Lecture 11: Quantization, Pruning and Distillation

Giới thiệu

Trong bối cảnh hiện nay, các mô hình xử lý ngôn ngữ tự nhiên (NLP) đang được triển khai ở quy mô lớn. Việc đào tạo các mô hình lớn là một quá trình tốn kém, điều này chắc hẳn ai cũng đã trải nghiệm qua các bài tập thực hành. Khi bạn huấn luyện một mạng nơ-ron sâu, bạn cần tài nguyên GPU, điều mà chúng ta đều hiểu.

Tuy nhiên, một khía cạnh thường bị bỏ qua là chi phí suy diễn (inference). Sau khi đã có mô hình đã được huấn luyện, việc triển khai và tạo ra dự đoán cho người dùng có thể tốn kém hơn nhiều. Theo một phân tích, chi phí trong suốt vòng đời của một mô hình có thể vượt qua chi phí đào tạo chỉ sau một tuần sử dụng. Nếu mô hình được sử dụng trong nhiều tháng hoặc nhiều năm với nhiều người, chi phí này sẽ vượt xa chi phí đào tạo, vốn chỉ là một lần.

Đây là một vấn đề lớn, đặc biệt khi chúng ta muốn các hệ thống AI có thể phục vụ nhiều người ở những nơi khác nhau, bao gồm cả những người có ít tài nguyên hoặc không có đủ nguồn điện. Chúng ta cần giảm chi phí phục vụ các hệ thống AI cho công chúng. Thách thức này càng trở nên khó khăn hơn khi các mô hình ngày càng lớn. Mặc dù có một chút xu hướng giảm kích thước mô hình trong hai năm qua, nhưng các mô hình hiện tại vẫn có kích thước lên tới hàng tỷ tham số, điều này khiến việc phục vụ trở nên tốn kém.

Main Question

Làm thế nào để triển khai các hệ thống NLP một cách tiết kiệm, hiệu quả và công bằng mà không làm giảm hiệu suất?

Answer: Model Compression

Có ba phương pháp chính để nén mô hình.

- Quantization: phương pháp này không thay đổi kiến trúc hay tham số của mô hình mà chỉ giảm độ chính xác của chúng và loại bỏ phần dư thừa.
- Pruning: tức là loại bỏ hoàn toàn các thành phần hoặc tham số không cần thiết của mô hình.
- Distillation: phương pháp này thay đổi toàn bộ tham số nhưng cô đọng kiến thức của một mô hình lớn vào một mô hình nhỏ hơn, thường được huấn luyện lại từ đầu để tái tạo hành vi của mô hình lớn.

Why is this even possible?

Ý tưởng nén mô hình rất hấp dẫn: chỉ cần thu nhỏ mô hình mà vẫn giữ nguyên hiệu suất, từ đó giảm chi phí triển khai. Tuy nhiên, câu hỏi đặt ra là tại sao điều này lại khả thi? Tại sao không bắt đầu với một mô hình nhỏ ngay từ đầu mà phải thu nhỏ từ mô hình lớn?

Một khái niệm quan trọng cần hiểu là "overparameterize" - tức là mô hình có nhiều tham số hơn dữ liệu huấn luyện. Ví dụ, mô hình GPT-3 ban đầu có 170 tỷ tham số, rõ ràng là

overparameterized. Nhiều nghiên cứu đã chỉ ra rằng các mô hình overparameterized dễ huấn luyện hơn, đặc biệt với các tác vụ phức tạp.

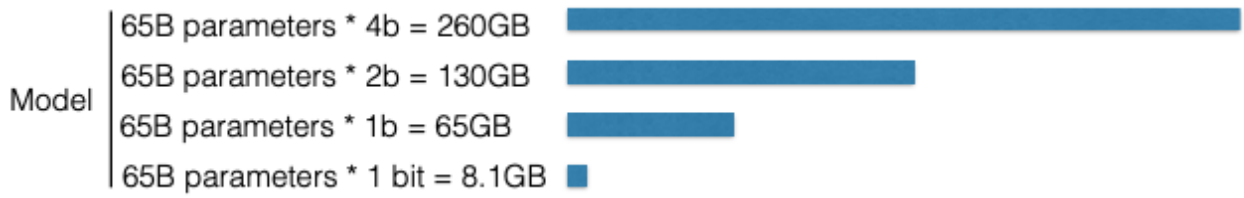
Huấn luyện mạng nơ-ron sâu thường yêu cầu tối ưu hóa một hàm mục tiêu không lồi (non-convex objective), và không có gì đảm bảo rằng ta sẽ tìm được điểm tối ưu toàn cục. Tuy nhiên, khi có nhiều tham số, ta có thể "né" các điểm yên ngựa (saddle points) hoặc các điểm tối ưu cục bộ không phải là tối ưu toàn cục. Điều này giúp quá trình tối ưu hóa trở nên linh hoạt hơn.

Tóm lại, mô hình với nhiều tham số dễ huấn luyện hơn và tạo ra mô hình tốt hơn, nhưng có thể không cần tất cả các tham số đó cho quá trình suy luận. Chúng chủ yếu là một "thủ thuật" trong giai đoạn huấn luyện.

Quantization

Post-Training Quantization

Chúng ta sẽ bắt đầu với khái niệm lượng tử hóa (quantization) và phương pháp rõ ràng nhất là lượng tử hóa sau huấn luyện (post-training quantization). Bạn có thể huấn luyện một mô hình lớn tùy ý và sau đó giảm độ chính xác của tất cả các trọng số trong mô hình đó. Ví dụ, nếu bạn có một mô hình 65 tỷ tham số như LLaMA 2 và sử dụng độ chính xác 4 bit, tức là 4 byte, thì chỉ riêng việc tải mô hình này vào bộ nhớ đã cần tới 260 GB bộ nhớ GPU, vượt quá khả năng của hầu hết các GPU đơn lẻ hiện nay.



Tuy nhiên, nếu bạn giảm độ chính xác của các tham số trọng số trong mô hình, bạn sẽ thấy sự giảm đáng kể về dung lượng bộ nhớ, tỷ lệ thuận với mức giảm độ chính xác. Trong trường hợp cực đoan nhất, nếu bạn thay thế mỗi float 32 bằng một bit đơn (0 hoặc 1), thì mô hình chỉ còn khoảng 8 GB, có thể tải vào hầu hết các GPU. Đây rõ ràng là một đề xuất hấp dẫn về mặt chi phí.

Floating point numbers

Trong mạng nơron, trọng số (weights) thường được biểu diễn bằng số thực dấu phẩy động (floating point) để có phạm vi giá trị rộng hơn. Theo chuẩn IEEE, số thực dấu phẩy động gồm 3 phần:

- Sign bit: Xác định dấu dương/âm

- Fraction: Xác định phạm vi giá trị
- Exponent: Xác định tỷ lệ (scale) lớn/nhỏ của số

Ví dụ với Float16:

- 10 bits cho fraction: cho phép nhiều giá trị khả dụng
- 5 bits cho exponent: cho phép tỷ lệ lên tới 2^5
- 1 bit cho sign: +1 hoặc -1

Tuy nhiên, Float16 thường không đủ cho machine learning vì các mô hình neural network thường có giá trị rất lớn hoặc rất nhỏ (underflow/overflow). Vì vậy, BFloat16 (Brain Floating Point 16) được thiết kế riêng cho machine learning bằng cách:

- Di chuyển một số bits từ phần fraction sang phần exponent
- Cho phép phạm vi giá trị rộng hơn
- Đánh đổi bằng việc giảm số lượng giá trị có thể biểu diễn trong phạm vi đó

Khi số bits < 16, khả năng biểu diễn giá trị bị giới hạn nghiêm trọng do:

- Cần 1 bit cho sign
- Phạm vi exponent nhỏ làm giảm đáng kể phạm vi giá trị có thể biểu diễn

Int8 Quantization

Một phương pháp phổ biến để giảm dung lượng mô hình là lượng tử hóa sang số nguyên (integer quantization). Một cách tiếp cận là ABS Max (Absolute Maximum) quantization, có nguyên lý hoạt động như sau:

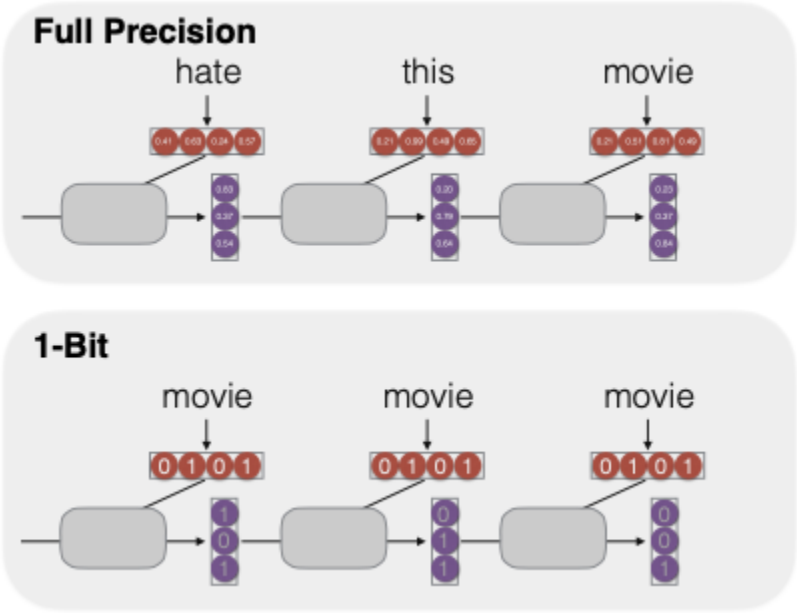
$$\mathbf{X}_{i8} = \left\lfloor \frac{127 \cdot \mathbf{X}_{f16}}{\max_{ij}(|\mathbf{X}_{f16_{ij}}|)} \right\rceil$$

- Ánh xạ danh sách số thực sang một khoảng số nguyên
- Với Int8: Khoảng từ -127 đến 127 (tổng 256 giá trị với 2^8)

- Tìm giá trị tuyệt đối lớn nhất trong mảng và ánh xạ nó thành 127
- Các giá trị khác được ánh xạ tỷ lệ thuận

Ví dụ: Ánh xạ [0.5, 20, -0.0001, -.01, -0.1]
- Giá trị lớn nhất là 20 → ánh xạ thành 127
- Giá trị 0.5 (= 1/40 của 20) → ánh xạ thành 3 (≈ 1/40 của 127)

Trường hợp cực đoan nhất là lượng tử hóa nhị phân (binary quantization):



- Chuyển đổi các tham số float (từ 0 đến 1) thành 0 hoặc 1
- Ưu điểm: Chỉ cần 1 bit cho mỗi vector
- Nhược điểm: Trong các tác vụ như dịch máy:
- Các hidden states và activations ban đầu khác nhau
- Sau khi làm tròn thành 0/1, các vector embedding có thể trở nên giống hệt nhau
- Điều này làm giảm nghiêm trọng khả năng biểu diễn của mô hình

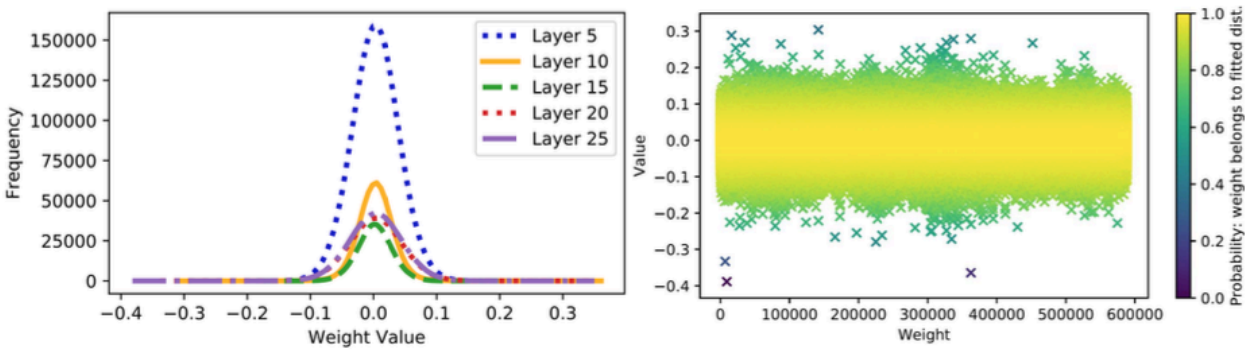
Việc chuyển đổi trực tiếp từ số thực chính xác cao sang số nhị phân thường không hiệu quả. Cần có thêm các bước xử lý sau huấn luyện để cải thiện kết quả.

Model-Aware Quantization: GOBO (Zadeh et al. 2020)

Model aware quantization là một phương pháp lượng tử hóa hiệu quả sau huấn luyện, dựa trên việc phân tích thống kê của mô hình. Nguyên lý chính của phương pháp này:

Nghiên cứu phân bố trọng số:
- Ví dụ với mô hình BERT: Phần lớn trọng số trong mỗi layer tập trung quanh giá trị trung bình
- Một số ít trọng số nằm xa giá trị trung bình

- Phân bố này có thể được mô tả bằng phân phối chuẩn (Gaussian distribution)



Vấn đề với các giá trị ngoại lai (outliers):

- Khi dùng ABS Max quantization, khoảng giá trị được xác định bởi giá trị min và max
- Các giá trị ở giữa, dù gần nhau, có thể bị gom vào cùng một nhóm
- Điều này làm giảm khả năng phân biệt giữa các trọng số trong mạng

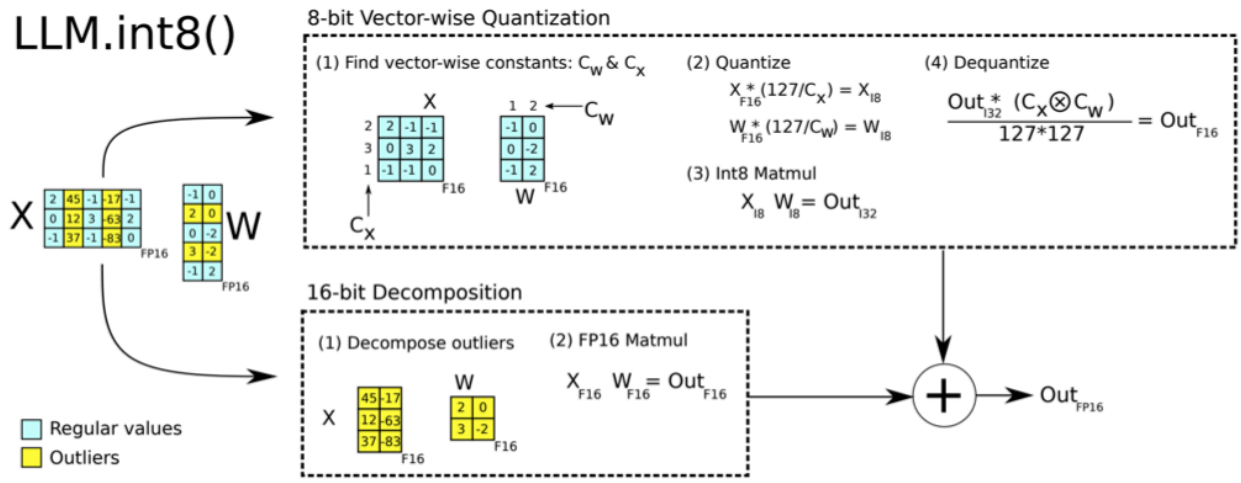
Giải pháp:

- Lưu trữ riêng các giá trị ngoại lai với độ chính xác đầy đủ (full precision)
- Các giá trị tập trung được lượng tử hóa với độ chính xác thấp hơn
- Phương pháp này cho kết quả tốt trong thực tế

Model-Aware Quantization: LLM.int8 (Dettmers et al. 2022)

Khi thực hiện lượng tử hóa các mô hình ngôn ngữ lớn, việc định nghĩa các giá trị ngoại lai (outliers), giá trị nhỏ nhất và lớn nhất một cách đồng nhất cho mỗi layer thường gặp một số hạn chế. Để khắc phục điều này, phương pháp LLM.int8 đề xuất thực hiện lượng tử hóa riêng biệt cho từng hàng hoặc cột của vector trong phép nhân ma trận.

Động lực đằng sau cách tiếp cận này xuất phát từ việc phần lớn tham số trong các mô hình Transformer được sử dụng cho phép nhân ma trận. Bằng cách lượng tử hóa theo từng hàng/cột, ta có thể đạt được độ chính xác cao hơn do phạm vi giá trị được xác định cụ thể hơn cho mỗi phần của ma trận.



Một điểm đáng chú ý là tần số phân bố trọng số thay đổi theo từng layer. Các layer sau thường xuất hiện nhiều giá trị ngoại lai có độ lớn cao hơn. Điều này là do trọng số ở mỗi layer có thể tập trung vào một giá trị cụ thể hoặc phân tán rộng hơn.

Tuy nhiên, phương pháp này cũng có chi phí phụ trợ. Quá trình ánh xạ vector thành danh sách số và giải mã ngược lại thành số thực trong quá trình inference tốn thời gian xử lý. Với mô hình nhỏ, kỹ thuật này có thể không mang lại lợi ích về tốc độ. Nhưng với mô hình lớn, nó có thể tăng gấp đôi tốc độ inference và cho phép tải được những mô hình mà bình thường không thể load vào bộ nhớ.

Hardware Concerns (Shen et al. 2019)

Trong lĩnh vực phần cứng, một trong những thách thức lớn đối với quantization là sự hạn chế của phần cứng và các hệ thống cấp thấp như framework mà bạn đang sử dụng, ví dụ như PyTorch. Một số kiểu dữ liệu không được phần cứng hỗ trợ, chẳng hạn như int3, không phải là thứ mà các bộ xử lý thường hỗ trợ. Nếu phương pháp quantization của bạn sử dụng int3, thực tế nó sẽ sử dụng int4 và bạn sẽ không đạt được tốc độ mong muốn.

PyTorch cũng có những yêu cầu riêng, chẳng hạn như không hỗ trợ int4 và nhiều module không hỗ trợ quantization. Ví dụ, RNN hiện đang trở nên phổ biến trở lại nhưng vẫn chưa hỗ trợ quantization.

Nếu bạn muốn áp dụng quantization vào ứng dụng thực tế, bạn cần biết rõ phần cứng và framework của mình có thể hỗ trợ những gì. Cả hai phương pháp tôi đã trình bày đều có các bộ tăng tốc phần cứng tùy chỉnh để làm cho chúng hoạt động, điều này đòi hỏi rất nhiều công sức và thời gian mà hầu hết mọi người không có.

Tuy nhiên, vẫn có những phương pháp quantization hiệu quả mà không cần phải viết lại framework hay bộ tăng tốc phần cứng của bạn. Đây là điều cần cân nhắc khi bạn muốn áp dụng quantization.

Quantization Aware Training

Binarized Neural Networks (Courbariaux et al. 2016)

Post-training quantization gặp khó khăn vì việc giảm độ chính xác (precision) có thể ảnh hưởng đến hiệu suất của mạng neural đã được huấn luyện. Một giải pháp thay thế là huấn luyện mô hình ngay từ đầu với việc quantization.

Ví dụ điển hình là trường hợp của Binarized Neural Networks. Theo một nghiên cứu năm 2016, mạng neural nhị phân có thể hoạt động hiệu quả nếu được huấn luyện với việc binary hóa ngay từ đầu. Trong nghiên cứu này:

- Tất cả trọng số (weights) được giới hạn ở giá trị -1 hoặc 1
- Các activation cũng chỉ nhận giá trị -1 hoặc 1
- Gradients được lan truyền ngược qua mô hình cũng ở dạng rời rạc

Mặc dù điều này có vẻ khó tin, nhưng các nhà nghiên cứu đã sử dụng các cơ chế cốt lõi trong huấn luyện mạng neural kết hợp với các phương pháp thống kê phức tạp để làm việc này khả thi với giá trị nhị phân.

Kết quả đạt được rất ấn tượng. Trên tập dữ liệu CIFAR-10, mô hình đạt tỷ lệ lỗi trên tập test là 10%, trong khi phương pháp state-of-the-art tại thời điểm đó đạt xấp xỉ 12%. Với cùng một kiến trúc mạng, việc binary hóa không chỉ cho phép thu nhỏ kích thước mô hình mà còn đạt được hiệu suất tương đương hoặc thậm chí tốt hơn các mô hình mạnh nhất thời bấy giờ.

Đây là bằng chứng cho thấy nếu thực hiện quantization trong quá trình huấn luyện, ta có thể đạt được hiệu suất tương đương trong khi giảm đáng kể kích thước mô hình - một phát hiện đáng ngạc nhiên tại thời điểm đó.

Layer-by-Layer Quantization - Aware Distillation (Yao et al. 2022)

Một công trình nghiên cứu thú vị gần đây liên quan đến việc lượng tử hóa mô hình. Thay vì lượng tử hóa toàn bộ mô hình ngay từ đầu, bạn có thể bắt đầu với mô hình có độ chính xác đầy đủ (full Precision) và sau đó huấn luyện từng lớp một để tái tạo lại đối tác của nó trong không gian độ chính xác đầy đủ. Cụ thể, bạn có thể chạy các đầu vào qua mô hình full Precision để lấy các đầu ra, ví dụ như xác suất của từng từ, sau đó điều chỉnh mô hình đã lượng tử hóa để tái tạo và đạt gần nhất với các trọng số đó.

Quá trình này được thực hiện từng lớp một. Ví dụ, sau khi hoàn thành lớp đầu tiên, bạn sẽ có các "logits" từ các trạng thái ẩn của lớp thứ hai từ cuối lên. Tiếp theo, bạn huấn luyện lớp đã lượng tử hóa để khớp với các trạng thái ẩn đó, và tiếp tục như vậy cho đến khi hoàn thành toàn bộ mô hình. Ý tưởng ở đây là bằng cách thực hiện "layer by layer distillation", bạn không chỉ tái tạo đầu ra, vốn có thể thưa thớt và khó tái tạo, mà còn tái tạo luồng dữ liệu qua từng bước của mô hình. Điều này giúp mô hình lượng tử hóa có thể hoạt động hiệu quả hơn mà không gặp phải các vấn đề khi huấn luyện từ đầu đến cuối.

Q-LORA (Dettmers et al. 2023)

Trong bài viết gần đây, tôi đã đề cập đến phương pháp "parameter efficient finetuning" để huấn luyện mô hình với độ lượng tử hóa cao, cụ thể là mô hình 4-bit. Phương pháp này đã áp dụng nhiều kỹ thuật tiên tiến khác và hiện tại "Q-LORA" đang rất phổ biến. Nếu bạn đang cân nhắc sử dụng phương pháp lượng tử hóa, đây có thể là lựa chọn phù hợp nhất hiện nay.

Pruning

Pruning vs Quantization

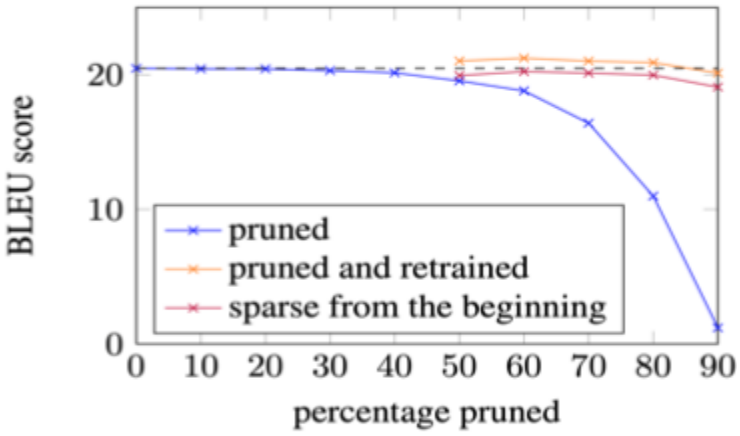
Trong quantization (lượng tử hóa), bạn thực hiện việc giảm độ chính xác của từng tham số trong mô hình, giống như việc "gọt giũa" từng tham số một cách tinh tế. Ngược lại, trong pruning, bạn hoàn toàn loại bỏ một số tham số nhất định, trong khi giữ nguyên các tham số còn lại. Điều này có nghĩa là một số tham số sẽ được đặt về 0, còn các tham số khác thì không thay đổi.

Magnitude Pruning (Han et al. 2015, See et al. 2016)

Một trong những cách trực quan nhất để giảm kích thước mô hình là Magnitude Pruning - phương pháp cắt tỉa tham số dựa trên độ lớn. Ý tưởng cốt lõi của phương pháp này là:

- Trong số lượng lớn tham số của mô hình, nhiều tham số có giá trị gần bằng 0 và gần như không ảnh hưởng đến kết quả
- Ta có thể đặt những tham số này chính xác bằng 0 và bỏ qua chúng trong quá trình tính toán
- Phương pháp này chọn một tỷ lệ phần trăm các tham số có độ lớn nhỏ nhất và đặt chúng về 0

Trong lĩnh vực machine translation, các nghiên cứu đã chỉ ra rằng có thể loại bỏ gần 50% tham số của mô hình mà hầu như không ảnh hưởng đến hiệu suất. Điều này củng cố quan điểm về over-parameterization: nhiều tham số chỉ cần thiết trong quá trình huấn luyện nhưng không thực sự quan trọng khi sử dụng mô hình.

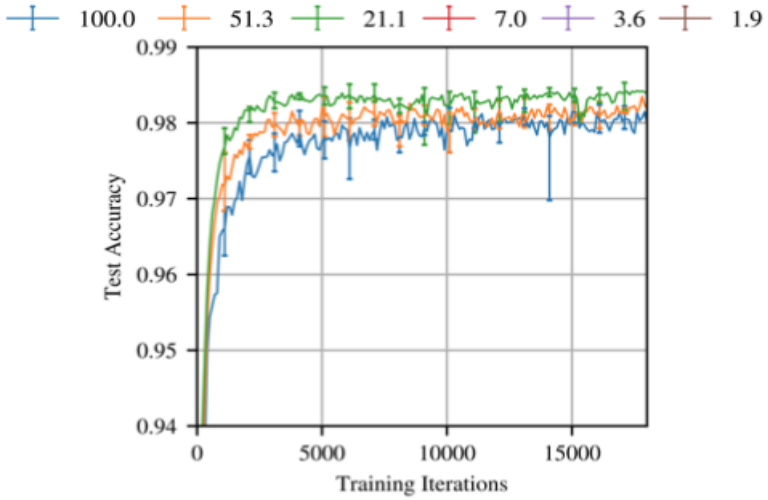


Đây được gọi là unstructured pruning vì việc loại bỏ tham số được thực hiện một cách ngẫu nhiên trong toàn bộ mô hình mà không tuân theo bất kỳ cấu trúc hay quy tắc nào cụ thể.

Lottery Ticket Hypothesis (Frankle et al. 2018)

Ý tưởng "lottery ticket hypothesis" cho rằng khi bạn huấn luyện một mô hình lớn, có những mạng con ngẫu nhiên trong mô hình đó có thể là khởi tạo tốt hơn so với mô hình ban đầu. Ví dụ, với một mô hình có 100 tỷ tham số, có thể tồn tại các mạng con chỉ với 1% kích thước (tức khoảng 1 tỷ tham số) nhưng lại hoạt động hiệu quả hơn mô hình đầy đủ.

Điều này liên quan đến kỹ thuật "pruning", nơi mà mô hình được cắt giảm kích thước và sau đó huấn luyện lại. Đáng ngạc nhiên là một mô hình chỉ còn 20% kích thước ban đầu, sau khi được pruning và huấn luyện lại, có thể hoạt động hiệu quả và tổng quát hóa tốt hơn so với mô hình gốc.



Ý tưởng chính là tìm ra những khởi tạo tốt cho các mạng con có thể mang lại hiệu quả cao hơn so với khởi tạo ngẫu nhiên của một mô hình lớn. Đây là một bước tiến xa hơn so với chỉ đơn thuần pruning, vì sau khi pruning, việc huấn luyện lại có thể cải thiện hiệu suất. Tuy nhiên, nhìn chung, pruning không phải là phương pháp để cải thiện hiệu suất mà là để duy trì hiệu suất trong khi cải thiện hiệu quả và giảm kích thước của mô hình.

Wanda (Sun et al. 2023)

Gần đây, nhóm nghiên cứu từ CMU đã công bố một phương pháp pruning mới có tên WANDA, chỉ ra hạn chế của phương pháp magnitude pruning truyền thống. Hãy xem xét một ví dụ đơn giản để hiểu vấn đề. Giả sử có một mô hình 2 tham số:

- Output = A*X + B*Y
- Trong đó A, B là trọng số (weights), X, Y là đầu vào
- Magnitude của A lớn hơn nhiều so với B

Theo magnitude pruning truyền thống:

- Ta sẽ đặt B = 0 vì nó có độ lớn nhỏ hơn
- Mô hình trở thành: Output = A*X
- Lý do: tham số có độ lớn cao hơn được cho là có ảnh hưởng lớn hơn đến output

Tuy nhiên, điều này có thể sai khi:

- Giá trị trung bình của X là 1
- Giá trị trung bình của Y là 1000

Trong trường hợp này, mặc dù B nhỏ hơn nhưng nó xử lý đầu vào có giá trị lớn hơn nhiều, do đó tác động thực tế của nó đến output có thể quan trọng hơn A.

WANDA cải tiến bằng cách:

- Xem xét cả độ lớn của tham số và độ lớn của đầu vào vào mỗi lớp
- Sử dụng dữ liệu hiệu chuẩn để học được giá trị trung bình của các đầu vào
- Kết hợp hai thông tin này để quyết định tham số nào nên được giữ lại hoặc loại bỏ

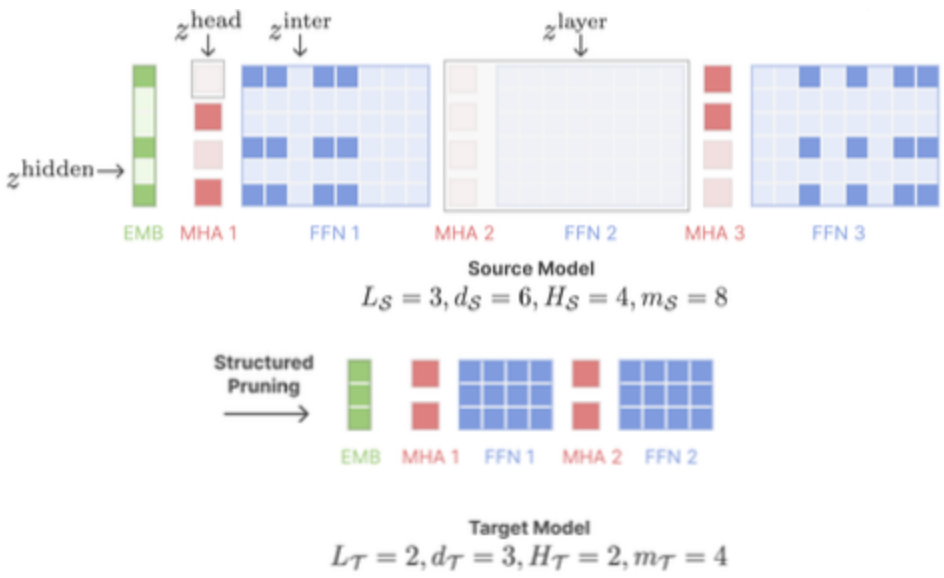
Problems with Unstructured Pruning

Mặc dù “unstructured pruning” có thể làm cho mô hình trở nên thưa thớt (sparse) bằng cách loại bỏ một số tham số, nhưng nếu phần cứng không tận dụng được tính thưa thớt này, thì hiệu suất cũng không được cải thiện.

Ví dụ, khi chúng ta "tắt" một nửa số tham số, nhưng vẫn thực hiện các phép nhân với số không trong một phép toán dày đặc (dense operation), thì khối lượng công việc thực hiện vẫn không thay đổi và không mang lại lợi ích nào. Thực tế hiện nay là phần cứng cho học máy chưa hỗ trợ tốt cho các cấu trúc dữ liệu thưa thớt hoặc các phép tính toán thưa thớt. Các phép nhân ma trận thường thực hiện những thao tác phức tạp bên trong, và do đó, không hoạt động hiệu quả với các cấu trúc dữ liệu thưa thớt.

Structured Pruning (Xia et al. 2022)

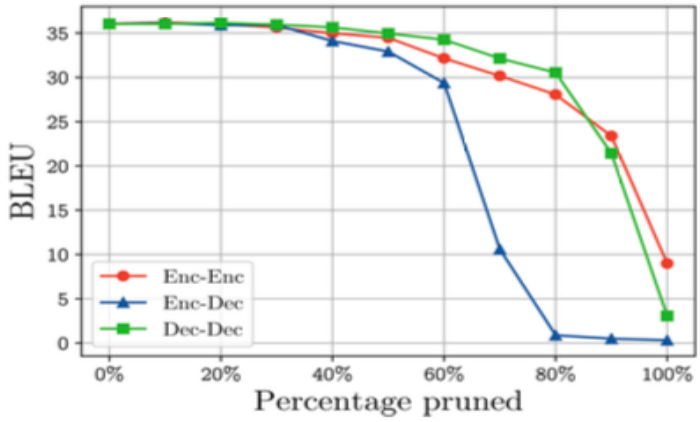
Thay vì chọn ngẫu nhiên các tham số trên toàn bộ mô hình, phương pháp “structure pruning” loại bỏ toàn bộ thành phần hoặc lớp của mô hình (remaining components aren’t pruned). Điều này giúp việc cắt tỉa mô hình trở nên có cấu trúc hơn và thực sự tạo ra sự khác biệt đáng kể về thời gian chạy của mô hình.



Are Sixteen Heads Really Better than One? (Michel and Neubig 2019)

Một nghiên cứu thú vị từ Graham và học viên của ông về pruning cấu trúc trong các mô hình Transformer như BERT đã chỉ ra những phát hiện quan trọng:

- Mô hình Transformer thường có nhiều attention heads (ví dụ như Llama có 8 heads)
- Trong thực tế, phần lớn các attention heads này có thể được loại bỏ mà không ảnh hưởng đáng kể đến hiệu suất của mô hình
- Với mô hình Machine Translation (MT), có thể loại bỏ 50% attention heads mà chỉ tạo ra tác động không đáng kể đến hiệu suất



Điểm khác biệt so với magnitude pruning truyền thống:

- Thay vì loại bỏ các tham số riêng lẻ dựa trên độ lớn của chúng
- Phương pháp này loại bỏ toàn bộ attention heads
- Việc loại bỏ được thực hiện bất kể các heads đó có chứa weights với magnitude lớn hay nhỏ

- Cách tiếp cận này mang lại tác động trực tiếp đến hiệu suất của mô hình

Coarse-to-Fine Structured Pruning (Xia et al. 2022)

Một nghiên cứu cách đây 2 năm đã đề xuất phương pháp pruning sử dụng hai cấp độ mask khác nhau. Cấp độ đầu tiên là "Coarse masks", cho phép tắt các thành phần lớn như toàn bộ self-attention layers hoặc toàn bộ feed-forward layers bằng cách thay thế chúng bằng ma trận đơn vị.

Cấp độ thứ hai là "fine masks", cho phép kiểm soát chi tiết hơn như tắt các attention heads riêng lẻ hoặc loại bỏ các chiều riêng lẻ. Ví dụ, thay đổi trạng thái ẩn (hidden states) từ 512 chiều xuống còn 200 chiều. Ý tưởng ở đây là cung cấp hai mức độ chi tiết khác nhau để tắt các thành phần khác nhau của mô hình.

Các mặt nạ này được học thông qua một tập dữ liệu xác thực (validation data) để xác định những gì có thể tắt mà không làm giảm hiệu suất của mô hình. Kết quả từ bài báo cho thấy rằng phương pháp này có thể đạt được hiệu quả đáng kể mà không làm suy giảm hiệu suất của mô hình.

Pruning w/ Forward Passes (Dery et al. 2024)

Khi nói về pruning, một trong những thách thức lớn nhất là vấn đề về tài nguyên máy tính. Quá trình này đòi hỏi việc học các mask để kiểm soát mô hình, điều này tiêu tốn rất nhiều bộ nhớ GPU. Ví dụ, để thực hiện pruning trên một mô hình LLaMA 70 tỷ tham số, bạn sẽ cần lượng tài nguyên máy tính tương đương với việc huấn luyện mô hình ban đầu.

Gần đây, Graham và các nghiên cứu sinh của ông đã đề xuất một phương pháp pruning mới không đòi hỏi việc tính toán gradient. Ý tưởng chính của phương pháp này bao gồm:

1. Tạo ngẫu nhiên khoảng 100-1000 biến thể của mô hình bằng cách mask các module khác nhau
2. Đo lường hiệu suất của các mô hình đã được điều chỉnh này
3. Thực hiện hồi quy để xác định mức độ ảnh hưởng của từng module đến hiệu suất tổng thể
4. Sử dụng trọng số hồi quy để xác định các module có thể tắt mà không ảnh hưởng nhiều đến hiệu suất

Sự khác biệt chính giữa pruning và dropout là:

- Pruning: Thực hiện một lần và sau đó huấn luyện mạng đã được pruning cố định. Mục đích nhằm giảm kích thước mô hình.
- Dropout: Thực hiện pruning ngẫu nhiên tại mỗi bước cập nhật. Mục đích để regularization, tránh overfitting.

So sánh Structured và Unstructured Pruning:

- Wanda (unstructured pruning): Dù đạt được cùng số lượng tham số như structured pruning. Có thể dẫn đến tốc độ chậm hơn (negative speedup)
- Bonsai (structured pruning): Với cùng số lượng tham số. Có thể tăng tốc độ lên đến 50%

Distillation

Distillation vs Quantization vs Pruning

Distillation là quá trình huấn luyện một mô hình để tái tạo hành vi của một mô hình khác. Trong distillation, bạn có thể thay đổi mọi tham số trong mô hình và thậm chí sử dụng một kiến trúc hoàn toàn khác.

Ngược lại, trong quantization, bạn không thay đổi tham số mà chỉ điều chỉnh độ chính xác của chúng. Còn trong pruning, một tập hợp tham số được giữ cố định. Distillation cho phép thay đổi toàn bộ tham số nhưng theo cách tối ưu để giảm thiểu số lượng tham số cần thiết.

Weak Supervision (Yarowski 1995)

Distillation có liên quan mật thiết đến một khái niệm cổ điển trong học máy gọi là weak supervision (giám sát yếu). Đây là phương pháp sử dụng dữ liệu chưa được gán nhãn (văn bản, hình ảnh hoặc bất kỳ loại dữ liệu nào) để tạo ra các pseudo-label (nhãn giả) thay vì nhãn được gán bởi con người. Các pseudo-label này sau đó được sử dụng để huấn luyện mô hình và có thể đạt được hiệu suất khá tốt.

Một trong những ví dụ nổi tiếng là self-training - phương pháp hoạt động theo các bước sau:

1. Khởi tạo mô hình với một số lượng rất nhỏ điểm dữ liệu (khoảng 3-5 mẫu)
2. Huấn luyện bộ phân loại trên tập dữ liệu nhỏ này (kết quả ban đầu sẽ kém do thiếu dữ liệu)
3. Sử dụng mô hình để dự đoán nhãn cho dữ liệu chưa gán nhãn
4. Dùng các pseudo-label này để cập nhật mô hình
5. Lặp lại quá trình trên

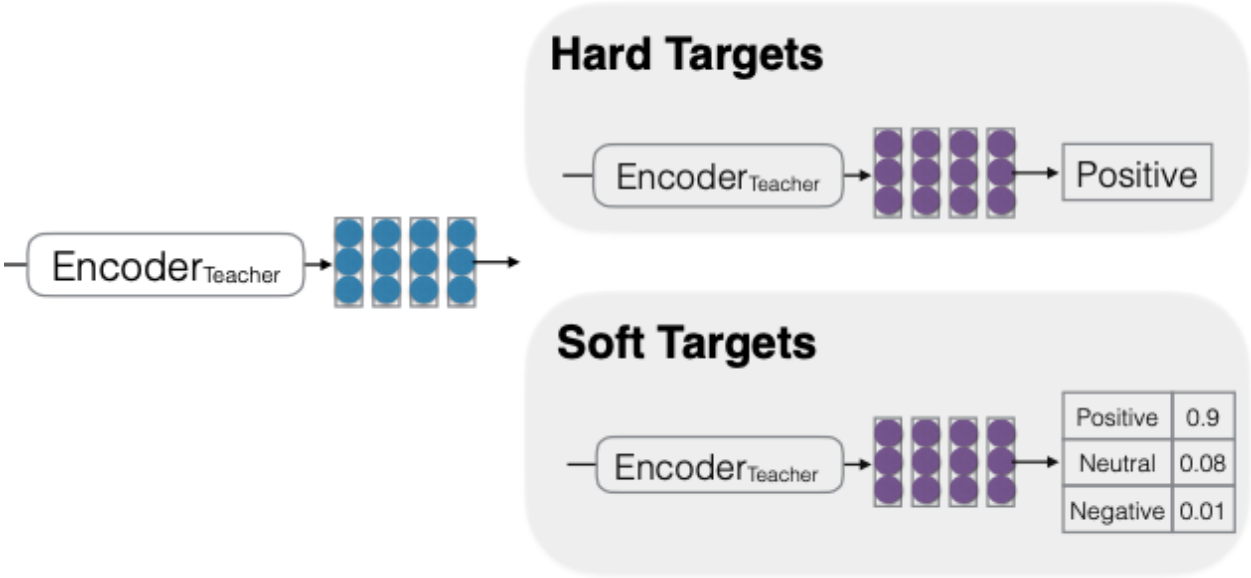
Pseudo-label còn được ứng dụng trong trường hợp không thể gán nhãn cho hàng nghìn mẫu, thay vào đó ta có thể viết các quy tắc cơ bản. Ví dụ với bài toán phân tích cảm xúc đánh giá phim:

- Nếu review có từ "awesome" → nhãn tích cực
- Nếu review có "I hated it" → nhãn tiêu cực

Mặc dù quy tắc này khá đơn giản và dễ bị lỗi, nhưng khi có đủ số lượng pseudo-label từ nhiều quy tắc khác nhau, ta có thể huấn luyện được một mô hình với từ vựng đầy đủ và đạt hiệu quả tốt. Ý tưởng này đã được startup Snorkel phát triển thành sản phẩm và công bố nhiều nghiên cứu liên quan. Weak supervision chính là nền tảng cho kỹ thuật knowledge distillation.

Hard vs. Soft Targets (Hinton et al 2015)

Knowledge distillation là quá trình huấn luyện một mô hình nhỏ (student model) để mô phỏng các dự đoán của một mô hình lớn (teacher model). Mô hình lớn sẽ tạo ra pseudo-label trên dữ liệu chưa gán nhãn, và đây chính là mục tiêu cho mô hình nhỏ hơn học theo. Một điểm quan trọng cần lưu ý là ta cần có dữ liệu chưa gán nhãn phù hợp với input mong đợi. Ví dụ, với bài toán phân loại đánh giá phim, ta cần thu thập hàng nghìn review phim tương tự với dữ liệu mà mô hình sẽ xử lý trong thực tế.



Có hai phương pháp chính để huấn luyện knowledge distillation:

1. Hard targets: Đây là phương pháp đơn giản và trực quan nhất
- Lấy dữ liệu chưa gán nhãn
- Sử dụng teacher model để dự đoán nhãn
- Dùng dự đoán này làm mục tiêu cho student model.
- Ví dụ: Nếu LLaMA 70B dự đoán nhãn "positive" cho một review, ta sẽ dùng nhãn "positive" này làm mục tiêu.

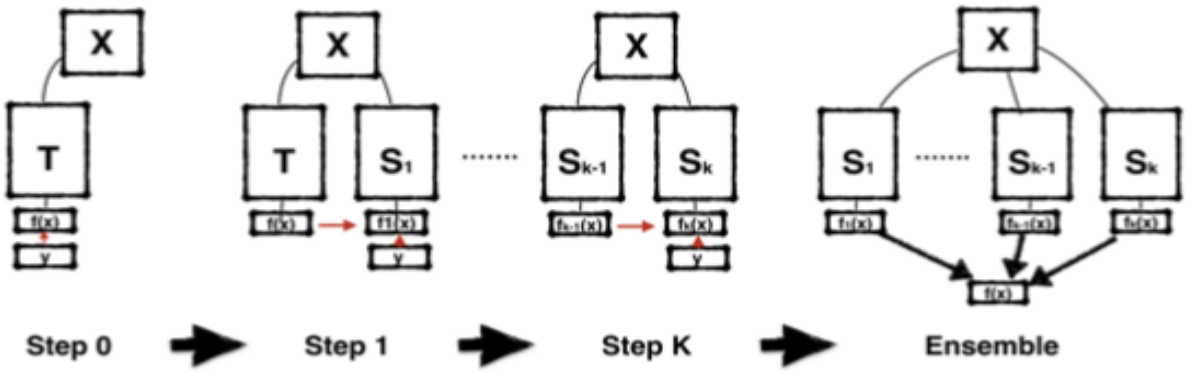
2. Soft target distillation: Phương pháp này thường hiệu quả hơn
- Thay vì chỉ học theo nhãn dự đoán, student model cố gắng tạo ra phân phối xác suất trên tất cả các nhãn giống với phân phối của teacher model
- Ví dụ: LLaMA 70B dự đoán xác suất cho 3 nhãn khác nhau, student model sẽ học để tạo ra phân phối xác suất tương tự

Một ưu điểm của soft target là nó cung cấp thông tin phong phú hơn so với học có giám sát truyền thống. Trong khi người gán nhãn thường chỉ đưa ra một câu trả lời duy nhất, mạng nơ-ron teacher có thể cung cấp thông tin về mức độ tin cậy của các nhãn khác nhau. Việc tối ưu hóa cũng thay đổi: thay vì tối ưu xác suất của nhãn đúng, ta tối ưu sự khác biệt giữa các phân phối xác suất.

Nghiên cứu của Jeff Hinton trong lĩnh vực nhận dạng giọng nói đã chứng minh: khi có ít dữ liệu chưa gán nhãn, phương pháp soft target hiệu quả hơn đáng kể so với hard target, ngay cả khi hard target được huấn luyện trên toàn bộ tập dữ liệu.

Born Again Neural Networks (Furlanello, Lipton, et al 2018)

Một kết quả nghiên cứu (Furlanello, Lipton, et al 2018) đã chỉ ra một mô hình được huấn luyện bằng học có giám sát (ví dụ cho bài toán phân loại ảnh) có thể cải thiện hiệu suất bằng cách liên tục thực hiện self-distillation với soft targets. Cụ thể:



- Mô hình dự đoán phân phối xác suất trên các nhãn cho một tập ảnh
- Sau đó tự huấn luyện lại để khớp với phân phối xác suất này
- Quá trình này được lặp lại nhiều lần

Trực quan cho hiện tượng này là: mục tiêu soft target khác với mục tiêu trong học có giám sát thông thường. Nó mang nhiều thông tin hơn, bao gồm cả độ không chắc chắn về nhãn, tạo ra một "giao diện tri thức" (knowledge interface) phong phú hơn giữa teacher và student so với việc chỉ cung cấp một câu trả lời duy nhất.

Có sự tương đồng giữa phương pháp này với kỹ thuật Dropout. Trong khi Dropout tạo ra các phiên bản nhiễu của mô hình bằng cách tắt ngẫu nhiên các nơ-ron, phương pháp self-distillation tạo ra các phiên bản nhiễu thông qua việc distill từ phiên bản trước đó. Cả hai đều có thể xem như tập hợp (ensemble) các biến thể ngẫu nhiên của cùng một mô hình.

Về vấn đề chất lượng nhãn, một nghiên cứu gần đây về phân loại ảnh đã chỉ ra điều đáng ngạc nhiên: mô hình vẫn có thể học được thông tin hữu ích ngay cả khi 99% nhãn trong tập huấn luyện bị đổi ngẫu nhiên sang nhãn sai. Hiện tượng này chỉ xảy ra với các mạng neural sâu và không qua pre-training. Điều này cho thấy mạng neural có khả năng chống chịu rất tốt với nhiễu nhãn.

Tuy nhiên, một điểm quan trọng cần lưu ý là nhiễu nhãn trong nghiên cứu trên là ngẫu nhiên đồng đều (uniform noise). Nếu sử dụng nhiễu có định hướng (biased noise), mạng neural có

thể bị ảnh hưởng nghiêm trọng trong việc học ánh xạ từ input sang nhãn. Trong lĩnh vực thị giác máy tính, các nghiên cứu về nhiễu nhãn thường xem xét đến mức 90% nhiễu, nhưng đều là nhiễu đồng đều để đảm bảo mô hình vẫn hội tụ về hướng đúng sau đủ số vòng lặp.

Sequence-Level Distillation (Kim and Rush 2016)

Ban đầu, kỹ thuật distillation được thiết kế cho bài toán gán một nhãn cho mỗi input. Tuy nhiên, trong xử lý ngôn ngữ tự nhiên, ta thường làm việc với chuỗi và cần sinh ra các câu. Vậy làm thế nào để mở rộng distillation cho bài toán sinh chuỗi? Có hai cách tiếp cận chính:

1. Word-level distillation:
- Tại mỗi bước trong quá trình sinh từ, student model cố gắng khớp với phân phối xác suất các từ được đề xuất bởi teacher model
- Ví dụ: Cho prefix "this movie is [blank]", student model sẽ học phân phối xác suất của các từ tiếp theo từ teacher model
- Tuy nhiên, cách này gặp vấn đề "exposure bias": khi sinh text, teacher và student có thể diverge mạnh - nghĩa là teacher có thể sinh ra văn bản nhất quán nhưng rất khác với khả năng sinh của student model.

2. Sequence-level distillation:
- Thay vì sử dụng soft targets ở mức từ, ta để teacher model sinh ra câu hoàn chỉnh
- Student model sẽ tối đa hóa xác suất của toàn bộ câu được sinh ra này

$$\mathcal{L}_{\text{WORD-KD}} = -\sum_{j=1}^{J}\sum_{k=1}^{|\mathcal{V}|} q(t_j = k \mid \mathbf{s}, \mathbf{t}_{<j}) \times \log p(t_j = k \mid \mathbf{s}, \mathbf{t}_{<j})$$

$$\begin{aligned}\mathcal{L}_{\text{SEQ-KD}} &\approx -\sum_{\mathbf{t}\in\mathcal{T}} \mathbb{1}\{\mathbf{t} = \hat{\mathbf{y}}\} \log p(\mathbf{t} \mid \mathbf{s}) \\ &= -\log p(\mathbf{t} = \hat{\mathbf{y}} \mid \mathbf{s})\end{aligned}$$

$$\mathcal{L} = (1-\alpha)\mathcal{L}_{\text{SEQ-NLL}} + \alpha\mathcal{L}_{\text{SEQ-KD}}$$

Nghiên cứu chỉ ra rằng kết hợp cả hai mục tiêu trên - word-level và sequence-level distillation - mang lại hiệu quả cao nhất. Đây được xem là phương pháp phù hợp để thực hiện distillation cho bài toán sinh chuỗi văn bản.

DistilBERT (Sanh et al 2019)

DistilBERT là một trong những mô hình distilled phổ biến nhất trong NLP. Mô hình này được phát triển từ BERT. Mục tiêu của DistilBERT là giảm kích thước BERT xuống một nửa nhưng vẫn duy trì hiệu năng tương đương. Để đạt được điều này, nhóm nghiên cứu đã sử dụng một số kỹ thuật:

1. Layer reduction:
- Chỉ giữ lại một nửa số layer của BERT (ví dụ: từ 12 layer xuống còn 6 layer)
- Các layer được khởi tạo từ các layer tương ứng của BERT gốc thay vì khởi tạo ngẫu nhiên

2. Soft target distillation:
- Áp dụng phương pháp soft target distillation hiệu quả
- Kết hợp với mục tiêu học có giám sát từ language modeling
- Cụ thể, họ mask một số token trong văn bản và huấn luyện mô hình dự đoán:
- Token thực sự bị ẩn đi (supervised objective)
- Token mà teacher model dự đoán cho vị trí đó (distillation objective)
- Bất ngờ là mục tiêu supervised không cải thiện nhiều kết quả, cho thấy một teacher model tốt là đủ cho quá trình distillation

3. Embedding space alignment:
- Thêm các kỹ thuật đảm bảo không gian embedding của mô hình nhỏ và mô hình lớn có cấu trúc hình học tương tự nhau

Kết quả chính là DistilBERT đạt được hiệu năng gần như tương đương với BERT gốc trên hầu hết các tác vụ, trong khi chỉ có một nửa kích thước.

Model	IMDb (acc.)	SQuAD (EM/F1)
BERT-base	93.46	81.2/88.5
DistilBERT	92.82	77.7/85.8
DistilBERT (D)	-	79.1/86.9

Model	# param. (Millions)	Inf. time (seconds)
ELMo	180	895
BERT-base	110	668
DistilBERT	66	410

Khi định nghĩa kiến trúc cho mô hình student từ mô hình teacher model, bạn có rất nhiều sự linh hoạt trong distillation, khác với pruning. Nghiên cứu cho thấy distillation hiệu quả nhất khi mô hình student và teacher có kiến trúc tương tự nhau. Ví dụ, nếu mô hình teacher là một mô hình tự hồi quy (auto regressive) như GPT-2 hoặc GPT-3, bạn có thể muốn mô hình student cũng là tự hồi quy.

Tuy nhiên, trực giác của tôi cho thấy có rất nhiều sự linh hoạt ở đây, đặc biệt khi bạn thực hiện distillation với mục tiêu cứng (Hard Target distillation) bằng cách tạo ra các chuỗi và coi chúng như nhãn để huấn luyện bất kỳ mô hình student nào. Vì vậy, tôi không nghĩ rằng có nhiều ràng buộc trong việc này.

Self-Instruct (Wang et al 2022)

Trong distillation LMs, một khái niệm thú vị là "self-instruct", đã được đề cập trước đó. Ý tưởng cơ bản là sử dụng một mô hình để tự tạo dữ liệu và sau đó huấn luyện lại chính mô hình đó trên dữ liệu tự tạo ra. Cụ thể, họ sử dụng một mô hình ngôn ngữ cơ bản, chỉ được huấn luyện để tạo văn bản, và cố gắng dạy nó theo các chỉ dẫn thông qua việc tinh chỉnh hướng dẫn (instruction tuning).

Quá trình này bắt đầu bằng việc để mô hình ngôn ngữ cơ bản tự tạo ra các prompt, ví dụ như "write a poem about dogs", sau đó tạo ra các phản hồi cho những prompt đó, như một bài thơ về chó. Mô hình sau đó được huấn luyện để bắt chước hành vi của chính nó. Một trong những

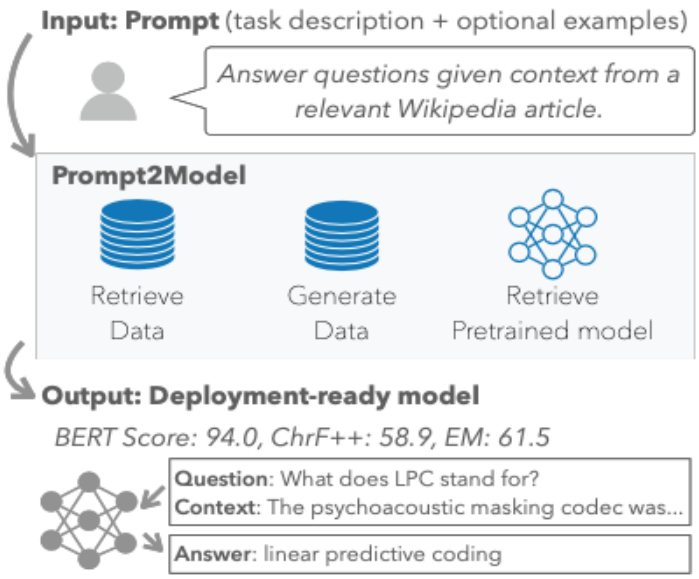
thủ thuật quan trọng là khi tạo dữ liệu, thay vì tạo đầu vào trước rồi gán nhãn đầu ra, có thể tạo nhãn trước rồi tạo đầu vào dựa trên nhãn đó. Điều này giúp cải thiện chất lượng dữ liệu tổng hợp.

Khái niệm "task asymmetry" cũng được đề cập, nơi mà nếu một nhiệm vụ từ X đến Y khó nhưng từ Y đến X dễ, có thể bắt đầu với Y, tạo ra X tổng hợp, sau đó huấn luyện mô hình để chuyển từ X sang Y. Ví dụ, trong trích xuất thông tin, thay vì chuyển từ câu sang thông tin, có thể tạo câu từ thông tin và sau đó đảo ngược quá trình. Kết quả cho thấy hiệu suất cải thiện đáng kể, gấp đôi so với mô hình tốt nhất trước đó.

Prompt2Model (Viswanathan et al 2023)

Prompt2Model là một hướng tiếp cận mới, phát triển từ ý tưởng sử dụng distillation như một công cụ tạo dữ liệu. Thay vì chỉ dựa vào dữ liệu được tạo ra từ mô hình, phương pháp này kết hợp cả dữ liệu có sẵn từ internet và dữ liệu được sinh ra từ các mô hình ngôn ngữ lớn.

Quy trình bắt đầu bằng việc người dùng chỉ định nhiệm vụ của họ thông qua một "prompt", tương tự như cách sử dụng GPT-3. Người dùng có thể cung cấp một vài ví dụ nếu muốn. Dựa trên "prompt" này, chúng tôi truy xuất các tập dữ liệu hiện có có thể liên quan. Ví dụ, nếu người dùng yêu cầu "answer biomedical questions about cancer for doctors", hệ thống có thể tìm thấy tập dữ liệu như BioASQ.



Tuy nhiên, tập dữ liệu truy xuất có thể không hoàn toàn phù hợp với nhiệm vụ cụ thể của người dùng. Do đó, chúng tôi bổ sung tập dữ liệu này bằng dữ liệu được tạo ra bởi mô hình ngôn ngữ, có thể không chất lượng cao nhưng phù hợp hơn với ý định của người dùng.

Chúng ta cũng có thể sử dụng một mô hình đã được huấn luyện trước trong lĩnh vực liên quan để cải thiện kết quả. Sau đó, tất cả các nguồn dữ liệu này được kết hợp lại và tinh chỉnh trên

một mô hình nhỏ hơn. Kết quả thú vị là các mô hình nhỏ này thường vượt trội hơn GPT-3, mặc dù GPT-3 là mô hình được sử dụng để tạo dữ liệu.

Phương pháp này không chỉ tận dụng "distillation" mà còn khai thác các tập dữ liệu có sẵn trên internet, mở ra một hướng đi mới đầy hứa hẹn trong việc phát triển khả năng của mô hình ngôn ngữ. Hiện nay, "distillation" thường được gọi dưới tên khác là "synthetic data generation", và đây là một trong những chủ đề nghiên cứu nóng nhất trong NLP.

A Toolkit for Synthetic Data Generation (Patel et al 2024)

Gần đây, tôi đã thấy một tài liệu trên internet giới thiệu một bộ công cụ tương tự như PyTorch dành cho việc thực hiện distillation. Bộ công cụ này định nghĩa các thao tác cơ bản như tạo dữ liệu từ prompt hoặc từ rag, thực hiện truy xuất, lọc và xếp hạng các ví dụ, hoặc đánh giá đầu vào bằng cách sử dụng một mô hình ngôn ngữ lớn khác. Họ cũng tích hợp việc huấn luyện mô hình vào quy trình này. Tôi cho rằng đây là một hướng đi rất thú vị trong việc biến quá trình tạo dữ liệu thành một vấn đề kỹ thuật thực sự có thể quản lý và phát triển.

Type		Examples
Steps	Load a Dataset	`DataSource`, `HFHubDataSource`, `JSONDataSource`, `CSVDataSource`, ...
	Prompting	`Prompt`, `RAGPrompt`, `ProcessWithPrompt`, `FewShotPrompt`, `DataFromPrompt`, `DataFromAttributedPrompt`, `FilterWithPrompt`, `RankWithPrompt`, `JudgeGenerationPairsWithPrompt`, ...
	Other	`Embed`, `Retrieve`, `CosineSimilarity`, ...
Models		`OpenAI`, `OpenAIAssistant`, `HFTransformers`, `CTransformers`, `VLLM`, `Petals`, `HFAPIEndpoint`, `Together`, `MistralAI`, `Anthropic`, `Cohere`, `AI21`, `Bedrock`, `Vertex`, ...
Trainers		`TrainOpenAIFineTune`, `TrainHFClassifier`, `TrainHFFineTune`, `TrainSentenceTransformer`, `TrainHFDPO`, `TrainHFPPO`, ...

Resources

1. https://phontron.com/class/anlp2024/lectures/#distillation-and-quantization-feb-20
2. Recommended Reading: Theia Vogel's blog on "How to make LLMs go fast"
3. Recommended Reading: Lilian Weng's blog on "Inference Optimization"
4. Reference: Over-parametrization is provably useful in training neural nets (Du and Lee 2018)
5. Reference: Model-Aware Quantization: GOBO (Zadeh et al. 2020)
6. Reference: Model-Aware Quantization: LLM.int8 (Dettmers et al. 2022)
7. Software: Binarized Neural Networks (Courbariaux et al. 2016)
8. Reference: Layer-by-Layer Quantization-Aware Distillation (Yao et al. 2020)
9. Reference: QLoRA (Dettmers et al. 2023)
10. Reference: Magnitude pruning (in general) (Han et al. 2015)
11. Reference: An analysis of magnitude pruning for machine translation) (See et al. 2016)
12. Reference: The Lottery Ticket Hypothesis) (Frankle et al. 2018)
13. Reference: Wanda (Pruning by Weights and Activations) (Frankle et al. 2018)

14. Reference: Coarse-to-fine Structured Pruning (Xia et al. 2022)
15. Reference: Are Sixteen Heads Really Better than One? (Michel and Neubig 2019)
16. Reference: Pruning with Forward Passes (Dery et al 2024)
17. Reference: Self-Training (Yarowski 1995)
18. Reference: Hard vs Soft Target Distillation (Hinton et al 2015)
19. Reference: Sequence-Level Distillation (Kim and Rush 2016)
20. Reference: DistilBERT (Sanh et al 2019)
21. Reference: Deep Learning is Robust to Massive Label Noise (Furlanello et al 2018)
22. Reference: Born Again Neural Networks (Rolnick et al 2018)
23. Reference: Self-Instruct (Wang et al 2022)
24. Reference: Prompt2Model (Viswanathan et al 2023)
25. Reference: SynthIE (Exploiting Asymmetry for Synthetic Training Data Generation) (Josifoski et al 2023)
26. Reference: DataDreamer: A Toolkit for Synthetic Data Generation (Patel et al 2024)