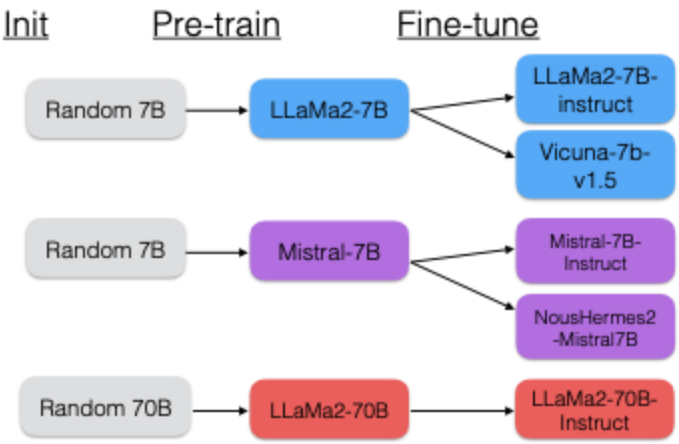
Lecture 14: Ensembling and Mixture of Experts

- Giới thiệu
- Model Ensembling
  - Ensembling
  - Linear Interpolation
  - Log-linear Interpolation
  - Linear or Log Linear
  - Using Models As Negative Evidence
- Efficient Methods for Using Multiple Models
  - Problem with Ensembling Cost
  - Parameter Averaging (e.g. Utans 1996)
  - Model Soups (Wortsman et al. 2022)
  - Task Vectors
  - Meta Learning
  - Software: mergekit
  - Ensemble Distillation (e.g. Kim et al. 2016)
- Sparse Mixture of Experts Models
  - Sparse Computation
  - GPU-level Sparsity
  - Sparsely Gated Mixture of Experts Layer (Shazeer+ 2017)
- Pipeline Systems
  - Cascaded Systems
  - Stacking
  - Iterative Refinement
  - Discussion questions
- Resources

Giới thiệu

Trong bài viết này, chúng ta sẽ thảo luận về việc kết hợp nhiều mô hình để cải thiện độ chính xác. Đây là một phương pháp đáng tin cậy để nâng cao hiệu suất của mô hình. Có nhiều lý do khiến chúng ta có nhiều mô hình khác nhau. Thứ nhất, có thể có các kiến trúc mô hình khác nhau. Thứ hai, có thể có các khởi tạo khác nhau cho cùng một kiến trúc mô hình.



Ví dụ, khi khởi tạo một kiến trúc như LLaMA, chúng ta bắt đầu với 7 tỷ tham số ngẫu nhiên, sau đó huấn luyện để có LLaMA 7B cho giai đoạn tiền huấn luyện. Tương tự, có thể khởi tạo một mô hình khác, có thể cùng hoặc khác kiến trúc, và huấn luyện trên cùng hoặc khác dữ liệu để có một mô hình như Mistral 7B. Chúng ta cũng có thể mở rộng hoặc thu nhỏ mô hình, ví dụ như LLaMA2 70B.

Sau khi có mô hình tiền huấn luyện, có nhiều chiến lược tinh chỉnh khác nhau, như LLaMA2-7B Instruct, Vicuna 7B v1.5, …. Do đó, chúng ta có một cây phân nhánh gồm nhiều kiến trúc, khởi tạo ngẫu nhiên, mô hình tiền huấn luyện và mô hình tinh chỉnh.

Khi kết hợp nhiều mô hình, một số phương pháp chỉ áp dụng cho các mô hình hoàn toàn khác nhau, một số chỉ áp dụng cho các mô hình có cùng kiến trúc, và một số chỉ áp dụng cho các mô hình có cùng khởi tạo và quá trình huấn luyện. Tôi sẽ cố gắng phân biệt các phương pháp này trong các phần tiếp theo.

Model Ensembling

Ensembling

Ensembling là một kỹ thuật trong học máy, cho phép kết hợp dự đoán từ nhiều mô hình khác nhau. Mặc dù có những nhược điểm nhất định, ensembling thường mang lại lợi ích đáng kể.

Tại sao nên sử dụng Ensembling?

- Giảm thiểu độ lệch (bias): Khi sử dụng một mô hình duy nhất, kết quả có thể bị ảnh hưởng bởi độ lệch của mô hình đó. Kết hợp nhiều mô hình giúp giảm thiểu vấn đề này.
- Quan điểm Bayes: Ensembling có thể được xem như một cách tiếp cận theo quan điểm Bayes, giúp cải thiện độ chính xác của dự đoán.
- Khả năng chuyên biệt của mô hình: Mỗi mô hình có thể giỏi ở một số khía cạnh và kém ở những khía cạnh khác. Bằng cách kết hợp, chúng ta có thể tận dụng điểm mạnh của từng mô hình.

Ý tưởng cơ bản của ensembling là các lỗi của các mô hình thường không nhất quán. Ví dụ, nếu có hai mô hình với các xác suất dự đoán khác nhau cho một chuỗi hành động như "chó sủa, chạy, và nhảy", việc trung bình hóa các xác suất này thường dẫn đến kết quả chính xác hơn. Khi trung bình hóa, xác suất tổng hợp thường chính xác hơn do các lỗi ít tương quan.

Lợi ích của Ensembling

- Ensembling giúp làm mượt các đặc điểm riêng biệt của từng mô hình. Ngay cả khi kết hợp các mô hình từ các checkpoints khác nhau, kỹ thuật này vẫn cải thiện độ chính xác. Điều này là do mỗi mô hình có thể đã được huấn luyện với dữ liệu khác nhau gần đây, dẫn đến sự thiên vị nhất định.
- Ensembling là một trong những phương pháp hiếm hoi gần như luôn cải thiện độ chính xác. Trong thực tế, rất hiếm khi việc kết hợp hai mô hình không mang lại sự cải thiện nào về độ chính xác.

Linear Interpolation

Việc kết hợp các mô hình khác nhau có thể mang lại hiệu quả cao trong các tình huống khác nhau. Một phương pháp phổ biến là nội suy tuyến tính, trong đó ta tính trung bình có trọng số của xác suất từ các mô hình. Về mặt toán học, xác suất của một token tiếp theo theo mô hình M được tính bằng cách lấy xác suất của mô hình đó và nhân với trọng số của mô hình.

$$P(y_j \mid X, y_1, \ldots, y_{j-1}) = \sum_{m=1}^{M} \underbrace{P_m(y_j \mid X, y_1, \ldots, y_{j-1})}_{\text{Probability according to model } m} \underbrace{P(m \mid X, y_1, \ldots, y_{j-1})}_{\text{Probability of model } m}$$

Trọng số này có thể được định nghĩa là hằng số hoặc được mô hình hóa một cách động. Trong trường hợp đơn giản, các trọng số hỗn hợp được định nghĩa sao cho tổng của chúng bằng 1 và mỗi trọng số nằm trong khoảng từ 0 đến 1.

Ngoài ra, có thể học trọng số này một cách động dựa trên tình huống cụ thể. Ví dụ, nếu bạn tin rằng mô hình M sẽ dự đoán tốt trong một tình huống cụ thể, bạn có thể đặt phần lớn xác suất vào mô hình đó.

Phương pháp này thực tế và dễ thực hiện vì bạn có thể tính xác suất theo từng mô hình tại mỗi bước thời gian mà không cần tải toàn bộ các mô hình vào bộ nhớ trong quá trình huấn luyện. Điều này giúp tối ưu hóa tài nguyên và tăng hiệu quả của quá trình xử lý.

Log-linear Interpolation

Nội suy tuyến tính là việc kết hợp tuyến tính các xác suất của từng mô hình. Trong khi đó, nội suy log tuyến tính kết hợp các log xác suất của từng mô hình và sau đó chuẩn hóa lại để đảm bảo đầu ra là một xác suất thực sự.

Để thực hiện điều này, chúng ta sử dụng một hệ số nội suy (interpolation coefficient), tương tự như trước đây, nhưng lần này là kết hợp các log xác suất. Ở đây, chúng ta cần sử dụng hàm softmax. Có thể bạn thắc mắc tại sao trong một số trường hợp không cần dùng softmax. Đó là vì xác suất đã được đảm bảo nằm trong khoảng từ 0 đến 1 và tổng bằng 1.

Tuy nhiên, khi làm việc trong không gian log, điều này không còn đúng nữa vì hàm không còn tuyến tính. Do đó, cần phải chuẩn hóa lại. May mắn thay, việc này rất dễ dàng trong PyTorch: chỉ cần cộng các giá trị lại và áp dụng hàm softmax để có đầu ra mong muốn. Nếu không làm như vậy, kết quả có thể sẽ không chính xác.

$$P(y_j \mid X, y_1, \ldots, y_{j-1}) = \underbrace{\text{softmax}}_{\text{Normalize}}\left(\sum_{m=1}^{M} \underbrace{\lambda_m(X, y_1, \ldots, y_{j-1})}_{\text{Interpolation coefficient for model } m} \underbrace{\log P_m(y_j \mid X, y_1, \ldots, y_{j-1})}_{\text{Log probability of model } m}\right)$$

Hệ số nội suy có thể được thiết lập như một hằng số hoặc học một cách động. Cách tiếp cận này cho phép kết hợp các mô hình ngôn ngữ khác nhau để tối ưu hóa khả năng dự đoán. Ví dụ, khi kết hợp hai mô hình, hệ số nội suy chỉ có một bậc tự do tại mỗi bước thời gian, giúp đơn giản hóa quá trình học so với việc dự đoán từ tiếp theo.

Nếu có năm mô hình, bạn sẽ thực hiện một phép "softmax" trên năm đầu ra, ít hơn nhiều so với toàn bộ từ vựng. Điều này làm cho việc học hệ số nội suy trở nên dễ dàng hơn. Hệ số này có thể được tinh chỉnh trên một tập dữ liệu rất nhỏ và thậm chí có thể không phụ thuộc vào ngữ cảnh, chỉ cần tính toán năm tham số.

Trong trường hợp bạn có một miền đặc biệt hoặc một nhiệm vụ cụ thể với chỉ 50 hoặc 100 ví dụ huấn luyện, bạn có thể học hệ số nội suy này rất hiệu quả. Ví dụ, nếu bạn có một mô hình ngôn ngữ y khoa với 1,3 tỷ tham số và một mô hình ngôn ngữ chung với 70 tỷ tham số, bạn có thể học hệ số nội suy giữa hai mô hình này. Mô hình ngôn ngữ chung sẽ xử lý các cấu trúc ngữ pháp, trong khi mô hình y khoa sẽ được ưu tiên khi gặp các thuật ngữ kỹ thuật y khoa. Phương pháp này rất hiệu quả khi bạn có lượng dữ liệu hạn chế.

Linear or Log Linear

Linear interpolation có thể được hiểu như một phép "hoặc" logic. Nó cố gắng tìm ra các ví dụ mà ít nhất một trong hai mô hình gán xác suất cao. Ví dụ, với các từ "bark", "run", và "diet", nếu ta lấy trung bình trong không gian tuyến tính, ta có thể có các xác suất như sau: "bark": 0.2, "run": 0.26, "diet": 0.21.

Trong trường hợp này, "run" sẽ được chọn vì nó có xác suất cao nhất. Linear interpolation hữu ích khi bạn có một mô hình nhỏ chuyên về từ vựng cụ thể và muốn nâng cao xác suất cho từ vựng đó, đặc biệt khi mô hình tổng quát không nhận diện được.

Log linear giống như phép "và" logic, chỉ chọn các lựa chọn mà tất cả các mô hình đều đồng ý. Điều này đặc biệt hữu ích khi bạn muốn hạn chế các câu trả lời có thể, chẳng hạn như ngăn chặn ngôn ngữ độc hại. Trong ví dụ trên, mô hình log linear có thể chọn từ khác vì nó có thể giảm điểm cho các từ có xác suất thấp.

Khi làm việc với các mô hình có từ vựng khác nhau, việc kết hợp chúng có thể trở nên phức tạp. Linear interpolation có thể giúp bằng cách chỉ kết hợp các từ vựng chung và bỏ qua các từ không có trong một trong hai mô hình. Tuy nhiên, cần cân nhắc kỹ trước khi thực hiện điều này.

Nếu lỗi của các mô hình hoàn toàn tương quan, như khi sử dụng cùng một mô hình hai lần, thì không có gì thay đổi. Tuy nhiên, ensembling thường giúp cải thiện độ chính xác vì các lỗi thường không hoàn toàn tương quan.

Using Models As Negative Evidence

Trong nội suy log-linear, các hệ số nội suy không nhất thiết phải dương mà có thể âm, cho phép chúng ta "phạt" xác suất từ một mô hình cụ thể. Cấu trúc điển hình bao gồm ba thành phần:

$$\text{logit} = \underbrace{\log P_{\text{core}}(y_t|X, y_{<t})}_{\text{Core}} + \lambda(\underbrace{\log P_{+}(y_t|X, y_{<t})}_{\text{Positive}} - \underbrace{\log P_{-}(y_t|X, y_{<t})}_{\text{Negative}})$$

- Mô hình cốt lõi (core model): Thường là một mô hình ngôn ngữ đa năng mạnh
- Mô hình dương (positive model): Mô hình cần tăng cường
- Mô hình âm (negative model): Mô hình cần giảm thiểu ảnh hưởng

Ví dụ thực tế từ nghiên cứu Dou et. al 2019:

- Core: Mô hình dịch máy
- Negative: Mô hình ngôn ngữ ngoài miền
- Positive: Mô hình ngôn ngữ trong miền

Ứng dụng trong dịch máy chuyên ngành: Khi có dữ liệu dịch máy chủ yếu trong lĩnh vực tin tức nhưng cần dịch văn bản y tế, ta có thể:

- Sử dụng mô hình dịch máy làm core
- Trừ đi xác suất từ mô hình ngôn ngữ tin tức (negative)
- Cộng thêm xác suất từ mô hình ngôn ngữ y tế (positive)

Một ứng dụng khác là DExperts (Liu et. al 2021):

- Core: Mô hình ngôn ngữ mạnh
- Negative: Mô hình ngôn ngữ độc hại yếu
- Positive: Mô hình ngôn ngữ không độc hại yếu

Phương pháp này giúp giảm thiểu nội dung độc hại trong đầu ra của mô hình ngôn ngữ.

Efficient Methods for Using Multiple Models

Trong Học sâu, Dropout được sử dụng để điều chuẩn hóa mạng nơ-ron. Ý tưởng cơ bản của nó là trong quá trình huấn luyện, một số nút trong mạng nơ-ron sẽ bị "drop out" ngẫu nhiên. Thông thường, khi kiểm tra, ta sử dụng toàn bộ mô hình mà không "drop out" nút nào. Tuy nhiên, một cách tiếp cận khác là "drop out" nhiều lần trong quá trình kiểm tra và kết hợp các mô hình khác nhau thông qua phương pháp ensemble. Đây là một ý tưởng đã được thử nghiệm trong bài báo về Dropout và giúp tạo ra nhiều mô hình khác nhau.

Một phương pháp khác đã tồn tại từ lâu là bagging. Bagging hoạt động bằng cách lấy mẫu lại dữ liệu với việc thay thế, tạo ra các tập dữ liệu có kích thước bằng nhau. Sau đó, ta huấn luyện nhiều mô hình khác nhau trên các tập dữ liệu này và kết hợp chúng lại. Cả hai phương pháp này đều giúp cải thiện độ bền vững của mô hình bằng cách cung cấp các góc nhìn khác nhau về dữ liệu, làm mượt đi những điểm bất thường.

Ngoài ra, có thể tạo ra nhiều mô hình từ các điểm kiểm tra khác nhau trong quá trình huấn luyện và kết hợp chúng. Tất cả các phương pháp này đều liên quan đến việc tận dụng các mô hình đã thấy dữ liệu khác nhau hoặc theo thứ tự khác nhau. Ngay cả khi các nút khác nhau trong mạng nơ-ron thấy các phần khác nhau của dữ liệu do Dropout, điều này cũng tạo ra các mô hình đủ khác biệt để cải thiện khi kết hợp.

Problem with Ensembling Cost

Việc sử dụng nhiều mô hình (ensembling) có thể mang lại hiệu quả cao, nhưng cũng đi kèm với chi phí lớn. Vấn đề chính của ensembling là chi phí, bởi vì phương pháp này yêu cầu chạy nhiều mô hình cùng lúc trong quá trình kiểm tra và suy luận. Điều này không lý tưởng khi triển khai dịch vụ, vì chi phí sẽ tăng tuyến tính theo số lượng mô hình bạn chạy. Cụ thể, nó đòi hỏi n lần tính toán và n lần bộ nhớ, trong đó bộ nhớ có thể là vấn đề lớn nhất. Bạn cần triển khai thêm các máy GPU và các thiết bị khác. Vậy câu hỏi đặt ra là: Có cách nào để tận dụng lợi ích của ensembling mà không cần tạo ra nhiều mô hình không? Phương pháp đơn giản nhất là trung bình tham số (parameter averaging).

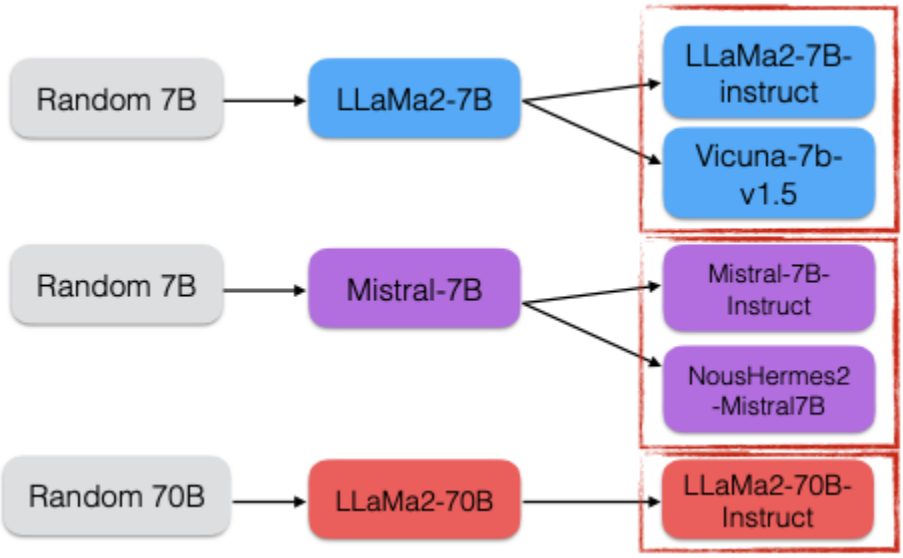
Parameter Averaging (e.g. Utans 1996)

Phương pháp trung bình tham số bao gồm việc trung bình các tham số của nhiều mô hình với nhau. Tuy nhiên, phương pháp này chỉ hoạt động dưới một số điều kiện nhất định.

Một câu hỏi đặt ra là liệu có thể trung bình tham số của "llama 7B" và "Llama 70B" không? Câu trả lời là không, vì hai mô hình này có số lượng tham số hoàn toàn khác nhau, không thể tìm thấy sự tương ứng một-một giữa 7 tỷ và 70 tỷ tham số.

Tương tự, việc trung bình tham số giữa "Llama 7B" và "Mistral 7B" cũng không khả thi nếu chúng có kiến trúc khác nhau. Ngay cả khi kiến trúc giống nhau, sự khởi tạo ngẫu nhiên khác nhau cũng có thể dẫn đến kết quả không mong muốn. Điều này là do trong mạng nơ-ron, không có ý nghĩa cố định cho từng tham số cụ thể.

Tuy nhiên, có thể trung bình các mô hình được phát triển từ cùng một mô hình tiền huấn luyện. Ví dụ, từ một mô hình khởi tạo như "Llama 2 7B", có thể có hàng trăm mô hình khác được phát triển từ đó. Các tham số của những mô hình này có ý nghĩa tương tự nhau, do đó có thể trung bình chúng.



Có hai cách để thực hiện điều này:

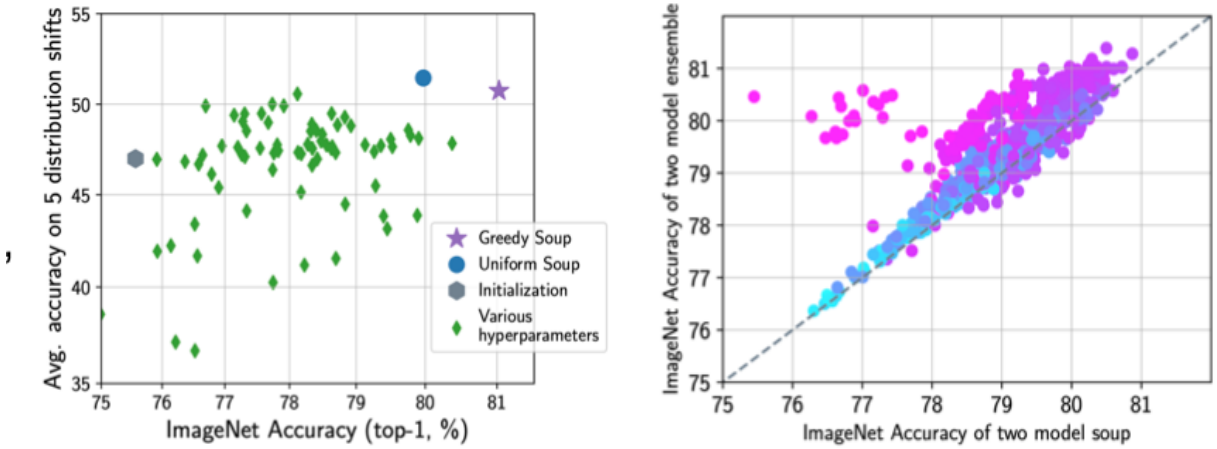
- Trung bình các checkpoints trong quá trình huấn luyện: Phương pháp này giúp giảm nhiễu từ quá trình tối ưu hóa gradient ngẫu nhiên và cải thiện độ chính xác tổng thể.
- Kết hợp các mô hình đã tinh chỉnh. Ví dụ, có thể kết hợp "Llama 2 7B instruct" và "Vicuna 7B 1.5" để cải thiện kết quả.

Một câu hỏi thú vị là liệu trung bình tham số có thể làm cho mô hình ít độc hại hơn không. Câu trả lời phức tạp hơn, vì việc kết hợp một mô hình đã được tinh chỉnh để giảm độc tính với một mô hình gốc có thể tạo ra một mô hình trung gian, không an toàn bằng mô hình đã được tinh chỉnh. Tuy nhiên, có thể có những cách sáng tạo để sử dụng phương pháp này, chẳng hạn như kết hợp một mô hình đã được tinh chỉnh tốt về hướng dẫn với một phiên bản ít độc hại hơn của cùng mô hình gốc.

Model Soups (Wortsman et al. 2022)

Trong những năm gần đây, có một số nghiên cứu mới về phương pháp "model soup". Phương pháp này được xem xét nhiều trong bối cảnh các mạng hiện đại. Bài báo "model soup" nghiên cứu hai chiến lược: "uniform averaging" và "greedy averaging".

Với "uniform averaging", tất cả các tham số được trung bình cộng lại. Trong khi đó, "greedy averaging" thêm từng mô hình một và kiểm tra xem mô hình trung bình có cải thiện không. Nếu có, mô hình đó được giữ lại, nếu không, nó bị loại bỏ. Kết quả cho thấy "greedy averaging" có hiệu suất tốt hơn so với mô hình đơn lẻ tốt nhất trên tập dữ liệu ImageNet, nhưng "uniform averaging" lại kém hơn mô hình tốt nhất. Tuy nhiên, "uniform averaging" lại tỏ ra mạnh mẽ hơn khi đối mặt với các tập dữ liệu có sự thay đổi phân phối.

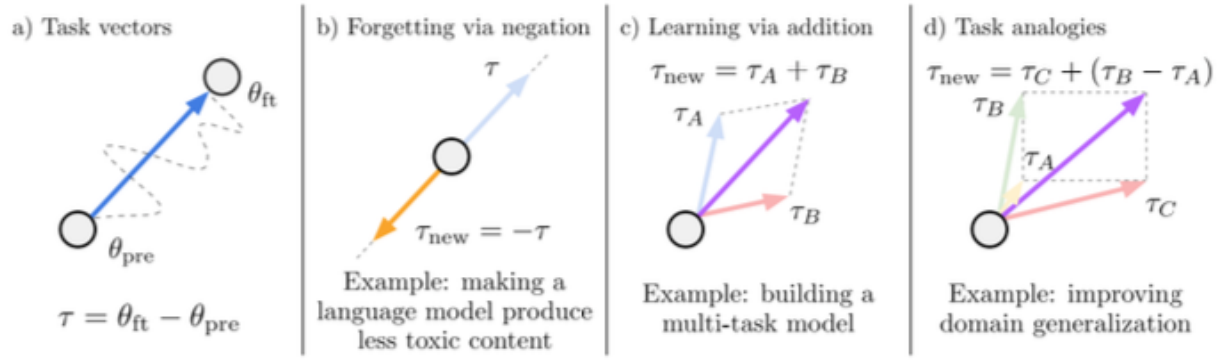


Một điểm thú vị khác là có sự tương quan mạnh giữa "averaging" và "ensembling". "Averaging" gần như không bao giờ tốt hơn "ensembling", nhưng nhanh hơn. Tuy nhiên, có những trường hợp "ensembling" vượt trội hơn hẳn so với "averaging", cho thấy rằng "parameter averaging" là an toàn nhưng có thể làm giảm độ chính xác trong một số trường hợp.

Các thí nghiệm này được thực hiện trên các mô hình CNN cho ImageNet. Một câu hỏi đặt ra là liệu có sự khác biệt nào khi áp dụng cho Transformers, vì các biểu diễn của Transformers thường tập trung ở một số phần cụ thể của không gian. Mặc dù việc hợp nhất các mô hình Transformer đã được chứng minh là cải thiện độ chính xác, nhưng có thể có những phương pháp "parameter averaging" hoặc "model merging" đặc biệt cho Transformers để tận dụng đặc điểm này. Đây có thể là một hướng nghiên cứu thú vị trong tương lai.

Task Vectors

Tiếp theo, chúng ta sẽ tìm hiểu về "Task vectors". Ý tưởng cơ bản của task vectors là kết hợp hai mô hình bằng cách lấy trung bình các tham số của chúng. Task vectors tận dụng sự khác biệt giữa các mô hình đã được fine-tune từ một mô hình gốc. Task vector được định nghĩa là sự khác biệt giữa vector tham số của mô hình gốc và mô hình đã fine-tune.



Một ứng dụng thú vị của task vectors là khả năng "trừ" đi các "tasks" không mong muốn. Ví dụ, nếu có một mô hình được huấn luyện trên dữ liệu văn bản độc hại, ta có thể trừ đi task vector để cố gắng loại bỏ khả năng của mô hình trong việc thực hiện các tác vụ không mong muốn đó.

Ngoài ra, ta có thể kết hợp hai task vectors để tạo ra một mô hình mới từ sự kết hợp của cả hai. Điều này mang lại sự linh hoạt hơn so với việc chỉ đơn giản là trung bình các tham số. Task vectors cũng cho phép giải quyết xung đột giữa các vector của các tác vụ khác nhau. Ví dụ, nếu có ba mô hình với các vector xung đột, phương pháp này sẽ xác định các vector mạnh nhất và giải quyết xung đột để cải thiện khả năng thực hiện tất cả các tác vụ cùng lúc.

Cuối cùng, phương pháp này được chứng minh là hiệu quả hơn so với việc chỉ trung bình các tham số, giúp cải thiện khả năng thực hiện đồng thời nhiều tác vụ.

Meta Learning

Chúng ta sẽ khám phá một ví dụ về meta learning trong dịch máy cho các ngôn ngữ ít tài nguyên. Meta learning là một phương pháp học nhằm tìm ra một khởi tạo tốt, giúp dễ dàng tinh chỉnh mô hình cho các nhiệm vụ mới. Quá trình này bao gồm hai bước gradient descent. Đầu tiên, mô hình được cập nhật với dữ liệu từ một ngôn ngữ, ví dụ như French. Sau đó, mô hình tiếp tục được cập nhật với dữ liệu từ một ngôn ngữ khác. Quá trình này giúp tạo ra một khởi tạo tốt cho việc huấn luyện trên các ngôn ngữ hoặc nhiệm vụ khác.

Một điểm thú vị là mặc dù meta learning có thể khó triển khai do yêu cầu tính toán phức tạp, nhưng nó có thể cải thiện khả năng của mô hình trong việc trả lời câu hỏi từ các tài liệu mới. Trong một nghiên cứu gần đây, thay vì sử dụng meta learning, nhóm nghiên cứu đã thực hiện một bước tiền huấn luyện trên các cặp câu hỏi-đáp và tài liệu từ một miền khác, sau đó huấn luyện trên tài liệu từ miền mục tiêu. Ví dụ, họ huấn luyện trên các cặp câu hỏi-đáp từ Wikipedia trước khi chuyển sang tài liệu y khoa. Kết quả cho thấy mô hình có khả năng trả lời câu hỏi tốt hơn trên các tài liệu đã được tinh chỉnh.

Software: mergekit

Chúng ta đã thảo luận về một số phương pháp kết hợp các mô hình lại với nhau. Một công cụ phổ biến được sử dụng cho mục đích này là "merge kit", giúp việc kết hợp mô hình trở nên

tương đối dễ dàng. Công cụ này triển khai nhiều mô hình mà tôi đã đề cập, bao gồm các phương pháp tuyến tính, phương pháp "task arithmetic" và "ties". Nếu bạn muốn kết hợp các mô hình lại với nhau, từ góc độ phần mềm, điều này khá dễ thực hiện.

Method	`merge_method` value	Multi-Model	Uses base model
Linear (Model Soups)	`linear`	✅	❌
SLERP	`slerp`	❌	✅
Task Arithmetic	`task_arithmetic`	✅	✅
TIES	`ties`	✅	✅
DARE TIES	`dare_ties`	✅	✅
DARE Task Arithmetic	`dare_linear`	✅	✅
Passthrough	`passthrough`	❌	❌

Ensemble Distillation (e.g. Kim et al. 2016)

Một phương pháp đơn giản khác là “Ensemble Distillation". Chúng ta đã nói về "distillation" và ý tưởng này khá đơn giản. Việc trung bình tham số chỉ thực sự hiệu quả với các mô hình trong cùng một lần chạy, cùng kiến trúc mô hình và cùng khởi tạo. "Knowledge distillation" huấn luyện một mô hình để sao chép một "ensemble" và cố gắng khớp với phân phối trên các từ dự đoán.

Điều này cho phép mô hình đưa ra các dự đoán tốt giống như "ensemble" và cũng có thể đưa ra các dự đoán sai giống như "ensemble". Nó giúp bạn học hiệu quả hơn, giống như cách "distillation" thường làm. Động lực ban đầu cho "model distillation" khi Jeff Hinton đề xuất vào năm 2015 trong một bài báo là để sao chép một "ensemble". Ngày nay, chúng ta sử dụng nó cho nhiều mục đích khác, như trong "distillation" để giảm số lượng lớp, nhưng đó là mục đích ban đầu.

Sparse Mixture of Experts Models

Sparse Computation

Trong phần này, chúng ta sẽ tìm hiểu về mô hình "sparse mixture of experts" và tầm quan trọng của nó trong các mô hình hiện đại. Đây là một khái niệm quan trọng, được cho là đã được sử dụng trong GPT 4 và chắc chắn được áp dụng trong Mistral, một trong những mô hình mở tiên tiến nhất hiện nay.

Mô hình "sparse mixture of experts" tận dụng tính toán thưa. Hãy tưởng tượng khi bạn thực hiện phép nhân giữa một tensor và một số vô hướng 0. Kết quả của phép nhân này sẽ là một tensor toàn số 0, và do đó, bạn không cần thực hiện phép tính này. Điều này giúp tối ưu hóa quá trình tính toán trong nhiều mô hình hiện đại.

Một ví dụ cụ thể là khi bạn thực hiện phép nhân ma trận hoặc phép nhân ma trận-véc tơ mà một số hàng của ma trận là toàn số 0. Trong trường hợp này, bạn có thể bỏ qua các phép tính liên quan đến những hàng đó.

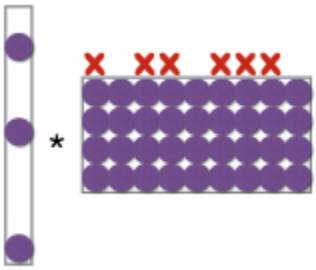
Ngoài ra, tính toán thưa còn có thể áp dụng cho các tensor lớn hơn và thậm chí cho toàn bộ mô hình trong một tập hợp (ensemble). Trong trường hợp này, nếu xác suất của một mô hình trong tập hợp là 0, bạn có thể loại bỏ mô hình đó mà không cần sử dụng.

Tóm lại, việc tối ưu hóa này có thể được thực hiện tự động bởi GPU trong trường hợp đầu tiên. Trong trường hợp thứ hai, nó thường xuất hiện trong các mô hình "sparse mixture of experts". Và cuối cùng, trong một tập hợp mô hình, bạn có thể loại bỏ hoàn toàn những mô hình không cần thiết nếu xác suất của chúng là 0.

GPU-level Sparsity

Các GPU của Nvidia hỗ trợ nhiều loại độ thưa khác nhau, và đội ngũ tuyệt vời tại Nvidia đã nỗ lực để đảm bảo rằng tính năng này hoạt động hiệu quả, ít nhất là ở một mức độ nào đó.

Một trong những công cụ hữu ích trong việc này là thư viện có tên là “cuSPARSE”, được sử dụng trong PyTorch và nhiều thư viện khác. Để minh họa, hãy xem xét phép nhân giữa vector và ma trận với một vector thưa, chẳng hạn như vector xuất phát từ hàm kích hoạt ReLU.

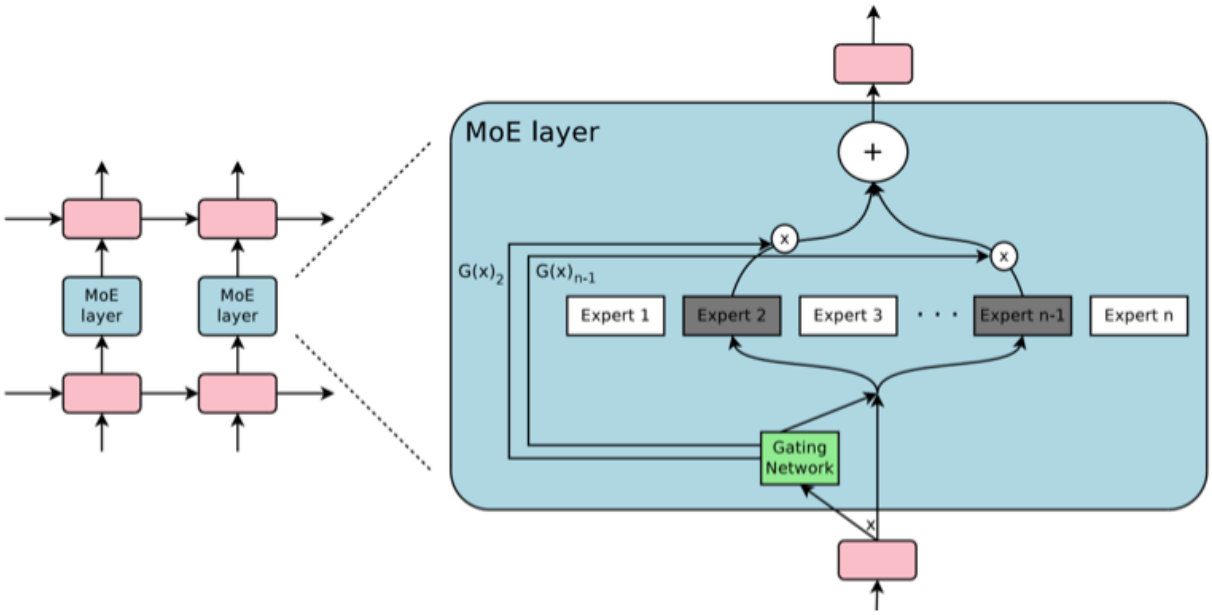


Giả sử bạn chỉ có ba phần của vector này là hoạt động. Trong trường hợp này, bạn thực sự không cần phải tính toán bất kỳ cột nào không có giá trị, điều này giúp đơn giản hóa quá trình tính toán và tiết kiệm tài nguyên.

Sparsely Gated Mixture of Experts Layer (Shazeer+ 2017)

Thông thường, trong một Transformer, mạng nơ-ron truyền thẳng là một cấu trúc rất rộng, được sử dụng để trích xuất nhiều đặc trưng ở mỗi lớp. Đây cũng là nơi tiêu tốn nhiều tài nguyên tính toán nhất trong Transformer.

Lớp "sparsely gated mixture of experts" hoạt động bằng cách sử dụng một mạng nơ-ron điều khiển (gating network) để tính toán xác suất kết hợp. Đặc điểm nổi bật là xác suất này thường bằng 0 cho nhiều phần của mạng nơ-ron truyền thẳng. Đối với những phần có xác suất bằng 0, chúng ta không cần thực hiện tính toán. Khi kết hợp các phần lại, chúng ta sử dụng tỷ lệ kết hợp.



Ý tưởng cơ bản của phương pháp này rất đơn giản và có thể được triển khai chỉ với 7-8 dòng mã Pytorch. Chúng ta có một hàm điều khiển (gating function) để tính toán dựa trên đầu vào, sau đó thực hiện thao tác "keep top K". Tiếp theo, chúng ta áp dụng hàm softmax. Thao tác "keep top K" giữ lại các giá trị nằm trong top K và loại bỏ các giá trị không nằm trong top K.

$$g(x) = \text{softmax}(\text{keep\_top\_k}(f_{\text{gating}}(x), k))$$

$$\text{keep\_top\_k}(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ -\infty & \text{otherwise.} \end{cases}$$

Lợi ích lớn của phương pháp này là giảm đáng kể khối lượng tính toán. Ví dụ, nếu bạn giữ lại top 2 trong số 8, bạn có thể giảm khối lượng tính toán xuống bốn lần cho phần này. Điều này giúp tối ưu hóa hiệu suất của mô hình mà không làm giảm chất lượng đầu ra.

Trong buổi thảo luận gần đây, một câu hỏi thú vị đã được đặt ra về khả năng song song hóa của các mô hình AI, đặc biệt là khi sử dụng GPU. Khi làm việc với các mô hình nhỏ, lợi ích từ việc song song hóa không rõ ràng do giới hạn không phải ở tính toán mà ở việc di chuyển dữ liệu và các yếu tố khác của GPU. Tuy nhiên, với các mô hình lớn hơn, việc giảm tải tính toán có thể mang lại lợi ích đáng kể.

Một điểm quan trọng khác là việc triển khai phần cứng có thể gặp khó khăn khi thực hiện batching. Nguyên nhân là do các expert khác nhau sẽ hoạt động cho các phần khác nhau của batch, đòi hỏi phải có những kỹ thuật xử lý phức tạp.

Pipeline Systems

Cascaded Systems

Hệ thống pipeline là những hệ thống mà đầu ra của một mô hình trở thành đầu vào của một mô hình khác. Một ví dụ điển hình của hệ thống này là hệ thống dạng chuỗi (cascaded system), nơi mà đầu ra của một hệ thống được đưa vào làm đầu vào cho hệ thống khác. Một ví dụ rất quen thuộc là dịch giọng nói, đó là âm thanh được chuyển thành văn bản thông qua nhận dạng giọng nói, sau đó văn bản này được dịch sang ngôn ngữ khác bằng máy dịch.

$$\widehat{Y_1} = argmax_Y P\left(Y|\, X;\, \theta_1\right)$$
$$\widehat{Y_2} = argmax_Y P\left(Y|\, Y_1;\, \theta_2\right)$$

Một trong những thách thức lớn của dịch giọng nói là các hệ thống pipeline thường hoạt động tốt hơn so với các hệ thống end-to-end (từ giọng nói sang văn bản ở ngôn ngữ khác). Nguyên nhân chính là do dữ liệu. Dữ liệu cho nhận dạng giọng nói trong cùng một ngôn ngữ và dịch văn bản giữa các ngôn ngữ có sẵn nhiều hơn so với dữ liệu cho dịch giọng nói trực tiếp giữa các ngôn ngữ. Ngoài ra, nhận dạng giọng nói trong cùng một ngôn ngữ là một nhiệm vụ đơn giản hơn, dễ học hơn.

Một yếu tố quan trọng khác là khả năng diễn giải. Khi giao tiếp qua hệ thống dịch giọng nói, nếu nhận dạng giọng nói không chính xác, thì khả năng cao là bản dịch cũng sẽ không chính xác. Do đó, việc có thể kiểm tra độ chính xác của nhận dạng giọng nói là rất quan trọng, trong khi việc kiểm tra bản dịch thì khó khăn hơn nhiều.

Stacking

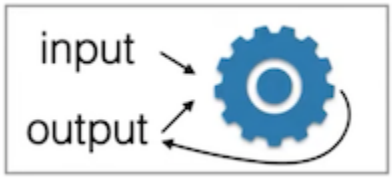
Khái niệm "stacking" được nhắc đến như một phương pháp tương tự "cascading", nhưng với một số điểm khác biệt quan trọng. Stacking cho phép sử dụng hai mô hình khác nhau cho cùng một nhiệm vụ, nhưng với cách dự đoán khác nhau.

$$\widehat{Y_1} = argmax_Y P\left(Y|\, X;\, \theta_1\right)$$
$$\widehat{Y_2} = argmax_Y P\left(Y|X,\, Y_1;\, \theta_2\right)$$

Ví dụ, trong dịch giọng nói, mô hình đầu tiên thực hiện nhận dạng giọng nói để chuyển đổi thành văn bản tiếng Anh. Sau đó, mô hình dịch sẽ sử dụng cả giọng nói và văn bản tiếng Anh để tạo ra bản dịch tiếng Nhật. Điều này không chỉ giúp kiểm tra lại độ chính xác của bản chép mà còn giữ lại thông tin độc đáo chỉ có trong giọng nói. Chẳng hạn, câu "I read the book" có thể được hiểu theo thì hiện tại hoặc quá khứ, và cả hai đều được chép lại giống nhau. Trong trường hợp này, hệ thống cascading có thể làm mất thông tin, trong khi hệ thống stacking thì không.

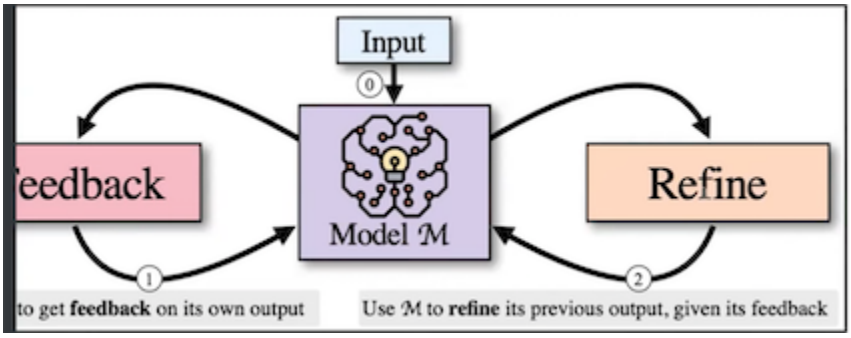
Iterative Refinement

Hiện nay, các mô hình ngôn ngữ lớn đã mở ra nhiều khả năng mới. Một trong những kỹ thuật thú vị là việc tinh chỉnh đầu ra thông qua các mạng Deliberation. Kỹ thuật này cho phép chúng ta nhập dữ liệu vào mô hình, nhận đầu ra, sau đó đưa đầu ra này trở lại mô hình để dần dần cải thiện chất lượng. Mạng Deliberation sử dụng thuật toán học tăng cường để thực hiện quá trình này, giúp cải thiện đầu ra qua từng bước.



Một xu hướng khác đang được quan tâm là các mô hình khuếch tán (diffusion models). Về cơ bản, mô hình khuếch tán hoạt động bằng cách bắt đầu từ một trạng thái không có gì và dần dần cải thiện nó. Điểm khác biệt chính giữa mạng Deliberation và mô hình khuếch tán là mô hình khuếch tán có thể được huấn luyện từ đầu bằng cách thêm nhiễu vào đầu vào, giúp huấn luyện hiệu quả hơn. Mô hình này được sử dụng rộng rãi trong việc tạo hình ảnh, nhưng chưa phổ biến trong xử lý văn bản do các mô hình tự hồi quy truyền thống đã rất hiệu quả.

Cuối cùng, một phương pháp khác là tự tinh chỉnh (self-refine). Phương pháp này yêu cầu mô hình tạo ra đầu ra, sau đó tự đánh giá và phản hồi về chất lượng đầu ra đó. Ví dụ, trong việc tạo mã, mô hình có thể chỉ ra lỗi trong đầu ra. Sau đó, cả đầu ra và phản hồi được đưa trở lại mô hình để cải thiện. Phương pháp này đã chứng minh hiệu quả trong nhiều tác vụ, nhưng yêu cầu mô hình phải có chất lượng cao, như GPT-4, để hoạt động ổn định.



Discussion questions

Nếu mô hình Stacked có khả năng giảm thiểu mất mát thông tin, tại sao chúng ta vẫn chọn mô hình Cascade?

Có hai lý do chính cho điều này:

- Khả năng tiếp cận dữ liệu: Để huấn luyện mô hình Stacked, cần có đầu ra cho mỗi ví dụ huấn luyện. Điều này đòi hỏi phải chạy nhận dạng giọng nói cho từng ví dụ và không thể

sử dụng các ví dụ chỉ có đầu vào văn bản, trừ khi bạn tổng hợp giọng nói từ văn bản cho các tác vụ như dịch máy. Điều này làm tăng yêu cầu về dữ liệu.

- Độ phức tạp và hiệu suất: Mô hình Stacked yêu cầu thiết kế phức tạp hơn, cần xử lý cả đầu vào giọng nói và văn bản. Trong khi đó, mô hình Cascade có thể đơn giản hơn khi chỉ cần kết hợp một hệ thống nhận dạng giọng nói với một hệ thống dịch thuật.

Cách đo lường đóng góp của từng thành phần trong Ensemble?

Một cách đơn giản để đo lường đóng góp của từng thành phần trong một hệ thống ensemble là xem xét các hệ số nội suy. Nếu bạn huấn luyện các hệ số này, chúng có thể cung cấp thông tin về mức độ đóng góp của từng thành phần.

So sánh giữa Iterative Refinement và Boosting?

Iterative refinement và boosting có một số điểm khác biệt. Iterative refinement thường liên quan đến việc lấy đầu ra phức tạp của một hệ thống và chỉnh sửa nó, không chỉ đơn thuần là điều chỉnh xác suất của một bộ phân loại đơn lẻ. Trong khi đó, boosting thường làm việc với đầu ra phân loại đơn giản. Iterative refinement có thể phức tạp hơn khi nó liên quan đến việc tạo ra và tinh chỉnh đầu ra văn bản qua nhiều bước.

Resources

1. https://phontron.com/class/anlp2024/lectures/#ensembling-and-mixture-of-experts-feb-29
2. Reference: Domain Differential Adaptation (Dou et al. 2019)
3. Reference: Dexperts (Liu et al. 2021)
4. Reference: Knowledge Distillation (Hinton et al. 2015)
5. Reference: cuSPARSE
6. Reference: NVIDIA Block Sparsity
7. Reference: Sparsely Gated MOE (Shazeer et al. 2017)
8. Code Example: Mistral MOE Implementation
9. Reference: Weight Averaging for Neural Networks (Utans 1996)
10. Reference: Model Soups (Wortsman et al. 2022)
11. Software: MergeKit
12. Reference: Task Vectors (Ilharco et al. 2022)
13. Reference: TIES (Yadav et al. 2023)
14. Reference: Stacking (Niehues et al. 2017)
15. Reference: Deliberation Networks (Xia et al. 2017)
16. Reference: Diffuser (Reid et al. 2022)
17. Reference: Self-refine (Madaan et al. 2023)