Lecture 10: Retrieval and RAG

Giới thiệu

Standard Prompting

Hôm nay, tôi sẽ nói về việc truy xuất thông tin và việc tạo ra thông tin tăng cường từ truy xuất. Trong quá trình prompting tiêu chuẩn, chúng ta thường kết hợp một prompt template với một đầu vào. Ví dụ, nếu chúng ta nói "please answer this question: I think Vin Diesel has been a voice actor for several pictures in TV series, do you know what their names are?", chúng ta có thể nhận được phản hồi từ một mô hình ngôn ngữ.

Problems

Có một số vấn đề cần được lưu ý. Đầu tiên là vấn đề về độ chính xác. Các mô hình thường có một "knowledge cut-off", nghĩa là các tham số chỉ được cập nhật đến một thời điểm nhất định. Ví dụ, nếu có một series TV mới của Vin Diesel ra mắt, mô hình được huấn luyện trước thời điểm đó sẽ không biết gì về nó.

Thứ hai là vấn đề về dữ liệu riêng tư. Dữ liệu được lưu trữ trong các văn bản hoặc kho dữ liệu riêng tư không phù hợp để huấn luyện mô hình vì nhiều lý do. Thứ nhất, dữ liệu này không có sẵn cho các nhà cung cấp dịch vụ huấn luyện mô hình ngôn ngữ như OpenAI hay Google. Thứ hai, có vấn đề về kiểm soát truy cập. Ngay cả khi bạn ở trong một tổ chức có nhiều dữ liệu riêng tư và có thể huấn luyện mô hình ngôn ngữ trên đó, không phải ai trong tổ chức cũng có quyền truy cập vào tất cả các loại dữ liệu.

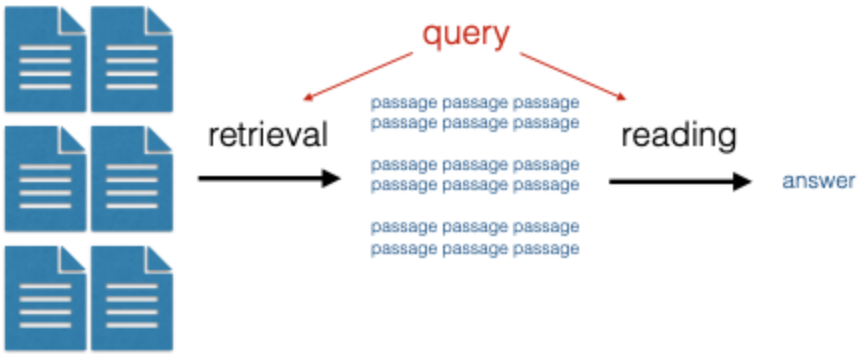
Ngoài ra, còn có vấn đề về thất bại trong việc học. Ngay cả với dữ liệu mà mô hình đã được huấn luyện, đôi khi vẫn không đủ để đưa ra câu trả lời đúng, đặc biệt là với các tập dữ liệu huấn luyện rất lớn và các mô hình có kích thước vừa phải. Mô hình thường không thể học từ một lần xem duy nhất về một thông tin cụ thể, đặc biệt là ở giai đoạn đầu của quá trình huấn luyện.

Cuối cùng, ngay cả khi câu trả lời là đúng, nó có thể không thể xác minh được. Để đảm bảo mô hình không gặp vấn đề về độ chính xác, con người thường cần quay lại nguồn dữ liệu gốc để kiểm tra.

Retrieval-augmented Generation (Chen et al. 2017)

Trong bài viết này, chúng ta sẽ tìm hiểu về phương pháp "retrieval augmented generation" (RAG). Phương pháp này hoạt động bằng cách truy xuất các đoạn văn bản liên quan, có khả năng chứa câu trả lời cho một câu hỏi cụ thể. Sau đó, hệ thống sẽ đọc các đoạn văn này để đưa ra câu trả lời cho truy vấn.

Quá trình này bao gồm các bước: từ truy vấn, hệ thống sẽ thực hiện truy xuất, thu thập một loạt các đoạn văn, sau đó đọc và đưa ra câu trả lời. Phương pháp này đã được triển khai trong nhiều dịch vụ mô hình ngôn ngữ, bao gồm cả OpenAI.



Ví dụ, khi tôi hỏi về việc lồng tiếng của Vin Diesel trong các loạt phim truyền hình, ChatGPT đã cung cấp câu trả lời kèm theo các trích dẫn nguồn thông tin. Tuy nhiên, câu trả lời không hoàn toàn chính xác. Mặc dù tôi chỉ hỏi về loạt phim truyền hình, nhưng hệ thống lại cung cấp thông tin về phim điện ảnh như "Groot trong Guardians of the Galaxy volume 3 (2023)". Thực tế, Vin Diesel không lồng tiếng cho nhân vật này, đây là một lỗi về độ chính xác.

Ngoài ra, có một đoạn thông tin sai lệch khi hệ thống cho rằng Nick Kroll lồng tiếng cho Vin Diesel trong chương trình "Big Mouth", trong khi thực tế Nick Kroll chỉ đóng vai Vin Diesel. Điều này cho thấy sự hiểu nhầm của mô hình.

Retrieval Methods

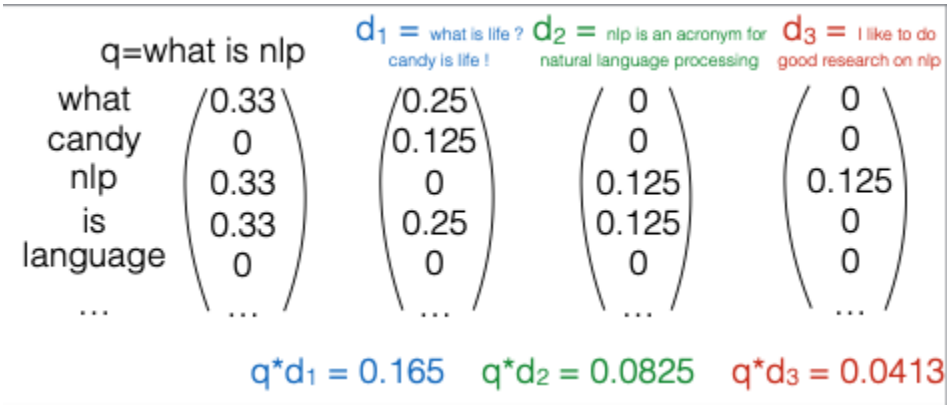
Trong phần này, tôi sẽ trình bày về các phương pháp truy xuất thông tin. Chúng ta có nhiều lựa chọn khác nhau cho các phương pháp truy xuất, bao gồm:

- Sparse retrieval (truy xuất thưa)
- Document-level dense retrieval (truy xuất dày đặc mức tài liệu)
- Token-level dense retrieval (truy xuất dày đặc mức token)
- Cross-encoder reranking (xếp hạng lại bằng bộ mã hoá chéo)
- Black-box retrieval (just ask Google/Bing) (truy xuất hộp đen)

Về truy xuất hộp đen, tôi sẽ không đi sâu vào chi tiết. Về cơ bản, phương pháp này sử dụng một công cụ tìm kiếm hộp đen để truy xuất ngữ cảnh liên quan và lấy ra các kết quả hàng đầu. Đây là một phương pháp hợp lý nếu bạn muốn tìm kiếm trên lượng dữ liệu lớn có sẵn trên internet. Ví dụ, ChatGPT thực hiện điều này bằng cách tạo truy vấn để tìm kiếm trên Bing.

Sparse Retrieval

Đầu tiên, chúng ta sẽ tìm hiểu phương pháp Sparse Retrieval (truy xuất thưa). Phương pháp này hoạt động bằng cách biểu diễn truy vấn và tài liệu dưới dạng vector tần suất từ thưa, thường được chuẩn hóa theo độ dài. Ví dụ, nếu bạn đặt câu hỏi "what is nlp?", bạn sẽ nhận được một vector mà mỗi hàng tương ứng với một token khác nhau. Các vị trí cho "what", "NLP", và "is" sẽ có giá trị khác 0, trong khi các vị trí khác sẽ có giá trị 0. Chúng ta cũng chuẩn hóa theo độ dài vector để có được các giá trị như 0.333333.



Giả sử chúng ta có một số tài liệu: tài liệu đầu tiên có nội dung "what is life? candy is life!" diễn ta một ai đó rất thích kẹo, và tài liệu thứ hai là "nlp is an acronym for natural language processing", đây là một câu trả lời khá tốt cho câu hỏi của chúng ta. Tài liệu thứ ba là "I like to do good research on NLP", mang lại một cảm xúc tích cực nhưng không phải là câu trả lời tốt nhất cho câu hỏi.

Khi phân tích các vector, chúng ta nhận thấy rằng trong tài liệu đầu tiên, các từ "what", "candy", và "is" có điểm số khá cao, trong khi ở tài liệu thứ hai, "NLP" và "is" nổi bật với điểm số cao. Để xác định tài liệu nào tương tự nhất, chúng ta có thể sử dụng tích vô hướng hoặc độ tương đồng cosine giữa các vector. Nếu áp dụng tích vô hướng, tài liệu đầu tiên sẽ có điểm số cao nhất nhờ vào giá trị lớn của các từ "what" và "is". Tuy nhiên, các từ phổ biến như "what" và "is" có thể có điểm số cao bất kể tài liệu có liên quan hay không.

Term Weighting (See Manning et al. 2009)

Một cách để khắc phục điều này là sử dụng trọng số từ (term weighting). Trọng số từ giúp tăng trọng số cho các từ có tần suất thấp vì chúng thường mang nhiều thông tin hơn về sự liên quan của tài liệu.

Một phiên bản dễ hiểu và được sử dụng rộng rãi là tf-idf (term frequency-inverse document frequency). Tần suất từ (tf) được định nghĩa là tần suất của từ T trong tài liệu d, chuẩn hóa theo tổng tần suất từ trong tài liệu. Ngược lại, tần suất ngược tài liệu (idf) được tính bằng log của tổng số tài liệu chia cho số lần từ đó xuất hiện trong bất kỳ tài liệu nào. Nếu một từ xuất hiện nhiều lần, nó sẽ có điểm idf thấp, gần bằng không, nhưng nếu hiếm khi xuất hiện, nó sẽ có điểm idf cao. Tf-idf là tích của hai thành phần này, giúp tăng trọng số cho các từ có tần suất thấp.

$$\mathrm{TF}(t,d) = \frac{\mathrm{freq}(t,d)}{\sum_{t'} \mathrm{freq}(t',d)} \qquad \mathrm{IDF}(t) = \log\left(\frac{|D|}{\sum_{d' \in D} \delta(\mathrm{freq}(t,d') > 0)}\right)$$

$$\text{TF-IDF}(t,d) = \mathrm{TF}(t,d) \times \mathrm{IDF}(t)$$

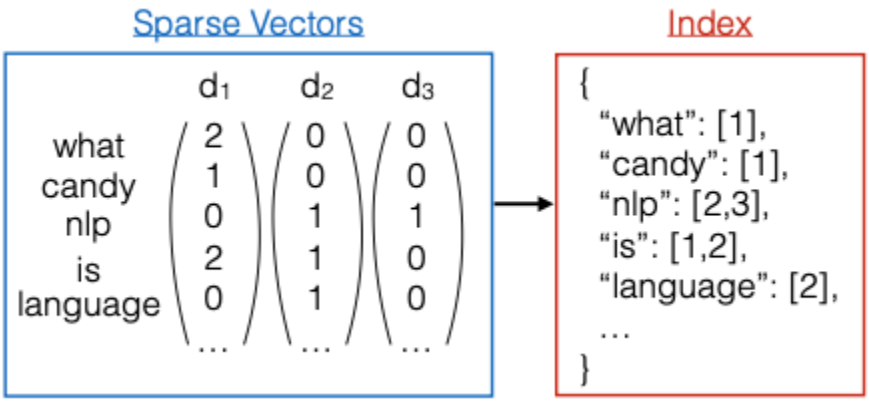
Một phiên bản khác là bm25, được sử dụng rộng rãi. Phương pháp này phức tạp hơn và sử dụng kỹ thuật làm mịn (smoothing) để điều chỉnh tần suất từ. Công thức của bm25 bao gồm các tham số K và B, cùng với độ dài tài liệu trung bình avgdl. Mặc dù chi tiết không quá quan trọng, nhưng bạn nên biết rằng phương pháp này thực hiện một số làm mịn trên tần suất từ.

$$\text{BM-25}(t,d) = \text{IDF}(t) \cdot \frac{\text{freq}(t,d) \cdot (k_1+1)}{\text{freq}(t,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)}$$

Inverted Index

Việc tìm kiếm tài liệu liên quan trong một tập hợp lớn, chẳng hạn như toàn bộ Internet, đòi hỏi một cấu trúc dữ liệu cho phép tra cứu nhanh chóng và hiệu quả. Để giải quyết vấn đề này, chúng ta sử dụng một cấu trúc gọi là "Inverted Index".

Inverted Index hoạt động bằng cách chuyển đổi các sparse vector dựa trên thuật ngữ thành một index. Cụ thể, chúng ta có thể hình dung nó như một dictionary hoặc map trong Python, nơi mỗi từ là một khóa (key) và giá trị (value) là chỉ số của tài liệu chứa từ đó. Ví dụ, từ "what" chỉ xuất hiện trong tài liệu 1, nên nó sẽ trỏ đến tài liệu 1. Tương tự, "candy" cũng xuất hiện trong tài liệu 1, trong khi "NLP" xuất hiện trong tài liệu 2 và 3.



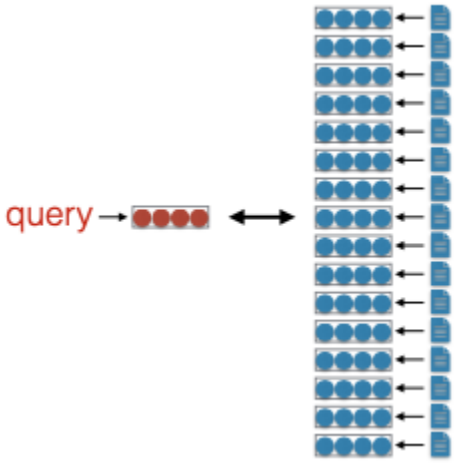
Một phần mềm phổ biến để xây dựng large index sử dụng cấu trúc này là Apache Lucene.

Dense Retrieval

Tiếp theo, chúng ta sẽ tìm hiểu về dense retrieval. Phương pháp này hoạt động bằng cách mã hóa tài liệu và truy vấn thành các dense vector, sau đó tìm kiếm láng giềng gần nhất. Để thực hiện mã hóa này, có thể sử dụng nhiều phương pháp khác nhau, bao gồm cả các embeddings có sẵn hoặc các embeddings được học và tạo ra đặc biệt cho mục đích truy xuất.

Quy trình thực hiện như sau: đầu tiên, tất cả các tài liệu được chuyển đổi thành embeddings bằng phương pháp mà chúng ta lựa chọn. Sau đó, khi có một truy vấn, chúng ta cũng chuyển đổi truy vấn đó thành embedding và tìm kiếm láng giềng gần nhất.

Nếu chỉ sử dụng các embeddings có sẵn, bạn không cần thực hiện điều gì đặc biệt cho việc truy xuất. Bạn có thể sử dụng các embeddings yêu thích như "sentence BERT embeddings" hoặc "OpenAI embeddings". Tuy nhiên, loại embeddings cần thiết cho việc truy xuất thực sự rất đặc biệt. Vì vậy, nếu bạn nghiêm túc muốn thực hiện tốt việc truy xuất, điều quan trọng là sử dụng các embeddings được thiết kế riêng cho mục đích này.



Learning Retrieval-oriented Embeddings

Khi xây dựng hệ thống tìm kiếm thông tin, việc học các embedding hướng đến truy vấn là rất quan trọng. Phương pháp TF-IDF (Term Frequency-Inverse Document Frequency) thường được sử dụng để gán trọng số cao cho các từ có nội dung quan trọng và hiếm gặp. Tuy nhiên, việc sử dụng các embedding ngẫu nhiên không đảm bảo điều này. Ví dụ, nếu chúng ta chỉ đơn giản lấy trung bình các word embedding trong một câu, tất cả các từ sẽ được gán cùng một trọng số. Thực tế, các từ phổ biến thường có chuẩn (norm) thấp hơn một chút so với các từ ít gặp, điều này phù hợp với mục tiêu của chúng ta là giảm thiểu ảnh hưởng của các từ thông dụng.

Để tối ưu hóa vấn đề này, chúng ta cần học các embedding hướng đến truy vấn. Phương pháp phổ biến là sử dụng kỹ thuật học đối lập (contrastive learning). Trong phương pháp này, chúng ta chọn các tài liệu tích cực (positive documents) và tiêu cực (negative documents) cho mỗi truy vấn, sau đó huấn luyện mô hình bằng cách sử dụng hàm mất mát đối lập như hinge loss hoặc các hàm mất mát xác suất tương tự.

$$\mathcal{L}(\theta, q) = \sum_{d_{\text{pos}} \in D_{\text{pos}}} \sum_{d_{\text{neg}} \in D_{\text{neg}}} \max(0, s(q, d_{\text{neg}}; \theta) - s(q, d_{\text{pos}}; \theta))$$

Nếu có sẵn các tài liệu tích cực chuẩn (gold standard positive documents), quá trình huấn luyện tương đối đơn giản. Chúng ta chỉ cần tìm cách chọn các tài liệu tiêu cực. Một cách phổ biến là tạo một batch dữ liệu, sử dụng các tài liệu tích cực đã được gán nhãn cho mỗi truy vấn, và coi các tài liệu tích cực của các truy vấn khác trong batch là tài liệu tiêu cực. Ví dụ, nếu chúng ta có

32 cặp truy vấn-tài liệu trong một batch, chúng ta sẽ sử dụng các cặp đã được gán nhãn làm tài liệu tích cực và 31 cặp còn lại làm tài liệu tiêu cực.

Tuy nhiên, phương pháp này thường chưa đủ hiệu quả vì các tài liệu tiêu cực được chọn có thể quá dễ phân biệt với tài liệu tích cực. Do đó, một kỹ thuật phổ biến khác là sử dụng "hard negatives" - những ví dụ tiêu cực có vẻ phù hợp nhưng thực tế lại không chính xác. Phương pháp DPR (Dense Passage Retrieval - Karpukhin et al. 2020) nổi tiếng đã sử dụng cả các ví dụ tiêu cực trong batch và hard negatives được tạo ra bằng cách tìm kiếm tài liệu với BM25.

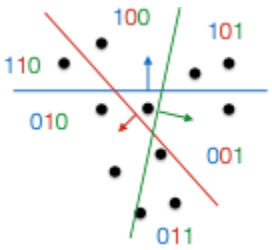
Ngoài ra, còn có các phương pháp học các bộ truy vấn (retrievers) mà không cần dữ liệu có nhãn. Ví dụ, phương pháp Contriever (Izacard et al. 2022) sử dụng hai đoạn văn ngẫu nhiên trong cùng một tài liệu làm cặp tích cực, và các đoạn văn từ các tài liệu khác làm cặp tiêu cực. Phương pháp này có thể được sử dụng cho việc tiền huấn luyện mô hình ở quy mô lớn. Sau đó, mô hình có thể được tinh chỉnh trên dữ liệu được gán nhãn để cải thiện hiệu suất.

Approximate Nearest Neighbor Search

Khi làm việc với các dense embedding models, chúng ta thường phải đối mặt với thách thức tính toán khi tìm kiếm trên tập dữ liệu lớn. Việc tính tích vô hướng giữa vector truy vấn và mọi vector tài liệu trong cơ sở dữ liệu có thể rất tốn kém về mặt tính toán. Để giải quyết vấn đề này, các phương pháp approximate nearest neighbor search đã được phát triển.

Các phương pháp này cho phép truy xuất các embedding có tích vô hướng lớn nhất với vector truy vấn trong thời gian dưới tuyến tính. Quá trình này còn được gọi là tìm kiếm tích vô hướng lớn nhất (maximum inner product search - MIPS). Dưới đây là hai phương pháp phổ biến:

Locality Sensitive Hashing (LSH):
LSH, còn được gọi là inverted index, tạo ra các phân vùng trong không gian liên tục và sử dụng chúng như một inverted index. Phương pháp này hoạt động như sau:



- Định nghĩa nhiều mặt phẳng để phân chia các điểm trong không gian embedding thành hai phần.
- Gán giá trị nhị phân (0 hoặc 1) cho mỗi điểm dựa trên vị trí của nó so với các mặt phẳng.
- Tạo ra các sparse vector dựa trên các giá trị nhị phân này.
- Sử dụng các sparse vector để tìm kiếm trong inverted index, tương tự như cách tìm kiếm trong bảng tra cứu thưa.

Ví dụ, một điểm có thể được biểu diễn bằng vector "110" nếu nó nằm ở phía phải của mặt phẳng thứ nhất và thứ hai, nhưng ở phía trái của mặt phẳng thứ ba.

Tìm kiếm dựa trên đồ thị (Graph-based search):
Phương pháp này tạo ra các "hub" - những điểm gần nhau trong không gian. Quá trình tìm kiếm được thực hiện như sau:

- Xác định các hub, tương tự như các tâm cụm (cluster centroids).
- Bắt đầu tìm kiếm từ các hub, giúp giảm đáng kể số lượng điểm cần xem xét.
- Tìm kiếm chi tiết hơn trong số các điểm đã được lọc.
- Có thể mở rộng thành cấu trúc cây với các hub chính, hub phụ và các điểm dữ liệu.

Thay vì tự triển khai các thuật toán phức tạp này, các nhà phát triển thường sử dụng các thư viện có sẵn. Hai lựa chọn phổ biến là:

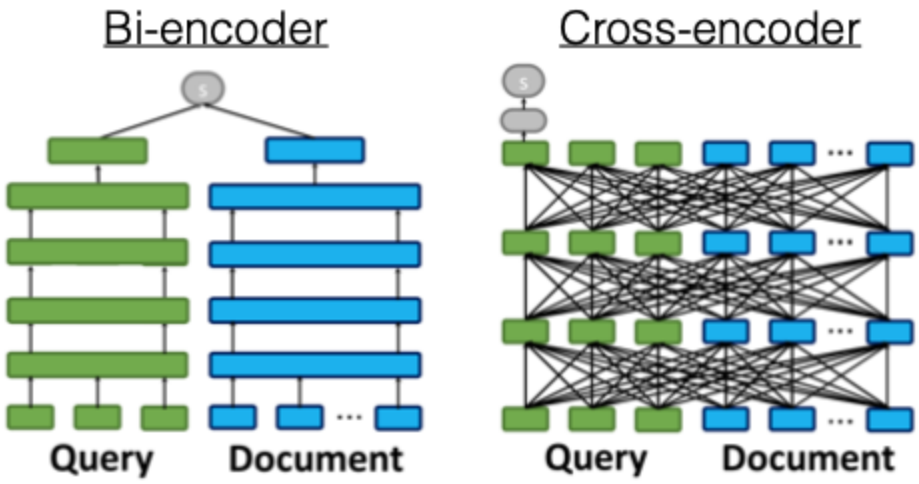
- FAISS (Facebook AI Similarity Search): Một thư viện được phát triển bởi Facebook (nay là Meta) và được sử dụng rộng rãi.
- Chroma DB: Một cơ sở dữ liệu tìm kiếm embedding "AI-native" mới nổi.

Mặc dù các phương pháp tìm kiếm gần đúng này giúp cải thiện đáng kể hiệu suất, việc sử dụng dense embedding vẫn còn một số thách thức, đặc biệt khi so sánh với các phương pháp mã hóa chéo (cross-encoders) trong một số tác vụ cụ thể.

Cross-encoder Reranking

Để tạo ra một embedding dày đặc (dense embedding) hiệu quả, bạn cần biết trước mình đang tìm kiếm điều gì. Quá trình này liên quan đến việc nén một tài liệu hoặc đoạn văn dài thành một embedding duy nhất. Tuy nhiên, trong quá trình nén này, có thể có những thông tin quan trọng liên quan đến truy vấn bị loại bỏ do khả năng lưu trữ hạn chế của embedding, dẫn đến thất bại trong việc truy xuất thông tin một cách chính xác.

Có một số phương pháp để khắc phục vấn đề này. Phương pháp đầu tiên là sử dụng cross-encoder thay vì bi-encoder. Bi-encoder mã hóa toàn bộ các truy vấn và tài liệu riêng biệt, sau đó thực hiện tìm kiếm tích vô hướng (inner product search) để tính điểm. Ngược lại, cross-encoder hoạt động bằng cách nối truy vấn và tài liệu lại với nhau, sau đó đưa chúng qua một mô hình như Transformer để tính toán điểm đầu ra.



Cross-encoder có ưu điểm là cung cấp sự linh hoạt tối đa. Các mô hình Transformer rất mạnh mẽ và có thể đánh giá mức độ liên quan một cách hiệu quả. Tuy nhiên, nhược điểm của phương pháp này là không thể sử dụng tìm kiếm láng giềng gần đúng (approximate nearest neighbor lookup) do phải xử lý qua nhiều phép biến đổi phi tuyến. Vì vậy, cross-encoder chỉ có thể được sử dụng để xếp hạng lại các tài liệu hoặc truy xuất trên một số lượng rất nhỏ tài liệu.

Tuy nhiên, nếu bạn muốn đạt được độ chính xác tối đa, tôi khuyên bạn nên sử dụng phương pháp này. Nó cho phép bạn thực hiện một bước lọc thứ cấp trên các tài liệu có vẻ liên quan nhất để xác định những tài liệu thực sự cần thêm vào ngữ cảnh của bạn.

Token-level Dense Retrieval

Trong lĩnh vực truy xuất thông tin, ngoài hai phương pháp chính là bi-encoder và cross-encoder, còn có những phương pháp trung gian kết hợp ưu điểm của cả hai. Một trong những phương pháp nổi bật nhất là ColBERT (Contextualized Late Interaction over BERT), được giới thiệu trong bài báo "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT - Khattab et al. 2020".

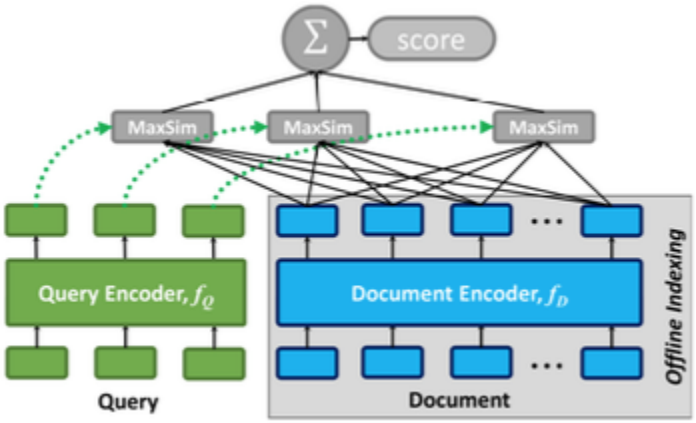
ColBERT được gọi là token-level dense retrieval hoặc "tương tác muộn" (late interaction). Đây là cách gọi đặc trưng cho cơ chế hoạt động của nó:

1. Token-level dense retrieval:
- Thay vì tạo một vector embedding duy nhất cho cả tài liệu (như bi-encoder) hoặc kết hợp truy vấn và tài liệu (như cross-encoder), ColBERT tạo ra các vector embedding cho từng token trong cả truy vấn và tài liệu.
- Mỗi token được biểu diễn bằng một vector có tính đến ngữ cảnh xung quanh nó, nhờ sử dụng mô hình ngôn ngữ như BERT.

2. Tương tác muộn:
- "Muộn" ở đây ám chỉ việc tương tác giữa truy vấn và tài liệu xảy ra ở giai đoạn cuối của quá trình, sau khi đã mã hóa riêng rẽ truy vấn và tài liệu.

- Điều này khác với cross-encoder, nơi tương tác xảy ra ngay từ đầu khi kết hợp truy vấn và tài liệu.

Cách hoạt động của ColBERT:



1. Giai đoạn lập chỉ mục (offline):
- Mỗi tài liệu được chia thành các token.
- Mỗi token được mã hóa thành một vector embedding có tính đến ngữ cảnh, sử dụng mô hình như BERT.
- Các vector embedding này được lưu trữ trong cơ sở dữ liệu.

2. Giai đoạn truy vấn (online):
- Truy vấn cũng được chia thành các token và mã hóa tương tự.
- Hệ thống thực hiện so khớp giữa mỗi token trong truy vấn với các token có điểm số cao nhất trong mỗi tài liệu.
- Điểm số cuối cùng của một tài liệu được tính bằng cách tổng hợp các điểm số của các cặp token truy vấn-tài liệu có độ tương đồng cao nhất.

Ưu điểm của ColBERT:
- Hiệu quả: Vẫn cho phép sử dụng các phương pháp tìm kiếm láng giềng gần đúng (approximate nearest neighbor search) hiệu quả, không như cross-encoder.
- Chính xác: Cung cấp độ chính xác cao hơn so với bi-encoder truyền thống, gần với cross-encoder.
- Linh hoạt: Có thể xử lý hiệu quả các truy vấn dài và phức tạp.

Nhược điểm:
- Yêu cầu lưu trữ cao: Cơ sở dữ liệu vector sẽ lớn hơn n lần so với bi-encoder, với n là số lượng token trung bình trong mỗi tài liệu. Ví dụ, nếu mỗi tài liệu có trung bình 100 token, cơ sở dữ liệu sẽ lớn hơn khoảng 100 lần so với phương pháp bi-encoder.
- Tính toán phức tạp hơn: Quá trình so khớp và tính toán điểm số phức tạp hơn so với bi-encoder.

Để giảm bớt vấn đề về lưu trữ, có một số phương pháp tối ưu hóa như:

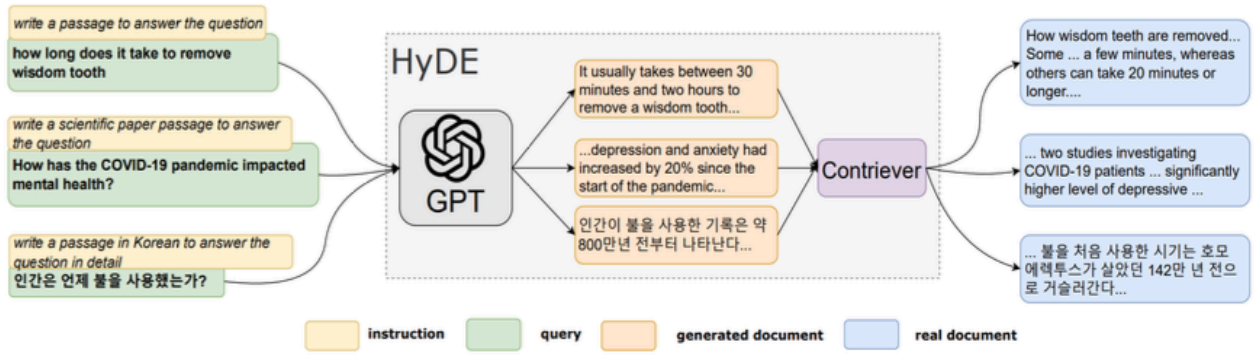
- Token pruning: Loại bỏ các token ít quan trọng.
- Dimension reduction: Giảm kích thước của các vector embedding.
- Quantization: Sử dụng ít bit hơn để biểu diễn mỗi giá trị trong vector.

ColBERT là một lựa chọn tốt khi bạn cần độ chính xác cao hơn bi-encoder nhưng vẫn muốn duy trì khả năng mở rộng tốt hơn cross-encoder. Tuy nhiên, khi triển khai, bạn cần cân nhắc kỹ về tài nguyên lưu trữ và tính toán có sẵn.

Hypothetical Document Embeddings (Gao et al. 2022)

Cuối cùng, chúng ta sẽ khám phá một phương pháp mới trong việc cải thiện độ chính xác của việc truy xuất thông tin, đặc biệt là khi đối mặt với các truy vấn ngắn và không chuẩn ngữ pháp. Phương pháp này được phát triển tại CMU bởi Lal, và được gọi là "hypothetical document embedding".

Ý tưởng chính của phương pháp này là thay vì cố gắng khớp một truy vấn ngắn với một tài liệu dài, chúng ta sẽ sử dụng một mô hình ngôn ngữ lớn (LLM) để tạo ra một tài liệu giả định từ truy vấn. Cụ thể, chúng ta đưa truy vấn vào mô hình cùng với một prompt và yêu cầu mô hình tạo ra một tài liệu có vẻ như là câu trả lời cho truy vấn đó. Tài liệu giả định này sau đó sẽ được sử dụng để so sánh với các tài liệu thực tế, với hy vọng rằng nó sẽ giống với các tài liệu cần truy xuất hơn so với truy vấn gốc.



Phương pháp này đã chứng minh được hiệu quả trong việc cải thiện độ chính xác, đặc biệt là trong các nhiệm vụ khó và nằm ngoài miền dữ liệu mà mô hình đã được huấn luyện.

Khi bắt đầu với việc truy xuất thông tin, một phương pháp cơ bản và dễ triển khai là sử dụng BM25. Phương pháp này khá mạnh mẽ khi áp dụng cho các miền dữ liệu mới, trong khi các mô hình dựa trên embedding có thể cho kết quả rất tốt hoặc rất tệ tùy thuộc vào mức độ khác biệt của miền dữ liệu so với mô hình embedding cơ bản.

Tuy nhiên, nếu mục tiêu là đạt được độ chính xác cao nhất, các mô hình đã được tinh chỉnh sẽ là lựa chọn tốt hơn. Nếu không lo ngại về hiệu suất tính toán, việc sử dụng các phương pháp như ColBERT với truy xuất ở mức token sẽ mang lại độ chính xác cao. Ngoài ra, các mô hình

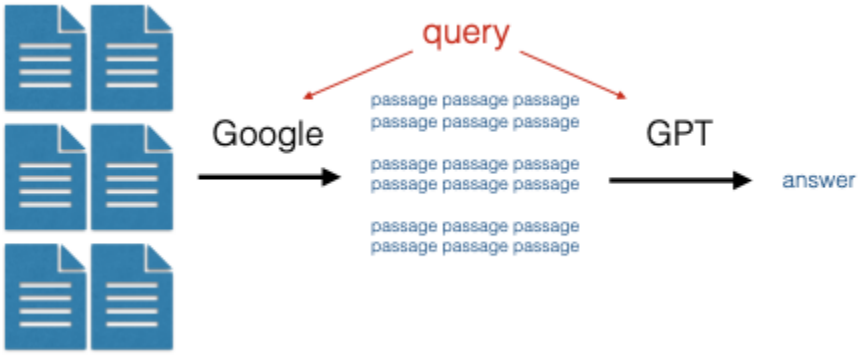
kiểu bi-encoder cũng được hỗ trợ tốt hơn trong các cơ sở dữ liệu vector tiêu chuẩn như FAISS và Chroma, giúp việc triển khai nhanh chóng và dễ dàng hơn.

Retrieval-Reader Models

Simple: Just Chain Retrieval + Reading

Khi nói đến các mô hình sinh tăng cường truy xuất thực tế (retrieval augmented generation models), chúng ta có mô hình truy xuất-đọc (retriever-reader models). Cách hoạt động đơn giản nhất của những mô hình này là kết hợp quá trình truy xuất và đọc. Bạn sử dụng một bộ truy xuất (retriever) và một mô hình đọc (reader model) có sẵn, sau đó kết nối chúng lại với nhau.

Ví dụ, với một truy vấn, bạn có thể tìm kiếm trên Google để lấy một số đoạn văn, rồi đưa chúng vào một mô hình GPT để nhận được câu trả lời. Phương pháp này khá hiệu quả, dễ triển khai và cho kết quả tốt.



Tuy nhiên, cần lưu ý rằng các đoạn văn được nối vào ngữ cảnh (context), khiến ngữ cảnh có thể trở nên khá dài và tốn kém khi xử lý. Đây là một thách thức cần giải quyết khi sử dụng phương pháp này.

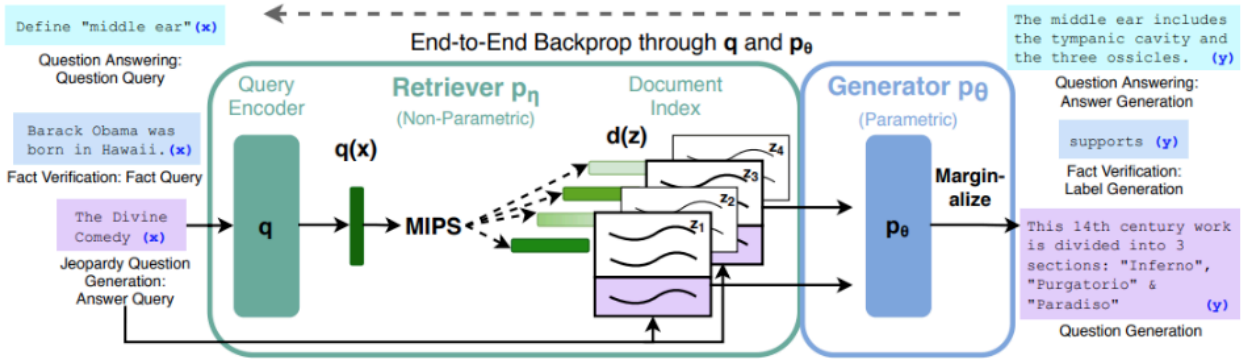
Retriever + Generator End-to-end Training (“RAG") (Lewis et al. 2020)

Ngoài ra, còn có nhiều phương pháp để huấn luyện mô hình Retriever và Generator theo cách end-to-end. Một trong những nghiên cứu nổi bật là bài báo đã đặt nền móng cho cái tên "RAG" (Retrieval-Augmented Generation). Bài báo này đề xuất một số phương pháp để huấn luyện Retriever và Reader nhằm cải thiện độ chính xác.

Cụ thể, trong phương pháp RAG-P của Lewis và cộng sự, Reader được huấn luyện bằng cách tối đa hóa khả năng sinh (generation likelihood) dựa trên một tài liệu được truy xuất duy nhất. Đối với Retriever, phương pháp này tối ưu hóa trọng số hỗn hợp (mixture weight) trên các tài liệu để tối đa hóa khả năng tổng thể.

Dưới đây là một sơ đồ minh họa cho quy trình này: Đầu tiên, bạn có bộ mã hóa truy vấn (query encoder). Sau đó, bạn chạy Retriever với phương pháp tìm kiếm tích vô hướng lớn nhất

(maximum inner product search), từ đó nhận được một số tài liệu, mỗi tài liệu đi kèm với một điểm số. Dựa trên các tài liệu và điểm số này, bạn tiến hành sinh (generate) với mỗi tài liệu trong ngữ cảnh và sau đó tổng hợp các xác suất, nhân với các trọng số tương ứng.



End-to-end Training Equations (Lewis et al. 2020)

Để hiểu rõ hơn về cơ chế hoạt động của mô hình RAG, ta cần xem xét các phương trình cụ thể. Quá trình sinh là một mô hình kết hợp, trong đó ta chọn một tài liệu và sinh nội dung từ tài liệu đó. Công thức tổng quát như sau:

$$P_{\text{RAG}}(y|x) \approx \prod_i \sum_{z \in \text{top-k}(p(\cdot|x))} \underbrace{p_\eta(z|x)}_{\text{Retriever}} \underbrace{p_\theta(y_i|x, z, y_{1:i-1})}_{\text{Generator}}$$

Trong đó:
- $p_\eta(z|x)$ là xác suất chọn tài liệu z cho truy vấn x
- $p_\theta(y_i|x, z, y_{1:i-1})$ là xác suất của token tiếp theo y, với điều kiện có tài liệu z

Phần $p_\eta(z|x)$ có thể được xem là bộ truy xuất (retriever), còn $p_\theta(y_i|x, z, y_{1:i-1})$ là bộ sinh/đọc (generator/reader).

Một điểm quan trọng cho phép huấn luyện end-to-end là xác suất của bộ truy xuất dựa trên các embedding. Cụ thể:

$$p_\eta(z|x) \propto \exp(\mathbf{d}(z)^\top \mathbf{q}(x)) \quad \mathbf{d}(z) = \text{enc}_d(z), \quad \mathbf{q}(x) = \text{enc}_q(x)$$

Xác suất này tỷ lệ thuận với hàm mũ của tích vô hướng giữa embedding tài liệu và embedding truy vấn. Điều này cho phép bộ truy xuất điều chỉnh để tăng độ tương đồng cho các tài liệu hữu ích, mà không cần chú thích cụ thể về tính hữu ích của từng tài liệu.

Tuy nhiên, một vấn đề quan trọng khi huấn luyện các mô hình như vậy là chỉ mục tìm kiếm (search index) sẽ trở nên lỗi thời. Việc tạo chỉ mục cho hàng triệu hoặc hàng tỷ tài liệu rất tốn

kém về mặt tính toán. Trong khi đó, việc tra cứu sử dụng tìm kiếm láng giềng gần đúng (approximate nearest neighbor) có thể thực hiện trong thời gian cận tuyến tính hoặc logarit.

Để giải quyết vấn đề này, bài báo RAG của Lewis và cộng sự đề xuất chỉ huấn luyện embedding truy vấn và giữ nguyên embedding tài liệu. Một giải pháp thay thế được đề xuất trong bài báo REALM là sử dụng một quá trình bất đồng bộ để cập nhật lại chỉ mục tìm kiếm trong quá trình huấn luyện, nhưng phương pháp này khá phức tạp và tốn kém.

Do đó, việc sử dụng embedding tài liệu cố định và chỉ cập nhật embedding truy vấn là một phương pháp phổ biến và hiệu quả trong thực tế.

When we do Retrieve?

Một vấn đề quan trọng cần xem xét trong quá trình truy xuất thông tin là thời điểm thực hiện truy xuất. Có nhiều phương pháp khác nhau để thực hiện điều này. Bài báo RAG (Lewis et al. 2020) đã đề cập trước đó chỉ thực hiện truy xuất một lần ngay từ đầu quá trình tạo ra. Cụ thể, nó lấy một tài liệu duy nhất và tạo ra toàn bộ đầu ra. Đây là phương pháp mặc định được sử dụng bởi hầu hết các hệ thống.

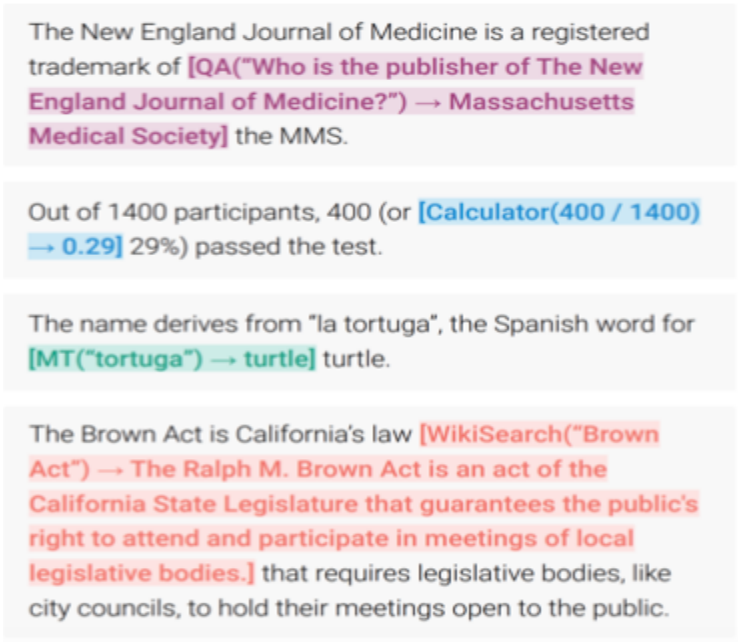
Tuy nhiên, còn có các lựa chọn khác. Chúng ta có thể thực hiện truy xuất nhiều lần trong quá trình tạo ra khi cần thiết. Cách thức hoạt động của phương pháp này có thể được thực hiện bằng cách tạo ra một "search token" (Schick et al. 2023), chỉ ra rằng chúng ta nên bắt đầu tìm kiếm, hoặc tìm kiếm khi mô hình không chắc chắn (Jiang et al. 2023).

Một cách tiếp cận khác là thực hiện truy xuất cho mỗi token. Điều này có thể được thực hiện bằng cách tìm các embedding cuối cùng tương tự và sử dụng chúng để ảnh hưởng đến xác suất (Khandelwal et al. 2019), hoặc xấp xỉ sự chú ý (attention) bằng cách sử dụng các neighbors gần nhất (Bertsch et al. 2023).

Triggering Retrieval w/ Tokens

Phương pháp kích hoạt truy xuất bằng token embeddings được đề xuất trong Toolformer (Schick et al. 2023). Cách thức hoạt động của nó là tạo ra các token để kích hoạt quá trình truy xuất hoặc các công cụ khác. Trong phương pháp cụ thể này, có nhiều công cụ khác nhau, bao gồm việc hỏi một mô hình QA (Question Answering), sử dụng máy tính, hoặc hệ thống dịch máy.

Đối với truy xuất tăng cường (retrieval augmented generation), nó có chức năng tìm kiếm Wikipedia, về cơ bản là tra cứu thông tin trên Wikipedia và sử dụng kết quả để ảnh hưởng đến xác suất cuối cùng của việc tạo ra văn bản.

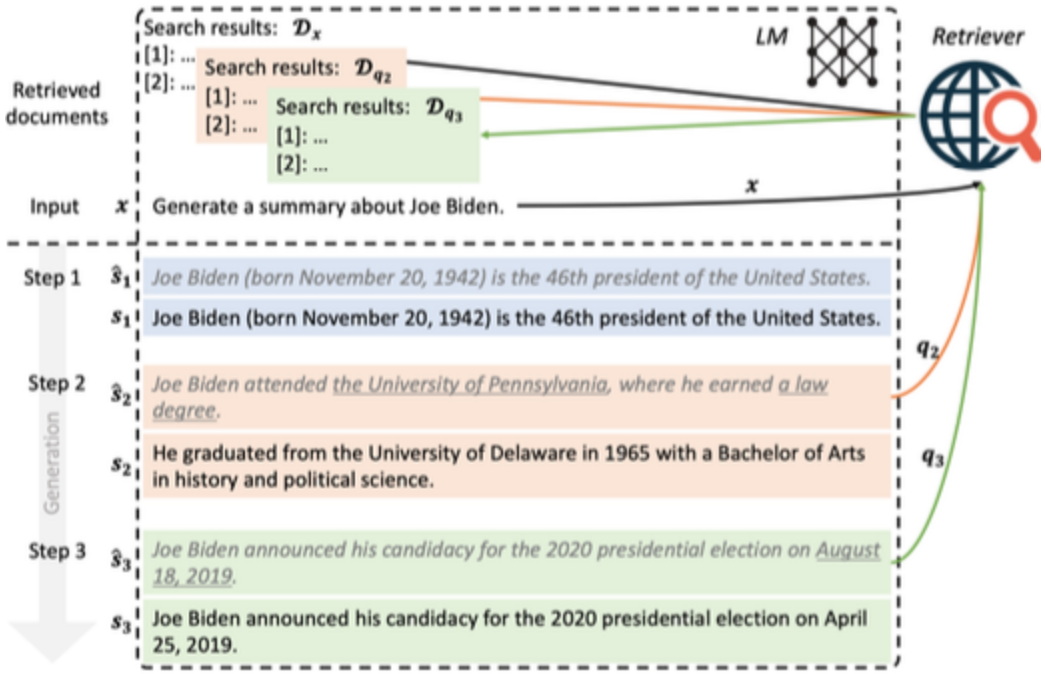


Quá trình đào tạo được thực hiện theo cách lặp đi lặp lại. Nó tạo ra các ví dụ về việc sử dụng công cụ một cách hữu ích. Khi các công cụ cải thiện xác suất của đầu ra tiếp theo, điều này được coi là một ví dụ tích cực và được sử dụng để tiếp tục đào tạo mô hình.

Phương pháp này đã có ảnh hưởng rất lớn. Thực tế, đây là cách mà các chức năng được triển khai trong ChatGPT ngày nay, không chỉ để thực hiện truy xuất mà còn để sử dụng các công cụ khác. Ví dụ như tạo mã hoặc tạo hình ảnh và các chức năng khác tương tự.

Triggering Retrieval w/ Uncertainty

Một phương pháp khác là kích hoạt quá trình truy xuất thông tin với các ước lượng độ không chắc chắn (Jiang et. al 2023). Các tác giả cố gắng tạo ra nội dung và sau đó thực hiện truy xuất nếu mô hình ngôn ngữ có độ chắc chắn thấp. Dưới đây là sơ đồ minh họa cách thức hoạt động của phương pháp này.



Cụ thể, nếu chúng ta có một số tài liệu đã được truy xuất, chúng ta có thể yêu cầu mô hình "generate a summary about Joe Biden". Khi mô hình tạo ra bản tóm tắt, có thể trong lần đầu tiên, mô hình ngôn ngữ có độ tin cậy cao và vì vậy chúng ta chỉ cần tạo ra đầu ra. Tuy nhiên, trong bước tiếp theo, nếu mô hình tạo ra thông tin như "Joe Biden attended the University of Pennsylvania where he earned a law degree", nhưng mô hình có thể không chắc chắn về thông tin này, nó có thể có xác suất thấp đối với một số thực thể quan trọng.

Dựa trên điều này, chúng ta sẽ tạo ra một truy vấn bằng cách loại bỏ các phần có xác suất thấp và thực hiện tìm kiếm. Phương pháp này cũng tương tự như phương pháp hypothetical edings, nơi chúng ta tạo ra một tài liệu mà chúng ta nghĩ sẽ giống với tài liệu mà chúng ta muốn tìm, sử dụng nó để tạo ra kết quả tìm kiếm và sau đó tạo ra đầu ra. Chúng ta tiếp tục làm như vậy và bất cứ khi nào có đầu ra có độ tin cậy cao, chúng ta không thực hiện truy xuất mà chỉ tạo ra trực tiếp từ các tham số của mô hình. Nhưng bất cứ khi nào có đầu ra có độ tin cậy thấp, chúng ta thực hiện truy xuất và dựa vào đó để tạo ra đầu ra.

Tôi nghĩ đây là một phương pháp hay có thể được sử dụng. Nhược điểm của phương pháp này là đôi khi bạn cần tạo ra đầu ra hai lần vì bạn sẽ tạo ra đầu ra một lần, sau đó tìm các phần có độ tin cậy thấp và tạo ra lần nữa. Nhưng nếu bạn thực sự quan tâm đến chất lượng của đầu ra, tôi nghĩ đây là một điều hợp lý để làm.

Token-level Softmax Modification

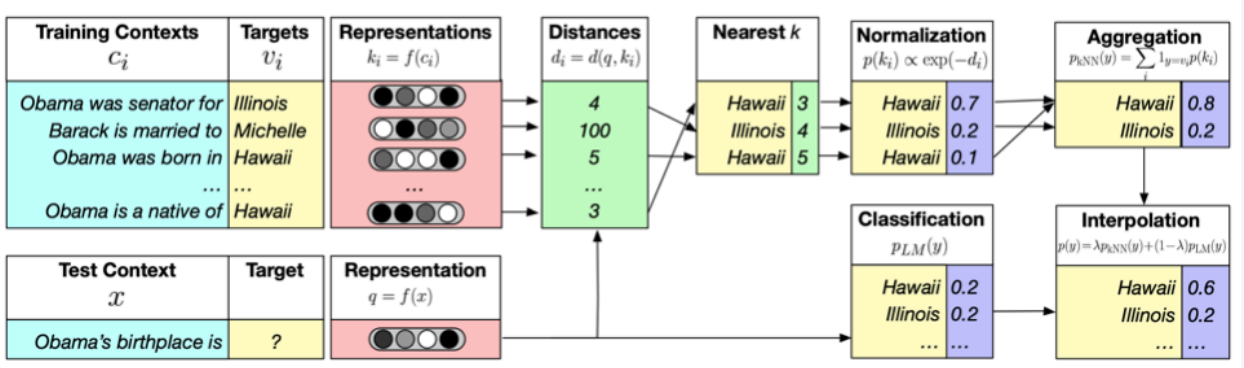
Tiếp theo, chúng ta sẽ tìm hiểu về các phương pháp truy xuất theo từng token (token by token retrieval). Một trong những phương pháp đã phổ biến ý tưởng này là kNN-LM (Khandelwal et al. 2019). Cách thức hoạt động của nó là truy xuất các ví dụ tương tự và sử dụng các token tiếp

theo từ những ví dụ này. Về một khía cạnh nào đó, có thể coi đây như một mô hình ngôn ngữ dựa trên đếm (count-based language model).

Nhắc lại về mô hình ngôn ngữ dựa trên đếm, chúng ta sẽ lấy token trước đó và tính xác suất của token tiếp theo bằng cách cộng tất cả các token tiếp theo và chia cho tổng số lần xuất hiện của token trước đó.

Với nền tảng đó, chúng ta có thể hiểu cách kNN-LM hoạt động như sau:

- Chúng ta có ngữ cảnh văn bản X và muốn tạo ra đầu ra mục tiêu
- Riêng biệt với điều này, chúng ta có tất cả các ngữ cảnh huấn luyện (training contexts) từ dữ liệu huấn luyện
- Chúng ta mã hóa tất cả các ngữ cảnh huấn luyện này bằng cách tính toán biểu diễn của lớp cuối cùng hoặc gần cuối cùng của mô hình
- Song song với việc đó, chúng ta ghi nhớ từ tiếp theo xuất hiện sau ngữ cảnh này



Kết quả là chúng ta có một kho dữ liệu gồm các biểu diễn và các từ tiếp theo. Sau đó:

- Lấy biểu diễn của ngữ cảnh hiện tại
- Tính khoảng cách giữa ngữ cảnh hiện tại và tất cả các ngữ cảnh tương tự trong cơ sở dữ liệu
- Lấy K ví dụ gần nhất (ví dụ như "Hawaii", "Illinois" và "Hawaii")
- Thực hiện chuẩn hóa dựa trên khoảng cách để có được phân phối xác suất cho tất cả các token tiếp theo
- Khi các token được lặp lại nhiều lần, chúng ta tổng hợp tất cả số lần đếm (ví dụ: Hawaii: 0.8, Illinois: 0.2)
- Cuối cùng, nội suy kết quả này với xác suất từ mô hình ngôn ngữ tiêu chuẩn bằng hệ số nội suy Lambda

Ưu điểm của phương pháp này là cho phép chúng ta định vị rõ ràng đầu ra của mình trong các ví dụ cụ thể. Đây là một cách hiệu quả để cải thiện xác suất của các mô hình, cải thiện dịch máy và các ứng dụng khác.

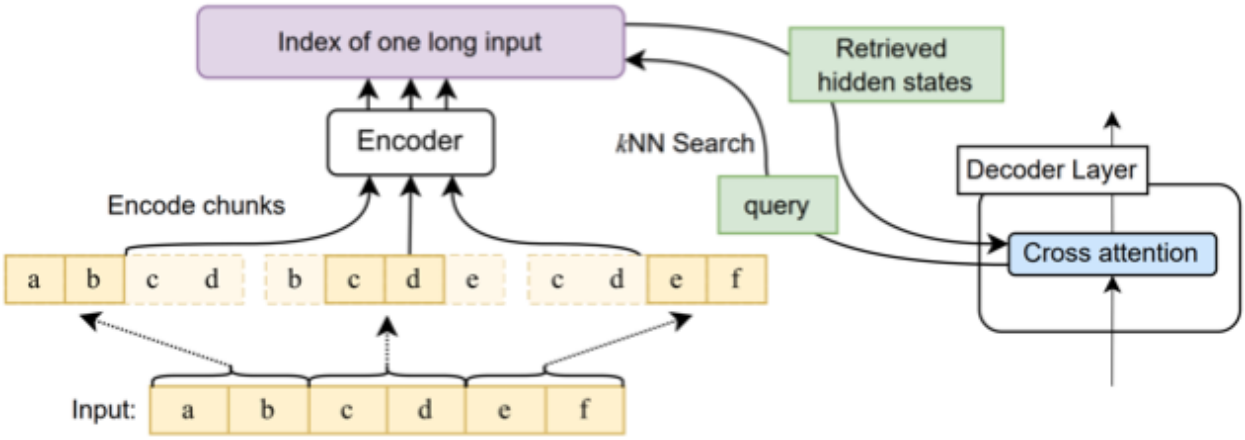
Tuy nhiên, nhược điểm là nó thêm vào một thành phần phụ cho mô hình và các siêu tham số bổ sung như Lambda. Điều này làm cho việc tinh chỉnh trở nên phức tạp hơn và không hoạt động hiệu quả trong mọi tình huống.

Token-level Approximate Attention

Một phương pháp khác được đề xuất bởi Amanda Bertsch là Unlimiformer (Bertsch et al. 2023). Về cơ bản, Unlimiformer nhận thấy rằng attention tự nó là một phép tìm kiếm tích trong (inner product search) và nó thực hiện top-k attention.

Cách thức hoạt động của phương pháp này như sau:

- Đầu tiên, xử lý đầu vào bằng cửa sổ trượt (sliding window)
- Sau đó thực hiện attention sử dụng chỉ mục vector (vector index)



Khi có một đầu vào rất dài cần mã hóa, quy trình sẽ là:

- Mã hóa từng phần nhỏ: ví dụ mã hóa AB, sau đó mã hóa CD, rồi mã hóa EF
- Nối chúng lại thành một chỉ mục lớn của một đầu vào dài

Điều này tương tự như những gì đã làm trong kNN-LM - nối tất cả các embedding thành một đầu vào duy nhất. Tuy nhiên, sự khác biệt là việc này được thực hiện với các giá trị mà chúng ta đang chú ý đến (attending to), thay vì chỉ ở lớp cuối cùng.

Điểm thú vị là khi đã có một chỉ mục của đầu vào dài, để thực hiện phiên bản attention tiếp theo:

- Thực hiện tìm kiếm kNN từ query
- Lấy các hidden states đã được truy xuất
- Thực hiện attention trên chúng

Ưu điểm của phương pháp này là trong trường hợp cực đoan, nó không thay đổi gì về mô hình. Nghĩa là, nếu đầu vào đủ nhỏ để có thể mã hóa trong một chunk duy nhất và chúng ta thực hiện tìm kiếm kNN chính xác trên tất cả các embedding trong chunk đó, kết quả sẽ giống hệt như attention thông thường.

Tuy nhiên, có một số xấp xỉ trong quá trình này:

- Khi mã hóa các chunk, chúng có thể không hoàn toàn giống như khi mã hóa toàn bộ cùng một lúc
- Chúng ta cũng loại bỏ một số giá trị có tích trong (inner products) rất thấp

Do những xấp xỉ này, có một số sai khác, nhưng trong trường hợp cực đoan khi không có xấp xỉ nào được thực hiện, mô hình sẽ hoàn toàn giống với mô hình ban đầu. Đây là một điểm hấp dẫn của phương pháp này.

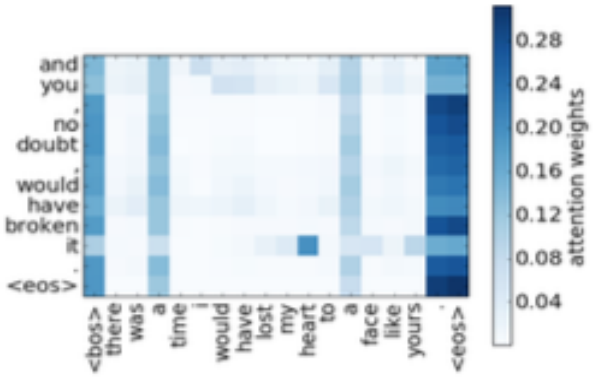
Về mặt thực nghiệm, phương pháp này cho kết quả rất tốt trên các khoảng cách dài và chúng ta luôn có thể cải thiện các xấp xỉ để nâng cao hiệu quả của mô hình. Đây là một phương pháp đáng để nghiên cứu và tìm hiểu thêm.

Long-context Transformers

Training Transformers Over Longer Sequences

Trong phần cuối cùng của bài viết này, tôi muốn thảo luận về các mô hình Transformer với ngữ cảnh dài. Đây là những mô hình được huấn luyện đặc biệt để có thể xử lý các ngữ cảnh dài một cách hiệu quả.

Một cách để huấn luyện mô hình với ngữ cảnh dài là gộp tất cả ngữ cảnh lại với nhau. Thực tế, ngay sau khi Transformer được giới thiệu, một nghiên cứu (Voita et al. 2018) đã chỉ ra rằng việc làm này có thể giúp mô hình học được các hiện tượng ở cấp độ tài liệu. Ví dụ, mô hình có thể xác định khi nào nhiều từ cùng tham chiếu đến một đối tượng, hay còn gọi là "co-reference", và các hiện tượng khác tương tự.

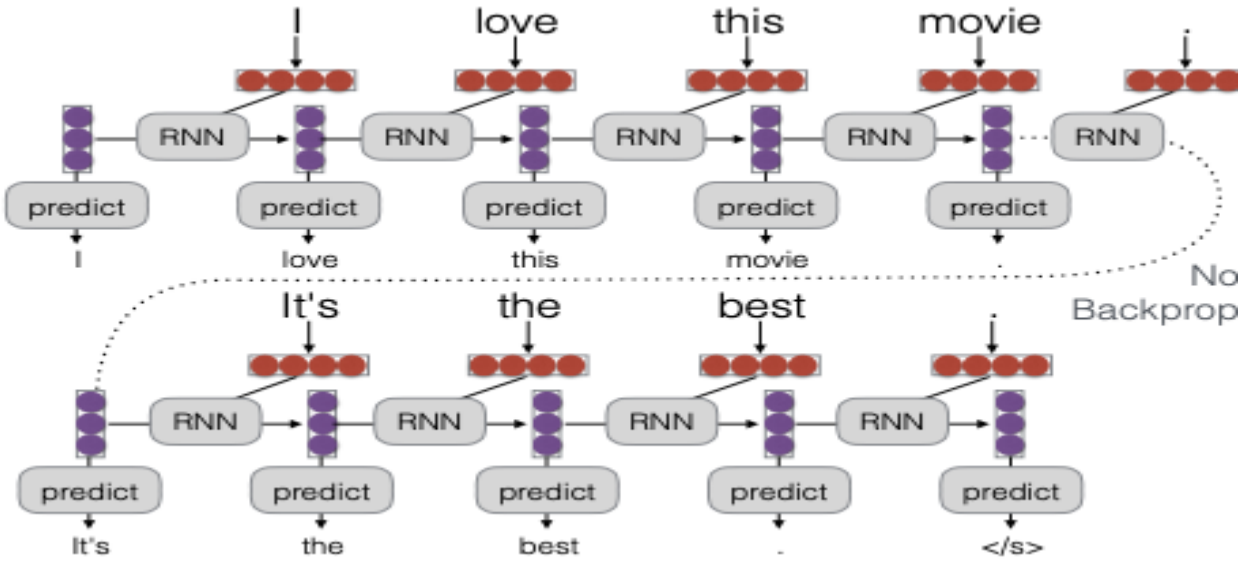


Tuy nhiên, vấn đề của Transformer là tính toán có độ phức tạp bậc hai theo độ dài chuỗi, do phải nhân tất cả các vector truy vấn (query vectors) với tất cả các vector khóa (key vectors). Điều này gây ra vấn đề lớn khi các chuỗi trở nên rất dài.

In RNNs: Pass State + Truncated Backprop

Nếu nhìn lại các mô hình RNN (Recurrent Neural Networks) mà chúng ta đã học, chúng không gặp phải vấn đề về độ phức tạp tính toán như các mô hình attention. Lý do là vì tính toán trong RNN có độ phức tạp tuyến tính theo độ dài của chuỗi - chúng ta chỉ cần truyền trạng thái RNN và thực hiện cùng một phép tính mỗi lần. Do đó, không có thành phần bậc hai trong việc tính toán RNN.

Một ưu điểm khác của RNN là khả năng lưu trữ trạng thái vô hạn trong quá trình lan truyền xuôi (forward pass). Điều này được thực hiện bằng cách tính toán trạng thái ẩn (hidden state) và sau đó loại bỏ phần còn lại của đồ thị tính toán đã được sử dụng để tính toán trạng thái ẩn đó. Không giống như trong UNLIMIformer mà chúng ta đã thảo luận trước đó - nơi cần phải thực hiện các phép xấp xỉ - trong trường hợp này không cần bất kỳ xấp xỉ nào.



Tuy nhiên, có một vấn đề khi thực hiện lan truyền ngược (backprop). Thông thường, để thực hiện lan truyền ngược, chúng ta cần duy trì toàn bộ trạng thái của đồ thị tính toán. Một phương pháp phổ biến để giải quyết vấn đề này là truyền trạng thái RNN từ câu trước đó nhưng không thực hiện lan truyền ngược vào câu trước đó. Phương pháp này được gọi là "truncated backprop" hoặc "truncated back propagation through time". Điều này cho phép chúng ta về cơ bản huấn luyện các mô hình với ngữ cảnh vô hạn, hoặc ít nhất là các mô hình có thể truyền ngữ cảnh vô hạn, ngay cả khi chúng ta không thực hiện lan truyền ngược vào các phần trước đó.

Tất nhiên, một hạn chế của các mô hình RNN trên các ngữ cảnh dài là chúng có thể chậm do phụ thuộc tuần tự. Chúng không lý tưởng cho việc chạy trên GPU hoặc các thiết bị tương tự. Vấn đề này đã được cải thiện bởi các kiến trúc gần đây như Mamba và RW KV, những kiến trúc này phù hợp hơn với việc huấn luyện trên GPU trong khi vẫn duy trì độ phức tạp thời gian tuyến tính.

Truncated BPTT + Transformers

Ý tưởng “truncated back propagation through time” cũng có thể được áp dụng cho các mô hình Transformer. Điều này được thể hiện rõ trong bài báo "Transformer-XL" (Dai et al. 2019). Mô hình này cố gắng cố định các vector từ câu trước đó.

Trong Transformer tiêu chuẩn, mỗi vector sẽ thực hiện attention với tất cả các vector khác trong ngữ cảnh hiện tại. Ngược lại, Transformer-XL hoạt động theo cách khác:

- Khi có một đoạn mới mà bạn muốn thực hiện lan truyền ngược
- Bạn vẫn thực hiện attention đến tất cả các token trước đó trong đoạn trước
- Tuy nhiên, không thực hiện lan truyền ngược vào chúng

Về cơ bản, đây chính là “truncated back propagation through time” nhìn từ góc độ của Transformer. Phương pháp này có một ưu điểm đáng chú ý so với Transformer đa tầng, nó cho phép thực hiện attention rất xa về phía trước:

- Lớp cuối cùng thực hiện attention với các phần tử trong cửa sổ ngữ cảnh trước đó
- Lớp áp cuối không chỉ thực hiện attention với một cửa sổ ngữ cảnh trước đó mà với nhiều cửa sổ ngữ cảnh trước đó

Điều này cho phép thực hiện attention hiệu quả trên ngữ cảnh rất dài vì mỗi lần ngữ cảnh mở rộng theo cách thức mũ (exponential). Gần đây, một mô hình phổ biến có tên là Mistral (Jiang et al. 2023) sử dụng sliding window attention, về cơ bản là cùng một cơ chế được đề xuất bởi Transformer-XL. Điều này cho thấy phương pháp này vẫn đang được sử dụng trong các hệ thống thực tế.

Sparse Transformers (Child et al. 2019)

Một trong những nghiên cứu có ảnh hưởng lớn trong lĩnh vực Transformer là "Sparse Transformer". Điểm đột phá của kiến trúc này nằm ở cách nó xử lý cơ chế attention.

Thay vì thực hiện attention với tất cả các trạng thái (states) trước đó như Transformer truyền thống, Sparse Transformer chỉ tập trung vào mỗi n trạng thái trước đó. Cụ thể, với một trạng thái hiện tại, mô hình sẽ:

- Thực hiện attention với n tokens gần nhất (local attention)
- Đồng thời thực hiện attention với các chunks của M tokens ở khoảng cách xa hơn (longer-range attention)

Cách tiếp cận này tương tự như việc kết hợp:

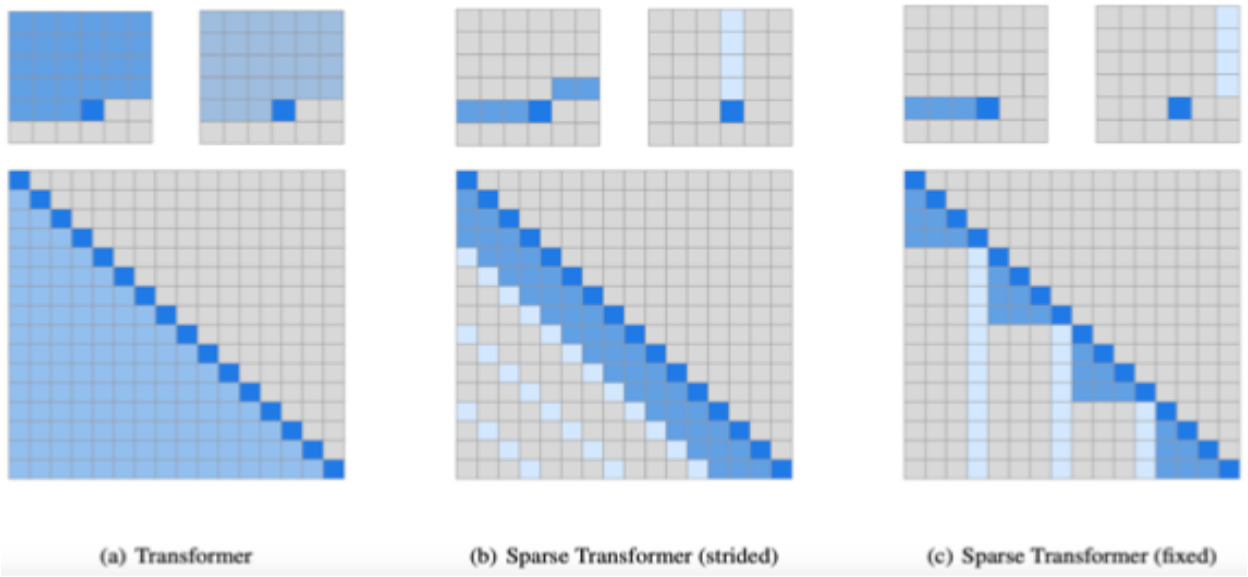
- Strided convolution (tích chập có bước nhảy): cho phép giảm độ phức tạp tính toán
- Pyramidal recurrent neural networks (mạng neural hồi quy dạng kim tự tháp): xử lý thông tin ở nhiều tầng trừu tượng khác nhau

Ưu điểm chính của Sparse Transformer là khả năng xử lý context dài hơn nhiều so với Transformer thông thường, trong khi chỉ tăng độ phức tạp tính toán một cách tối thiểu. Điều này

đạt được nhờ việc kết hợp thông minh giữa attention cục bộ (local) và attention tầm xa (longer-range), cho phép mô hình nắm bắt được cả mối quan hệ ngắn hạn lẫn dài hạn trong dữ liệu.

Cơ chế Attention của Sparse Transformers (bổ sung):

Thay vì thực hiện attention với tất cả các tokens trước đó như Transformer gốc, Sparse Transformer áp dụng hai loại attention patterns chính:



1. Strided Attention Pattern

- Với mỗi token tại vị trí i, mô hình chỉ tính attention với các token cách nhau một khoảng cách cố định n
- Ví dụ: Với stride n = 3, token tại vị trí i sẽ attend tới các token ở vị trí i-3, i-6, i-9,...
- Pattern này giúp nắm bắt các mối quan hệ có tính chu kỳ và giảm đáng kể số lượng phép tính

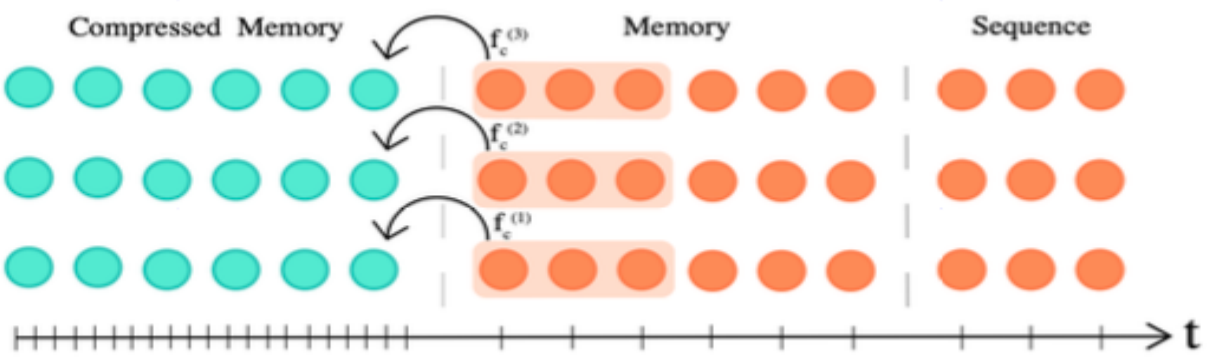
2. Fixed Attention Pattern

- Chia chuỗi input thành các chunks có kích thước cố định M
- Mỗi token sẽ attend tới:
- Tất cả các token trong chunk hiện tại (local attention)
- Token đầu tiên của mỗi chunk trước đó (global attention)
- Ví dụ: Với M = 128, chuỗi được chia thành các chunks 0-127, 128-255,...
- Pattern này đảm bảo thông tin có thể truyền giữa các chunks khác nhau

Compressing Previous States (Rae et al. 2019)

Một phương pháp khác có phần tương tự nhưng khác biệt trong cách triển khai là Compressive Transformer. Trong Compressive Transformer, chúng ta cũng có ý tưởng về bộ nhớ cục bộ và bộ nhớ nén dài hạn, nhưng có một bước nén rõ ràng, giúp tạo ra bộ nhớ nén (compressed memory) một cách trực tiếp. Phương pháp này mang lại sự linh hoạt hơn, cho phép bạn thu

thập tất cả những thông tin liên quan từ bộ nhớ cục bộ và nén chúng lại. Đây là một phương pháp đáng để suy nghĩ và khám phá thêm.



Low-rank Approximation

Một hướng tiếp cận đầy hứa hẹn để tối ưu hóa Transformer là sử dụng các phương pháp xấp xỉ low-rank cho ma trận attention. Phương pháp này giải quyết thách thức về chi phí tính toán cao của attention matrix trong Transformer truyền thống.

Thay vì tính toán ma trận attention đầy đủ, ta có thể xấp xỉ nó bằng một ma trận có rank thấp hơn. Điều này dựa trên quan sát rằng:

- Ma trận attention thường có nhiều thông tin dư thừa
- Có thể capture được thông tin quan trọng nhất thông qua xấp xỉ rank thấp
- Giảm đáng kể độ phức tạp tính toán mà vẫn duy trì được hiệu năng

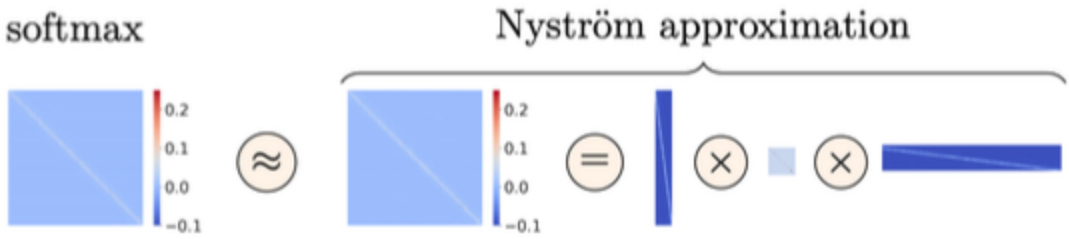
Các phương pháp chính:

1. Linformer (Wang et al. 2020)

- Thêm các low-rank linear projections vào các vị trí phù hợp trong mô hình
- Giảm độ phức tạp từ $O(n^2)$ xuống $O(n)$ với n là độ dài chuỗi
- Vẫn duy trì được khả năng học các dependencies dài trong dữ liệu

2. Nyströmformer (Nystrom-based Transformer - Xiong et al. 2021)

- Sử dụng phương pháp Nyström để xấp xỉ ma trận attention
- Dựa trên việc lấy mẫu các landmark points
- Hiệu quả trong việc xấp xỉ softmax trên các vectors attention lớn



Thay vì tính softmax trực tiếp trên một attention vector lớn như Transformer truyền thống:

- Các phương pháp này sử dụng vectors rank thấp (low-rank vectors)
- Xấp xỉ phép tính softmax
- Tương tự như kỹ thuật được sử dụng trong Linear Attention (LARA)
- Giảm đáng kể chi phí tính toán và bộ nhớ

Benchmarks for Long-context Models

Khi nói đến việc đánh giá các mô hình xử lý chuỗi dài (long sequence models), có một số tiêu chuẩn đánh giá (benchmarks) đáng chú ý. Một trong những benchmark nổi tiếng nhất là "Long Range Arena" (Tay et al. 2020) - đây là một bộ đánh giá tổng hợp chủ yếu bao gồm các tác vụ không thuộc lĩnh vực xử lý ngôn ngữ tự nhiên (non-NLP tasks). Mặc dù Long Range Arena được sử dụng rộng rãi cho việc mô hình hóa chuỗi dài, nhưng kết quả từ benchmark này thường có sự khác biệt đáng kể so với kết quả thu được từ các tác vụ mô hình hóa ngôn ngữ khoảng cách xa (long-distance language modeling).

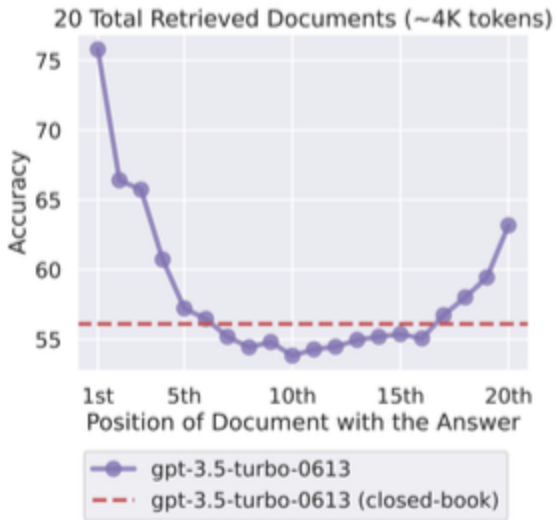
Ngoài ra, một benchmark khác đáng chú ý là "Scrolls" (Shaham et al. 2022) - một bộ công cụ kết hợp nhiều tác vụ theo phong cách hỏi đáp (QA-style) và tóm tắt (summarization) với ngữ cảnh rất dài. Scrolls làm việc trên nhiều loại văn bản như tiểu thuyết, sách, báo cáo chính phủ và các tài liệu dài khác. Đây là một lựa chọn tốt cho việc đánh giá hiệu suất của các mô hình có khả năng xử lý ngữ cảnh dài.

Effective Using Long Contexts

As Context Increases, Models Miss Relevant Info

Cuối cùng, chúng ta sẽ thảo luận về cách sử dụng hiệu quả các mô hình retriever và reader khi làm việc với các ngữ cảnh dài. Gần đây, một nghiên cứu thú vị của Nelson Leo tại Stanford đã chỉ ra một hiện tượng được gọi là "lost in the middle". Nghiên cứu này cho thấy rằng nhiều mô hình, bao gồm cả các mô hình tiên tiến nhất, thường ít chú ý đến các thông tin nằm ở giữa các cửa sổ ngữ cảnh dài.

Cụ thể, nếu chúng ta có một câu trả lời và đặt nó ở vị trí đầu tiên trong tài liệu hoặc ở vị trí thứ 20 trong một ngữ cảnh được nối lại, mô hình có xu hướng chú ý nhiều hơn đến các thông tin ở đầu hoặc cuối, trong khi các thông tin ở giữa thường bị "mất" - điều này giải thích cho tên gọi "lost in the middle".

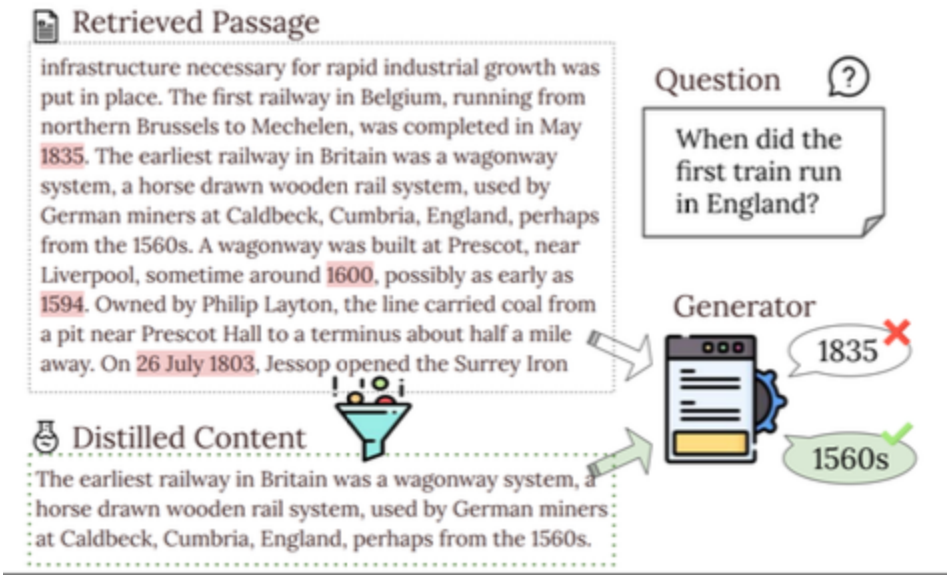


Vấn đề này có thể không quá nghiêm trọng khi chúng ta thực hiện việc truy xuất và đọc tài liệu, vì chúng ta có thể đặt các tài liệu có điểm số cao nhất ở đầu, điều này có thể hiệu quả hơn so với việc nối nhiều tài liệu có điểm số thấp lại với nhau. Tuy nhiên, nếu chúng ta muốn đọc một tài liệu dài và tổng hợp thông tin mà không thực hiện thêm một bước đánh giá nào, thì hiện tượng này có thể gây ra khó khăn. Hơn nữa, vì mô hình retriever không hoàn hảo, chúng ta mong muốn mô hình reader có thể làm tốt công việc với các đầu ra mà nó nhận được.

Ensuring the Use of Relevant Context

Việc sử dụng ngữ cảnh phù hợp là rất quan trọng để cải thiện hiệu suất của các mô hình. Có nhiều phương pháp để đảm bảo rằng ngữ cảnh được sử dụng là liên quan. Một trong những cách tiếp cận là sử dụng các bộ truy xuất (retrievers) tốt hơn, giúp cung cấp ngữ cảnh có liên quan hơn. Ngoài ra, chúng ta có thể áp dụng các kỹ thuật như reranking để chỉ bao gồm những ngữ cảnh có liên quan nhất.

Bên cạnh đó, cũng có những phương pháp giúp quyết định xem liệu có nên sử dụng hay bao gồm các đoạn văn (passages) hay không (Asai et al. 2021). Gần đây, chúng tôi đã đề xuất một phương pháp để lọc các đoạn văn đã truy xuất, chỉ giữ lại những nội dung phù hợp (Wang et al. 2023). Mô hình này được gọi là "filco", có chức năng lọc ngữ cảnh xuống còn những nội dung liên quan nhất mà chúng tôi cho là phù hợp. Điều này cho phép chúng tôi đạt được kết quả tốt hơn khi ngữ cảnh này được đưa vào cho bộ sinh (generator).



Resources

1. https://phontron.com/class/anlp2024/lectures/#retrieval-and-rag-feb-15
2. Recommended Reading: ACL 2023 RAG Tutorial (Asai et al. 2023)
3. Reference: Retrieval-based QA (Chen et al. 2017)
4. Reference: Dense Passage Retrieval (Karpukhin et al. 2020)
5. Reference: Introduction to Information Retrieval (Manning et al. 2009)
6. Software: Apache Lucene
7. Reference: DPR (Karpukhin et al. 2020)
8. Reference: Contriever (Izacard et al. 2022)
9. Software: FAISS
10. Software: ChromaDB
11. Reference: Cross-encoder Reranking (Nogueira et al. 2019)
12. Reference: Token-level Retrieval (Khattab and Zaharia 2020)
13. Reference: Hypothetical Document Embeddings (Gao et al. 2022)
14. Reference: End-to-end RAG Training (Lewis et al. 2020)
15. Reference: Toolformer (Schick et al. 2023)
16. Reference: FLARE (Jiang et al. 2023)
17. Reference: kNN-LM (Khandelwal et al. 2019)
18. Reference: Unlimiformer (Bertsch et al. 2023)
19. Reference: Training Transformers with Context (Voita et al. 2018)
20. Reference: Transformer-XL (Dai et al. 2019)
21. Reference: Mistral (Jiang et al. 2023)
22. Reference: Sparse Transformers (Child et al. 2019)
23. Reference: Compressive Transformer (Rae et al. 2019)
24. Reference: Linformer (Wang et al. 2020)
25. Reference: Nystromformer (Xiong et al. 2021)
26. Reference: Long Range Arena (Tay et al. 2020)

27. Reference: SCROLLS (Shaham et al. 2022)
28. Reference: Lost in the Middle (Liu et al. 2023)
29. Reference: Deciding Whether to Use Passages (Asai et al. 2021)
30. Reference: Learning to Filter Context (Wang et al. 2023)