Lecture 13: Debugging and Interpretation

Resources

Giới thiệu

Trong bài viết này, tôi sẽ đi sâu vào chủ đề debugging (gỡ lỗi) và cách hiểu sâu về các mô hình NLP. Chúng ta sẽ tìm hiểu cách phát hiện các tình huống như khi cách triển khai của bạn đều sai, khi các giả định nền tảng không chính xác, hoặc khi mô hình thất bại trên một số phân khúc dữ liệu cụ thể. Bài trình bày sẽ bao quát nhiều vấn đề có thể phát sinh trong quá trình thực nghiệm với mô hình NLP.

A Typical Situation

Trong một tình huống điển hình, bạn đã triển khai một hệ thống xử lý ngôn ngữ tự nhiên dựa trên mạng nơ-ron. Bạn đã xem xét mã nguồn và thấy nó có vẻ ổn, nhưng hệ thống lại có độ chính xác thấp hoặc mắc phải những lỗi khó hiểu. Bạn muốn khắc phục những vấn đề này hoặc cải thiện độ chính xác của hệ thống. Vậy bạn nên làm gì?

Three Model Understanding Dimensions

Chúng ta sẽ khám phá ba khía cạnh quan trọng để hiểu rõ hơn về mô hình và hành vi của nó:

- Gỡ lỗi trong quá trình triển khai: xác định các vấn đề phát sinh khi bạn thực hiện một điều gì đó.
- Đánh giá có thể thực hiện (actionable evaluation): nhận diện các trường hợp lỗi điển hình và cách bạn có thể khắc phục chúng.
- Diễn giải dự đoán: hiểu rõ những gì đang diễn ra bên trong mô hình, điều này có thể giúp bạn có cái nhìn sâu sắc hơn về các trường hợp cụ thể.

Có nhiều lý do để bạn muốn thực hiện những điều này, không chỉ để cải thiện mô hình mà còn để đảm bảo rằng mô hình của bạn không vi phạm pháp luật, chẳng hạn như phân biệt đối xử dựa trên các thuộc tính được bảo vệ.

Debugging

In Neural Net Models, Debugging is Paramount!

Trong các mô hình mạng nơ-ron, việc gỡ lỗi là rất quan trọng vì chúng thường không rõ ràng và khó dự đoán. Những sai sót nhỏ có thể gây ra vấn đề lớn cho đầu ra của bạn. Một điểm cần lưu ý là mọi thứ đều là hyperparameters, bao gồm kích thước mạng, biến thể mô hình, kích thước batch, chiến lược, bộ tối ưu hóa và tốc độ học.

Khác với các phương pháp học máy truyền thống như "logistic regression" hay "support vector machines", tối ưu hóa ngẫu nhiên không đảm bảo hội tụ. Độ mất mát có thể giảm xuống rồi

tăng lên mà không có gì sai với quá trình huấn luyện, hoặc có vấn đề nghiêm trọng. Đây là một thách thức khác mà bạn cần phải đối mặt.

Possible Causes

Trong quá trình triển khai mô hình, có thể gặp phải nhiều vấn đề khác nhau. Để giải quyết, trước tiên cần xác định nguyên nhân gây ra vấn đề. Dưới đây là một số nguyên nhân phổ biến và cách khắc phục:

1. Vấn đề trong quá trình huấn luyện:

- Mô hình thiếu khả năng: Mô hình không đủ khả năng để mô phỏng hiện tượng cần thiết.
- Thuật toán huấn luyện kém: Có thể do thuật toán không hiệu quả.
- Lỗi trong mã nguồn: Lỗi xảy ra trong quá trình huấn luyện.

2. Vấn đề trong quá trình kiểm tra:

- Sự không nhất quán giữa huấn luyện và kiểm tra: Có sự khác biệt giữa những gì thực hiện trong quá trình huấn luyện và kiểm tra.
- Thất bại của thuật toán tìm kiếm: Thuật toán tìm kiếm không hoạt động như mong đợi.

3. Overfitting: Mô hình hoạt động tốt trên tập huấn luyện nhưng kém trên tập kiểm tra.

4. Không khớp giữa hàm tối ưu hóa và đánh giá: Hàm tối ưu hóa trong quá trình huấn luyện không phù hợp với tiêu chí đánh giá trong quá trình kiểm tra.

Lời khuyên tốt nhất là không nên cố gắng giải quyết tất cả các vấn đề cùng một lúc. Hãy bắt đầu từ những nguyên nhân dễ chẩn đoán nhất và dần dần mới tới các nguyên nhân phức tạp hơn.

Debugging at Training Time

Identifying Training Time Problems

Khi debug hệ thống trong quá trình huấn luyện, điều quan trọng nhất là theo dõi loss function trên tập huấn luyện. Thay vì tập trung vào error hay accuracy - vốn khó để tối ưu hóa trực tiếp, chúng ta nên quan sát likelihood hoặc loss function trên training set trước khi xem xét accuracy trên test set. Có một số điểm cần chú ý khi đánh giá loss function:

1. Giá trị tối ưu của loss:

- Trong hầu hết trường hợp, loss function tốt nhất sẽ tiến về 0 (ví dụ như với log likelihood)
- Loss function nên có xu hướng giảm dần và hội tụ về một điểm tốt

2. Các dấu hiệu bất thường:

- Loss function có giá trị cao bất thường trên training set thường là dấu hiệu của vấn đề nghiêm trọng
- Nếu thấy loss function cao trên dev set hoặc test set, có thể là do overfitting
- Nếu loss function không giảm về gần 0 sau nhiều epochs, ngay cả với tập training set nhỏ, đây có thể là dấu hiệu của lỗi trong implementation

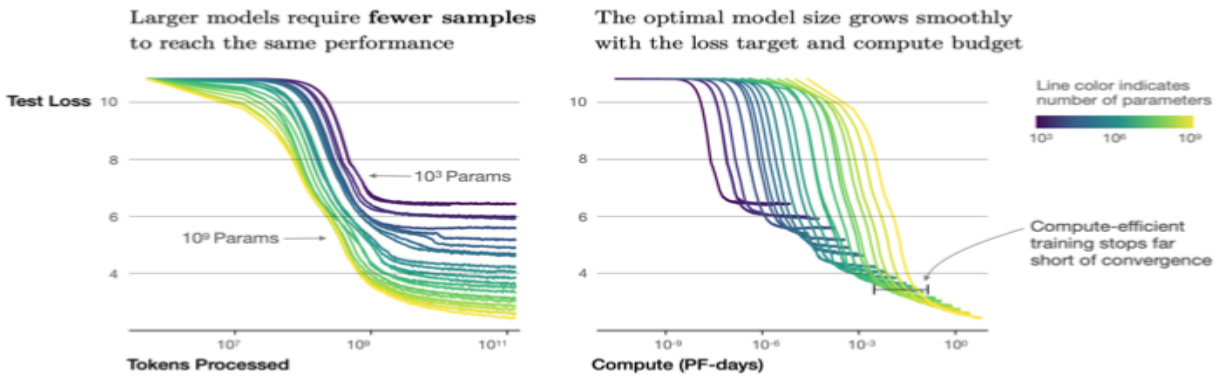
Đây là những kiểm tra đầu tiên và cơ bản nhất khi huấn luyện một mô hình. Việc theo dõi và phân tích loss function sẽ giúp phát hiện sớm các vấn đề trong quá trình huấn luyện.

Is My Model Too Tweak?

Trong lĩnh vực học máy, có nhiều lý do khiến mô hình của bạn không đạt hiệu quả như mong đợi. Một trong những lý do chính là mô hình của bạn có thể quá nhỏ. Thông thường, các mô hình lớn hơn sẽ hoạt động tốt hơn, đặc biệt khi bạn sử dụng mô hình đã được huấn luyện trước. Ví dụ, trong bài báo T5, họ đã mở rộng mô hình T5 từ một mô hình tương đối nhỏ lên một mô hình lớn với 11 tỷ tham số. Mặc dù hiện nay mô hình này có thể được coi là vừa phải hoặc thậm chí nhỏ, nhưng nó vẫn cho thấy hiệu suất tăng lên khi quy mô tăng.

Model	GLUE Average	CoLA Matthew's	SST-2 Accuracy	MRPC F1	MRPC Accuracy	STS-B Pearson	STS-B Spearman
Previous best	89.4[a]	69.2[b]	97.1[a]	**93.6**[b]	**91.5**[b]	92.7[b]	92.3[b]
T5-Small	77.4	41.0	91.8	89.7	86.6	85.6	85.0
T5-Base	82.7	51.1	95.2	90.7	87.5	89.4	88.6
T5-Large	86.4	61.2	96.3	92.4	89.9	89.9	89.2
T5-3B	88.5	67.1	97.4	92.5	90.0	90.6	89.8
T5-11B	**90.3**	**71.6**	**97.5**	92.8	90.4	**93.1**	**92.8**

Một hiện tượng thú vị là các mô hình lớn có thể học nhanh hơn hoặc ít bước hơn so với các mô hình nhỏ. Một nghiên cứu nổi bật về vấn đề này là bài báo về "neural scaling", trong đó cho thấy rằng các mô hình lớn (màu vàng) hội tụ nhanh hơn so với các mô hình nhỏ (màu tím) khi xử lý số lượng token. Điều này có thể gây ngạc nhiên, vì nhiều người có thể nghĩ rằng mô hình lớn sẽ khó khăn hơn trong việc khớp dữ liệu do sự phức tạp và nhiễu.



Một lý do cho hiện tượng này là mô hình lớn có nhiều tham số hơn, cho phép nó hội tụ theo nhiều chiều khác nhau. Điều này liên quan đến "lottery ticket hypothesis", một giả thuyết nổi tiếng cho rằng chỉ một số ít tham số trong mô hình lớn là thực sự hữu ích. Khi bạn bắt đầu với một mô hình lớn hơn, khả năng cao là nó sẽ có nhiều tập hợp tham số hữu ích hơn.

Ngoài ra, ngay cả khi bạn mở rộng tính toán, các mô hình lớn vẫn vượt trội hơn về hiệu quả trong việc mô hình hóa dữ liệu. Các mô hình thường học tốt trong một thời gian, nhưng sau đó sẽ đạt đến giới hạn của chúng và bắt đầu học chậm lại. Khi đến giai đoạn này, các mô hình lớn sẽ hoạt động tốt hơn.

Cuối cùng, nếu bạn thấy mô hình của mình đang dừng lại hoặc không cải thiện, có thể mô hình của bạn quá nhỏ và bạn cần thử nghiệm với một mô hình lớn hơn.

Trouble with Optimization

Một vấn đề khác thường gặp là tối ưu hóa. Để giải quyết vấn đề này, bạn nên kiểm tra bộ tối ưu của mình. Hiện nay, nhiều người sử dụng các biến thể của Adam như "Adam" hoặc "AdamW". Hãy đảm bảo rằng bạn đang sử dụng một learning rate phù hợp với kích thước mô hình của bạn. Cách tốt nhất để xác định learning rate là tham khảo các tài liệu nghiên cứu trước đó để xem họ đã sử dụng gì.

Nếu bạn đang huấn luyện mô hình từ đầu, việc khởi tạo mô hình là rất quan trọng. Thông thường, bạn nên khởi tạo với một số loại noise ngẫu nhiên đồng nhất (uniform random noise). Bạn có thể chọn noise này một cách thông minh dựa trên kích thước dữ liệu mà bạn có.

Ngoài ra, hãy chú ý đến kích thước mini-batch. Nếu bạn sử dụng các mini-batch quá nhỏ, có thể bạn sẽ gặp phải quá nhiều noise trong quá trình huấn luyện, dẫn đến việc mô hình không hội tụ. Đây là những yếu tố quan trọng mà bạn cần cân nhắc khi tối ưu hóa mô hình của mình.

Debugging at Test Time

Training/ Test Disconnects

Tiếp theo là việc gỡ lỗi trong giai đoạn kiểm tra là rất quan trọng, đặc biệt khi bạn tự triển khai các thuật toán thay vì sử dụng các thư viện tiêu chuẩn như Hugging Face. Khi bạn thực hiện các chức năng tính toán hàm mất mát và dự đoán (prediction) trong các hàm khác nhau, điều này có thể dẫn đến sự không nhất quán giữa quá trình huấn luyện và kiểm tra.

Một trong những nguyên nhân chính gây ra sự ngắt kết nối này là do mã nguồn bị trùng lặp (duplicated code), có thể dẫn đến việc bạn triển khai một chức năng theo cách này ở một nơi và theo cách khác ở nơi khác. Điều này đặc biệt đúng với các mô hình dự đoán có cấu trúc, nơi bạn không chỉ thực hiện một dự đoán đơn lẻ mà là nhiều dự đoán liên tiếp.

Ngoài ra, khi tự tính toán mất mát, bạn cần chú ý rằng quá trình này thường được thực hiện theo mini-batch, trong khi việc sinh dữ liệu có thể không được thực hiện theo cách tương tự. Trong các phiên bản tối ưu hóa cao của quá trình suy diễn (inference), bạn có thể sử dụng batching động (Dynamic batching), điều này có thể làm cho quá trình trở nên phức tạp và dễ dẫn đến sai sót.

Debugging Minibatching

Trong quá trình tính toán hàm mất mát cho các mini batch, việc đảm bảo không mắc phải sai sót là rất quan trọng. Một phương pháp đơn giản để kiểm tra tính chính xác của các phép tính này là so sánh kết quả giữa việc tính toán hàm mất mát cho một batch lớn và tính toán cho từng câu riêng lẻ.

Cụ thể, bạn có thể tính toán hàm mất mát với kích thước batch lớn, chẳng hạn như 32, sau đó tính hàm mất mát cho từng câu riêng lẻ và cộng tổng lại. Kết quả từ hai phương pháp này nên giống nhau, điều này giúp bạn phát hiện các vấn đề liên quan đến padding, masking hoặc các yếu tố khác.

Điều này đặc biệt quan trọng khi bạn không chỉ sử dụng các mô hình có sẵn mà có thể đang làm việc với một mô hình có cấu trúc hơi khác thường, như "hierarchical encoding". Trong trường hợp này, bạn cần phải cẩn thận hơn.

Ngoài ra, bạn cũng có thể tạo các unit test để kiểm tra tính chính xác này. Trong mã máy học, đặc biệt là với các mô hình dựa trên mạng nơ-ron, việc viết các unit test không thường xuyên được thực hiện do có nhiều yếu tố ngẫu nhiên. Tuy nhiên, việc kiểm tra này là một trong những điều dễ dàng thực hiện và giúp bạn tránh được những sai sót không đáng có.

Debugging Structured Generation

Trong quá trình phát triển các thuật toán sinh, việc đảm bảo rằng mã giải mã (decoding code) cho ra cùng một điểm số (score) như khi tính toán hàm mất mát là rất quan trọng. Một cách đơn giản để thực hiện điều này là gọi hàm giải mã để tạo ra một đầu ra.

Khi thực hiện tìm kiếm hoặc lấy mẫu, bạn sẽ tính toán các logits hoặc xác suất log cho mỗi bước mà bạn lấy mẫu. Bạn cần theo dõi các giá trị này trong suốt quá trình lấy mẫu. Sau khi hoàn thành, hãy gọi hàm mất mát trên đầu ra đã tạo ra và tính toán giá trị mất mát theo hàm này.

Điểm số từ hai quá trình này nên giống nhau. Cụ thể, bạn sẽ thực hiện các bước sau:

- Gọi hàm sinh (generate) để nhận đầu ra và điểm số.
- Tính toán hàm mất mát (loss) trên đầu ra và nhận được điểm số thứ hai.
- So sánh hai điểm số này.

Theo kinh nghiệm của tôi, phương pháp này đã giúp tôi phát hiện hầu hết các lỗi trong quá trình làm việc với các mô hình phức tạp liên quan đến sinh và các vấn đề khác. Đây là một khu vực

phổ biến để phát sinh lỗi, ngay cả khi bạn đã quen thuộc với các mô hình. Do đó, tôi rất khuyến khích bạn áp dụng phương pháp này.

Beam Search - Debugging Search

Beam search, như đã biết từ bài giảng trước, không chỉ chọn một từ có xác suất cao nhất ở bước tiếp theo mà duy trì nhiều đường đi khác nhau. Một cách để cải thiện điều này là khi bạn làm cho tìm kiếm tốt hơn, điểm số của mô hình cũng nên được cải thiện. Điều này có nghĩa là log likelihood của đầu ra nên tăng lên gần như mọi lúc. Bạn có thể thử nghiệm với các kích thước beam khác nhau để đảm bảo rằng bạn đạt được điểm số mô hình tổng thể tốt hơn ở cuối quá trình.

Ngoài ra, bạn có thể tạo một unit test để kiểm tra điều này. Mặc dù không nhiều người sẽ phải triển khai lại beam search, nhưng nếu bạn đang làm việc với các thuật toán tìm kiếm, đây là một điều quan trọng cần biết.

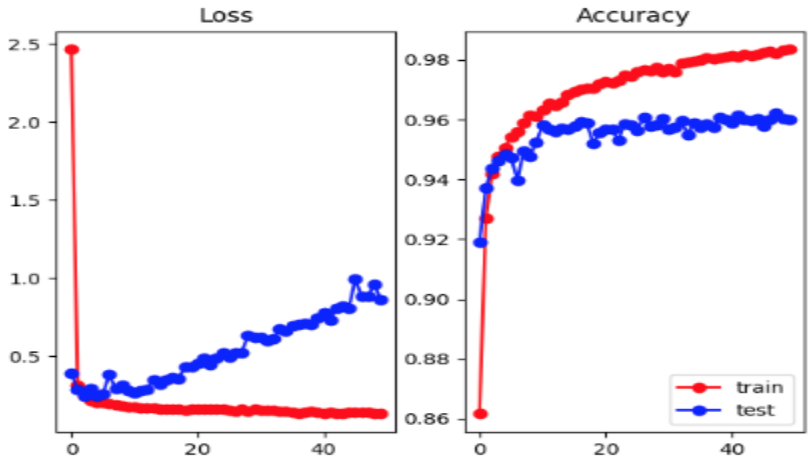
Mismatch b/t Optimized Function and Evaluation Metrics

Loss Function, Evaluation Metrics

Một vấn đề quan trọng mà nhiều người thường ít chú ý đến, nhưng nó có thể ảnh hưởng đến việc xây dựng hệ thống. Đó là sự không khớp giữa hàm mục tiêu mà bạn tối ưu hóa trong quá trình huấn luyện và chỉ số đánh giá mà bạn sử dụng để đánh giá mô hình.

Như đã đề cập trong bài viết về học tăng cường, việc tối ưu hóa cho maximum likelihood trong huấn luyện là rất phổ biến. Tuy nhiên, có nhiều vấn đề liên quan đến điều này, chẳng hạn như việc nhạy cảm với các sai lầm mà thuật toán sinh ra. Mặc dù likelihood của mô hình có thể cải thiện, độ chính xác (accuracy) lại có thể giảm.

Một ví dụ đơn giản về phân loại hình ảnh trên tập dữ liệu MNIST cho thấy điều này. Ở bên trái, chúng ta có loss trên tập huấn luyện và tập kiểm tra, và ở bên phải là độ chính xác trên tập huấn luyện và tập kiểm tra.



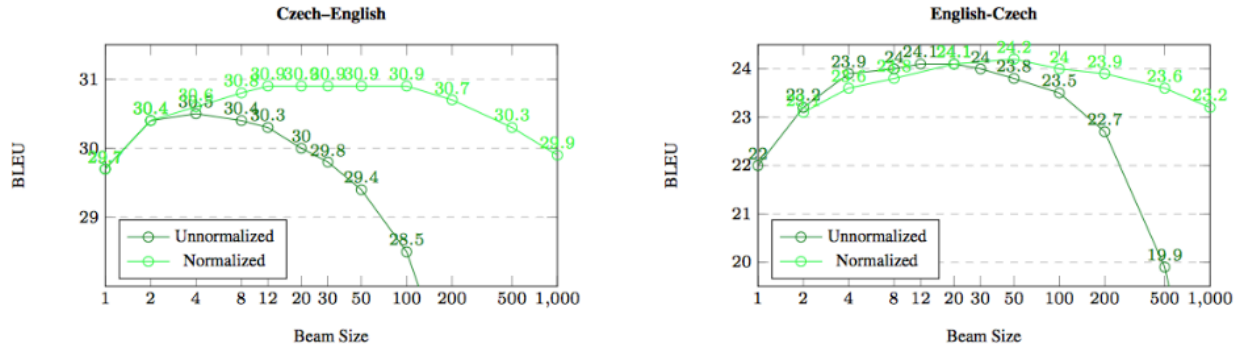
Vấn đề ở đây là loss mà bạn đang tính toán là xác suất của câu trả lời đúng, trong khi độ chính xác là số lần mà mô hình đưa ra câu trả lời đúng. Khi bạn huấn luyện một mô hình để nó tự tin hơn, nó có thể trở nên tốt hơn trong việc đưa ra câu trả lời đúng, nhưng đồng thời cũng có thể trở nên tự tin hơn vào những câu trả lời sai. Nếu có bất kỳ ví dụ nào mà mô hình rất kém, nó có thể trở nên rất tự tin vào câu trả lời sai đó, dẫn đến việc log likelihood của câu trả lời sai tăng lên, và do đó, negative log likelihood sẽ giảm.

Tóm lại, loss mà bạn tính toán và điều bạn thực sự quan tâm, độ chính xác, có thể không tương quan với nhau.

A Staker Example (Koehn and Knowles 2017)

Một vấn đề quan trọng là trong các mô hình được huấn luyện bằng phương pháp maximum likelihood, việc tìm kiếm một mô hình tốt hơn không nhất thiết mang lại kết quả sinh văn bản tốt hơn.

Một ví dụ từ dịch máy cho thấy rằng khi sử dụng beam search với kích thước beam ngày càng lớn, kích thước beam tối ưu để tìm ra đầu ra có điểm số cao nhất lại chỉ là 4. Tuy nhiên, độ chính xác lại giảm dần khi kích thước beam tăng lên. Điều này xảy ra vì các mô hình huấn luyện bằng maximum likelihood thường ưa chuộng các đầu ra ngắn hơn. Khi đầu ra trở nên dài hơn, xác suất của các đầu ra này giảm xuống, dẫn đến việc beam search tạo ra các đầu ra ngắn hơn.



Kết quả là, điểm số BLEU không thích các đầu ra quá ngắn, do đó độ chính xác giảm. Một số phương pháp cải thiện cho beam search bao gồm việc sử dụng trung bình log likelihood của từng token thay vì tổng log likelihood của toàn bộ đầu ra, nhưng vẫn không thể khắc phục hoàn toàn vấn đề này. Tóm lại, khi tìm kiếm nhiều hơn, độ chính xác vẫn có xu hướng giảm.

Managing Loss Function/ Eval Metrics Differences

Trong quá trình huấn luyện mô hình, có nhiều cách để cải thiện kết quả. Một phương pháp có tính nguyên tắc là sử dụng học tăng cường hoặc các thuật toán huấn luyện có cấu trúc khác để đảm bảo mô hình không tạo ra các đầu ra không mong muốn. Một cách đơn giản hơn là áp dụng "early stopping" dựa trên chỉ số đánh giá thay vì dựa trên hàm mất mát. Bằng cách này, bạn sẽ dừng lại khi đạt được chỉ số đánh giá cao nhất mà bạn quan tâm.

Tuy nhiên, "early stopping" theo cách này có thể dẫn đến một số vấn đề. Một trong những vấn đề là mô hình có thể trở nên quá tự tin vào các dự đoán sai của nó. Điều này liên quan đến khái niệm "calibration", tức là độ chính xác của các ước lượng xác suất. Một mô hình không được hiệu chỉnh tốt có thể rất tự tin dù đúng hay sai, điều này có thể gây ra vấn đề trong các nhiệm vụ như “downstream”.

Ngoài ra, có một hiện tượng thú vị mà nhiều người trong lĩnh vực diễn giải mô hình quan tâm, đó là "generalization Beyond overfitting on small algorithmic data sets". Nghiên cứu cho thấy rằng, trong một số trường hợp, mô hình cần thời gian huấn luyện rất dài để bắt đầu tổng quát hóa tốt và đạt độ chính xác cao. Điều này đặc biệt đúng với các tập dữ liệu yêu cầu nhiều bước liên tiếp phải đúng trước khi có thể đưa ra câu trả lời cuối cùng chính xác. Ví dụ, bạn cần thực hiện đúng 20 hoặc 50 bước liên tiếp trước khi có thể đưa ra câu trả lời đúng. Độ chính xác của từng quyết định riêng lẻ có thể tăng lên, nhưng chỉ khi tất cả các bước đều đúng thì mới được đánh dấu là chính xác. Điều này càng rõ rệt hơn khi nói về các nhiệm vụ yêu cầu nhiều bước suy luận hoặc nhiều bước tạo token chính xác.

Actionable Evaluation

Look At Your Data!

Trong quá trình cải thiện các mô hình, “actionable evaluation” là rất quan trọng. Đầu tiên, bạn cần xem xét dữ liệu mà mình đang sử dụng. Cả lỗi và hướng nghiên cứu mới đều có thể được phát hiện thông qua việc phân tích đầu ra của mô hình.

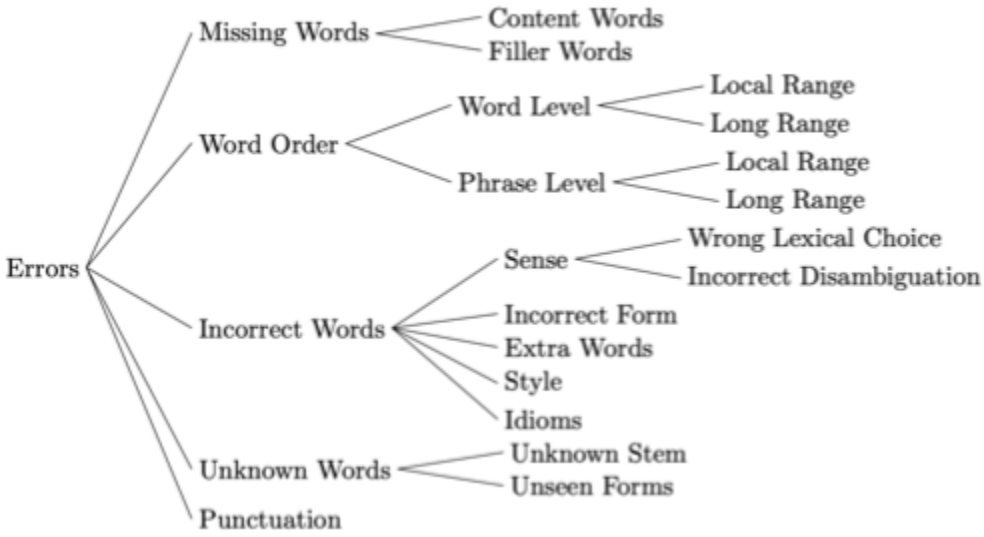
Một ví dụ điển hình về lỗi phổ biến là "off by one errors". Giả sử bạn đã triển khai một hệ thống dịch thuật và nó tạo ra các đầu ra như "went to the store yesterday bought a dog". Ngay lập tức, bạn có thể nhận thấy rằng câu này không giống như tiếng Anh tự nhiên. Vấn đề ở đây có thể là do bạn đã sử dụng chỉ số đầu ra sai, chẳng hạn như "output 1" thay vì "output 0". Đây là một lỗi đơn giản mà bạn có thể mắc phải trong quá trình tiền xử lý hoặc hậu xử lý dữ liệu.

Nếu bạn chỉ dựa vào các chỉ số đánh giá như điểm BLEU, bạn có thể thấy mình chỉ kém hơn một hoặc hai điểm so với mô hình tốt nhất, mà không hiểu rõ nguyên nhân. Tuy nhiên, nếu bạn xem xét kỹ lưỡng dữ liệu, bạn sẽ dễ dàng phát hiện ra vấn đề.

Ngoài ra, con người có khả năng nhận diện mẫu rất tốt. Nếu bạn có thể xem qua các điểm dữ liệu, bạn có thể nhận ra các mẫu về lý do tại sao mô hình lại thất bại. Ví dụ, bạn có thể nhận thấy rằng mô hình của bạn kém trong việc trả lời các câu hỏi liên quan đến con người, từ đó bạn sẽ cần cải thiện mô hình của mình về khía cạnh này. Hoặc nếu hệ thống RAG mà bạn xây dựng cho bài tập gặp khó khăn với các câu hỏi liên quan đến nghiên cứu, bạn có thể cần thu thập thêm dữ liệu nghiên cứu.

Systematic Qualitative Analysis of Model Errors

Trong quá trình phát triển các hệ thống dịch máy, việc phân loại và phân tích lỗi một cách có hệ thống là rất quan trọng. Một phương pháp đơn giản để thực hiện điều này là lấy mẫu ngẫu nhiên 100 đầu ra và xem xét 100 lỗi, sau đó nhóm chúng thành các loại lỗi khác nhau. Ví dụ, một phân loại lỗi đã được V et al. định nghĩa, trong đó họ nhóm các lỗi dịch máy thành các loại như: "content words", "filler words", "word level", "word order", "local range", "long range", v.v.



Tuy nhiên, khi các hệ thống ngày càng cải thiện, các phân loại lỗi cũ có thể không còn phù hợp. Ví dụ, các hệ thống dịch máy hiện nay ít mắc lỗi "local range word level" hơn. Thay vào đó, chúng ta cần phân tích chi tiết hơn, như xem xét lỗi liên quan đến "named entities".

Khoảng bốn năm trước, khi mọi người bắt đầu cho rằng các hệ thống dịch máy đã gần đạt đến trình độ của con người, chúng tôi đã so sánh các hệ thống này với con người và tạo ra một phân loại lỗi mới, phù hợp với bối cảnh năm 2020 thay vì 2006. Việc này rất hữu ích vì nó giúp xác định các loại lỗi nổi bật nhất, từ đó tìm ra cách cải thiện độ chính xác của hệ thống.

Nguyên tắc chung là không nên tối ưu hóa hệ thống dựa trên những lỗi không thực sự tồn tại. Điều này tương tự như trong lập trình, bạn không nên tối ưu mã nguồn cho đến khi chạy một profiler, vì có thể mã của bạn chậm ở một chỗ mà bạn không ngờ tới.

Quantitative Analysis

Trong phần này, chúng ta thảo luận về phân tích định lượng. Khi bạn chọn một hiện tượng để tập trung nghiên cứu, điều quan trọng là phải đo lường xem hiện tượng đó có được cải thiện hay không. Ví dụ, nếu bạn tập trung vào việc cải thiện chất lượng của các từ có tần suất thấp, bạn có thể kiểm tra xem độ chính xác trên các từ này có tăng lên không. Tương tự, nếu bạn đang cải thiện cú pháp trong một ngôn ngữ ít tài nguyên, bạn có thể đo lường xem việc sắp xếp từ hoặc các phụ thuộc dài có được cải thiện không.

Một công cụ hỗ trợ cho việc này được phát triển bởi giáo sư và Alex Cabrera là Zeno. Nó cho phép bạn xem dữ liệu và thực hiện các thao tác như lọc các ví dụ dịch máy từ ngôn ngữ Hausa hoặc xem các ví dụ có độ chính xác thấp. Bạn cũng có thể tạo biểu đồ để so sánh hiệu suất của các mô hình trên các tập dữ liệu khác nhau. Ví dụ, bạn có thể tìm các trường hợp mà "chat GPT" hoạt động kém hơn "GPT-4" và phân tích lý do, chẳng hạn như việc tạo ra văn bản sai định dạng.

Để sử dụng Zeno, bạn cần tạo một DataFrame của pandas chứa tất cả data và metadata bạn muốn sử dụng. Sau đó, bạn có thể tải lên và phân tích dữ liệu này.

Cuối cùng, khi áp dụng regularization trong mô hình, điều này có thể ảnh hưởng đến kỳ vọng về tổn thất của mô hình. Khi áp dụng regularization, tổn thất có thể không hội tụ về 0 vì cần phải điều chỉnh trọng số để giảm tổn thất, nhưng điều này lại làm tăng thành phần regularization. Tuy nhiên, bạn có thể đo lường riêng biệt các thành phần tổn thất để có cái nhìn rõ ràng hơn. Với một mô hình được tham số hóa hợp lý, tổn thất thực tế nên tiến gần về 0, mặc dù điều này có thể khó khăn hơn với các mô hình nhỏ.

Introduction to Model Interpretability

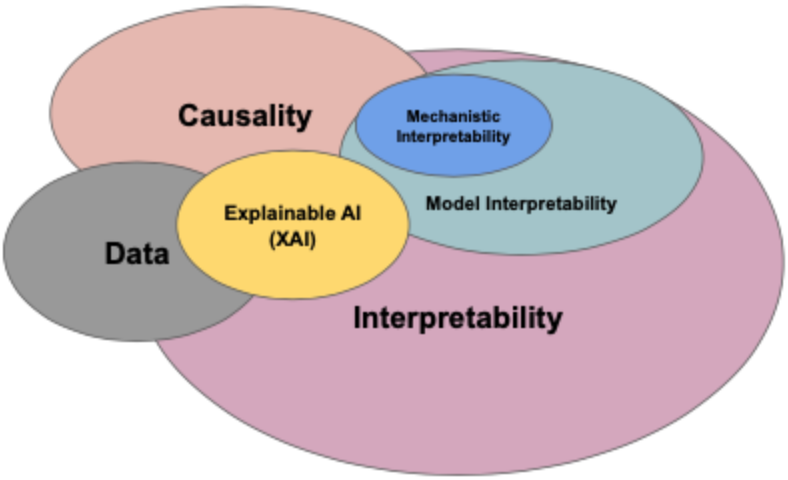
What I want to take away

Tiếp theo, tôi sẽ nói về tính diễn giải của mô hình (model interpretability). Có hai điểm chính mà tôi muốn bạn lưu ý trong phần này. Đầu tiên, tôi muốn thuyết phục bạn rằng tính diễn giải của mô hình là một lĩnh vực quan trọng cần nghiên cứu. Thứ hai, tôi hy vọng bạn sẽ tìm thấy chủ đề này thú vị và muốn khám phá thêm.

Interpretability (AI)

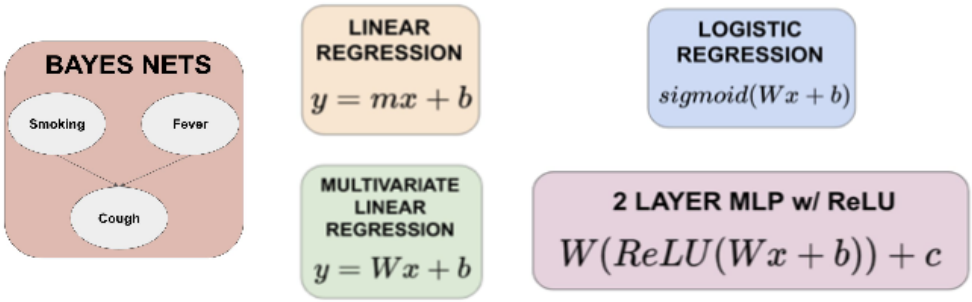
Trong AI, khả năng diễn giải được định nghĩa là việc nghiên cứu để hiểu các quyết định mà hệ thống AI đưa ra và diễn giải chúng thành những thuật ngữ dễ hiểu đối với con người. Điều này có thể mang nhiều ý nghĩa khác nhau và thường rất khó thực hiện. Mục tiêu của việc này là sử dụng sự hiểu biết đó để thiết kế các hệ thống ngày càng tốt hơn, không chỉ về hiệu suất mà còn về khả năng dễ hiểu đối với con người.

Khả năng diễn giải là một chủ đề lớn, nhưng còn nhiều lĩnh vực khác giao thoa với nó. Danh sách này chưa đầy đủ, nhưng có thể kể đến: nhân quả (causality) và dữ liệu có liên quan đến khả năng diễn giải. Explainable AI là một khái niệm bạn có thể đã nghe qua, nằm trong phạm vi của interpretability và kết nối với nhân quả và dữ liệu. Model interpretability nằm ở một khía cạnh khác, giao thoa một chút với causality và explainable AI, nhưng vẫn tách biệt. Mechanistic interpretability, một chủ đề đang được chú ý, nằm trong model interpretability, là một trường hợp đặc biệt của nó.

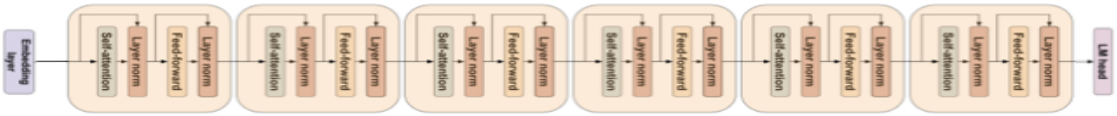


Historically models are small - Now they are not!

Lịch sử cho thấy chúng ta từng làm việc với các mô hình rất nhỏ. Ví dụ, Bayes Nets là một mô hình rất nhỏ với chỉ 8 tham số, trong đó chỉ có 4 tham số độc lập nếu tất cả các biến là nhị phân. Chúng ta cũng từng sử dụng nhiều hồi quy tuyến tính, với trường hợp đơn biến chỉ có 2 tham số và trường hợp đa biến cũng chỉ có một số lượng tham số nhỏ.



Tuy nhiên, hiện nay chúng ta đã chuyển sang các mô hình lớn hơn như MLPs với ma trận trọng số lớn hơn. Những mô hình này vẫn có thể hiểu và diễn giải được. Nhưng giờ đây, chúng ta đang đối mặt với các mô hình ngôn ngữ lớn, phức tạp và khó giải thích. Ví dụ, một mô hình Transformer 6 lớp, nhỏ hơn nhiều so với hầu hết các mô hình hiện tại, đã làm cho việc giải thích trở nên rất khó khăn.

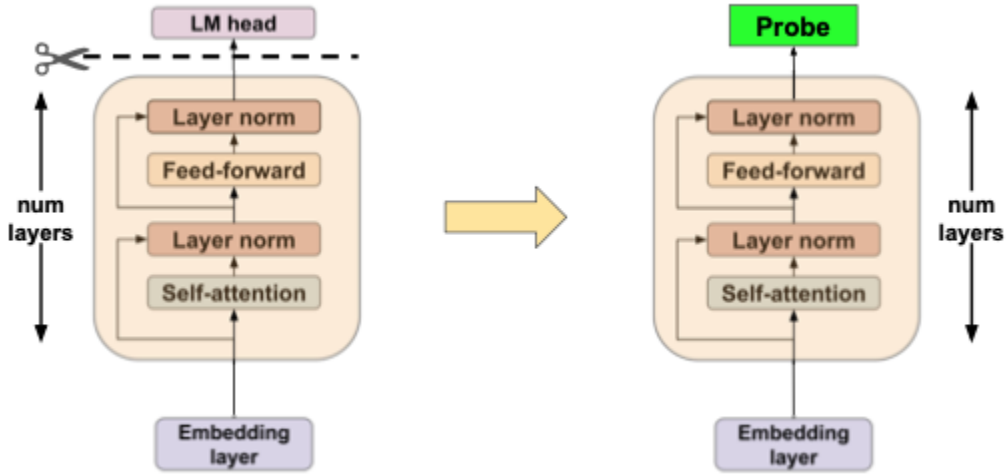


☹️

Probing

How we do make sense of this huge model?

Vậy làm thế nào để chúng ta hiểu được một mô hình khổng lồ? Đây là một cách tiếp cận: chúng ta lấy mô hình lớn của mình và cắt bỏ phần trên cùng. Sau đó, chúng ta gắn một "probe" vào. Thông thường, "probe" này là một mạng nơ-ron truyền thẳng nhỏ với một hoặc hai lớp. Chúng ta coi mô hình như một thực thể tồn tại và chỉ thực sự quan tâm đến đầu ra của mô hình.



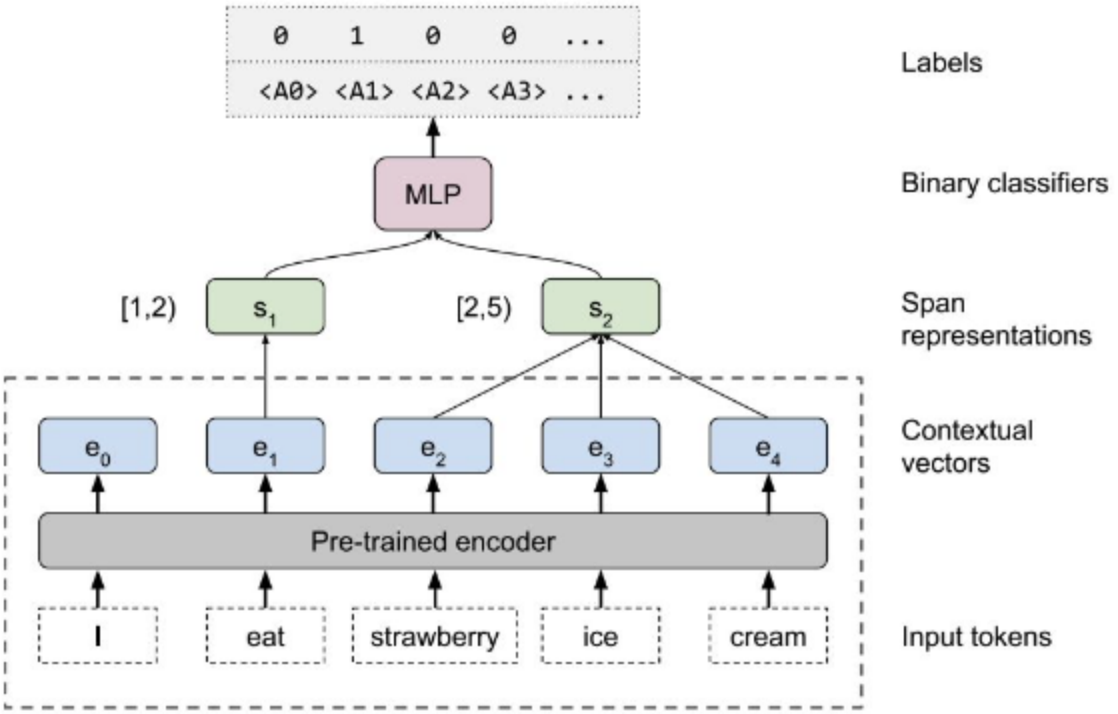
What is a Probe?

Probe là một bộ phân loại, được biểu diễn bằng hình ảnh màu xanh lá cây trong ví dụ, được huấn luyện đặc biệt để dự đoán một thuộc tính cụ thể nào đó chỉ dựa trên các biểu diễn của mô hình đã được huấn luyện trước.

Edge Probing (Tenney et al. 2019)

Năm 2019, Ian Tenny và các cộng sự đã giới thiệu phương pháp Edge probing, một phương pháp tổng quát cho phép khai thác các loại thông tin khác nhau từ một mô hình.

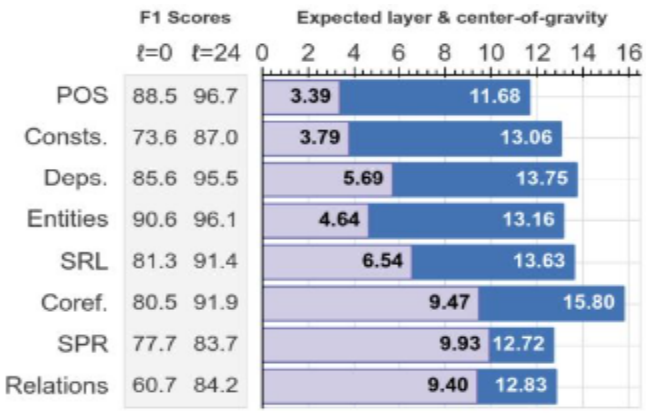
Quá trình hoạt động của Edge probing như sau: bạn đưa vào một chuỗi văn bản vào mô hình (thường là BERT trong các thí nghiệm) và mô hình sẽ xuất ra một tập hợp các vector ngữ cảnh. Các vector này có thể được lấy từ bất kỳ lớp nào trong mô hình, mặc dù thường thì chúng được lấy từ gần lớp trên cùng. Sau đó, một mạng nơ-ron đa lớp (MLP) sẽ được huấn luyện để đưa ra dự đoán. Trong các trường hợp này, mô hình gốc luôn được giữ cố định.



Một ví dụ điển hình là nhiệm vụ part of speech tagging - xác định loại từ cho mỗi từ trong câu. Trong trường hợp này, chỉ một trong hai khoảng S1 và S2 được kích hoạt, vì bạn đang dự đoán cho từng vector ngữ cảnh xem nó là danh từ, động từ hay một loại từ khác. Ngoài ra, bạn cũng có thể thực hiện các nhiệm vụ khác như trong bài toán "entailment", nơi bạn có hai chuỗi và hai khoảng, và bạn sử dụng các embedding cho các khoảng đó từ câu một và câu hai, sau đó kết hợp chúng và đưa vào MLP để xem liệu MLP có thể giải quyết bài toán đó hay không.

BERT rediscovers the NLP pipeline (Tenney et al. 2019)

Một bài báo khác có tựa đề "Bert Rediscovers the NLP Pipeline" đã khám phá nhiều khía cạnh thú vị trong mô hình BERT.



Trong hình minh họa, điểm nổi bật chính là các con số được hiển thị bằng màu hồng tím. Đây là các thuộc tính khác nhau như part of speech và nhiều yếu tố khác. Nghiên cứu cho thấy, ở các lớp đầu tiên của mô hình, những thông tin gần với biểu diễn cấp độ Token dễ dàng được trích xuất hơn bằng cách sử dụng probe. Trong khi đó, những thông tin đòi hỏi ngữ cảnh hóa nhiều hơn thì có thể trích xuất từ các lớp sau của mô hình.

Các nhiệm vụ được mô tả ngắn gọn như sau: những nhiệm vụ ở phía dưới mang tính ngữ nghĩa và ngữ cảnh hóa nhiều hơn, chẳng hạn như semantic role labeling và relation classification. Ngược lại, các nhiệm vụ ở phía trên như chunking, part of speech tagging và dependency labeling lại ít đòi hỏi ngữ cảnh hơn.

Issues with Probing (Belinkov et al. 2021)

Việc sử dụng phương pháp probing để kiểm tra các mô hình đã gặp phải nhiều vấn đề. Mặc dù trước đây có rất nhiều nghiên cứu về probing, nhưng hiện nay số lượng đã giảm đi đáng kể. Khi một probe hoạt động tốt, có thể là do biểu diễn thực sự mã hóa thông tin đó, nhưng cũng có khả năng probe tự giải quyết được nhiệm vụ mà không cần đến thông tin từ biểu diễn. Điều này là do probe được huấn luyện trên dữ liệu có nhãn.

Nếu probe không hoạt động có thể do biểu diễn thiếu thông tin cần thiết, hoặc probe không thể tách rời thông tin từ biểu diễn. Nguyên nhân có thể do probe không thuộc lớp hàm phù hợp, hoặc do quá trình huấn luyện probe chưa tốt, bao gồm cả việc điều chỉnh siêu tham số.

Một vấn đề khác là việc probe thường yêu cầu nhiều dữ liệu giám sát, nhưng việc thu thập dữ liệu này không phải lúc nào cũng khả thi. Kết quả là, chúng ta chỉ có một tập dữ liệu mẫu tiện lợi, và thực chất, probe chỉ đang kiểm tra tập dữ liệu đó. Do những hạn chế này, phương pháp probing đã không còn được ưa chuộng như một công cụ cốt lõi để giải thích mô hình.

Ngoài ra, các probe được thiết kế theo cách này thường chỉ mang tính tương quan, không phải nhân quả. Mô hình cơ bản được huấn luyện theo một cách cụ thể, và thông tin được tách rời, chỉ còn lại biểu diễn đầu ra. Việc kiểm tra chỉ dừng lại ở mức độ xem liệu biểu diễn đầu ra có tương quan với thuộc tính mà probe đang kiểm tra hay không, mà không có khái niệm can thiệp vào không gian tiềm ẩn hay xác định quan hệ nhân quả. Do đó, cộng đồng nghiên cứu đã dần chuyển hướng khỏi lĩnh vực này.

Other Probing Works

Trong lĩnh vực nghiên cứu mô hình, có nhiều vấn đề mà các nhà nghiên cứu đang cố gắng giải quyết. Một trong những khái niệm quan trọng là "model interpretability" (khả năng diễn giải mô hình), được định nghĩa là việc hiểu rõ các thành phần bên trong của mô hình, chẳng hạn như trọng số và các hàm kích hoạt của nó, và diễn đạt những hiểu biết này bằng ngôn ngữ dễ hiểu cho con người. Mục tiêu là không chỉ sửa chữa các mô hình hiện tại mà còn phát triển những mô hình tốt hơn.

Một nhánh con của khả năng diễn giải mô hình là "mechanistic interpretability", tập trung vào việc đảo ngược kỹ thuật các mô hình tham số, thường là mạng nơ-ron, để chuyển đổi trọng số đã học thành các đơn vị thuật toán dễ hiểu hơn, thường được gọi là "circuits". Những "circuits" này là các hàm có thể được mô tả một cách dễ hiểu và nằm bên trong các mô hình.

Có nhiều nghiên cứu đáng chú ý trong lĩnh vực này. Một trong số đó là phân tích các MLP nhỏ và Transformers để xây dựng trực giác về các "circuits" tồn tại. Một phát hiện quan trọng là khái niệm "induction heads", là các đầu chú ý đặc biệt giúp mô hình thực hiện "in-context learning". Cụ thể, khi được cung cấp một tiền tố, induction head cho phép mô hình sao chép token cần thiết từ dữ liệu huấn luyện mà nó đã thấy trước đó.

Ngoài ra, nghiên cứu cũng đề cập đến "polysemanticity", tức là một nơ-ron có thể đại diện cho nhiều đặc điểm khác nhau. Khi xử lý một từ vựng lớn, mô hình sẽ nén thông tin xuống một không gian ẩn nhỏ hơn, dẫn đến việc một nơ-ron có thể đại diện cho nhiều đặc điểm cùng lúc. Điều này có thể gây khó khăn trong việc sử dụng hiệu quả các đặc điểm này, vì số lượng đặc điểm có thể lớn hơn số lượng nơ-ron trong không gian kích hoạt.

Cuối cùng, có một số nghiên cứu gần đây cho thấy rằng Transformers có khả năng tốt hơn trong việc thực hiện các tác vụ như induction heads so với các mô hình khác. Điều này mở ra câu hỏi về việc liệu có cấu trúc nào tốt hơn Transformers trong việc học tập theo ngữ cảnh hay không.

Model interpretability

Weights and Activations

Tiếp theo, chúng ta sẽ tìm hiểu về tính diễn giải của mô hình, tập trung vào trọng số và các kích hoạt. Trọng số của một mô hình đã được huấn luyện cho phép chúng ta chỉnh sửa và quan sát những thay đổi xảy ra.

Tương tự, chúng ta có thể xem xét các kích hoạt cho các đầu vào khác nhau, thực hiện các thao tác để xem phản ứng của mô hình. Nghiên cứu của tôi chủ yếu xoay quanh việc "chọc" vào các mô hình và quan sát các kích hoạt. Thuật ngữ kỹ thuật cho quá trình này là can thiệp vào không gian ẩn bằng cách thêm một vector hoặc thực hiện các thao tác khác. Tuy nhiên, thực chất, chúng ta chỉ đang "chọc" vào mô hình để tìm hiểu sâu hơn về cách nó hoạt động.

Looking at weights

Model Editing

Chỉnh sửa mô hình (model editing) là một phương pháp quan trọng, trong đó fine-tuning là một phiên bản cực đoan nhất. Mục tiêu của chỉnh sửa mô hình là thay đổi một khái niệm hoặc thông tin cụ thể trong mô hình mà không làm ảnh hưởng đến các hành vi khác của mô hình. Ví dụ,

nếu muốn thay đổi thông tin rằng Graham không còn là giáo sư tại CMU mà là giáo sư tại Stanford, ta không muốn mọi người ở CMU đều bị gán là giáo sư tại Stanford.

Một nghiên cứu (Meng et al. 2022) đã sử dụng phương pháp causal tracing để xác định tác động nhân quả của các trạng thái ẩn liên quan đến thông tin cần chỉnh sửa. Họ liên tục thay đổi đầu vào, thực hiện nhiều lần lan truyền thuận (forward passes) và tìm ra các trạng thái ẩn cụ thể liên quan đến thông tin đó. Sau đó, họ thực hiện chỉnh sửa bằng cách coi cặp "Space Needle" và "Paris" như một cặp khóa-giá trị, và thay đổi đầu ra từ "Seattle" thành "Paris".

Phương pháp này rất tốn kém do cần nhiều lần lan truyền thuận, nhưng đã được cải thiện trong các nghiên cứu sau. Tuy nhiên, việc chỉnh sửa mô hình có thể gây ra các vấn đề khác như làm thay đổi hoặc tạo ra thiên lệch không mong muốn trong mô hình. Ngoài ra, thông tin thường xuất hiện ở nhiều nơi trong mô hình, đặc biệt là với các mô hình lớn, nên có thể cần can thiệp ở nhiều điểm khác nhau.

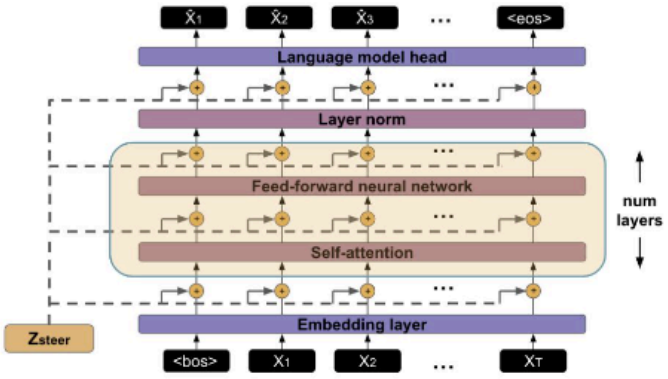
Looking at activations

Steering Vectors (Subramani et al. 2019; 2020; 2022)

Steering vectors là các vector có độ dài cố định được thêm vào các trạng thái ẩn của mô hình ngôn ngữ tại một vị trí cụ thể nhằm điều khiển mô hình sinh ra một chuỗi văn bản nhất định. Khác với các phương pháp như soft prompts hay model editing, steering vectors tương ứng với một chuỗi cụ thể và không liên quan đến bất kỳ chuỗi nào khác. Thực tế, có thể có nhiều vector khác nhau tương ứng với cùng một chuỗi, tùy thuộc vào cách chúng được trích xuất.

Extracting Steering Vectors

Quá trình trích xuất steering vectors bắt đầu bằng việc khởi tạo ngẫu nhiên vector Z_{steer} với giá trị nhỏ và phân bố đồng đều. Đối với mỗi chuỗi cụ thể mà chúng ta muốn mô hình sinh ra, vector này được tối ưu hóa để sinh ra chuỗi đó trong khi giữ nguyên toàn bộ mô hình. Ý tưởng là "không động" mô hình đã được huấn luyện để nó có thể sinh ra chuỗi mong muốn khi vector Z_{steer} được thêm vào.



ALGORITHM 1: Extracting z_{steer} for a sentence

Input : x – target sentence
M – pretrained language model
θ – pretrained language model weights
I_L – injection location
I_T – injection timestep
d – dimension of z_{steer}

Output : z_{steer} – extracted candidate steering vector

$z_{steer} \sim$ xavier_normal(d)
for $i \leftarrow [1, 2, ..., N]$ **do**
  logits $= M_\theta.forward(x, z_{steer}, I_L, I_T)$
  $\mathcal{L} = XENT(logits, x)$
  $\mathcal{L}.backward()$
  $z_{steer} = z_{steer} + lr * \frac{\partial \mathcal{L}}{\partial z_{steer}}$
end
return z_{steer}

Các bước thực hiện:

1. Khởi tạo: Z_{steer} được khởi tạo ngẫu nhiên với độ dài cố định.
2. Tối ưu hóa: Qua vài vòng lặp (thường từ 8 đến 10 lần), Z_{steer} được tinh chỉnh thông qua các lần lan truyền thuận để nó dần dần điều khiển mô hình sinh ra chuỗi mong muốn.
3. Chèn Vector: Trong thực tế, Z_{steer} thường được chèn vào bước thời gian đầu tiên và một vị trí nào đó ở giữa mô hình (không phải lớp đầu tiên hay lớp cuối cùng).

Trong thế giới của soft prompts, thường cần một vector có chiều rộng lớn (ví dụ, độ rộng 50) để đạt được hiệu quả tốt. Ngược lại, các steering vectors có chiều rộng bằng với kích thước ẩn của mô hình và thường nhỏ hơn nhiều. Điều này cho phép steering vectors thực hiện nhiệm vụ điều khiển một cách hiệu quả hơn với ít tài nguyên hơn.

Steering vector results

Các nghiên cứu đã chỉ ra rằng:

- Dễ dàng tìm thấy: Với hầu hết các chuỗi, thậm chí cả những chuỗi mà mô hình chưa từng thấy trước đó, steering vectors có thể được tìm thấy một cách dễ dàng.
- Tương thích ngữ nghĩa: Khoảng cách giữa các steering vectors phản ánh được sự tương đồng về ngữ nghĩa giữa các chuỗi. Ví dụ, hai câu gần nhau về mặt ý nghĩa sẽ có các vectors gần nhau trong không gian steering.
- Chuyển đổi phong cách: Bằng việc thực hiện các phép tính vector đơn giản, như trừ và cộng, có thể điều chỉnh phong cách của câu. Ví dụ:
  Vector ban đầu: "The taste is excellent."
  Sau khi điều chỉnh: "The taste is unpleasant."
- Interpolation: Có thể thực hiện nội suy giữa các steering vectors để tạo ra các chuỗi mới mà không bị sai lệch quá nhiều từ không gian vector.

Mặc dù steering vectors mang lại nhiều tiềm năng, nhưng cũng có một số hạn chế:

- Phụ thuộc vào độ lớn Vector: Nếu vector có độ lớn quá lớn, mô hình có thể "bị kẹt" và không thể sinh ra các chuỗi hợp lý.
- Không gian Vector không hoàn hảo: Có một số vùng trong không gian steering vectors mà khi nội suy hoặc thao tác có thể dẫn đến việc sinh ra các chuỗi không hợp lệ hoặc lặp lại.

Inference-time Interventions (Li et al. 2023)

Một nghiên cứu gần đây liên quan đến can thiệp trong quá trình suy luận. Nghiên cứu này áp dụng các ý tưởng đã được đề cập trước đó, sử dụng các đầu dò tuyến tính (linear probes) để

xác định các attention heads tương ứng với một thuộc tính mong muốn. Cụ thể, họ đã thực hiện điều này cho truthful QA, với hy vọng tìm ra các hướng trung thực trong không gian ẩn (latent space).

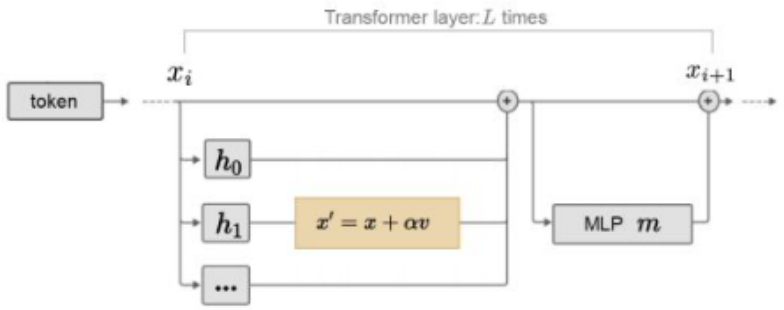


Figure 3: A sketch of the computation on the last token of a transformer with inference-time intervention (ITI) highlighted.

Quá trình này bao gồm việc điều chỉnh các kích hoạt của attention head trong quá trình suy luận theo các hướng được xác định bởi các phép đo. Cụ thể, họ sử dụng một bộ phân loại để phân biệt giữa các thông tin trung thực và không trung thực, từ đó xác định một hyperplane. Sau đó, có thể di chuyển vuông góc với siêu phẳng này theo hướng mong muốn, ví dụ như hướng tới sự trung thực.

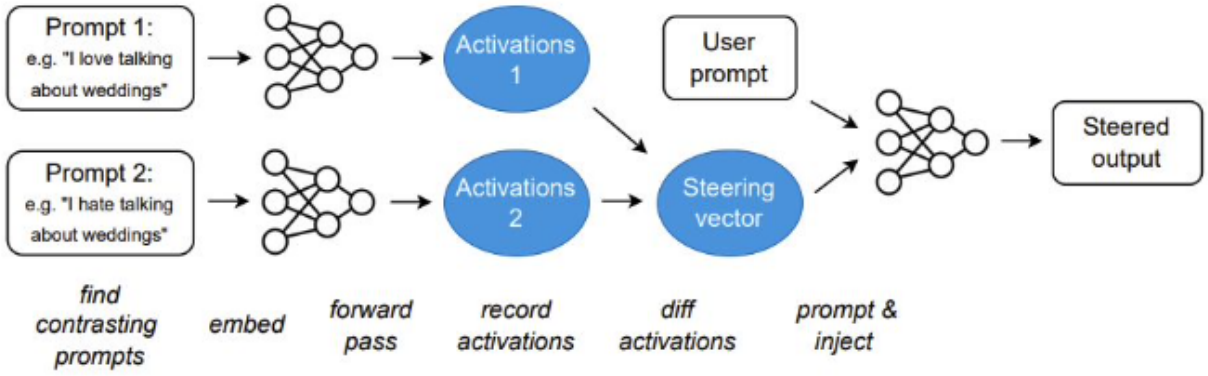
Nghiên cứu này được thực hiện trên các mô hình GPT và có thể là mô hình LLaMA. Một điểm đáng chú ý là việc thêm vector tìm được vào mô hình cần được điều chỉnh cẩn thận về độ lớn, vì nếu quá lớn có thể gây ra lỗi. Do đó, họ đã thực hiện tìm kiếm hyperparameter để xác định độ lớn phù hợp cho các kích hoạt.

Nghiên cứu này tập trung vào các attention head cụ thể và không áp dụng cho tất cả attention heads. Điều này dẫn đến câu hỏi liệu các attention head có chuyên biệt hóa hay không, và dường như là có. Nhiều attention head không có độ chính xác cao khi được kiểm tra và thậm chí có thể gây nhiễu cho hướng trung thực.

More activation manipulation

Gần đây có một số nghiên cứu về vector điều hướng đối lập (contrastive steering vectors) (Turner et al. 2023; Rimsky et al. 2023). Phương pháp này được áp dụng trong việc điều hướng cảm xúc, nơi chúng ta sử dụng các câu tích cực và tiêu cực không liên kết với nhau một cách rõ ràng. Chúng ta tìm các vector điều hướng của chúng một cách riêng biệt.

Một trường hợp hữu ích hơn có thể là khi có hai prompts được thiết kế để đi theo hai hướng khác nhau. Bằng cách tìm các biểu diễn của chúng và thực hiện thao tác trên sự khác biệt, chúng ta có thể bảo toàn ngữ cảnh.



Điều này đặc biệt hữu ích trong các tác vụ truy xuất thông tin, khi bạn có một tài liệu và một câu hỏi. Nếu bạn muốn đặt câu hỏi theo hai cách khác nhau cho hai mục đích khác nhau, việc sử dụng vector điều hướng sẽ hiệu quả hơn. Mặc dù không có sự so sánh trực tiếp với các phương pháp trước đây, nhưng phương pháp này dường như hoạt động tốt hơn, tổng quát hơn và hữu ích hơn.

What can model interpretability give us?

Tính diễn giải của mô hình ngôn ngữ mang lại cho chúng ta nhiều lợi ích quan trọng. Đầu tiên, nó giúp chúng ta hiểu rõ hơn về cách thức hoạt động, cấu trúc và nội bộ của các mô hình ngôn ngữ. Mặc dù hiện tại vẫn còn nhiều điều chưa rõ ràng về lý do tại sao chúng hoạt động hiệu quả, nhưng việc tìm ra các phương pháp nhẹ nhàng để kiểm soát và điều hướng mô hình là cần thiết khi chúng ngày càng trở nên hữu ích và ảnh hưởng đến nhiều người dùng hơn.

Ngành công nghiệp có thể không đầu tư nhiều vào việc này, do đó, vai trò của học thuật là thực hiện nhiều nghiên cứu khoa học hơn để tìm ra cách kiểm soát và điều hướng mô hình tốt hơn. Ngoài ra, chúng ta cũng cần tìm kiếm các phương pháp thay thế hoặc bổ sung để thực hiện việc căn chỉnh mô hình. Phương pháp RLHF hiện tại khá tốn kém, và nếu có thể thực hiện với dữ liệu hạn chế và khai thác cấu trúc, thông tin đã có trong mô hình, chúng ta có thể căn chỉnh chúng tốt hơn. Những phương pháp này không nhất thiết phải là thay thế mà có thể bổ sung cho RLHF.

Resources

1. https://phontron.com/class/anlp2024/lectures/#debugging-and-interpretation-feb-27
2. Recommended Reference: Interpretable Machine Learning (Molnar 2022)
3. Reference: T5: Larger Models are Better (Raffel et al. 2020)
4. Reference: Scaling Laws for Neural Language Models (Kaplan et al. 2020)
5. Reference: Train Large, then Distill (Li et al. 2020)
6. Reference: compare-mt (Neubig et al. 2019)
7. Reference: ExplainaBoard (Liu et al. 2021)
8. Reference: Edge Probing (Tenney et al. 2019a)
9. Reference: BERT Rediscovers the Classical NLP Pipeline (Tenney et al. 2019b)

10. Reference: Control Tasks for Probing (Hewitt et al. 2019)
11. Reference: Probing Classifiers: Promises, Shortcomings, and Advances (Belinkov 2022)
12. Reference: Information Theoretic Probing with MDL (Voita et al. 2020)
13. Reference: Amnesic Probing (Elazar et al. 2021)
14. Reference: Low-Complexity Probing (Cao et al. 2021)
15. Reference: Pareto Probing (Pimentel et al. 2020)
16. Reference: Zoom In: An Introduction to Circuits (Olah et al. 2020)
17. Reference: A Mathmatical Framework for Transformer Circuits (Elhage et al. 2021)
18. Reference: Induction Heads (Olsson et al. 2022)
19. Reference: Toy Models of Superposition (Elhage et al. 2022)
20. Reference: ROME (Meng et al. 2022)
21. Reference: Steering Vectors in LSTMs (Subramani et al. 2019)
22. Reference: Steering Vectors in Transformers v1 (Subramani and Suresh 2020)
23. Reference: Steering Vectors in Transformers v2 (Subramani et al. 2022)
24. Reference: Inference-time Interventions (Li et al. 2023)
25. Reference: Activation Addition (Turner et al. 2023)
26. Reference: Contrastive Activation Addition (Rimsky et al. 2023)