Lecture 8: Fine-tuning and Instruction Tuning

Giới thiệu

Trong bài viết này, chúng ta sẽ tìm hiểu về fine-tuning và instruction tuning - những bước đầu tiên trong quy trình chuẩn bị các mô hình để sử dụng làm chatbot. Đây là nền tảng cho các ứng dụng trò chuyện AI phổ biến như ChatGPT hay Gemini. Quá trình này đóng vai trò quan trọng trong việc tối ưu hóa mô hình ngôn ngữ để đáp ứng các yêu cầu cụ thể và tương tác hiệu quả với người dùng.

Plethora of Tasks in NLP

Trong lĩnh vực Xử lý Ngôn ngữ Tự nhiên, chúng ta có nhiều loại nhiệm vụ khác nhau, mỗi loại yêu cầu các dạng dữ liệu khác nhau. Một số nhiệm vụ chỉ cần dữ liệu văn bản, như mô hình ngôn ngữ, trong khi các nhiệm vụ khác cần dữ liệu tự nhiên, không cần tạo thủ công, như dịch máy. Dữ liệu dịch máy có sẵn nhiều vì dịch thuật là hoạt động phổ biến, ngay cả khi không có các công cụ như ChatGPT hay Google Dịch.

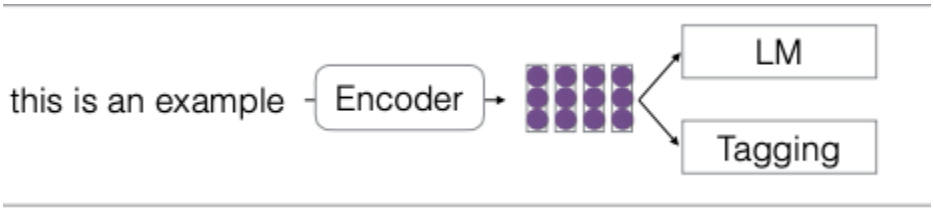
Ngoài ra, có những nhiệm vụ cần dữ liệu được gán nhãn thủ công, như trả lời câu hỏi hay nhận dạng thực thể tên (name entity recognition). Dữ liệu này không tự nhiên có sẵn, nên cần tạo ra bằng tay, điều này tốn kém và hạn chế về số lượng.

Một điểm thú vị là các mô hình ngôn ngữ lớn đã đạt được thành công đáng kể khi chỉ huấn luyện trên văn bản, nhờ vào lượng dữ liệu khổng lồ có sẵn trên internet. Ví dụ, một nghiên cứu tại hội nghị ACL đã tìm thấy hơn 30 triệu cặp dịch thuật từ dữ liệu web, không được tạo ra đặc biệt cho mục đích dịch thuật.

Tuy nhiên, việc chỉ dựa vào dữ liệu ngẫu nhiên từ internet có thể dẫn đến những kết quả không mong muốn. Ví dụ, khi sử dụng Chat GPT để dịch từ tiếng Anh sang tiếng Nhật, đôi khi nó dịch sang tiếng Nhật phiên âm thay vì ký tự tiếng Nhật, do học từ các nguồn dữ liệu như sách cụm từ dành cho người học tiếng Nhật. Điều này cho thấy rủi ro khi chỉ dựa vào mô hình ngôn ngữ mà không có dữ liệu huấn luyện được giám sát kỹ lưỡng.

Standard Multi-task Learning

Trong phần này, chúng ta sẽ thảo luận về một phương pháp học máy được gọi là học đa nhiệm (multitask learning). Học đa nhiệm là quá trình huấn luyện các mô hình để chúng có thể thực hiện tốt nhiều nhiệm vụ cùng lúc. Ví dụ, bạn có thể sử dụng học đa nhiệm để thực hiện mô hình hóa ngôn ngữ và gán nhãn (tagging) trên cùng một dữ liệu.

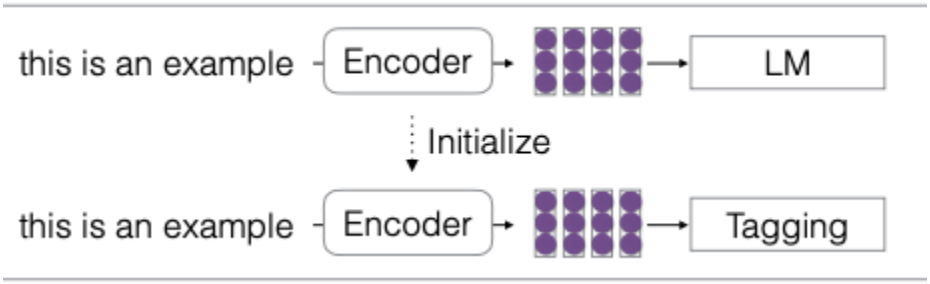


Điểm quan trọng trong học đa nhiệm là việc chia sẻ tham số giữa các mô hình được huấn luyện trên tất cả các nhiệm vụ. Nếu bạn chỉ huấn luyện một mô hình ngôn ngữ lớn, có thể bạn sẽ chia sẻ tất cả các tham số. Trong trường hợp bạn sử dụng mô hình như BERT, bạn có thể huấn luyện phần thân của mô hình trên nhiều nhiệm vụ, nhưng có các lớp phân loại riêng biệt cho từng nhiệm vụ.

Có nhiều cách để thực hiện điều này, nhưng ý tưởng cơ bản là cần có nhiều tham số được chia sẻ. Một cách đơn giản để thực hiện là huấn luyện mô hình bằng cách lấy mẫu một mini-batch cho một nhiệm vụ, sau đó lấy mẫu một mini-batch cho nhiệm vụ khác, và cứ thế luân phiên. Hoặc bạn có thể trộn tất cả dữ liệu lại với nhau, đặc biệt nếu tất cả đều là dữ liệu văn bản, thì bạn thậm chí không cần phải lo lắng về mini-batch.

Pre-train and Fine-tune

Trong lĩnh vực học máy, phương pháp "tiền huấn luyện và tinh chỉnh" (pre-train and fine-tune) là một chiến lược phổ biến. Quá trình này bắt đầu bằng việc huấn luyện mô hình trên một nhiệm vụ, chẳng hạn như mô hình hóa ngôn ngữ, sau đó chuyển sang huấn luyện trên một nhiệm vụ khác như gán nhãn (tagging). Có nhiều lý do để áp dụng phương pháp này thay vì học đa nhiệm tiêu chuẩn, nơi các nhiệm vụ được thực hiện đồng thời.



Một lý do chính là chi phí. Huấn luyện một mô hình ngôn ngữ lớn từ đầu là rất tốn kém và lãng phí tài nguyên. Do đó, chỉ một số ít người thực hiện tiền huấn luyện, sau đó nhiều người khác có thể tinh chỉnh mô hình trên một lượng dữ liệu nhỏ hơn. Điều này giúp tiết kiệm tài nguyên đáng kể.

Ngoài ra, nếu dữ liệu tiền huấn luyện lớn và không đồng nhất, như dữ liệu từ toàn bộ Internet chứa nhiều nội dung độc hại, việc tinh chỉnh có thể giúp mô hình an toàn hơn và loại bỏ các yếu tố không mong muốn.

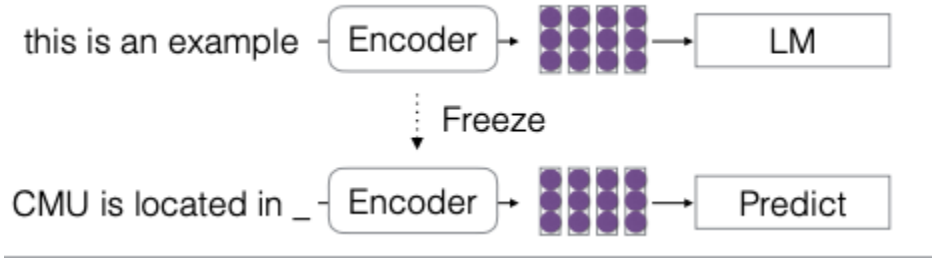
Tuy nhiên, có những lý do để cân nhắc việc không sử dụng phương pháp này. Một trong số đó là khả năng mất thông tin từ tập dữ liệu đa nhiệm. Nghiên cứu cho thấy rằng huấn luyện trên nhiều nhiệm vụ cùng lúc có thể mang lại kết quả tốt hơn cho nhiệm vụ cuối cùng. Điều này là do mô hình học được các biểu diễn hữu ích cho cả hai nhiệm vụ đồng thời, thay vì chỉ tập trung vào một nhiệm vụ cụ thể.

Một nghiên cứu khác từ Anthropic cũng chỉ ra rằng việc tích hợp khái niệm an toàn sớm trong quá trình huấn luyện có thể mang lại kết quả tốt hơn so với việc bắt đầu huấn luyện an toàn sau khi mô hình đã được huấn luyện một thời gian.

Mặc dù có những hạn chế, lợi ích của việc tiết kiệm tài nguyên và khả năng tinh chỉnh cho nhiều nhiệm vụ khác nhau khiến phương pháp tiền huấn luyện và tinh chỉnh vẫn là tiêu chuẩn hiện nay.

Prompting

Chúng ta đã tìm hiểu về prompting ở bài trước. Do đó, tôi sẽ nhắc lại một cách ngắn gọn để đảm bảo tính toàn diện của bài viết. Khi thực hiện "prompting", chúng ta sử dụng một bộ mã hóa (encoder) đã được huấn luyện trước đó, có thể là trên mô hình ngôn ngữ hoặc các tác vụ khác. Sau khi huấn luyện, bộ mã hóa này sẽ được "freeze", tức là không thay đổi nữa. Để xác định nhiệm vụ cụ thể mà chúng ta muốn thực hiện, chúng ta sẽ sử dụng một tiền tố (prefix) để chỉ định nhiệm vụ đó.



Instruction Tuning

Instruction tuning có thể được hiểu là sự kết hợp giữa fine-tuning và prompting. Quá trình này bắt đầu bằng việc pre-train mô hình, sau đó áp dụng các phương pháp huấn luyện cụ thể cho từng nhiệm vụ.

Thông thường, bạn sẽ có một prompt cho một nhiệm vụ này, một prompt cho nhiệm vụ khác, và cứ như vậy. Mục tiêu là huấn luyện mô hình để nó có thể hoàn thành tốt các yêu cầu từ những prompts này. Ví dụ, bạn sẽ thấy rằng mô hình sẽ được đào tạo để thực hiện các "good completions" cho các prompts đã được đưa ra.

Fine-tuning

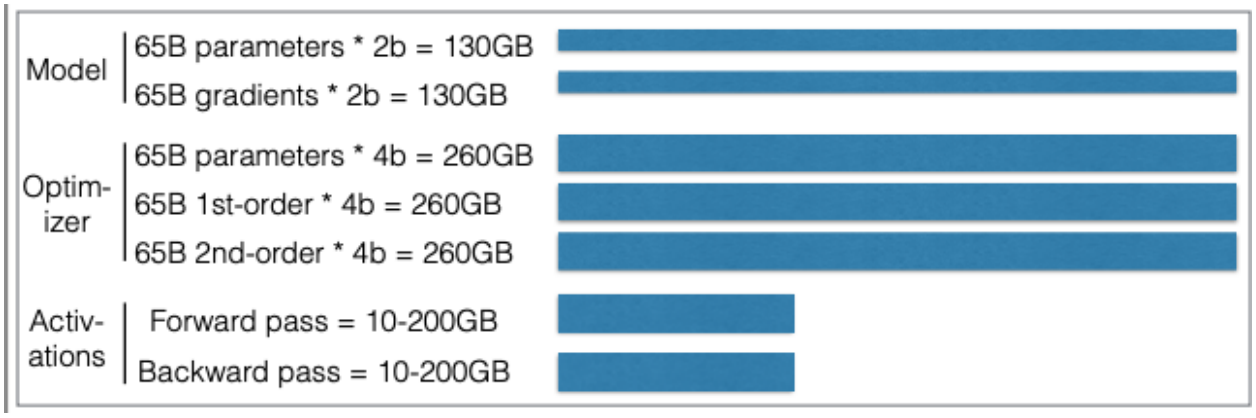
Full Fine-tuning

Trong NLP, việc tinh chỉnh (fine tuning) các mô hình ngôn ngữ lớn đang trở thành một xu hướng phổ biến. Tuy nhiên, quá trình này không hề đơn giản và đòi hỏi nhiều tài nguyên.

Tinh chỉnh đầy đủ là một khái niệm dễ hiểu nhưng khó thực hiện. Về cơ bản, chúng ta sẽ tiếp tục huấn luyện mô hình ngôn ngữ trên dữ liệu mà chúng ta muốn mô hình phù hợp, chẳng hạn

như "translation pairs" (cặp dịch thuật) hoặc "question answering pairs" (cặp hỏi đáp). Tuy nhiên, vấn đề nảy sinh từ phương pháp tối ưu hóa mà bạn sử dụng, vì nó có thể tiêu tốn nhiều bộ nhớ và đôi khi không ổn định.

Lấy ví dụ, việc huấn luyện một mô hình với 65 tỷ tham số, như phiên bản lớn nhất của LLaMA 1, với độ chính xác hỗn hợp 16 bit, thực tế yêu cầu nhiều bộ nhớ hơn bạn tưởng tượng. Để lưu trữ mô hình này, bạn cần khoảng 130 gigabytes bộ nhớ chỉ để chứa các tham số. Nếu bạn muốn lưu cả tham số và gradient của mô hình, bạn sẽ cần gấp đôi số lượng bộ nhớ, tức là 260 gigabytes.



Ngoài ra, bạn cũng cần bộ nhớ cho bộ tối ưu hóa, chẳng hạn như Adam, mà yêu cầu lưu trữ các giá trị trung bình và phương sai. Theo một nghiên cứu (Rajbhandari et al. 2019), các giá trị này cần được lưu trữ ở độ chính xác 32 bit để tránh các vấn đề về tràn số (overflow, underflow). Điều này có nghĩa là bạn cần một bản sao 32 bit của các tham số, cùng với bộ nhớ cho các phép toán lan truyền thuận và ngược, có thể tiêu tốn thêm 100 đến 200 gigabytes nữa. Tổng cộng, bạn có thể cần khoảng 1,000 - 1,400 gigabytes bộ nhớ GPU.

Mặc dù nghiên cứu này được công bố vào năm 2019, nhưng đã có nhiều tiến bộ trong việc tối ưu hóa mô hình. Trước đây, khi sử dụng fp16, bạn cần 32 bit để đảm bảo tính ổn định. Hiện nay, việc sử dụng BF16 (brain float 16) đã trở nên phổ biến, giúp giảm yêu cầu bộ nhớ xuống còn 2 byte thay vì 4 byte. Điều này cũng cho phép bạn không cần bản sao bổ sung của các tham số, nhưng vẫn yêu cầu khoảng 130 gigabytes bộ nhớ cho một mô hình 65 tỷ tham số.

An Aside: GPU Specs

Trong phần này, chúng ta sẽ tìm hiểu về các loại GPU và cách sử dụng chúng cho việc huấn luyện mô hình ngôn ngữ lớn.

GPU	Memory	Cost (2/2024)	(Cloud) Machines
T40 / K80	24GB	$150	Google Colab, AWS p2.*
V100	32GB	$2,500	Google Colab
A100	40GB or 80GB	$8,000/$16,000	Google Colab, AWS p3.*
H100	80GB	$44,000	AWS p4.*
6000 Ada, L40	48GB	$8000	N/A
Mac M*	Same as CPU	$2000	N/A

Dung lượng bộ nhớ của GPU:

- GPU tiêu chuẩn hiện nay có dung lượng bộ nhớ từ 24GB, 32GB, 40GB, 48GB đến 80GB.
- Tuy nhiên, không có GPU nào có bộ nhớ lên đến 130GB, do đó, không thể thực hiện huấn luyện hoặc tinh chỉnh mô hình lớn trên một GPU đơn lẻ.

Các tùy chọn phần cứng:

- Có nhiều tùy chọn phần cứng khác như GPU AMD, TPU của Google, và các thiết bị đặc biệt cho huấn luyện như Cerebras hay AWS.
- Đối với lớp học này, bạn có thể sử dụng các phần cứng tiêu chuẩn như GPU của Nvidia hoặc AMD. Nhiều người dùng Google Colab hoặc AWS cho các bài tập thực hành.

Ưu và nhược điểm của các loại phần cứng:

- GPU của Nvidia và AMD: Rất tốt cho việc huấn luyện mạng neural và các mô hình ngôn ngữ Transformer. Gần đây, AMD GPU đã được sử dụng nhiều hơn do giá cả hợp lý và hiệu suất cao.
- TPU: Tốt cho các tác vụ lớn và đồng bộ, nhưng không linh hoạt cho các biểu đồ tính toán động.

Chi phí huấn luyện mô hình:

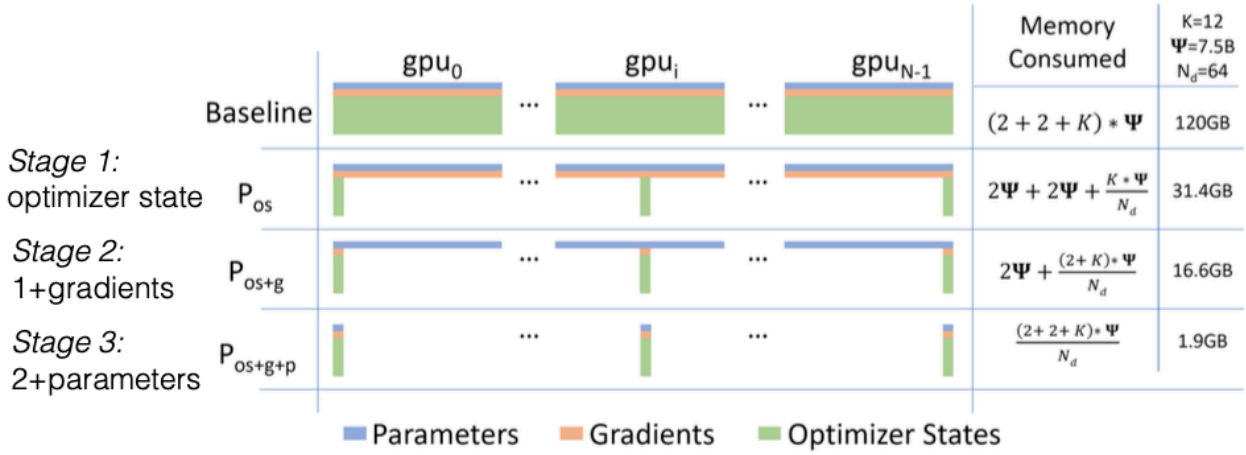
- Huấn luyện mô hình ngôn ngữ lớn từ đầu rất đắt đỏ. Meta đã thông báo họ sử dụng 350,000 H100 GPU, tương đương khoảng 10 đến 20 tỷ USD cho việc huấn luyện mô hình.
- Đó là lý do tại sao không phải ai cũng tiền huấn luyện mô hình từ đầu; thay vào đó, phần lớn sử dụng mô hình đã tiền huấn luyện và tinh chỉnh trên dữ liệu cụ thể của mình.

Multiple GPU Training

Việc huấn luyện các mô hình ngôn ngữ lớn thường vượt quá khả năng của một GPU đơn lẻ. Giải pháp phổ biến cho vấn đề này là sử dụng kỹ thuật huấn luyện đa GPU, trong đó

DeepSpeed Zero nổi bật như một công cụ được nhiều nhà nghiên cứu tin dùng cho việc tiền huấn luyện và tinh chỉnh mô hình.

DeepSpeed Zero hoạt động bằng cách phân chia quá trình tối ưu hóa trên nhiều thiết bị, với ba giai đoạn chính:



- Giai đoạn 1: Phân chia trạng thái của bộ tối ưu hóa (optimizer state) trên các GPU. Công thức tính bộ nhớ là 2 + 2 + K byte, với K là kích thước của trạng thái bộ tối ưu hóa. Ví dụ, một mô hình 7.5 tỷ tham số có thể giảm yêu cầu bộ nhớ từ 120GB xuống còn 31GB.
- Giai đoạn 2: Phân chia cả trạng thái bộ tối ưu hóa và gradient. Điều này làm giảm bộ nhớ hơn nữa nhưng có thể làm chậm quá trình do cần di chuyển dữ liệu giữa các thiết bị.
- Giai đoạn 3: Phân chia cả tham số mô hình, cho phép giảm bộ nhớ xuống mức cực thấp, nhưng có chi phí tính toán cao hơn do phải di chuyển dữ liệu nhiều.

Với DeepSpeed Zero, ta có thể huấn luyện một mô hình 7 tỷ tham số một cách dễ dàng trên một hoặc hai thiết bị, sử dụng chỉ 4 byte cho mỗi trạng thái bộ tối ưu hóa. Tuy nhiên, nên cân nhắc sử dụng giai đoạn 1 hoặc 2 để cân bằng giữa hiệu suất và yêu cầu bộ nhớ.

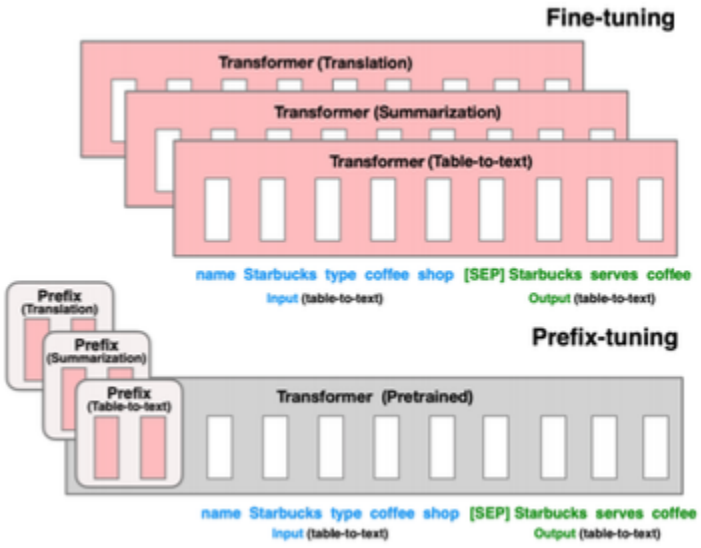
Nhiều thư viện phổ biến như Hugging Face Accelerate, GPT-NeoX, và TRL đã tích hợp DeepSpeed hoặc các kỹ thuật tương tự. Điều này cho phép các nhà nghiên cứu và kỹ sư dễ dàng áp dụng các kỹ thuật huấn luyện phân tán trong các dự án của họ, mở ra khả năng làm việc với các mô hình lớn hơn mà không cần đầu tư vào phần cứng đắt tiền như máy tính có 1000GB bộ nhớ.

Parameter-efficient Fine-tuning (PEFT)

Một lựa chọn khác mà bạn có thể sử dụng là không tinh chỉnh tất cả các tham số của mô hình mà chỉ tinh chỉnh một số tham số nhất định. Phương pháp này đang trở nên phổ biến trong thời gian gần đây, vì nó giúp cải thiện khả năng huấn luyện trên nhiều tập dữ liệu khác nhau mà không cần phải sử dụng các GPU mạnh mẽ hoặc nhiều thiết bị GPU.

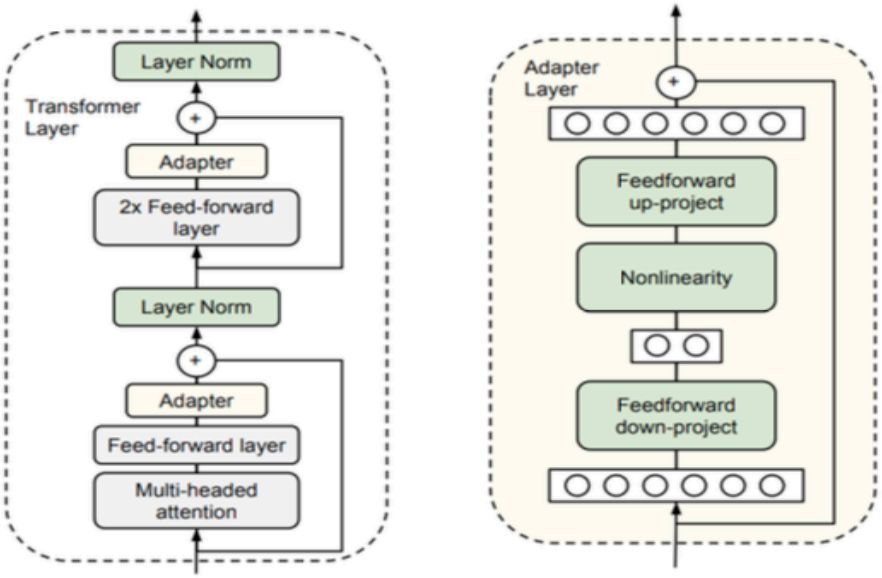
Reminder: Prefix Tuning (Li and Liang 2021)

Đầu tiên, chúng ta sẽ nhắc lại qua khái niệm "prefix tuning". Đây là một phương pháp mà tôi đã đề cập trong bài trước. Prefix tuning có thể được coi là một cầu nối giữa hai kỹ thuật: parameter efficient fine-tuning và prompting. Phương pháp này thực hiện việc tinh chỉnh một tiền tố (prefix) cho mỗi lớp trong mô hình.



Adapters (Houlsby et al. 2019)

Adapters là một phương pháp tinh chỉnh hiệu quả về tham số cho các mô hình Transformer. Kỹ thuật này thêm một lớp mới vào kiến trúc Transformer chuẩn, trong khi giữ nguyên các thành phần khác như lớp multi-head attention và feed-forward.



Cấu trúc của một adapter bao gồm:

1. Một phép chiếu xuống (down-projection) từ không gian biểu diễn lớn (ví dụ: 512 chiều) xuống một không gian nhỏ hơn nhiều (ví dụ: 16 chiều).
2. Một hàm phi tuyến.
3. Một phép chiếu lên (up-projection) trở lại không gian ban đầu.

Adapter được đặt trong một kết nối phần dư (residual connection), cho phép mô hình học các biểu diễn bổ sung mà không làm gián đoạn luồng thông tin chính.

Ưu điểm chính của adapters:

- Giảm số lượng tham số: Ví dụ, thay vì một ma trận 512x512 (2^18 tham số), ta chỉ cần 512x16 + 16x512 (2^14 tham số), giảm 16 lần.
- Hạn chế overfitting: Ít tham số hơn giúp mô hình tổng quát hóa tốt hơn, đặc biệt với bộ dữ liệu nhỏ.
- Tiết kiệm bộ nhớ: Khi tinh chỉnh, ta chỉ cần lưu trữ gradient cho các tham số của adapter.

Khi sử dụng adapters, ta chỉ cập nhật các tham số của adapter trong quá trình huấn luyện, giữ nguyên các phần khác của mô hình đã được tiền huấn luyện. Điều này cho phép tối ưu hóa quá trình lan truyền ngược:

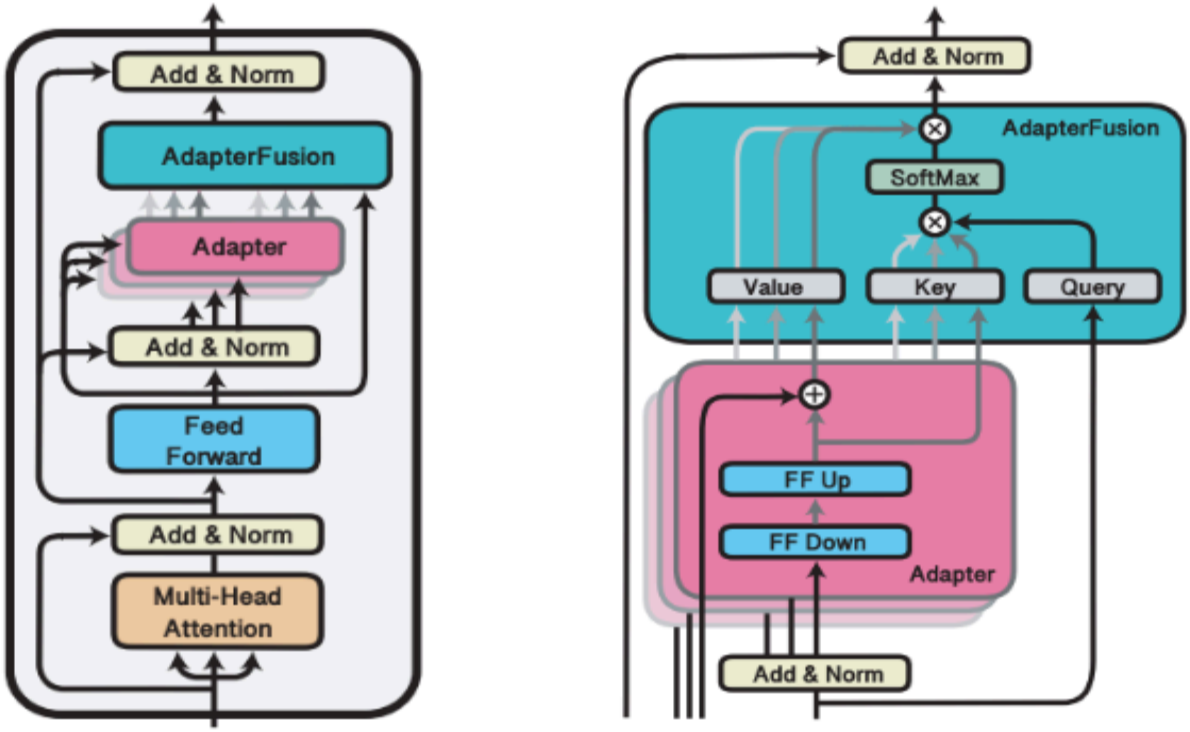
- Gradient chỉ cần được tính cho các tham số của adapter.
- Không cần tính gradient cho các phần không được cập nhật của mô hình.
- Có thể áp dụng kỹ thuật checkpointing để tiết kiệm bộ nhớ bằng cách chỉ lưu trữ một số trạng thái trung gian và tính toán lại khi cần thiết trong quá trình lan truyền ngược.
- Ví dụ, trong một đồ thị tính toán của block attention, gradient từ hàm mất mát sẽ truyền ngược lại qua các lớp để cập nhật các tham số của Adapters, trong khi các tham số khác không cần tính toán lại gradient.

Tóm lại, Adapters là một công cụ mạnh mẽ giúp mở rộng khả năng của mô hình Transformer bằng cách giảm số lượng tham số và bộ nhớ cần thiết. Kỹ thuật này giúp mô hình tổng quát hóa tốt hơn và tiết kiệm tài nguyên.

Adapter Fusion (Pfeiffer et al. 2020)

Trong phần này, chúng ta sẽ tìm hiểu về Adapter Fusion. Mặc dù nó không phải là phương pháp tiêu chuẩn, nhưng lại là một kỹ thuật thú vị.

Adapter Fusion là một kỹ thuật cho phép học các adapter cho nhiều tác vụ khác nhau và kết hợp chúng lại. Thay vì chỉ có một lớp adapter, chúng ta có thể có nhiều adapter và sau đó thực hiện "Adapter Fusion" ở phía trên.



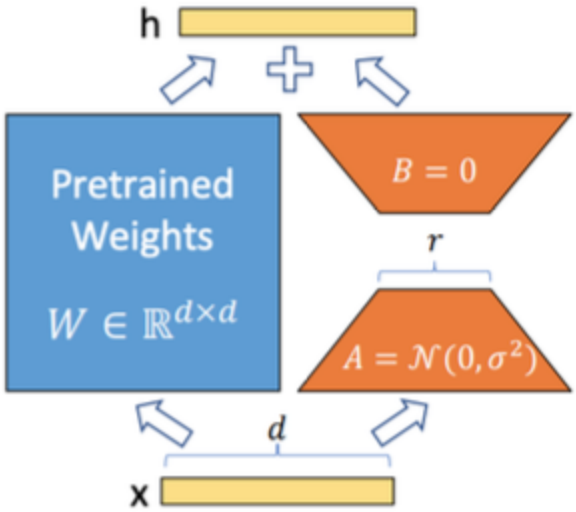
Ý tưởng cơ bản là sử dụng attention để quyết định adapter nào sẽ được sử dụng trong từng trường hợp cụ thể. Mỗi adapter được huấn luyện riêng biệt trên dữ liệu đặc thù của từng tác vụ. Ví dụ, bạn có thể huấn luyện một adapter cho tác vụ trả lời câu hỏi dựa trên dữ liệu từ nhiều bộ dữ liệu trả lời câu hỏi khác nhau, hoặc huấn luyện một adapter cho tác vụ dịch thuật dựa trên dữ liệu từ các bộ dữ liệu dịch thuật.

Khi sử dụng, chúng ta thực hiện attention để chọn adapter phù hợp và lấy giá trị từ adapter đó. Điều này cho phép chúng ta huấn luyện các module hữu ích cho từng tác vụ cụ thể và quyết định sử dụng module nào tại từng thời điểm. Ngoài ra, còn có các phiên bản đa ngôn ngữ, nơi bạn có thể huấn luyện adapter cho từng ngôn ngữ và từng tác vụ, sau đó kết hợp chúng lại.

Kỹ thuật này có thể được so sánh với mô hình "mixture of experts", một chủ đề mà chúng ta sẽ khám phá trong các bài viết sau.

LoRA (Hu et al. 2021)

Một phương pháp đang rất phổ biến hiện nay là LoRA (Low Rank Adaptation). Về mặt khái niệm, LoRA khá giống với adapters, nhưng có một điểm khác biệt quan trọng trong cách triển khai. Khác với adapters, LoRA không có lớp phi tuyến tính. Thay vào đó, nó sử dụng một ma trận giảm kích thước và một ma trận tăng kích thước để thực hiện phép biến đổi tuyến tính.



Trong hình minh họa từ bài báo về LoRA, các ma trận này được thể hiện như các đường tính toán riêng biệt. Tuy nhiên, thực tế là bạn có thể cộng chúng lại với nhau để có kết quả tương đương. Cụ thể, bạn cộng ma trận này nhân với ma trận kia vào trọng số trước khi huấn luyện, và điều này cho ra kết quả giống như khi bạn tính toán chúng riêng biệt rồi cộng lại sau đó.

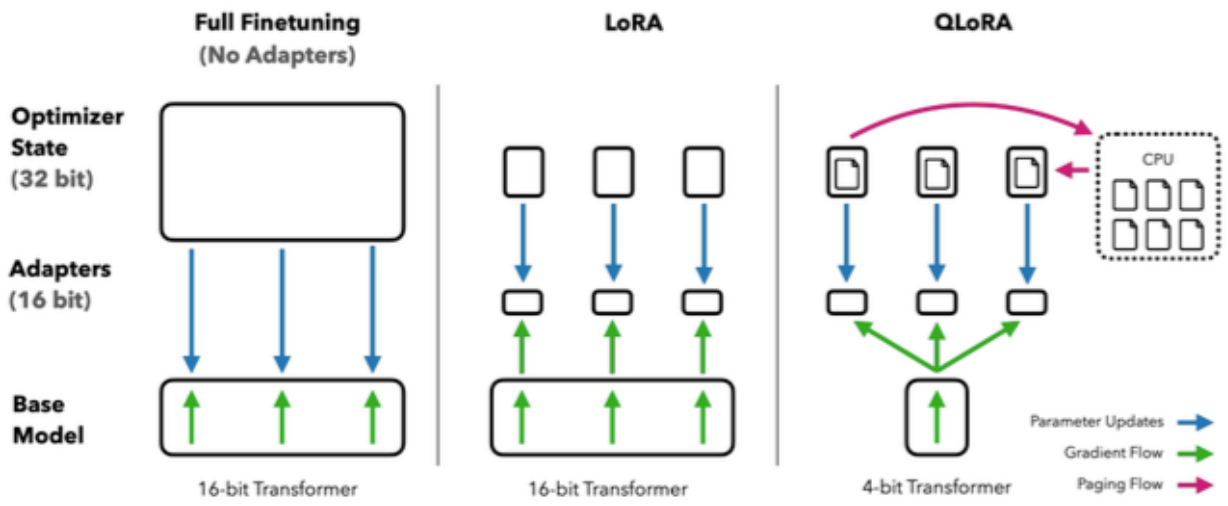
Vậy tại sao LoRA lại phổ biến? Lý do chính là sự tiện lợi. Sau khi hoàn tất quá trình huấn luyện với LoRA, bạn có thể cộng các ma trận đã học vào ma trận trọng số gốc, tạo ra một mô hình có cùng kích thước như ban đầu. Điều này không yêu cầu thêm bất kỳ thành phần nào khác hay thay đổi mã nguồn, chỉ cần cập nhật các tham số. Điều này khác với adapters, nơi bạn cần thêm các thành phần mô hình bổ sung và thay đổi mã nguồn để triển khai.

Tóm lại, LoRA không quá phức tạp mà khá đơn giản, nhưng lại rất quan trọng và tiện lợi trong việc triển khai mô hình.

Q-LoRA (Dettmers et al. 2023)

Một trong những xu hướng phổ biến hiện nay là việc kết hợp giữa quantization và parameter efficient tuning thông qua Quantization và LoRA. Quantization về cơ bản là những phương pháp giúp nén mô hình từ 16 bit xuống chỉ còn 4 bit. Điều này có nghĩa là mỗi tham số chỉ chiếm 4 bit, giúp mô hình trở nên rất gọn nhẹ.

Nếu quay lại với phép tính trước đó, một mô hình 16 bit như Llama cần khoảng 130 GB bộ nhớ. Tuy nhiên, với mô hình 4 bit, dung lượng cần thiết giảm xuống còn khoảng 32.5 GB, điều này cho phép nó chạy trên nhiều phần cứng, bao gồm cả A100, H100 và các GPU giá rẻ hơn. Thậm chí, bạn có thể chạy mô hình này trên máy Mac của mình nếu có đủ bộ nhớ.



Ý tưởng chính là nén mô hình để các tham số nhỏ hơn, từ đó tạo ra một lớp LoRA rất gọn nhẹ, không chiếm nhiều bộ nhớ. Điều này cho phép chúng ta huấn luyện mô hình trên phần cứng thông dụng như GPU 48 GB hoặc thậm chí trên MacBook. Ngoài ra, còn có cơ chế paging để chuyển dữ liệu giữa bộ nhớ CPU và GPU, giúp tăng hiệu suất.

Nếu bạn muốn huấn luyện một mô hình lớn trên phần cứng hạn chế, tôi khuyên bạn nên sử dụng phương pháp này. Đối với các mô hình không quá lớn như 65 GB, chỉ cần sử dụng LoRA là đủ.

Một câu hỏi thường gặp là liệu việc sử dụng độ chính xác thấp có gây ra vấn đề không. Thực tế, bạn cần lưu ý một chút, nhưng vì bạn không thực hiện tối ưu hóa trong độ chính xác thấp mà chỉ giữ mô hình gốc ở độ chính xác thấp, nên điều này vẫn có thể quản lý được. Bạn cũng có thể tham khảo bài báo, nơi có nhiều thí nghiệm chi tiết về vấn đề này.

BitFit (Ben Zaken et al. 2021)

Một phương pháp cuối cùng mà tôi muốn đề cập đến là BitFit. Đây là một kỹ thuật rất đơn giản, bạn chỉ cần huấn luyện các trọng số bias của mô hình cho bất kỳ mô hình nào có bias. Điều này rất dễ thực hiện vì bạn không cần phải thay đổi hay thêm bất kỳ mã nào. Bạn chỉ cần "freeze" tất cả các tham số ngoại trừ các bias. Từ góc độ đó, phương pháp này thực sự dễ dàng.

$$\mathbf{h}_2^\ell = \text{Dropout}\left(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell + \mathbf{b}_{m_1}^\ell\right) \quad (1)$$

$$\mathbf{h}_3^\ell = \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell \quad (2)$$

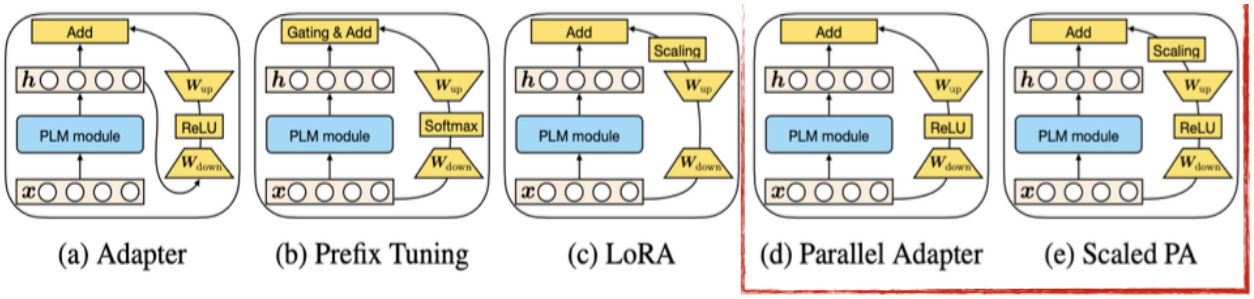
$$\mathbf{h}_4^\ell = \text{GELU}\left(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell + \mathbf{b}_{m_2}^\ell\right) \quad (3)$$

$$\mathbf{h}_5^\ell = \text{Dropout}\left(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell + \mathbf{b}_{m_3}^\ell\right) \quad (4)$$

$$\text{out}^\ell = \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell \quad (5)$$

A Unified View of PEFT (He et al. 2021)

Chúng ta đã thảo luận về các phương pháp tinh chỉnh tham số hiệu quả (PEFT). Trong bài báo nghiên cứu, các tác giả đã phân tích và chia nhỏ các phương pháp điều chỉnh này thành nhiều thành phần thiết kế khác nhau.



Có một số yếu tố cần xem xét khi nói về các phương pháp tinh chỉnh tham số hiệu quả. Đầu tiên là dạng hàm phi tuyến mà bạn sử dụng. Tiếp theo là vị trí bạn chèn mô hình, cách bạn điều chỉnh biểu diễn, và cuối cùng là hàm kết hợp để thêm biểu diễn đã điều chỉnh vào biểu diễn gốc.

Trong bài báo, các tác giả đã so sánh các phương pháp như adapters, LoRA, và prefix tuning. Các phương pháp này khá giống nhau, nhưng khác biệt ở chỗ lấy biểu diễn gốc từ đâu. Adapters thường lấy từ sau module mà bạn đang điều chỉnh, trong khi prefix tuning và LoRA lấy từ trước đó. Về hàm phi tuyến, có thể là ReLU, softmax hoặc không có gì. LoRA còn có một yếu tố không được đề cập trong bài báo nhưng có trong mã nguồn, đó là hệ số tỷ lệ, một siêu tham số cần lưu ý.

Bằng cách phân tích chi tiết, chúng ta có thể hiểu rõ hơn về từng phương pháp và cách chúng tương tác với nhau. Điều này cũng giúp chúng ta phát triển các biến thể mới hiệu quả hơn. Các tác gỉa đã đề xuất hai phương pháp mới là Parallel Adapter và Scaled Parallel Adapter, và chứng minh rằng chúng mang lại kết quả tốt hơn.

Which one to choose? (He et al. 2021)

Khi lựa chọn phương pháp tinh chỉnh tham số, câu hỏi đặt ra là nên chọn phương pháp nào. Đối với sự tiện lợi, LoRA và BitFit không thay đổi kiến trúc mô hình. Nếu bạn không quá quan tâm đến độ chính xác tuyệt đối, tôi khuyên bạn nên sử dụng các phương pháp này vì chúng dễ dàng nhất sau khi hoàn tất quá trình huấn luyện.

Trong nghiên cứu của bài báo, đối với các tác vụ đơn giản, sự khác biệt giữa các phương pháp không quá lớn. Ví dụ, với các tác vụ phân loại, BitFit có thể cạnh tranh với các phương pháp khác. Đối với các tác vụ phức tạp hơn và ngân sách tham số nhỏ, các tác giả nhận thấy rằng prefix tuning hoạt động khá tốt. Tuy nhiên, đây không phải là kết luận phổ quát mà chỉ là kết quả từ nghiên cứu của bài báo.

Đối với các tác vụ phức tạp hơn và ngân sách tham số lớn, việc sử dụng adapters hoặc kết hợp nhiều phương pháp có thể mang lại kết quả tốt hơn. Tất cả các chi tiết này đều có trong bài báo nếu bạn muốn tìm hiểu thêm.

NLP Tasks

Approaches to Model Construction

Trong phần này, tôi sẽ trình bày một số nhiệm vụ trong lĩnh vực Xử lý Ngôn ngữ Tự nhiên và lý do tại sao chúng ta cần thực hiện những nhiệm vụ này. Khi tiến hành tinh chỉnh mô hình, chúng ta cần tập trung vào từng nhiệm vụ cụ thể mà mình muốn giải quyết.

Cụ thể, tinh chỉnh cơ bản (basic fine-tuning) cho phép chúng ta xây dựng một mô hình có khả năng thực hiện tốt một nhiệm vụ đơn lẻ. Ngược lại, tinh chỉnh theo hướng dẫn (instruction tuning) giúp chúng ta phát triển một mô hình tổng quát, có khả năng thực hiện nhiều nhiệm vụ khác nhau. Trong phần tiếp theo, tôi sẽ đi sâu vào các nhiệm vụ NLP này.

Context-free Question Answering

Nhiệm vụ đầu tiên là trả lời câu hỏi không phụ thuộc ngữ cảnh, còn được gọi là trả lời câu hỏi mở (open book QA). Nhiệm vụ này đòi hỏi mô hình phải trả lời câu hỏi mà không cần dựa vào bất kỳ tài liệu cụ thể nào. Đây cũng chính là cách ChatGPT hoạt động khi trả lời câu hỏi của bạn mà không cần tra cứu thông tin trên web.

Một bộ dữ liệu phổ biến được sử dụng để đánh giá khả năng này là MMLU (Massive Multitask Language Understanding). Bộ dữ liệu này bao gồm các câu hỏi thuộc nhiều lĩnh vực khó, như luật chuyên nghiệp.

Ví dụ:
"Điều gì xảy ra khi một người bán hàng bỏ qua biển cảnh báo 'Cấm xâm phạm' và đi vào nhà của một người sống ẩn dật? Anh ta lái xe lên đường dẫn vào nhà và một thiết bị nổ phát nổ. Người bán hàng bị thương. Liệu người bán hàng có thể đòi bồi thường thiệt hại từ chủ nhà không?"

Mặc dù không phải là luật sư, câu trả lời cho câu hỏi này là: Có, nếu chủ nhà chịu trách nhiệm về thiết bị nổ dưới đường dẫn vào nhà. Điều này cho thấy bạn có thể đòi bồi thường nếu ai đó cố tình gây thương tích cho bạn khi bạn xâm phạm vào tài sản của họ.

Tập dữ liệu này có rất nhiều danh mục câu hỏi tương tự, giúp đánh giá khả năng của mô hình trong việc xử lý các tình huống phức tạp.

Contextual Question Answering

Nhiệm vụ tiếp theo là trả lời câu hỏi dựa trên ngữ cảnh. Đây là phương pháp trả lời câu hỏi dựa trên thông tin có sẵn trong một tài liệu cụ thể. Một ví dụ điển hình về tập dữ liệu được nhiều

người sử dụng là "Natural Questions", trong đó các câu hỏi được dựa trên các tài liệu từ Wikipedia.

Khi nói đến việc "dựa trên một tài liệu Wikipedia", điều này có nghĩa là người dùng sẽ nhận được tài liệu cụ thể mà họ cần trả lời câu hỏi. Hệ thống sẽ phải đọc và trả lời các câu hỏi liên quan đến tài liệu đó. Phương pháp này thường được gọi là "machine reading" (đọc máy), vì nó yêu cầu hệ thống không chỉ đọc mà còn hiểu và trả lời các câu hỏi dựa trên nội dung của tài liệu.

Ngoài ra, còn có một phương pháp khác được gọi là "retrieval-based question answering" (trả lời câu hỏi dựa trên truy xuất), trong đó hệ thống sẽ được cung cấp toàn bộ dữ liệu từ Wikipedia và yêu cầu cung cấp câu trả lời cho một câu hỏi cụ thể. Một biến thể của phương pháp này là "retrieval-augmented generation" (RAG), kết hợp giữa việc truy xuất thông tin và sinh ra câu trả lời.

Điều này thực sự rất quan trọng, và nhiều người mà tôi đã trò chuyện đều muốn xây dựng các hệ thống thực tế từ các mô hình ngôn ngữ hoặc hệ thống NLP theo hướng này.

Code Generation

Nhiệm vụ phổ biến tiếp theo là tạo mã (code generation). Cụ thể, đây là quá trình tự động sinh ra mã nguồn, chẳng hạn như Python hoặc SQL, từ các command bằng ngôn ngữ tự nhiên.

Một trong những bộ dữ liệu phổ biến nhất cho nhiệm vụ này là HumanEval (Chen et al. 2021). Bộ dữ liệu này chứa các câu hỏi liên quan đến thư viện chuẩn của Python, ví dụ như yêu cầu "trả về một danh sách với các phần tử được tăng thêm một". Bộ dữ liệu cung cấp văn bản mô tả và nhiều ví dụ về đầu vào và đầu ra mà bạn cần trả về dưới dạng chương trình.

```
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

HumanEval là một ví dụ đơn giản trong lĩnh vực này, vì nó không sử dụng bất kỳ thư viện bên ngoài nào và không yêu cầu ngữ cảnh phức tạp.

Summarization

Tiếp theo, tóm tắt tài liệu cũng là một trong những nhiệm vụ quan trọng. Có hai loại tóm tắt chính: tóm tắt tài liệu đơn và tóm tắt đa tài liệu. Tóm tắt tài liệu đơn giúp nén một tài liệu dài thành một phiên bản ngắn hơn, trong khi tóm tắt đa tài liệu tổng hợp thông tin từ nhiều tài liệu khác nhau về một chủ đề cụ thể thành một tóm tắt mạch lạc.

Hiện tại, tóm tắt tài liệu đơn bằng tiếng Anh hoạt động khá hiệu quả, mặc dù không hoàn hảo, nhưng gần như đạt được độ chính xác cao. Ngược lại, tóm tắt đa tài liệu vẫn còn nhiều thách thức. Tình huống này thường xảy ra khi có nhiều tài liệu liên quan đến một chủ đề, và mục tiêu là tạo ra một tóm tắt tổng quát về chủ đề đó.

Một ví dụ điển hình là tập dữ liệu WikiSum, nơi người dùng được cung cấp các liên kết đến các trang về một bài viết trên Wikipedia và cần tạo ra đoạn văn đầu tiên hoặc vài đoạn đầu của bài viết đó.

Chẳng hạn, khi tóm tắt thông tin từ nhiều bài viết không đồng nhất về Barack Obama, người dùng cần tổng hợp lại thành một nội dung mạch lạc. Ngoài ra, một số ứng dụng thú vị khác của tóm tắt đa tài liệu bao gồm việc tạo ra bảng khảo sát cho các bài báo khoa học hoặc tổng hợp báo cáo về những diễn biến trên thị trường chứng khoán trong một ngày.

Information Extraction

Một loại nhiệm vụ quan trọng khác trong lĩnh vực NLP là trích xuất thông tin. Mặc dù có nhiều ví dụ khác nhau, nhưng về cơ bản, tất cả đều xoay quanh việc trích xuất thông tin có cấu trúc từ văn bản. Các nhiệm vụ này bao gồm:

- Nhận dạng thực thể (Entity recognition): Xác định những từ nào là thực thể trong văn bản.
- Liên kết thực thể (Entity linking): Kết nối các thực thể với một cơ sở kiến thức.
- Đồng tham chiếu thực thể (Entity co-reference): Tìm ra những thực thể nào trong văn bản đầu vào ám chỉ cùng một đối tượng.
- Nhận dạng sự kiện (Event recognition): Xác định các sự kiện trong văn bản.
- Liên kết và đồng tham chiếu sự kiện (Event linking/co-reference): Tương tự như với thực thể, nhưng áp dụng cho sự kiện.

Một bộ dữ liệu ví dụ cho các nhiệm vụ này là OntoNotes. Mặc dù là bộ dữ liệu cũ, nhưng nó chứa tất cả các loại chú thích cần thiết cho các nhiệm vụ trên. Ngoài ra còn có nhiều bộ dữ liệu khác phục vụ cho mục đích này.

Ở mức độ tổng quát hơn, ta có thể xem xét khả năng của mô hình trong việc điền vào các cột trong một bảng tính Excel hoặc Google Sheets dựa trên thông tin có sẵn trên internet. Đây cũng là một loại nhiệm vụ quan trọng trong lĩnh vực trích xuất thông tin.

Translation

Dịch thuật là một loại nhiệm vụ quan trọng khác trong NLP. Nó là việc chuyển đổi nội dung từ một ngôn ngữ này sang một ngôn ngữ khác.

Về cơ bản, chất lượng dịch thuật được đánh giá dựa trên độ tương đồng với một bản dịch tham chiếu, sử dụng các phương pháp như BLEU score hoặc các chỉ số đánh giá dựa trên mạng nơ-ron.

Một ví dụ nổi bật trong lĩnh vực này là bộ dữ liệu FLORES. Bộ dữ liệu này bao gồm bản dịch của hàng trăm bài viết Wikipedia sang 101 ngôn ngữ khác nhau. Điều đáng chú ý về bộ dữ liệu FLORES là tiềm năng ứng dụng của nó: nếu một mô hình có thể dịch chính xác sang tất cả các ngôn ngữ này, nó sẽ góp phần đáng kể vào việc phổ biến thông tin trên toàn cầu và tạo ra sự bình đẳng hơn trong việc tiếp cận thông tin.

“General Purpose” Benchmarks

Ngoài các nhiệm vụ cụ thể, còn có những bộ đánh giá đa năng nhằm kiểm tra khả năng ngôn ngữ tổng quát của các mô hình ngôn ngữ. Một ví dụ điển hình là BIG-bench (Srivatsava et al. 2022), bao gồm nhiều bài kiểm tra đa dạng. Chẳng hạn như:

- Theo dõi các đối tượng bị xáo trộn: "Ellis, Bob và CLA là bạn, thỉnh thoảng trao đổi sách. Đầu học kỳ, mỗi người có một cuốn sách mới. Sau 4 lần trao đổi, Bob có cuốn sách nào?"
- Tính toán ngày tháng: "Hôm nay là Đêm Giáng sinh năm 1937. Ngày mai là ngày nào?"
- Suy luận logic: "Sherry nói thật. Vernal nói Sherry nói thật. Alexis nói Vernal nói dối. Michaela nói Alexis nói thật. Elanor nói Michaela nói thật. Elanor có nói thật không?"

Các mô hình ngôn ngữ hiện đại như Gemini thường được đánh giá dựa trên nhiều loại nhiệm vụ tương tự, ngoại trừ trích xuất thông tin.

Về vấn đề đảm bảo dữ liệu kiểm tra không xuất hiện trong dữ liệu huấn luyện, có một số phương pháp:

- Tạo dữ liệu mới hoàn toàn.
- Thực hiện các biến đổi nhỏ trên dữ liệu kiểm tra và quan sát sự thay đổi độ chính xác của mô hình.
- Thay đổi thứ tự câu trả lời hoặc các giá trị số trong bài toán.
- Nén và đặt mật khẩu cho tệp dữ liệu để tránh bị thu thập tự động.
- Giữ một phiên bản riêng tư của dữ liệu.

Về kiểm soát độ phức tạp của nhiệm vụ, hiện chưa có định nghĩa chuẩn. Tuy nhiên, có thể áp dụng một số phương pháp như:

- Kiểm soát độ dài văn bản.
- Đánh giá số bước suy luận cần thiết.
- Sử dụng các công cụ phân tích như BREAK benchmark để chia nhỏ câu hỏi thành các thao tác cơ bản.
- Biểu diễn bài toán dưới dạng đồ thị để đánh giá độ phức tạp.
- Phân đoạn dữ liệu và đánh giá độ chính xác trên từng phân đoạn.

Instruction Tuning

Basic Instruction Tuning (Wei et al. 2021, Sanh et al. 2021)

Trong lĩnh vực xử lý ngôn ngữ tự nhiên, instruction tuning là một khái niệm đơn giản nhưng rất quan trọng. Được đề xuất gần như đồng thời bởi các nhà nghiên cứu tại Google và Hugging Face, phương pháp này đã trở thành một phần không thể thiếu trong việc phát triển các mô hình ngôn ngữ hiện đại.

Instruction tuning hoạt động bằng cách huấn luyện mô hình trên nhiều nhiệm vụ khác nhau. Quá trình này bao gồm việc thêm vào một lời nhắc (prompt) vào đầu vào, sau đó huấn luyện mô hình để tạo ra đầu ra tương ứng. Điều này khác biệt so với việc huấn luyện mô hình ngôn ngữ cơ bản, bởi vì nó yêu cầu định dạng dữ liệu theo một cách cụ thể để giải quyết các nhiệm vụ. Về bản chất, đây là một dạng huấn luyện có giám sát, nhưng được thực hiện trên rất nhiều nhiệm vụ khác nhau.

Một điểm thú vị mà các nghiên cứu đã chỉ ra là khi áp dụng instruction tuning, mô hình không chỉ hoạt động tốt trên các nhiệm vụ đã được huấn luyện mà còn có khả năng giải quyết tốt các nhiệm vụ mới chưa từng gặp phải.

Learning to In-context Learn (Min et al. 2021)

Một phương pháp quan trọng khác là "học trong ngữ cảnh" (in-context learning). Thay vì chỉ cung cấp một lời nhắc (prompt) đơn giản, chúng ta có thể đưa ra các ví dụ huấn luyện trong ngữ cảnh. Điều này có nghĩa là chúng ta sẽ lấy một loạt các ví dụ huấn luyện và thêm chúng vào ngữ cảnh trước khi tiến hành huấn luyện mô hình.

Việc này mang lại lợi ích lớn, vì nó giúp mô hình cải thiện khả năng học trong ngữ cảnh. Nếu bạn muốn cung cấp các ví dụ huấn luyện, bạn có thể thực hiện theo cách này.

	Meta-training	Inference
Task	C *meta-training* tasks	An unseen *target* task
Data given	Training examples $\mathcal{T}_i = \{(x_j^i, y_j^i)\}_{j=1}^{N_i}$, $\forall i \in [1, C] \quad (N_i \gg k)$	Training examples $(x_1, y_1), \cdots, (x_k, y_k)$, Test input x
Objective	For each iteration, 1. Sample task $i \in [1, C]$ 2. Sample $k+1$ examples from $\mathcal{T}_i$: $(x_1, y_1), \cdots, (x_{k+1}, y_{k+1})$ 3. Maximize $P(y_{k+1} \vert x_1, y_1, \cdots, x_k, y_k, x_{k+1})$	$\text{argmax}_{c \in C} P(c \vert x_1, y_1, \cdots, x_k, y_k, x)$

Instruction Tuning Datasets

Trong nhiệm vụ tinh chỉnh mô hình, việc sử dụng các bộ dữ liệu đã được tổng hợp sẵn là rất quan trọng. Việc tự biên soạn các bộ dữ liệu có thể gây khó khăn và mất thời gian, mặc dù không quá phức tạp. Một tài liệu đáng chú ý mà tôi khuyên bạn nên tham khảo là bài báo về bộ dữ liệu FLAN (Longpre et al. 2023). Bài báo này cung cấp một cái nhìn tổng quan hữu ích với

bảng phân loại rõ ràng, bao gồm tên bộ dữ liệu, kích thước dữ liệu huấn luyện, loại prompt sử dụng (zero-shot hoặc few-shot), số lượng nhiệm vụ và các phương pháp chi tiết được áp dụng.

Một số bộ dữ liệu phổ biến mà nhiều người thường sử dụng bao gồm bộ dữ liệu FLAN, Natural Instructions và Self-Instruct. Những bộ dữ liệu này đã chứng minh được tính hiệu quả và được áp dụng rộng rãi trong nghiên cứu và phát triển mô hình.

Instruction Tuned Models

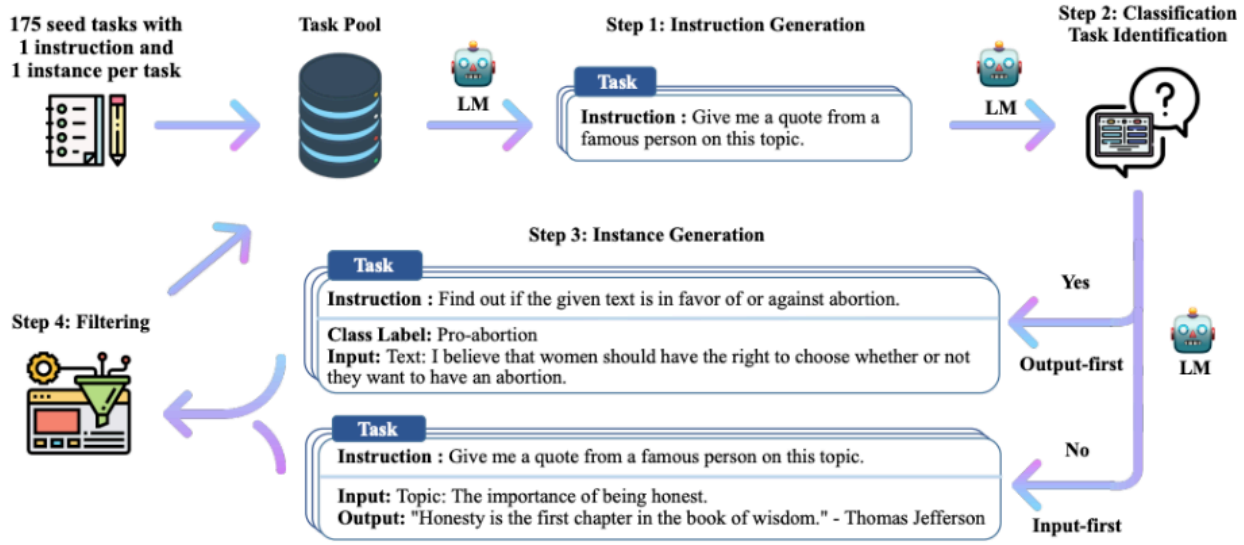
Trong phần này, tôi muốn giới thiệu về các mô hình được tinh chỉnh theo hướng dẫn. Một trong những mô hình đáng chú ý là FLAN-T5 (huggingface/google/flan-t5-xxl), nổi bật với kích thước đa dạng, từ mô hình nhỏ đến mô hình lên tới 11 tỷ tham số. Đây là mô hình encoder-decoder được đào tạo trên một lượng lớn dữ liệu, nó hoạt động tốt trong các tác vụ đơn giản theo kiểu đầu vào - đầu ra, chẳng hạn như tóm tắt văn bản.

Mô hình thứ hai là LLaMa-2 Chat (huggingface/meta-llama/Llama-2-70b-chat-hf), mô hình decoder-only, 70 tỉ tham số, được tinh chỉnh theo hướng dẫn và có khả năng tuân theo các chỉ dẫn một cách hiệu quả.

Cuối cùng, là Mixtral instruct (huggingface/mistralai/Mixtral-8x7B-Instruct-v0.1), một mô hình chỉ sử dụng decoder, 45 tỉ tham số. Mixtral là một mô hình hỗn hợp các chuyên gia (mixture of experts) với kích thước nhỏ nhưng mạnh mẽ, rất đáng để xem xét.

Dataset Generation

Phần cuối cùng, chúng ta sẽ thảo luận về việc tạo ra các bộ dữ liệu huấn luyện tự động. Một ví dụ điển hình cho phương pháp này là "self-instruct" (Wang et al. 2022).



Cách hoạt động của self-instruct là bắt đầu với một tập hợp các nhiệm vụ gốc, mỗi nhiệm vụ có một hướng dẫn và một ví dụ. Những nhiệm vụ này được đưa vào một "Task Pool" và từ đó, hệ

thống sẽ tạo ra các nhiệm vụ mới thông qua prompting. Sau đó, hệ thống xác định loại nhiệm vụ và tạo ra các đầu vào và đầu ra tương ứng. Quá trình này bao gồm việc lọc dữ liệu để loại bỏ các dữ liệu trùng lặp và những dữ liệu yêu cầu thông tin hình ảnh. Kết quả là, từ 175 ví dụ ban đầu, bộ dữ liệu được mở rộng để bao phủ nhiều nhiệm vụ khác nhau.

Một điểm thú vị là việc cải thiện mô hình đã được sử dụng để tạo ra các nhiệm vụ này. Ví dụ, GPT-3 đã được sử dụng để tạo ra các nhiệm vụ và sau đó được tinh chỉnh lại dựa trên chính các nhiệm vụ đó. Một số ví dụ khác bao gồm "Chain of Thought" (Mukherjee et al. 2023) và "Evol Instruct" (Xu et al. 2023). "Chain of Thought" tạo ra các giải thích cho quyết định của mô hình, giúp cải thiện khả năng suy luận. Trong khi đó, "Evol Instruct" bắt đầu với một tập hợp hướng dẫn gốc và sau đó làm phức tạp hóa chúng để cải thiện khả năng giải quyết vấn đề phức tạp của mô hình.

Một câu hỏi thường gặp là khi nào nên sử dụng Single Task Fine-Tuning so với Instruction Tuning. Nếu bạn có một định nghĩa nhiệm vụ rõ ràng và nhiều dữ liệu huấn luyện, việc Full Fine-Tuning có thể mang lại độ chính xác cao hơn, đặc biệt là với các mô hình nhỏ. Ví dụ, một mô hình chuyển đổi văn bản thành SQL dựa trên Llama 7B đã đạt được kết quả ấn tượng khi chỉ huấn luyện trên dữ liệu chuyển đổi văn bản thành SQL. Tương tự, trong các nhiệm vụ dịch thuật, một mô hình nhỏ với 3.3 tỷ tham số đã cạnh tranh tốt với GPT-4 trên các ngôn ngữ có nhiều dữ liệu huấn luyện.

Cuối cùng, nếu bạn có một định dạng cố định và luôn muốn đầu ra theo định dạng đó, việc Instruction Tuning có thể là lựa chọn phù hợp. Điều này cho thấy rằng, dù mô hình nhỏ hơn nhưng nếu được huấn luyện đặc biệt cho một nhiệm vụ cụ thể, nó vẫn có thể đạt được kết quả tốt.

Resources

1. https://phontron.com/class/anlp2024/lectures/#fine-tuning-and-instruction-tuning-feb-8
2. ZeRo (Rajbhandari et al. 2019)
3. Adapters (Houlsby et al. 2019)
4. Adapter Fusion (Pfeiffer et al. 2020)
5. LoRa (Hu et al. 2021)
6. QLoRA (Dettmers et al. 2023)
7. BitFit (Ben Zaken et al. 2023)
8. MMLU (Hendrycks et al. 2020)
9. Natural Questions (Kwiatkowski et al. 2019)
10. HumanEval (Chen et al. 2021)
11. WikiSum (Liu et al. 2018)
12. FLORES (Goyal et al. 2021)
13. OntoNotes (Weischedel et al. 2013)
14. BIGBench (Srivastava et al. 2022)
15. Instruction Tuning (1) (Wei et al. 2021)
16. Instruction Tuning (2) (Sanh et al. 2021)

17. Learning to In-context Learn (Min et al. 2021)
18. Self-instruct (Wang et al. 2022)
19. ORCA (Mukherjee et al. 2023)
20. Evol-Instruct (Xu et al. 2023)