 Ví dụ về Attention có KV Cache và không có KV Cache bước 4

2.2.4 Group Multi-Query Attention

Llama kết hợp một kỹ thuật được gọi là Group Multi-Query Attention (GQA) để giải quyết các thách thức về băng thông bộ nhớ trong quá trình giải mã tự hồi quy của các mô hình Transformer. Vấn đề chính xuất phát từ nhu cầu tải trọng số bộ giải mã và khóa/giá trị attention ở mỗi bước xử lý, điều này tiêu tốn quá nhiều bộ nhớ.

Trước tiên hãy tìm hiểu về Multi-Head Attention và Multi-Query Attention:

Multi-Head Attention là một thành phần quan trọng của các mô hình Transformer, cho phép chúng xử lý và hiểu các chuỗi phức tạp một cách hiệu quả trong các tác vụ như dịch ngôn ngữ, tóm tắt, v.v. Để nắm bắt được sự phức tạp của nó, chúng ta phải đi sâu vào nền tảng toán học và hiểu cách hoạt động của nhiều đầu trong cơ chế attention.

Cơ chế attention cơ bản tính toán tổng giá trị có trọng số, với trọng số phụ thuộc vào query và key. Về mặt toán học, điều này được thể hiện như sau:

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

Điều này được gọi là scaled dot product attention. Trong phương trình này, Q (Truy vấn) và K (Khóa) là các ma trận biểu thị các truy vấn và khóa. V (Value) là ma trận các giá trị. “d_k” là chiều của các khóa, được sử dụng để chia tỷ lệ.

Multi-Head Attention attention sử dụng nhiều 'head" của các lớp attention, cho phép mô hình attention đến thông tin từ các không gian biểu diễn khác nhau. Trong mỗi đầu, có một tập hợp độc lập các lớp tuyến tính (ma trận chiếu) cho các truy vấn, khóa, và giá trị (điều này là một điểm quan trọng mà chúng ta sẽ quay lại trong GQA). Đối với mỗi đầu (được đánh số là h):

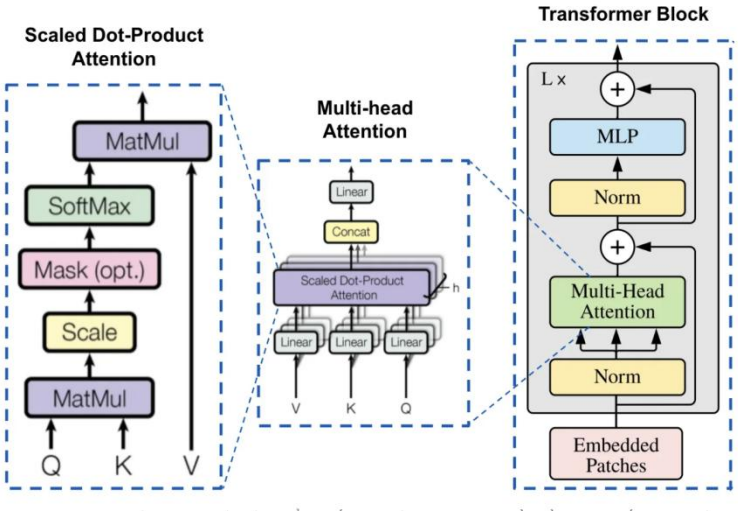
$$head^h = Attention(Q.Wq^h, K.Wk^h, V.Wv^h)$$

Đầu ra của các đầu riêng lẻ được nối và sau đó được chuyển đổi tuyến tính:

$$MultiHead(Q, K, V) = Concat(head^1, head^2, \ldots, head^h) . W^o$$

với W^o là một ma trận trọng số khác biến đổi tuyến tính vectơ nối thành thứ nguyên đầu ra cuối cùng.

Ý nghĩa Multi-Head Attention là bằng cách áp dụng song song cơ chế attention nhiều lần, mô hình có thể nắm bắt các loại mối quan hệ khác nhau trong dữ liệu.



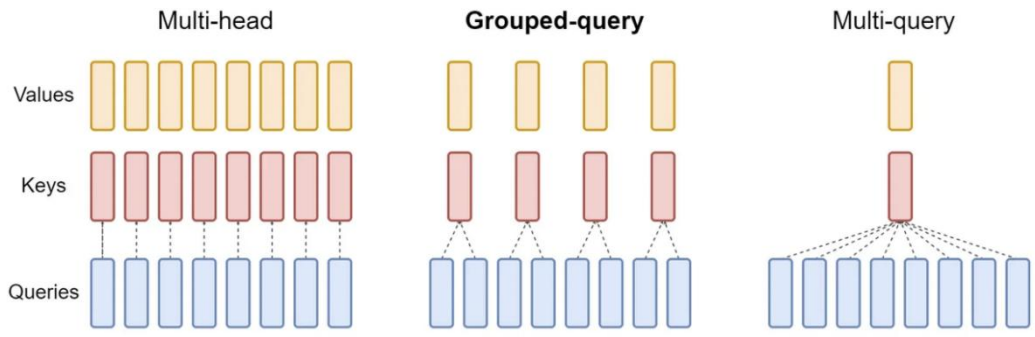
Hình 2.10 Cách thức hoạt động của MHA

Tuy nhiên, MHA cho phép hiểu rõ hơn về mối quan hệ giữa các phần khác nhau của đầu vào. Tuy nhiên, sự phức tạp này phải trả giá - nhu cầu đáng kể về băng thông bộ nhớ, đặc biệt là trong quá trình suy luận của Decoder.

Mấu chốt của vấn đề nằm ở chi phí bộ nhớ. Mỗi bước giải mã trong các mô hình tự hồi quy như Transformers yêu cầu tải trọng số bộ giải mã cùng với tất cả các khóa và giá trị attention. Quá trình này không chỉ đòi hỏi nhiều tính toán mà còn tốn nhiều băng thông bộ nhớ. Khi kích thước mô hình tăng lên, chi phí này cũng tăng lên, khiến việc mở rộng quy mô trở thành một nhiệm vụ ngày càng khó khăn.

Multi-Query Attention (MQA) nổi lên như một giải pháp để giảm thiểu nút thắt cổ chai này. Ý tưởng đơn giản nhưng hiệu quả: sử dụng nhiều đầu truy vấn nhưng chỉ có một đầu khóa và giá trị duy nhất. Cách tiếp cận này làm giảm đáng kể tải bộ nhớ, tăng cường tốc độ suy luận. Nó đã được sử dụng trong nhiều mô hình quy mô lớn như PaLM, StarCoder và Falcon.

Trong MQA, tính trung bình các phần đầu cho khóa và giá trị sao cho tất cả các phần đầu truy vấn có chung phần đầu khóa và giá trị. Điều này đạt được bằng cách sao chép “head” H lần được gộp trung bình, trong đó H là số lượng đầu truy vấn.



Hình 2.11 Kiến trúc của MHA so với MQA

Việc tạo mô hình MQA từ mô hình MHA có bao gồm quy trình gồm hai bước: chuyển đổi cấu trúc của mô hình và đào tạo trước sau đó.

Chuyển đổi điểm kiểm tra: Bước này chuyển đổi cấu trúc của mô hình MHA thành mô hình MQA. Nó đạt được bằng cách hợp nhất (gộp trung bình) các ma trận chiếu (lớp tuyến tính) cho các khóa và giá trị từ nhiều phần head của mô hình ban đầu thành các ma trận chiếu đơn cho các khóa và giá trị. Cách tiếp cận gộp trung bình này được cho là hiệu quả hơn so với việc chọn một trong các đầu khóa và giá trị hiện có hoặc khởi tạo các đầu khóa và giá trị mới từ đầu. Cấu trúc kết quả có phép chiếu khóa và giá trị hợp nhất, đặc trưng của mô hình đa truy vấn.

Đào tạo trước mô hình đã chuyển đổi: Sau khi chuyển đổi cấu trúc, mô hình sẽ trải qua quá trình đào tạo bổ sung. Khóa đào tạo này không rộng rãi như khóa đào tạo mô hình ban đầu; đó là một phần nhỏ (ký hiệu là α) trong các bước huấn luyện của mô hình ban đầu. Mục đích của giai đoạn đào tạo trước này là cho phép mô hình điều chỉnh và tối ưu hóa hiệu suất theo cơ chế attention mới, đơn giản hóa. Quá trình đào tạo tuân theo cùng một công thức như ban đầu, đảm bảo tính nhất quán trong động lực học tập.

Tuy nhiên, MQA không phải không có nhược điểm. Sự phức tạp giảm đi có thể dẫn đến suy thoái chất lượng và mất ổn định đào tạo.

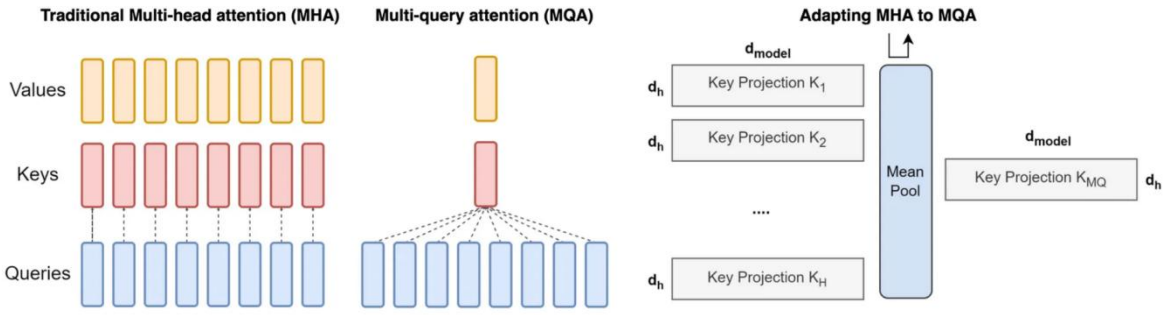
Group Multi-Query Attention(GQA) là một cách tiếp cận đơn giản kết hợp các yếu tố của MHA và MQA để tạo ra cơ chế attention hiệu quả hơn. Khung toán học của GQA có thể được hiểu như sau:

Phân chia thành các nhóm: Trong GQA, các đầu truy vấn (Q) từ mô hình MHA truyền thống được chia thành các nhóm G. Mỗi nhóm được gán một đầu khóa (K) và giá trị (V). Cấu hình này được ký hiệu là GQA-G, trong đó G đại diện cho số lượng nhóm.

Các trường hợp đặc biệt của GQA:

- GQA-1 = MQA: Chỉ với một nhóm (G = 1), GQA trở nên tương đương với MQA, vì chỉ có một đầu khóa và giá trị duy nhất cho tất cả các đầu truy vấn.

- GQA-H = MHA: Khi số nhóm bằng số đầu (G = H), GQA hoạt động giống như MHA truyền thống, với mỗi đầu truy vấn có đầu khóa và giá trị duy nhất.



Hình 2.12 Kiến trúc của MHA, MQA và GQA

Gộp các ma trận chiếu khóa và giá trị của các head ban đầu trong mỗi nhóm để chuyển đổi mô hình nhiều đầu thành mô hình GQA. Kỹ thuật này tính trung bình các ma trận chiếu của mỗi đầu trong một nhóm, dẫn đến một phép chiếu khóa và giá trị duy nhất cho nhóm đó.

Bằng cách sử dụng GQA, mô hình duy trì sự cân bằng giữa chất lượng MHA và tốc độ MQA. Vì có ít cặp khóa-giá trị hơn nên băng thông bộ nhớ và nhu cầu tải dữ liệu sẽ được giảm thiểu. Việc lựa chọn G thể hiện sự đánh đổi: nhiều nhóm hơn (gần MHA) mang lại chất lượng cao hơn nhưng hiệu suất chậm hơn, trong khi ít nhóm hơn (gần MQA) tăng tốc độ nhưng có nguy cơ hy sinh chất lượng. Hơn nữa, khi kích thước mô hình tăng lên, GQA cho phép giảm băng thông bộ nhớ và dung lượng mô hình theo tỷ lệ tương ứng với quy mô của mô hình. Ngược lại, đối với các mô hình lớn hơn, việc giảm xuống một đầu khóa và giá trị duy nhất có thể trở nên quá nghiêm trọng trong MQA.

2.2.5 SwiGLU Activation function

Trong mô hình Transformer thông thường activation function được sử dụng là:

Transformer ("Attention is all you need")

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

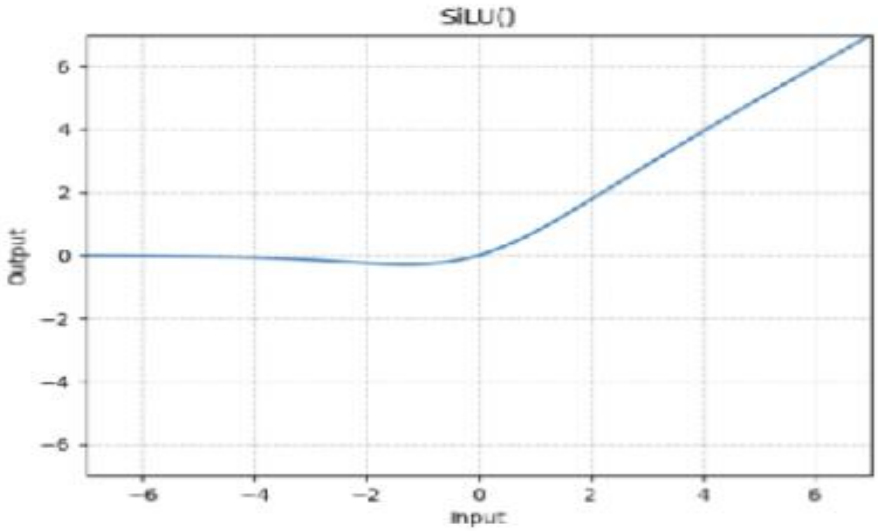
Nhưng trong mô hình LLaMA có cải tiến về activation function:

LLaMA

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(x, W, V, W_2) = (\mathrm{Swish}_1(xW) \otimes xV)W_2$$

Sử dụng Swish function với β=1. Nó được gọi là Sigmoid Linear Unit (SiLU) function

swish(x)=x . sigmoid(βx) = x/(1+ e^(-βx))



Hình 2.13 Biểu đồ biến thiên của SiLu()

Ưu điểm của SwiGLU :

- Hiệu suất: SwiGLU đã được chứng minh là có thể cải thiện hiệu suất của các mạng nơ-ron nhân tạo trên nhiều nhiệm vụ khác nhau, bao gồm phân loại hình ảnh, xử lý ngôn ngữ tự nhiên và học tăng cường.
- Tính ổn định: SwiGLU có độ ổn định cao hơn ReLU, giúp giảm nguy cơ xảy ra hiện tượng mất gradient trong quá trình huấn luyện.
- Dễ sử dụng: SwiGLU có thể được dễ dàng triển khai trong các thư viện học máy phổ biến như TensorFlow và PyTorch.

CHƯƠNG 3 - COUNTINUE PRE-TRAINING ANF FINE-TUNING

3.1 Continue pre-training

Continuous pre-training là sử dụng một số mô hình đã được pretrain và về cơ bản áp dụng phương pháp học chuyển giao - sử dụng các trọng số đã lưu từ mô hình được huấn luyện (checkpoint) và huấn luyện nó trên một số miền mới.

3.2 Fine-tuning

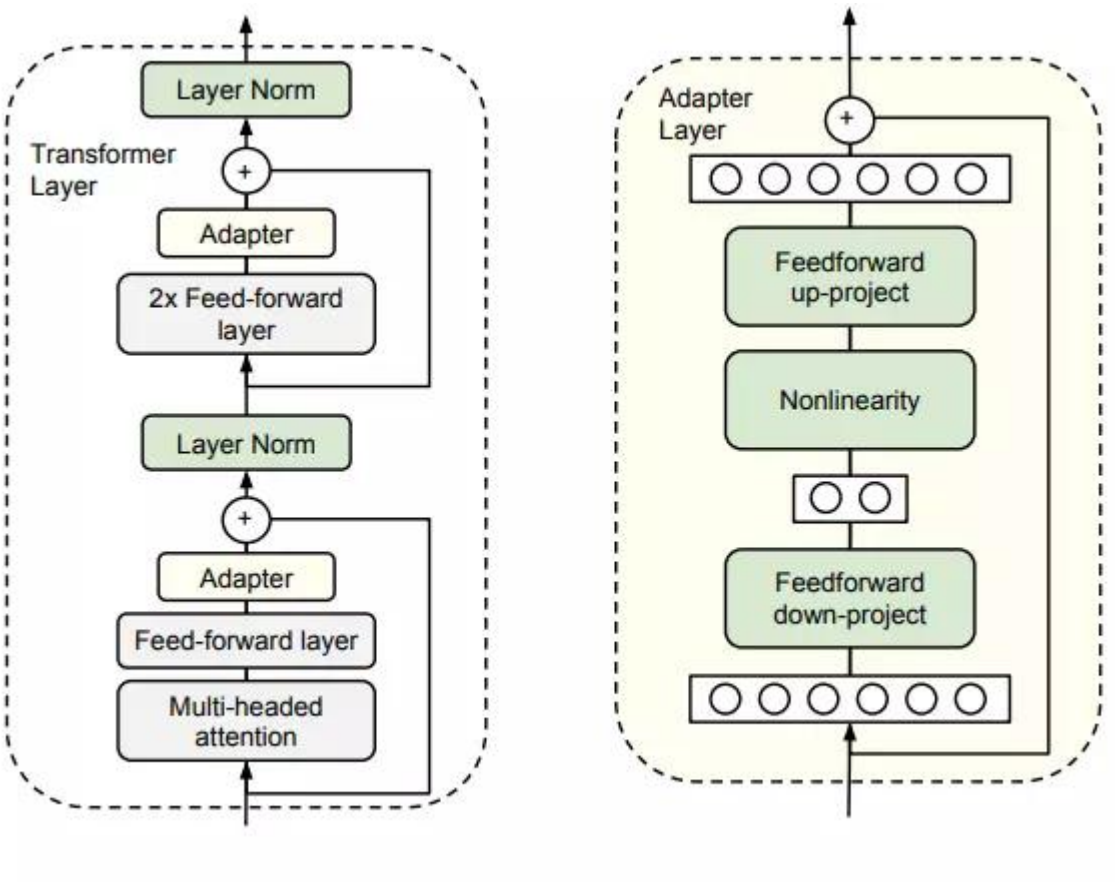
Là một phương pháp phổ biến trong các hệ thống học máy, trong đó lấy một mô hình được pretrain và huấn luyện lại mô hình đó bằng dữ liệu mới để cải thiện hiệu suất của mô hình đó trong một nhiệm vụ cụ thể. Trong bối cảnh của các mô hình ngôn ngữ, có thể tinh chỉnh mô hình được pretrain bằng một tập hợp các ví dụ được tuyển chọn cho một nhiệm vụ nhất định để tạo một mô hình tùy chỉnh có thể chính xác và phù hợp hơn cho nhiệm vụ cụ thể đó.

Các trường hợp nên fine tuning:

- Hallucinations
- Các mô hình pre-trained không đạt được độ chính xác mong muốn cho 1 nhiệm vụ cụ thể.

3.2.1 Parameter-efficient Fine-tuning (PEFT) với Adapters

Chèn thêm vào 2 lớp Adapters trước khi thực hiện fine-tuning. Chỉ thực hiện training các lớp Adapters được chèn vào model, và freeze toàn bộ pre-trained model trong quá trình fine-tuning.



Hình 3.1 Lớp Adapters được chèn vào mỗi khối Transformer

Việc freeze toàn bộ pre-trained model và chỉ train các lớp Adapters khiến số parameters cần phải train giảm đáng kể, do đó làm quá trình training ngốn ít tài nguyên hơn rất nhiều.

3.2.2 Low-rank Adaptation: LoRA

Điểm yếu của Adapters thông thường:

- Sử dụng Adapters tức là ta phải chèn thêm vào model các lớp Adapters. Vì vậy trong quá trình forward của model, độ nặng tính toán tăng lên.
- Chèn thêm nhiều layer nhỏ sẽ làm tăng số phép tính tuần tự của model, giảm khả năng tính toán song song của GPU.

Quá trình cập nhật weight của mô hình trong fine-tuning: $W' = W + \Delta W$

Tại mỗi iteration, thay vì cập nhật W thì ta cập nhật ΔW.

Mục đích của LoRA là tìm cách biểu diễn ma trận ΔW thành một dạng biểu diễn nhẹ hơn. LoRA chọn sử dụng Matrix decomposition để biểu diễn ma trận ΔW bằng tích của các ma trận con với độ nặng tính toán thấp hơn việc tính trên ma trận gốc.

LoRA phân tách ma trận ΔW thành 2 ma trận con A và B với số rank thấp hơn rất nhiều ma trận ban đầu. $\Delta W = AB$ với $\Delta W \in \mathbb{R}^{d\times k}$, $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$ với số rank $r \ll \min(d, k)$.

Ma trận A được khởi tạo theo random Gaussian init, còn ma trận B thì được khởi tạo toàn 0. Quá trình training sẽ tối ưu để tìm ra ma trận A và B.

Tại sao LoRA cũng thêm layers vào mà không bị chậm giống Adapters thông thường?

Bản chất LoRA cũng là thêm các layers vào trong model. Nhưng cái hay của LoRA là nó có thể Re-parameterize sau khi training xong.

Trước Re-parameterize:$y = Wx + BAx$.

Sau Re-parameterize:$y = (W + BA)x$.

3.2.3 QLoRA:Quantized LoRA

Là sự kết hợp giữa quantization vs LoRA để giúp training các mô hình siêu nặng một cách dễ dàng.

Thành phần pretrained được freeze. Weights của thành phần pretrained sẽ được quantize về NF4. Trong quá trình tính toán output, thành phần pretrained sẽ được dequantize từ NF4 về BF16, rồi kết hợp tính toán với thành phần LoRA ở dạng BF16. Khi tính xong thì thành phàn pretrained lại được quantize lại về NF4. Vì vậy, QLoRA giúp chúng ta có thể đưa được model to vào vừa với VRAM để training.

3.2.4 Những hạn chế của fine-tuning

- Yêu cầu dữ liệu huấn luyện chất lượng cao, đủ lớn và mang tính đại diện phù hợp với nhiệm vụ mục tiêu.
- Phải trả thêm chi phí liên quan đến việc đào tạo và lưu trữ mô hình fine-tuning.

- Việc fine-tuning có thể cần phải được lặp lại bất cứ khi nào dữ liệu được cập nhật.
- Các siêu tham số cần phải được thiết lập cẩn thận.

TÀI LIỆU THAM KHẢO

[1] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. ArXiv preprint arXiv:1907.11692.

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[3] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.

[4] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pretraining. URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf.

[5] Sennrich, R., Haddow, B., & Birch, A. (2016). Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909.