Lecture 3: Mô Hình Ngôn Ngữ

Generative vs Discriminative Models

Mô hình sinh (Generative) và mô hình phân biệt (Discriminative) là hai mô hình quan trọng trong Học máy. Mô hình phân biệt là mô hình tính toán đặc điểm tiềm ẩn (latent trait) dựa trên dữ liệu và được biểu diễn bởi công thức $P(Y|X)$ trong đó Y là đặc điểm chúng ta muốn tính toán và X là dữ liệu đầu vào. Ví dụ trong bài học trước, X là một đoạn text, Y là một nhãn cảm xúc (tích cực/ tiêu cực) trong nhiệm vụ phân tích cảm xúc.

Ngược lại, mô hình sinh là mô hình tính toán xác suất không có điều kiện của dữ liệu đầu vào. Mô hình này có thể tính toán xác suất độc lập $P(X)$ hoặc xác suất chung $P(X, Y)$. Tóm lại mô hình phân biệt tập trung vào dự đoán Y cho bởi X, trong khi mô hình sinh tập trung vào việc mô phỏng và hiểu dữ liệu X.

Probabilistic Language Model

Mô hình xác suất ngôn ngữ là một mô hình sinh tính toán xác suất của ngôn ngữ. Cụ thể, nó tính xác suất $P(X)$ trong đó X có thể là một câu hoặc tài liệu. Gần đây, định nghĩa mô hình ngôn ngữ đã mở rộng bao gồm cả những mô hình tính toán xác suất của văn bản và hình ảnh, được gọi là mô hình ngôn ngữ đa phương thức (multimodal language model). Trong bài này, chúng ta chỉ tập trung vào mô hình tính toán xác suất của văn bản.

What Can we do with Language Models?

Có hai kỹ thuật cơ bản chúng ta có thể thực hiện với mô hình ngôn ngữ. Thứ nhất, chúng ta tính xác suất câu. Ví dụ, nếu chúng ta tính xác suất cho câu “Jane went to the store”, câu này có xác suất cao. Ngược lại, câu “store to Jane went the” có xác suất thấp. Nếu chúng ta sử dụng một mô hình ngôn ngữ Tiếng Trung, nó cho câu đầu tiên xác suất thấp. Vì đây là mô hình ngôn ngữ Tiếng Trung, chứ không phải Tiếng Anh. Điều này cho thấy có nhiều mô hình ngôn ngữ khác nhau, tuỳ thuộc vào dữ liệu mà bạn sử dụng.

Thứ hai, chúng ta tạo câu. Có hai phương pháp chính để tạo câu. Một là, sampling đó là bạn lấy mẫu một câu từ phân phối xác suất của mô hình ngôn ngữ, có thể có một số điều chỉnh với phân phối xác suất. Thứ hai, tìm câu nào có xác suất cao nhất theo mô hình ngôn ngữ. Chúng ta sẽ tìm hiểu các phương pháp này trong các phần sau.

$$\tilde{x} \sim P(X)$$

How Can We Apply These?

Các kỹ thuật này có thể sử dụng để trả lời câu hỏi. Ví dụ, nếu chúng ta có một câu hỏi trắc nghiệm, chúng ta có thể đánh giá các câu trả lời có thể có. Để làm điều này, chúng ta tính xác

suất các câu trả lời. Giả sử câu hỏi là "Where is CMU located?" và các lựa chọn là "New York", "Birmingham", v.v. Chúng ta sẽ tính xác suất cho từng lựa chọn và chọn lựa chọn có xác suất cao nhất.

Một ứng dụng khác là tạo ra phần tiếp theo cho một câu hỏi. Chúng ta có thể sử dụng prompting để yêu cầu mô hình tạo ra phần hoàn thành có khả năng xảy ra nhất hoặc tạo ra nhiều phần hoàn thành khác nhau.

Chúng ta cũng có thể phân loại văn bản. Một cách để làm điều này là đánh giá câu có cảm xúc, ví dụ như “This is Great!” và tính xác suất cho các đánh giá từ 1 đến 5 sao, sau đó chọn đánh giá có khả năng nhất.

Ngoài ra, có một phương pháp thú vị là tạo ra đầu ra dựa trên một đánh giá sao cụ thể, ví dụ như "star rating five", và sau đó kiểm tra xem đầu ra có phù hợp với khái niệm của một câu tích cực hay không.

Cuối cùng, chúng ta có thể sửa lỗi ngữ pháp. Bằng cách tính xác suất cho từng từ, chúng ta có thể thay thế những từ có xác suất thấp bằng những từ có xác suất cao hơn. Hoặc yêu cầu mô hình vui lòng diễn giải lại đầu ra này để có phiên bản tốt hơn về mặt ngữ pháp.

Tóm lại, các mô hình ngôn ngữ rất đa dạng và có thể thực hiện nhiều nhiệm vụ khác nhau, nhưng hầu hết đều liên quan đến hai nhiệm vụ chính: đánh giá (scoring) hoặc tạo sinh (generating).

Auto-Regressive Language Models

Mô hình ngôn ngữ tự hồi quy (auto-regressive) tính toán xác suất một token dựa trên các token trước đó. Cụ thể là xác suất $P(x_i | x_1, ..., x_{i-1})$ được tính bằng dự đoán token tiếp theo dựa trên ngữ cảnh của các token trước đó. Thông thường, quá trình này diễn ra từ trái sang phải, từ đầu đến cuối. $P(X) = \prod_{i=1}^{I} P(x_i | x_1, ..., x_{i-1})$

Một câu hỏi đặt ra là tại sao chúng ta không cố gắng dự đoán xác suất của toàn bộ chuỗi token một cách trực tiếp $P(X)$? Lý do là nếu chúng ta làm như vậy kích thước của bài toán phân loại sẽ rất lớn V^N trong đó V là kích thước từ vựng, và N là độ dài chuỗi. Điều này rất khó khăn để xử lý. Thay vào đó, bằng cách phân tách chúng, ta có thể giải quyết N bài toán dự đoán với kích thước V, giúp cho việc huấn luyện mô hình trở lên khả thi hơn.

Ngoài mô hình tự hồi quy, còn có các mô hình ngôn ngữ khác như mô hình ngôn ngữ bị che khuất (masked language models) như BERT, DeBERTa, hay RoBERTa. Những mô hình này dự đoán một từ bị che khuất dựa trên các từ còn lại trong chuỗi. Tuy nhiên, chúng không cung cấp xác suất chính xác cho chuỗi ngôn ngữ và cũng khó khăn trong việc sinh ra văn bản vì chúng không xác định được thứ tự sinh ra.

Cuối cùng, còn có mô hình ngôn ngữ dựa trên năng lượng (energy-based) tạo ra một hàm điểm số không theo thứ tự cụ thể nào. Những mô hình này rất tiên tiến, sẽ được thảo luận sau nếu cần. Các mô hình ngôn ngữ hiện nay GPT, Lamma đều thuộc loại này.

Unigram Language Models

The Simplest Language Model: Count-based Unigram Models

Mô hình ngôn ngữ Unigram là mô hình ngôn ngữ đơn giản nhất và chúng hoạt động bằng cách tính xác suất của từ tiếp theo mà không quan tâm đến thứ tự các từ trước đó. Cách thực hiện là chúng ta chỉ cần dự đoán xác suất của từ tiếp theo độc lập với các từ trước đó.

$$P(x_i | x_1, ..., x_{i-1}) \approx P(x_i)$$

Giả sử bạn có một từ nào đó, việc dự đoán xác suất cho nó rất dễ dàng. Bạn chỉ cần đếm số lần xuất hiện của từ này trong tập huấn luyện và chia cho tổng số từ trong tập dữ liệu đó.

$$P_{MLE}(x_i) = c_{train}(x_i)/\Sigma_{x_j} c_{train}(x_j)$$

Bây giờ bạn đã có một mô hình ngôn ngữ, mặc dù là mô hình đơn giản nhất, nhưng chúng cũng gặp một số vấn đề.

Handling Unknown Words

Một vấn đề với mô hình ngôn ngữ là xử lý các từ chưa biết. Nếu một từ không tồn tại trong tập dữ liệu huấn luyện, xác suất của bất kỳ chuỗi nào chứa từ đó đều bằng không. Điều này không phải vấn đề lớn khi tạo ra văn bản từ mô hình ngôn ngữ, vì có thể bạn chỉ cần tạo ra các từ mà mô hình đã thấy trước đó. Tuy nhiên đây là vấn đề nghiêm trọng trong việc tính điểm các chuỗi hoặc trong nhiệm vụ dịch thuật. Nếu bạn gặp một từ không biết khi dịch, bạn muốn dịch nó một cách hợp lý, nhưng điều đó là không thể.

Để khắc phục vấn đề này, có hai lựa chọn. Lựa chọn đầu tiên là phân đoạn từ chưa biết thành các từ con. Đây là lựa chọn ưa chuộng hiện nay, bạn chỉ cần chạy “Sentence Piece" để phân đoạn từ vựng của mình, bạn sẽ không phải gặp từ chưa biết nữa, vì tất cả các từ chưa biết sẽ được chia thành các đơn vị ngắn hơn.

Lựa chọn thứ hai là xây dựng một mô hình từ chưa biết. Mô hình này dự đoán xác suất của từ chưa biết dựa trên các ký tự, và sau đó mô hình hoá xác suất các từ bằng cách sử dụng chính các từ. Như vậy, bạn sẽ có một mô hình phân cấp, nơi bạn cố gắng dự đoán các từ trước, nếu không được, bạn dự đoán các từ chưa biết. Mặc dù, phương pháp này không được sử dụng rộng rãi như trước, nhưng cũng đáng để nhắc đến.

Parameterizing in Log Space

Khi nhân nhiều xác suất, chẳng hạn như 30 xác suất của các token, có thể xảy ra vấn đề “numerical underflow" (tràn số dưới). Do khi thực hiện trên máy tính, chúng ta gặp giới hạn về

số lượng bit trong số thực 32 bit. Nếu tích xác suất trở nên quá nhỏ, nó sẽ bị underflow và trở thành 0, mặc dù không phải như vậy. Vì lý do này, việc sử dụng log sẽ giúp con người dễ dàng hơn trong khi nhìn các giá trị $-\ 30$ hơn là 10^{-30}.

$$P(X) = \prod_{i=1}^{|X|} P(x_i) \rightarrow log\ P(X) = \sum_{i=1}^{|X|} log\ P(x_i)$$

Mỗi xác suất unigram có thể coi là một tham số. Do đó số lượng tham số trong một mô hình unigram chính bằng kích thước từ vựng.

$$\theta_{x_i} := log\ P(x_i)$$

Higher-order Language Models

Higher-order n-gram Models

Mô hình n-gram bậc cao hoạt động bằng cách giới hạn độ dài ngữ cảnh đến một độ dài N, sau đó đếm số lần xuất hiện một chuỗi (bao gồm chuỗi ngữ cảnh và từ cần dự đoán) và chia cho số lần xuất hiện của chuỗi ngữ cảnh có độ dài N.

$$P_{ML}\left(x_i|x_{i-n+1},\ ...,\ x_{i-1}\right) := c\left(x_{i-n+1},\ ...,\ x_i\right)/c\left(x_{i-n+1},\ ...,\ x_{i-1}\right)$$

Ví dụ, chúng ta đếm số lần xuất hiện của "this is an example" và chia cho số lần xuất hiện của "this is an" để tính xác suất của "example" dựa trên các từ trước đó.

Tuy nhiên, vấn đề xảy ra khi chúng ta gặp một chuỗi chưa từng thấy trước đây, điều này dẫn đến việc xác suất bằng 0. Để khắc phục điều này, các mô hình ngôn ngữ n-gram sẽ quay lại sử dụng các mô hình n-gram ngắn hơn. Ví dụ, nếu mô hình hiện tại là 4-gram, nó sẽ tính toán và sau đó kết hợp với mô hình trigram, và mô hình trigram sẽ kết hợp với mô hình bigram, cuối cùng là mô hình unigram.

$$P\left(x_i|x_{i-n+1},\ ...,\ x_{i-1}\right) = \lambda P_{ML}\left(x_i|x_{i-n+1},\ ...,\ x_{i-1}\right) + (1\ -\ \lambda)P\left(x_i|x_{i-n+2},\ ...,\ x_{i-1}\right)$$

Điều này cho phép chúng ta xử lý các mô hình có độ chính xác cao nhưng lại hiếm gặp, và các mô hình ít chính xác hơn nhưng phổ biến hơn. Đây là một khái niệm về việc kết hợp (ensembling) các mô hình khác nhau, mỗi mô hình có ưu điểm riêng.

Mặc dù mô hình n-gram có thể không được sử dụng rộng rãi trừ khi bạn muốn xử lý các tập dữ liệu lớn, nhưng một số phương pháp làm mịn có thể hữu ích ngay cả khi bạn sử dụng các mô hình khác và kết hợp chúng lại với nhau.

Để xác định hệ số nội suy (interpolation coefficient), một cách đơn giản là đặt một giá trị cố định cho xác suất mà chúng ta sử dụng cho mỗi lần. Ví dụ, chúng ta có thể đặt Lambda là 0.8 và Lambda là 0.2, sau đó kết hợp hai giá trị này lại với nhau.

Smoothing Methods (e.g. Goodman 1998)

Một trong những phương pháp phổ biến để làm mịn là **additive smoothing**. Cách hoạt động của phương pháp này là thêm giá trị α vào cả tử số và mẫu số. Khi số lượng dữ liệu tăng lên, chúng ta sẽ tiến gần hơn đến phân phối thực tế.

$$P\left(x_i|x_{i-n+1},\ ...,\ x_{i-1}\right) := \frac{c\left(x_{i-n+1}, ..., x_i\right)+\alpha \cdot P\left(x_i|x_{i-n+2}, ..., x_{i-1}\right)}{c\left(x_{i-n+1}, ..., x_{i-1}\right)+\alpha}$$

Khi số lượng dữ liệu $c\left(x_{i-n+1},\ ...,\ x_i\right)$ tăng lên, chúng ta sẽ dựa nhiều hơn vào phân phối bậc cao hơn. Còn nếu tổng số lượng dữ liệu $c\left(x_{i-n+1},\ ...,\ x_i\right)$ nhỏ, chúng ta sẽ dựa nhiều hơn vào phân phối bậc thấp.

Ngoài ra, còn có phương pháp **discounting**, trong đó chúng ta trừ một giá trị từ các số lượng. Ví dụ, có thể trừ 0.5 từ mỗi số lượng, điều này giúp phù hợp hơn với phân phối dài đuôi của ngôn ngữ tự nhiên. Phân phối dài đuôi đề cập đến hiện tượng trong một tập hợp từ vựng, một số từ rất phổ biến có tần số xuất hiện cao trong khi phần lớn các từ khác lại xuất hiện với tần suất thấp.

$$P\left(x_i|x_{i-n+1},\ ...,\ x_{i-1}\right) := \frac{c\left(x_{i-n+1}, ..., x_i\right)-d+\alpha \cdot P\left(x_i|x_{i-n+2}, ..., x_{i-1}\right)}{c\left(x_{i-n+1}, ..., x_{i-1}\right)}$$

Giá trị nội suy được tính bằng tổng các discount: $\alpha = \sum_{\left\{\bar{x};\, c\left(x_{i-n+1}, ..., \bar{x}\right)>0\right\}} d$

Khi chúng ta trừ đi một giá trị d từ số lượng xuất hiện, xác suất của các từ phổ biến sẽ giảm xuống và phần xác suất này phân bố lại cho các từ ít phổ biến hơn. Điều này giúp mô hình linh hoạt hơn và có khả năng dự đoán tốt hơn cho những từ chưa thấy, đặc biệt là trong ngữ cảnh mới.

Cuối cùng, **Kneser-Ney smoothing** là một trong những phương pháp làm mịn tiên tiến nhất trước khi mạng nơron ra đời. Phương pháp này bao gồm ba phần chính:

$$P_{KN}(w_n|w_1, \ldots, w_{n-1}) = \frac{C(w_1, \ldots, w_n) - D}{\sum_{w' \in L} C(w_1, \ldots, w_{n-1}, w')} + \lambda(w_1, \ldots, w_{n-1}) P_{KN}(w_n|w_2, \ldots, w_{n-1})$$

1. Giảm giá trị tuyệt đối (Absolute Discounting): Để đảm bảo rằng tổng xác suất vẫn hợp lệ (tổng bằng 1), Kneser-Ney giảm một lượng cố định D từ tất cả các n-gram đã thấy. Điều này tạo ra một khối lượng xác suất để phân bổ cho các n-gram chưa thấy.

2. Nội suy: Khi gặp một n-gram chưa thấy, Kneser-Ney sử dụng các mô hình bậc thấp hơn (như bigram hoặc unigram) để ước lượng xác suất. Xác suất của n-gram được tính bằng cách kết hợp xác suất từ n-gram đã thấy và xác suất từ các mô hình bậc thấp hơn,

với một trọng số gọi là trọng số quay lại (back-off weight) để điều chỉnh xác suất cho các mô hình bậc thấp hơn.

3. Lịch sử từ (Word Histories): Kneser-Ney không chỉ xem xét ngữ cảnh hiện tại mà còn xem xét số lượng ngữ cảnh mà một từ xuất hiện. Điều này có nghĩa là nếu một từ chỉ xuất hiện sau một số ngữ cảnh hạn chế, nó sẽ ít có khả năng xuất hiện trong một ngữ cảnh mới. Xác suất của một từ được điều chỉnh dựa trên số lượng ngữ cảnh mà nó theo sau, thay vì chỉ dựa vào tần suất xuất hiện.

Nguồn: https://smithamilli.com/blog/kneser-ney/

Problems and Solutions

Khi tạo ra các mô hình ngôn ngữ n-gram, chúng ta gặp phải nhiều vấn đề. Đầu tiên, là các mô hình này không thể chia sẻ sức mạnh giữa các từ tương tự, ví dụ như “bought" và “purchase". Giải pháp cho vấn đề này là sử dụng các mô hình ngôn ngữ dựa trên lớp (class-based language model). Các mô hình này nhóm các từ tương tự lại với nhau thành các lớp và xử lý chúng như một đơn vị. Thay vì xem "bought" và "purchase" là hai từ riêng biệt, mô hình sẽ gán chúng vào cùng một lớp, ví dụ như lớp "mua". Khi đó, khi mô hình gặp một trong hai từ này, nó có thể sử dụng thông tin từ lớp để cải thiện khả năng dự đoán cho cả hai từ.

Thứ hai, chúng không thể điều kiện hoá ngữ cảnh với các từ xen kẽ, dẫn đến việc nếu có một từ hiếm trong ngữ cảnh, mô hình sẽ lập tức quay về phân phối unigram và trở lên kém hiệu quả. Giải pháp cho vấn đề này là sử dụng mô hình Skip-gram. Mô hình Skip-gram là một phần của kiến trúc Word2Vec, được thiết kế học các embedding từ bằng cách dự đoán các từ trong ngữ cảnh xung quanh một từ mục tiêu. Bằng cách này, mô hình skip-gram có thể tận dụng thông tin từ các từ xen kẽ để cải thiện khả năng dự đoán. Khi mô hình học được rằng từ "racquet" thường xuất hiện trong ngữ cảnh của "tennis class," nó sẽ có khả năng dự đoán từ này chính xác hơn, ngay cả khi có nhiều từ khác nằm giữa chúng.

Cuối cùng, chúng không thể xử lý được các phụ thuộc xa, xảy ra khi có mối quan hệ ngữ nghĩa và cú pháp giữa các từ nằm cách xa nhau trong câu. Điều này làm cho mô hình không thể nắm bắt ý nghĩa đầy đủ của câu, dẫn đến dự đoán từ tiếp theo không chính xác. Do các mô hình n-gram chỉ xem xét một số từ gần nhất để dự đoán từ tiếp theo. Giải pháp cho vấn đề này là sử dụng mô hình mạng nơ-ron như LSTM, Transformers. Những mô hình này được thiết kế để xử lý các phụ thuộc xa bằng cách giữ lại thông tin các từ trước đó trong một khoảng thời gian dài hơn.

When to Use N-gram Models?

Mặc dù các mô hình ngôn ngữ mạng nơ-ron đạt hiệu suất tốt hơn, nhưng mô hình n-gram lại cực kỳ nhanh để ước lượng và áp dụng. Chúng có thể tốt hơn trong việc mô hình hoá các hiện tượng có tần suất thấp, tức là những từ hoặc cụm từ không xuất hiện thường xuyên trong văn bản. Điều này do mô hình n-gram dựa vào các chuỗi từ ngắn (n từ liên tiếp) và có thể ghi nhớ

các mẫu ngữ nghĩa đơn giản mà không cần có một lượng lớn dữ liệu để học. Một công cụ tiêu chuẩn để ước lượng mô hình ngôn ngữ n-gram là kenlm: https://github.com/kpu/kenlm.

LM Evaluation

Likelihood

Log-likelihood được tính toán bằng cách sử dụng một tập kiểm tra mà lý tưởng là không bao gồm trong dữ liệu huấn luyện. Chúng ta sẽ lấy tất cả tài liệu hoặc câu trong tập kiểm tra và tính toán loga cho xác suất của chúng. Tuy nhiên, chỉ số này không được sử dụng rộng rãi vì nó phụ thuộc vào kích thước của tập dữ liệu. Nếu bạn có tập dữ liệu lớn, số này sẽ lớn hơn. Ngược lại, thì nó nhỏ hơn.

$$LL\left(X_{test}\right) = \sum_{X \in X_{test}} log(P(X))$$

Do đó, chỉ số phổ biến hơn là **Per-word Log Likelihood**, tức là loga của xác suất trên mỗi từ. Chúng ta tính toán chỉ số này bằng cách chia loga xác suất của toàn bộ tập hợp với số lượng từ trong tập hợp đó. Ngoài ra, trong tài liệu nghiên cứu, thường có **Negative Log-likelihood**, vì nó được sử dụng như hàm mất mát trong trường hợp này, giá trị càng thấp càng tốt.

$$WLL\left(X_{test}\right) = \frac{1}{\Sigma_{X \in X_{test}} |X|} \sum_{X \in X_{test}} log(P(X))$$

Entropy

Tại sao gọi là (Cross) Entropy? Bởi vì bạn đang ước lượng mô hình trên tập dữ liệu huấn luyện và sau đó đánh giá nó trên tập dữ liệu kiểm tra. Về mặt khái niệm, cross entropy đo lường sự khác biệt giữa hai phân phối xác suất: phân phối thực tế (thường là phân phối nhãn trong tập dữ liệu) và phân phối dự đoán của mô hình.

Per-word Cross Entropy thường được tính bằng log2 của xác suất chia cho số lượng từ hay đơn vị trong corpus.

$$H\left(X_{test}\right) = \frac{1}{\Sigma_{X \in X_{test}} |X|} - \sum_{X \in X_{test}} log_2(P(X))$$

Có ai biết tại sao lại sử dụng log2 thay vì một log thông thường không? Vâng, đúng vậy, nó được tính bằng bits!

An Aside: LMs and Compression

Bất kì mô hình xác suất nào cũng có thể sử dụng để nén dữ liệu. Điều này dựa trên nguyên tắc cơ bản của lý thuyết thông tin: thông tin bất ngờ hoặc dễ xảy ra cần nhiều bit để mã hoá hơn so với thông tin phổ biến hoặc dễ đoán.

Nguyên tắc nén là sử dụng các mã ngắn hơn cho các đầu vào có xác suất cao hơn, và các mã dài hơn cho các đầu vào ít có khả năng xảy ra. Điều này cho phép tiết kiệm không gian lưu trữ tổng thể vì các mẫu phổ biến thường xuất hiện thường xuyên hơn. Phương pháp nén là sử dụng mã hoá số học (**Arithmetic Coding**), nó hoạt động như sau:

1. Bắt đầu với một khoảng từ 0 đến 1
2. Khi xử lý từng ký tự trong chuỗi đầu vào, khoảng này được chia nhỏ dựa trên xác suất của ký tự đó.
3. Khoảng con tương ứng ký tự hiện tại trở thành khoảng mới cho cho ký tự tiếp theo.
4. Quá trình này tiếp tục cho đến khi xử lý hết chuỗi đầu vào.
5. Cuối cùng một số duy nhất trong khoảng cuối cùng được chọn để làm đại diện cho toàn bộ chuỗi.

Perplexity

Perplexity là thước đo về khả năng dự đoán của một mô hình xác suất, một khái niệm thường gặp trong NLP, nó được định nghĩa là 2 mũ entropy hoặc e mũ log-likelihood ở mức từ. Giá trị Perplexity càng nhỏ thì mô hình càng tốt.

$$PPL(X_{test}) = 2^{H(X_{test})} = e^{-WWL(X_{test})}$$

Để minh họa, chúng ta có một câu: "When a dog sees a squirrel it will usually ___" và mọi người được mời đoán từ tiếp theo. Một số người đã đoán “jump", "run", hay "try". Điều này cho thấy con người không giỏi trong việc dự đoán từ tiếp theo, vì chúng ta không được đào tạo để làm điều đó.

Perplexity thực chất là số lần lấy mẫu từ phân phối xác suất trước khi dự đoán đúng. Trong ví dụ này, nếu từ đúng là "start", perplexity sẽ là 4.66, có nghĩa là mô hình ngôn ngữ sẽ đoán đúng trong khoảng từ 4 đến 5 lần.

Evaluation and Vocabulary

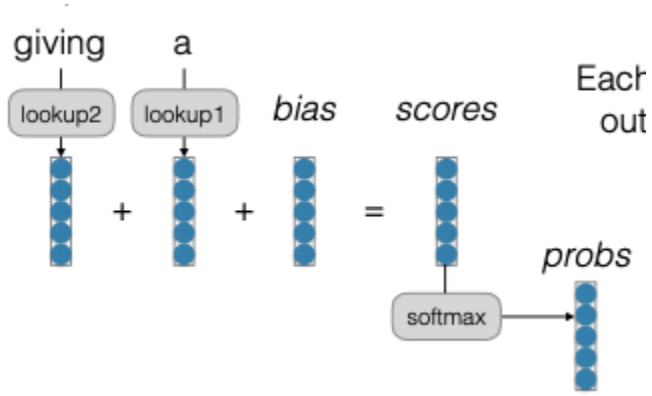
Về việc đánh giá trong từ vựng, để so sánh công bằng, hãy đảm bảo rằng mẫu số giống nhau. Nếu bạn đang tính perplexity, hãy chắc chắn rằng bạn chia cùng một số từ mỗi lần. Việc so sánh các mẫu số khác nhau là không công bằng. Nếu bạn cho phép các từ hoặc các ký tự không xác định, thì bạn cũng cần phải công bằng về điều đó.

An Alternative: Featured Log-Linear Models

An Alternative: Featured Models

Phần này đề cập đến các một số mô hình thay thế tương tự bộ phân loại dùng mạng nơ-ron hay bag-of-words mà tôi đã nói trước đó. Cụ thể, các mô hình thay thế này là các mô hình dựa trên đặc trưng. Chúng tính toán các đặc trưng của mô hình dựa trên ngữ cảnh và từ đó tính toán xác suất, tối ưu hoá trọng số đặc trưng bằng gradient descent.

Ví dụ, khi có một đầu vào, chúng ta sẽ tính toán các đặc trưng. Chúng ta có thể xem xét danh tính của hai từ trước đó, cộng thêm độ lệch (bias), sau đó tổng hợp lại để có được các điểm số và tính toán xác suất, trong đó mỗi vector có kích thước bằng kích thước của từ vựng đầu ra.



Trọng số đặc trưng (feature weights) được tối ưu hóa bằng SGD. Đây cơ bản là một bộ phân loại bag of words nhưng là bộ phân loại bag of words đa lớp cho từ tiếp theo.

Reminder: Training Algorithm

Chúng ta đã thực hiện một thuật toán huấn luyện, trong đó chúng ta tính toán gradient của hàm mất mát tương ứng với các tham số $\partial L_{train}(\theta)/\partial\theta$. Chúng ta có thể sử dụng quy tắc chuỗi (chain rule) và lan truyền ngược để cập nhật tham số và di chuyển theo hướng làm giảm hàm mất mát:

$$\theta \leftarrow \theta - \alpha * \partial L_{train}(\theta)/\partial\theta$$

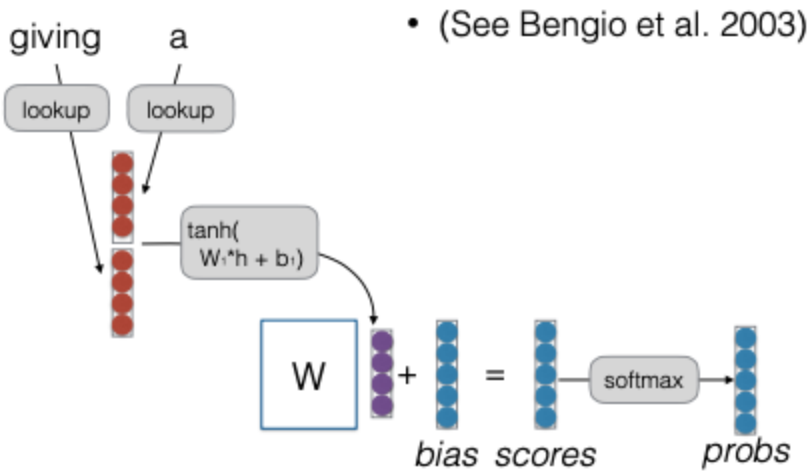
What Problems are Handled?

Phần này đề cập đến việc giải quyết một số vấn đề trong xử lý ngôn ngữ. Mặc dù nó không giải quyết được vấn đề chia sẻ sức mạnh giữa các từ tương tự, nhưng nó đã giải quyết được vấn đề điều kiện hóa dựa trên ngữ cảnh với các từ xen kẽ. Bây giờ, chúng ta có thể điều kiện hóa trực tiếp trên "Doctor" mà không cần phải kết hợp với "Smith". Tuy nhiên, nó vẫn không xử lý tốt các phụ thuộc xa vì chúng ta vẫn bị giới hạn trong ngữ cảnh của mô hình mà tôi vừa mô tả.

Back to Language Modeling

Feed-forward Neural Language Models

Trong quá trình lan truyền thuận của mô hình ngôn ngữ sử dụng mạng nơ-ron, thay vì sử dụng các đặc trưng rời rạc như mô hình bag-of-words, chúng ta sẽ sử dụng các dense embedding. Chúng ta sẽ nối các embedding này lại với nhau và dựa vào chúng để thực hiện các biến đổi ở lớp trung gian để trích xuất đặc trưng, giống như bộ phân loại dựa trên mạng nơ-ron. Sau đó chúng ta nhân với trọng số, cộng thêm một độ lệch (bias) và tính toán các điểm số. Cuối cùng, chúng ta sử dụng hàm softmax để thực hiện phân loại.

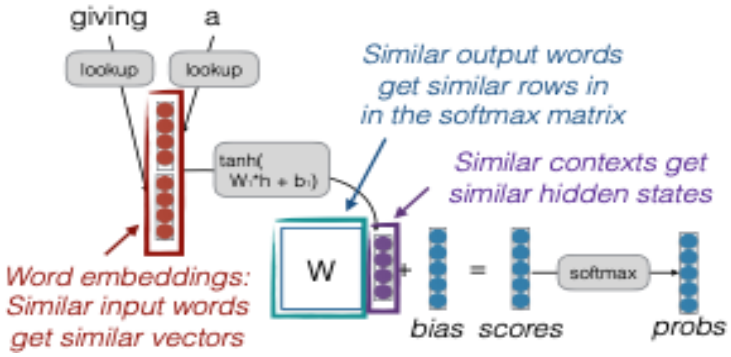


Examples of Combination Features

Các vector từ (word embeddings) nắm bắt các đặc trưng của từ, ví dụ, đặc trưng 1 chỉ ra động từ, đặc trưng 2 chỉ ra từ determiners. Một hàng trong ma trận trọng số (cùng với độ lệch) có thể nắm bắt các kết hợp cụ thể của những đặc trưng này. Ví dụ, hàng thứ 34 trong ma trận trọng số xem xét đặc trưng 1 trong từ thứ hai trước đó và đặc trưng 2 trong từ trước đó.

Where is Strength Shared?

Việc chia sẻ sức mạnh trong mô hình của chúng ta rất quan trọng để làm cho các bộ phân loại mạng nơ-ron hoạt động hiệu quả. Các từ tương tự sẽ có các vector tương tự, và điều này cũng áp dụng cho các từ đầu ra, khi chúng cũng nhận được các hàng tương tự trong ma trận softmax. Như đã đề cập trong bài trước, ma trận này có kích thước bằng số lượng từ vựng nhân với kích thước của vector từ.



Typing Input/Output Embeddings

Một mẹo phổ biến trong mô hình ngôn ngữ là tying embeddings. Điều này có nghĩa là chia sẻ các tham số giữa ma trận tra cứu và ma trận được sử dụng để tính toán softmax. Lợi ích của việc này là hai mặt: thứ nhất, nó cung cấp nhiều dữ liệu huấn luyện hơn để học các embedding, vì chúng ta học cùng một ma trận embedding cho cả ngữ cảnh và dự đoán. Thứ hai, ma trận embedding có thể rất lớn; ví dụ, nếu số lượng từ vựng là 100,000 và kích thước embedding là

512, thì sẽ có 51 triệu tham số. Mặc dù nghe có vẻ không nhiều, nhưng thực sự là rất nhiều để học, vì chúng chỉ được cập nhật khi từ hoặc token đó xuất hiện trong dữ liệu huấn luyện. Điều này giúp tiết kiệm tham số và tăng hiệu quả. Tuy nhiên, nó vẫn không giải quyết được tất cả các vấn đề.

What Problems are Hand

Nó giải quyết được hầu hết các vấn đề: chia sẻ sức mạnh giữa các từ tương tự, và điều kiện hoá dựa trên ngữ cảnh với các từ xen kẽ. Nhưng nó lại không giải quyết được vấn đề về các phụ thuộc xa vì vẫn bị giới hạn về độ dài tổng thể của ngữ cảnh mà chúng đang kết hợp. Chúng ta có thể làm mô hình dài hơn, nhưng điều đó làm mô hình lớn hơn và gây ra nhiều vấn đề khác.

Many Other Potential Designs!

Trong bài sau, tôi sẽ nói về cách giải quyết vấn đề mô hình hóa ngữ cảnh dài. Chúng ta sẽ tìm hiểu cách xây dựng các mạng nơ-ron hồi tiếp (recurrent neural networks), mạng nơ-ron tích chập (convolutional networks) và các mô hình dựa trên attention như Transformer. Hiện nay, Transformers là công nghệ chính được sử dụng, nhưng có nhiều phiên bản của Transformers đã vay mượn ý tưởng từ các mô hình hồi tiếp và tích chập. Gần đây, nhiều mô hình ngữ cảnh dài đã áp dụng ý tưởng từ mạng hồi tiếp, trong khi các mô hình như mô hình giọng nói (speech models) hay mô hình hình ảnh (image models) lại sử dụng ý tưởng từ mạng tích chập.

Other Desiderata of LMs

Calibration (Guo+ 2017)

Khái niệm calibration (hiệu chuẩn) có nghĩa là mô hình biết khi nào nó có câu trả lời đúng, tức là khả năng cung cấp độ tin cậy chính xác cho câu trả lời của nó. Cụ thể hơn, điều này có định nghĩa là xác suất mà mô hình đưa ra câu trả lời phải khớp với xác suất thực tế của việc câu trả lời đó đúng.

Ví dụ, nếu mô hình nói rằng xác suất câu trả lời đúng là 0.7, thì thực tế, câu trả lời đó sẽ đúng 70 lần trong 100 lần. Để hình dung điều này, chúng ta có thể sử dụng một biểu đồ gọi là "reliability diagram" (biểu đồ độ tin cậy). Trong biểu đồ này, chúng ta chia xác suất của mô hình thành các "buckets" (thùng) như từ 0 đến 0.1, từ 0.1 đến 0.2, và cứ thế. Chúng ta tính toán độ tin cậy trung bình trong mỗi thùng và so sánh với độ chính xác thực tế.

Một điều quan trọng cần lưu ý là độ hiệu chỉnh và độ chính xác không nhất thiết phải đi đôi với nhau. Ví dụ, một mô hình có độ chính xác kém (với tỷ lệ sai sót 44.9%) nhưng lại được hiệu chỉnh tốt, có nghĩa là khi nó tự tin về câu trả lời, nó thực sự biết câu trả lời đó. Ngược lại, một mô hình khác có độ chính xác tốt hơn nhưng lại kém hiệu chỉnh, thường rất tự tin nhưng lại không chính xác.

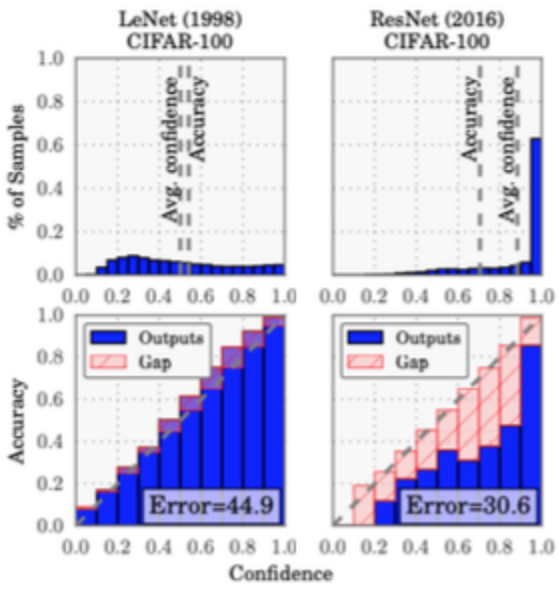


Figure 1. Confidence histograms (top) and reliability diagrams (bottom) for a 5-layer LeNet (left) and a 110-layer ResNet (right) on CIFAR-100. Refer to the text below for detailed illustration.

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} \left| acc(B_m) - conf(B_m) \right|$$

How to Calculate Answer Probability?

Đầu tiên, việc tính xác suất của câu trả lời khá đơn giản khi có một câu trả lời đúng. Ví dụ, nếu bạn muốn mô hình giải quyết các bài toán toán học và chỉ trả về câu trả lời mà không có gì khác, bạn có thể sử dụng xác suất của câu trả lời đó.

Tuy nhiên, nếu có nhiều câu trả lời có thể chấp nhận được, một cách để tính độ tin cậy là tính xác suất của câu trả lời cộng với xác suất của các cách diễn đạt khác của câu trả lời, sau đó tổng hợp xác suất cho tất cả câu trả lời chấp nhận được.

Một phương pháp khác là lấy mẫu nhiều đầu ra và đếm số lần bạn nhận được một câu trả lời cụ thể. Phương pháp này không giải quyết được vấn đề về các cách diễn đạt khác nhau, nhưng nó hữu ích trong trường hợp bạn không thể lấy xác suất từ một số mô hình. Đặc biệt, với các mô hình như Chain of Thought reasoning, nơi mô hình sẽ giải thích chi tiết trước khi đưa ra câu trả lời, bạn có thể lấy mẫu các chuỗi lý luận nhiều lần và xem câu trả lời nào xuất hiện nhiều nhất.

Cuối cùng, bạn có thể hỏi mô hình về độ tự tin của nó, và đôi khi nó sẽ cung cấp một câu trả lời hợp lý. Một nghiên cứu so sánh các phương pháp khác nhau cho thấy rằng việc lấy mẫu nhiều đầu ra là cách tốt nhất nếu bạn không thể tính xác suất trực tiếp.

Efficiency

Một tiêu chí mà ta muốn mô hình ngôn ngữ đạt được là efficiency (sự hiệu quả). Cụ thể, mô hình cần dễ dàng chạy trên phần cứng hạn chế, theo một số metrics nhất định. Một số metrics liên quan đến tính hiệu quả:

1. Số lượng tham số: Chẳng hạn, bạn có thể thấy rằng "đây là mô hình tốt nhất dưới 3 tỷ tham số" hoặc "chúng tôi đã huấn luyện một mô hình với 1 triệu tỷ tham số". Tuy nhiên, số lượng tham số không thực sự có ý nghĩa nhiều về mặt dễ sử dụng mô hình, trừ khi bạn cũng xem xét các yếu tố khác. Ví dụ, 7 tỷ tham số đó là ở độ chính xác 32-bit hay 4-bit? Điều này sẽ ảnh hưởng lớn đến dung lượng bộ nhớ và tốc độ.

2. Sử dụng bộ nhớ: Có hai loại sử dụng bộ nhớ:
- Bộ nhớ chỉ có mô hình: Khi bạn tải mô hình vào bộ nhớ, nó chiếm bao nhiêu không gian.
- Đỉnh tiêu thụ bộ nhớ: Khi bạn chạy mô hình trên một chuỗi có độ dài nhất định, nó tiêu tốn bao nhiêu bộ nhớ?

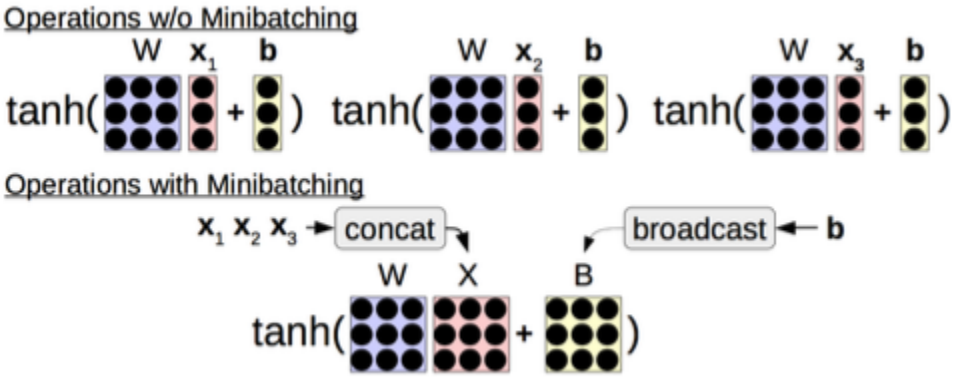
3. Độ trễ: có thể liên quan đến:
- Thời gian để bắt đầu xuất ra token đầu tiên.
- Thời gian để hoàn thành việc xuất ra một chuỗi có độ dài nhất định. Thời gian này thường liên quan đến thời gian mã hóa và thời gian sinh ra chuỗi.

4. Thông lượng: Số lượng câu mà mô hình có thể xử lý trong một khoảng thời gian nhất định.

Efficiency Trick

Efficiency Tricks: Mini-batching

Batching/ Minibatching có ý tưởng cơ bản là trên phần cứng hiện đại, việc thực hiện nhiều phép toán giống nhau cùng một lúc nhanh hơn nhiều so với việc thực hiện từng phép toán một. Mini batching kết hợp các phép toán nhỏ thành một phép toán lớn. Ví dụ, nếu chúng ta muốn tính toán một lớp tuyến tính với một hàm phi tuyến tính sau đó, chúng ta sẽ lấy nhiều đầu vào như X1, X2, X3, kết hợp chúng lại và thực hiện một phép nhân ma trận thay vì thực hiện ba phép nhân vector với ma trận.



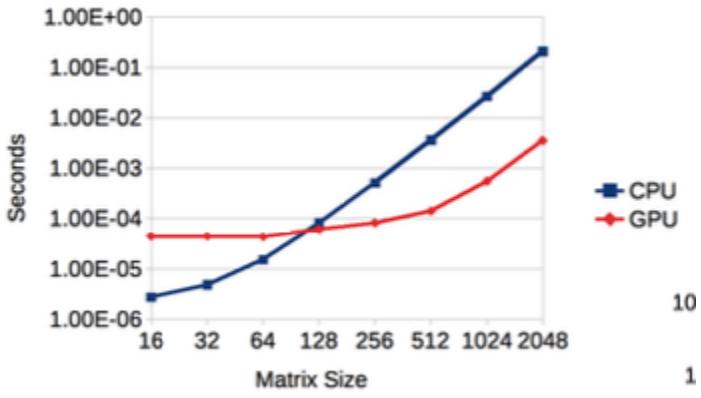
Một điều lưu ý khi làm việc với câu, có nhiều cách tính kích thước minibatch. Nhưng tôi khuyên bạn, nên tính kích thước mini-batch dựa trên số lượng token trong mini-batch. Trước đây, người ta thường tính dựa trên số lượng chuỗi, nhưng vấn đề là 50 chuỗi có độ dài 100 sẽ tiêu tốn nhiều bộ nhớ hơn so với 50 chuỗi có độ dài 5. Điều này dẫn đến các mini batch có kích thước rất khác nhau, gây ra vấn đề về tràn bộ nhớ và độ ổn định trong quá trình học.

GPUs vs CPUs

CPU có thể được ví như một chiếc xe máy, rất nhanh trong việc khởi động và thực hiện nhiều nhiệm vụ một cách nhanh chóng. Ngược lại, GPU giống như một chiếc máy bay; bạn phải chờ đợi lâu trong hàng đợi an ninh và mất thời gian để cất cánh, nhưng khi đã hoạt động, nó lại cực kỳ nhanh.

A Simple Example: How long does a matrix-matrix multiply take?

Khi thực hiện phép nhân ma trận kích thước 16, CPU thực hiện nhanh hơn do chi phí khởi động thấp. Tuy nhiên, khi kích thước ma trận tăng lên, chẳng hạn như 128x128, GPU sẽ nhanh hơn. Đặc biệt, với các ma trận lớn, GPU có thể nhanh hơn tới 100 lần.



Speed Trick

Có một số mẹo để tối ưu hóa hiệu suất GPU mà bạn nên biết. Một sai lầm phổ biến khi tạo mô hình là lặp lại các phép toán nhiều lần. Ví dụ, việc nhân một ma trận với một hằng số nhiều lần là không cần thiết và sẽ gây lãng phí nếu bạn chỉ sử dụng PyTorch mặc định.

Bạn cũng nên giảm số lượng phép toán cần thiết, chẳng hạn như sử dụng phép nhân ma trận-ma trận thay vì ma trận-vector. Một điều quan trọng khác là giảm thiểu việc di chuyển dữ liệu giữa CPU và GPU. Khi bạn cần di chuyển bộ nhớ, hãy cố gắng thực hiện càng sớm càng tốt và càng ít lần càng tốt.

Resources

https://phontron.com/class/anlp2024/lectures/#language-modeling-jan-23
https://smithamilli.com/blog/kneser-ney/