XỬ LÝ NGÔN NGỮ TỰ Nhiên-bao cao

Trí tuệ nhân tạo (Trường Đại học Sư phạm Thành phố Hồ Chí Minh)

MỤC LỤC

DANH MỤC HÌNH ẢNH

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

1.1. Self-attention networks: transformers

Trong phần này, chúng tôi giới thiệu kiến trúc của Transformers. Giống như LSTMs [4] trong Chương 9, Transformers [1] có thể xử lý thông tin xa. Tuy nhiên, khác với LSTMs, Transformers không dựa trên các kết nối lặp (recurrent connections) (mà có thể khó để song song hóa), điều này có nghĩa là Transformers có thể hiệu quả hơn trong việc triển khai quy mô lớn.

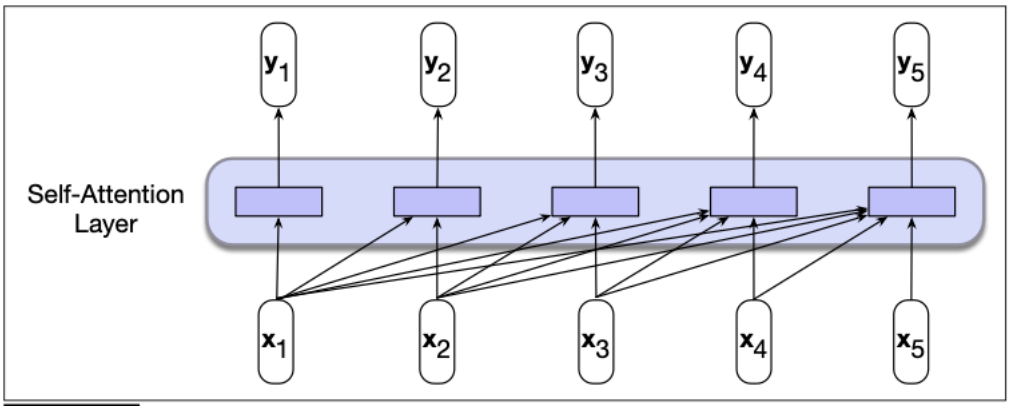
Transformers ánh xạ các chuỗi vectơ đầu vào (x_1, ..., x_n) thành các chuỗi vectơ đầu ra (y_1, ..., y_n) cùng độ dài. Transformers được tạo thành từ các khối Transformers, mỗi khối bao gồm một mạng đa tầng được kết hợp từ các lớp tuyến tính đơn giản, mạng truyền thẳng (Feedforward networks) và lớp self-attention (self-attention), đây là đổi mới chính của Transformers. Self-attention cho phép một mạng trực tiếp trích xuất và sử dụng thông tin từ ngữ cảnh có kích thước bất kỳ mà không cần truyền qua các kết nối lặp trung gian như trong RNNs. Chúng tôi sẽ bắt đầu bằng cách mô tả cách self-attention hoạt động, sau đó quay lại cách nó khớp vào các khối transformer lớn hơn.

Hình 10.1 minh họa quá trình truyền thông tin trong một lớp self-attention (self-attention) đơn lẻ theo hướng tiến, hay còn gọi là self-attention hướng lui. Giống như transformer tổng thể, một lớp self-attention ánh xạ các chuỗi đầu vào (x_1, ..., x_n) thành các chuỗi đầu ra cùng độ dài (y_1, ..., y_n). Khi xử lý từng phần tử trong đầu vào, mô hình có quyền truy cập vào tất cả các phần tử đầu vào cho đến và bao gồm phần tử đang xem xét, nhưng không có quyền truy cập vào thông tin về các phần tử vượt quá phần tử hiện tại. Ngoài ra, tính toán được thực hiện cho mỗi phần tử là độc lập với tất cả các tính toán khác. Điều đầu tiên đảm bảo rằng chúng ta có thể sử dụng phương pháp này để tạo ra các mô hình ngôn ngữ và sử dụng chúng để tạo ra dữ liệu tự sinh, và điều thứ hai có nghĩa là chúng ta có thể dễ dàng song song hóa cả việc suy luận tiến và huấn luyện các mô hình như vậy.

Ở trung tâm của phương pháp dựa trên chú ý là khả năng so sánh một phần tử quan tâm với một tập hợp các phần tử khác một cách tiết lộ sự liên quan của chúng trong ngữ cảnh hiện tại. Trong trường hợp self-attention, tập hợp các so sánh là với các phần

tử khác trong một chuỗi cho trước. Kết quả của những so sánh này sau đó được sử dụng để tính toán đầu ra cho đầu vào hiện tại. Ví dụ, quay trở lại Hình 10.1, tính toán của y_3 dựa trên một tập hợp các so sánh giữa đầu vào x_3 và các phần tử trước đó x_1 và x_2, và với chính x_3. Hình thức đơn giản nhất của so sánh giữa các phần tử trong một lớp self-attention là tích vô hướng (dot product). Hãy gọi kết quả của so sánh này là một điểm số (chúng ta sẽ cập nhật phương trình này để thêm sự chú ý vào việc tính toán điểm số này):

$$\text{score}(x_i, x_j) = x_i \cdot x_j \quad (10.1)$$



Hình 1. Luồng thông tin trong mô hình self-attention hướng tiến (hoặc masked).

Trong quá trình xử lý mỗi phần tử của chuỗi, mô hình chú ý đến tất cả các đầu vào cho đến và bao gồm phần tử hiện tại. Khác với RNNs, các tính toán tại mỗi bước thời gian là độc lập với tất cả các bước khác và do đó có thể được thực hiện song song.

Kết quả của một tích vô hướng là một giá trị vô hướng nằm trong khoảng từ -∞ đến ∞, giá trị càng lớn thì các vector được so sánh càng tương tự nhau. Tiếp tục với ví dụ của chúng ta, bước đầu tiên trong việc tính toán y_3 sẽ là tính toán ba điểm số: $x_3 \cdot x_1$, $x_3 \cdot x_2$ và $x_3 \cdot x_3$. Sau đó, để sử dụng hiệu quả các điểm số này, chúng ta sẽ chuẩn hóa chúng bằng một hàm softmax để tạo ra một vector trọng số, α_{ij}, chỉ ra sự liên quan tỷ lệ của mỗi đầu vào đối với phần tử đầu vào i đang là trọng tâm chú ý hiện tại.

$$\alpha_{ij} = softmax\big(score(x_i, x_j)\big) \forall j \leq i \quad (10.2)$$

$$= \frac{\exp\left(\text{score}\left(x_i, x_j\right)\right)}{\sum_{k=1}^{i} \exp\left(\text{score}\left(x_i, x_k\right)\right)} \forall j \leq i \quad (10.3)$$

Dựa trên các điểm số tỷ lệ trong α, chúng ta sau đó tạo ra giá trị đầu ra y_i bằng cách tính tổng các đầu vào đã được xem xét cho đến thời điểm hiện tại, được trọng số bởi giá trị α tương ứng của chúng.

$$y_i = \sum_{j \leq i} \propto_{ij} x_j$$

Các bước được thể hiện trong Công thức 10.1 đến 10.4 đại diện cho lõi của một phương pháp dựa trên chú ý: một tập hợp các so sánh với các phần tử liên quan trong ngữ cảnh nào đó, một quá trình chuẩn hóa các điểm số đó để tạo ra một phân bố xác suất, sau đó là tổng có trọng số sử dụng phân bố này. Đầu ra y là kết quả của quá trình tính toán trực tiếp này trên các đầu vào.

Loại chú ý đơn giản này có thể hữu ích và thực tế chúng ta đã thấy trong Chương 9 làm thế nào để sử dụng ý tưởng đơn giản này của chú ý cho các mô hình encoder – decoder dựa trên LSTM trong dịch máy.

Tuy nhiên, transformer cho phép chúng ta tạo ra một cách biểu diễn phức tạp hơn về cách các từ có thể đóng góp vào việc biểu diễn các đầu vào dài hơn. Hãy xem xét ba vai trò khác nhau mà mỗi input embedding đóng trong quá trình chú ý.

Dưới vai trò hiện tại của sự chú ý khi được so sánh với tất cả các đầu vào trước đó. Chúng ta sẽ gọi vai trò này là truy vấn (query).

Với vai trò là một đầu vào trước đó được so sánh với sự chú ý hiện tại. Chúng ta sẽ gọi vai trò này là khóa (key).

Và cuối cùng, với vai trò là một giá trị (value) được sử dụng để tính toán đầu ra cho sự chú ý hiện tại.

Để thu thập ba vai trò khác nhau này, transformers giới thiệu ma trận trọng số W^Q, W^K và W^V. Những trọng số này sẽ được sử dụng để chiếu mỗi vector đầu vào x_i thành một biểu diễn cho vai trò của nó là khóa, truy vấn hoặc giá trị.

$$q_i = W^Q x_i;\ k_i = W^K x_i;\ v_i = W^V x_i \qquad (10.5)$$

Các đầu vào x và đầu ra y của transformers, cũng như các vector trung gian sau các lớp khác nhau, đều có cùng số chiều 1 × d. Trong lúc này, hãy giả sử số chiều của các ma trận biến đổi là $W^Q \in R^{d\times d}$, $W^K \in R^{d\times d}$ và $W^V \in R^{d\times d}$. Sau này, chúng ta sẽ cần các số chiều riêng biệt cho các ma trận này khi giới thiệu multi-headed attention, vì vậy hãy ghi chú rằng chúng ta sẽ có một số chiều d_k cho các vector khóa và truy vấn, và một số chiều d_v cho các vector giá trị, cả hai số chiều này hiện tại chúng ta sẽ đặt là d. Trong công trình gốc về transformer (Vaswani et al., 2017), d là 1024.

Với những chiếu này, điểm số giữa sự chú ý hiện tại, x_i, và một phần tử trong ngữ cảnh trước đó, x_j, bao gồm một tích vô hướng giữa vector truy vấn q_i của nó và các vector khóa k_j của phần tử trước đó. Tích vô hướng này có hình dạng phù hợp vì cả truy vấn và khóa đều có số chiều 1 × d. Hãy cập nhật phép tính so sánh trước đó của chúng ta để phản ánh điều này, thay thế Công thức 10.1 bằng Công thức 10.6:

$$\text{score}(x_i,x_j) = q_i \cdot k_j \qquad (10.6)$$

Phép tính softmax kế tiếp dẫn đến $\alpha_{i,j}$ vẫn giữ nguyên, nhưng phép tính đầu ra cho y_i hiện tại dựa trên tổng có trọng số qua các vector giá trị v.

$$y_i = \sum_{j\le i} \propto_{ij} v_j \qquad (10.7)$$

Hình 10.2 minh họa phép tính này trong trường hợp tính toán đầu ra thứ ba y3 trong một chuỗi.

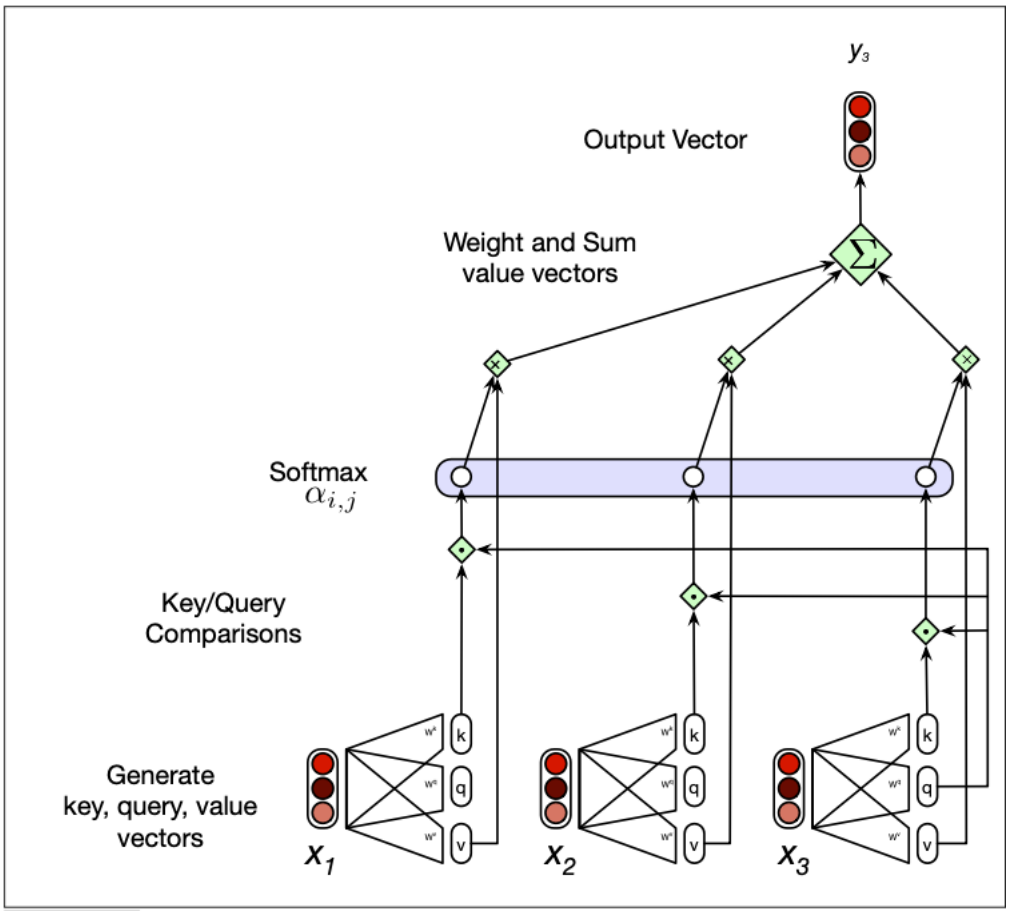
Kết quả của tích vô hướng có thể là một giá trị rất lớn (dương hoặc âm) tùy ý. Việc tính lũy thừa các giá trị lớn như vậy có thể dẫn đến vấn đề về số học và làm mất mát hiệu suất của các gradient trong quá trình huấn luyện. Để tránh điều này, tích vô hướng cần được tỉ lệ phù hợp. Phương pháp tích vô hướng được tỉ lệ sử dụng phép chia kết quả của tích vô hướng cho một hệ số liên quan đến kích thước embeddings trước khi

đưa chúng qua hàm softmax. Một phương pháp thông thường là chia tích vô hướng cho căn bậc hai của số chiều của các vector truy vấn và khóa (d_k), dẫn đến việc cập nhật hàm điểm số của chúng ta một lần nữa, thay thế Công thức 10.1 và Công thức 10.6 bằng Công thức 10.8:

$$score(x_i, x_j) = \frac{q_i k_j}{\sqrt{d_k}} \qquad (10.8)$$

Mô tả quá trình self-attention này đã từ quan điểm tính toán một đầu ra duy nhất tại một bước thời gian i. Tuy nhiên, vì mỗi đầu ra y_i được tính toán độc lập, toàn bộ quá trình này có thể được song song hóa bằng cách tận dụng các thuật toán nhân ma trận hiệu quả bằng cách đóng gói các input embeddings của N thành phần trong chuỗi đầu vào thành một ma trận duy nhất $X \in R^{N\times d}$.. Đó là, mỗi hàng của X là input embedding của một từ trong đầu vào. Sau đó, chúng ta nhân ma trận X với các ma trận khóa K, truy vấn Q và giá trị V (đều có số chiều d×d) để tạo ra các ma trận $Q \in R^{N\times d}$, $K \in R^{N\times d}$, and $V \in R^{N\times d}$, chứa tất cả các vector khóa, truy vấn và giá trị:

$Q=XW^Q$; $K=XW^K$; $V=XW^V$ (10.9)



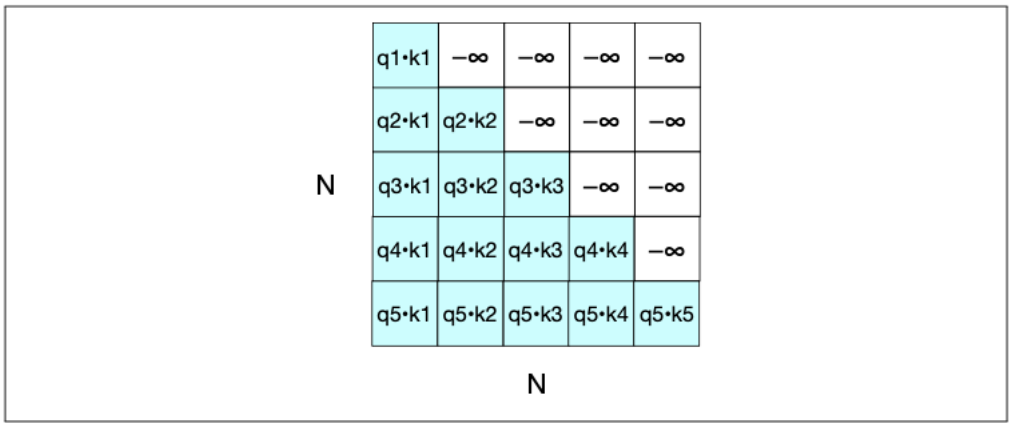
Hình 2. Tính toán giá trị của y_3, phần tử thứ ba trong một chuỗi bằng cách sử dụng self-attention hướng tới nguyên tắc gây ra (từ trái sang phải).

Dựa trên những ma trận này, chúng ta có thể tính toán tất cả các so sánh truy vấn-khóa cần thiết đồng thời bằng cách nhân Q và K^T trong một phép nhân ma trận duy nhất (kết quả có hình dạng N × N; Hình 10.3 thể hiện một biểu đồ minh họa). Đẩy điều này lên một bước xa hơn, chúng ta có thể tỉ lệ các điểm số này, áp dụng hàm softmax và sau đó nhân kết quả với ma trận V, kết quả là một ma trận có kích thước N × d: một biểu diễn vector embedding cho mỗi thành phần trong đầu vào. Chúng ta đã giảm bước self-attention cho một chuỗi đầu vào gồm N thành phần thành phép tính sau đây:

$$SelfAttention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (10.10)$$

Rất tiếc, quá trình này đi quá xa vì tính toán các so sánh trong QK^T dẫn đến một điểm số cho mỗi giá trị truy vấn đối với mọi giá trị khóa, bao gồm cả những giá trị khóa

tiếp theo truy vấn. Điều này không phù hợp trong bối cảnh mô hình ngôn ngữ vì việc đoán từ tiếp theo khá đơn giản nếu bạn đã biết nó. Để khắc phục điều này, các phần tử trong phần tam giác trên của ma trận được gán giá trị 0 (hoặc -∞), loại bỏ bất kỳ thông tin về các từ tiếp theo trong chuỗi. Hình 10.3 miêu tả ma trận QK^T. (chúng ta sẽ thấy trong Chương 11 cách sử dụng các từ trong tương lai cho các nhiệm vụ cần thiết).

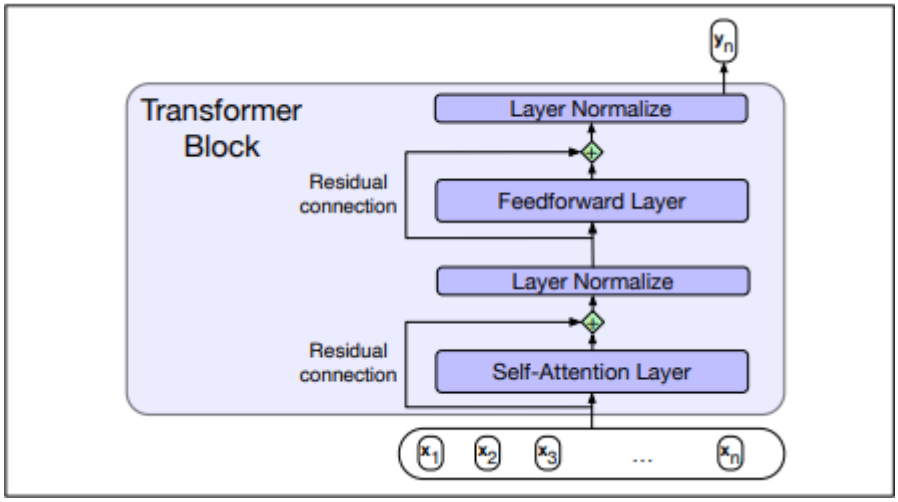


Hình 3. Ma trận QK^T kích thước $N \times N$ hiển thị các giá trị $q_i \cdot k_j$, với phần tam giác trên của ma trận so sánh bị gán giá trị 0 (được đặt thành -∞, mà hàm softmax sẽ chuyển thành giá trị 0).

Hình 10.3 cũng cho thấy rõ ràng rằng sự chú ý tuyến tính với độ dài của đầu vào, vì ở mỗi tầng chúng ta cần tính các tích vô hướng giữa mỗi cặp thành phần trong đầu vào. Điều này làm cho việc sử dụng chú ý trong transformer trở nên rất tốn kém cho đầu vào là các tài liệu dài (như trang Wikipedia hoặc tiểu thuyết), và do đó hầu hết các ứng dụng phải giới hạn độ dài đầu vào, ví dụ như tối đa là một trang hoặc một đoạn văn bản. Tìm kiếm các cơ chế chú ý hiệu quả hơn là một hướng nghiên cứu đang tiếp tục.

1.1.1. Transformer Blocks

Phép tính self-attention đóng vai trò quan trọng trong những gì được gọi là một khối transformer, bên cạnh lớp self-attention còn bao gồm các lớp feedforward bổ sung, kết nối dư thừa và các lớp chuẩn hóa. Kích thước đầu vào và đầu ra của các khối này được phù hợp với nhau để chúng có thể được xếp chồng như cách thực hiện với RNNs xếp chồng.



Hình 4. Minh họa một khối máy biến áp tiêu chuẩn bao gồm một bộ điều khiển

Một khối transformer được theo sau bởi một lớp Fully-connected Feedforward với kết nối dư thừa và layer normalization sau mỗi lớp. Chúng ta đã thấy các lớp feedforward trong Chương 7, nhưng kết nối dư thừa và layer normalization là gì? Trong các mạng sâu, kết nối dư thừa là các kết nối truyền thông tin từ một tầng thấp hơn đến một tầng cao hơn mà không đi qua tầng trung gian. Cho phép thông tin từ hoạt động điều hướng và gradient đi ngược lại bỏ qua một tầng cải thiện quá trình học và cho phép các tầng cấp cao truy cập trực tiếp vào thông tin từ các tầng thấp hơn (He et al., 2016). Kết nối dư thừa trong các transformer được thực hiện bằng cách thêm vector đầu vào của một lớp vào vector đầu ra của nó trước khi điều hướng nó tiếp theo. Trong khối transformer được hiển thị trong Hình 10.4, kết nối dư thừa được sử dụng cả với lớp chú ý và lớp feedforward. Những vector được tổng hợp này sau đó được chuẩn hóa bằng cách sử dụng layer normalization (Ba et al., 2016). Nếu chúng ta coi một lớp như một vector dài của các đơn vị, hàm kết quả được tính toán trong một khối transformer có thể được biểu diễn như sau:

$$z = LayerNorm\big(x + SelfAttention(x)\big) \qquad (10.11)$$

$$y = LayerNorm\big(z + FFN(z)\big) \qquad (10.12)$$

Layer normalization (hoặc layer norm) là một trong nhiều hình thức chuẩn hóa có thể được sử dụng để cải thiện hiệu suất huấn luyện trong các deep neural networks bằng

cách giữ các giá trị của một tầng ẩn trong một khoảng giá trị thuận tiện cho việc huấn luyện dựa trên gradient. Layer normalization là một biến thể của điểm số tiêu chuẩn, hoặc điểm z, từ thống kê được áp dụng cho một tầng ẩn duy nhất. Bước đầu tiên trong layer normalization là tính toán giá trị trung bình, μ, và độ lệch chuẩn, σ, qua các phần tử của vector cần chuẩn hóa. Với một tầng ẩn có kích thước dh, các giá trị này được tính toán như sau.

$$\mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i - \mu)}$$

Với các giá trị này, các thành phần của vector được chuẩn hóa bằng cách trừ giá trị trung bình từng phần tử và chia cho độ lệch chuẩn. Kết quả của phép tính này là một vector mới có giá trị trung bình bằng không và độ lệch chuẩn bằng một.

$$\hat{x} = \frac{(x - \mu)}{\sigma}$$

Cuối cùng, trong việc triển khai tiêu chuẩn của layer normalization, hai tham số có thể học được, γ và β, đại diện cho giá trị tăng và giá trị dịch chuyển, được giới thiệu.

$$LayerNorm = \hat{x}\gamma + \beta$$

1.1.2. Multihead Attention

Các từ khác nhau trong một câu có thể có liên quan đến nhau theo nhiều cách khác nhau đồng thời. Ví dụ, có thể tồn tại các mối quan hệ cú pháp, ngữ nghĩa và diễn ngôn khác nhau giữa động từ và các đối số của nó trong một câu. Sẽ khó cho một khối transformer đơn lẻ học cách ghi nhận tất cả các loại mối quan hệ song song khác nhau giữa các đầu vào của nó. Transformers giải quyết vấn đề này bằng cách sử dụng các lớp self-attention đa đầu vào. Đó là các tập hợp các lớp self-attention, được gọi là đầu, nằm

trong các lớp self-attention đa đầu vào ở cùng một độ sâu trong mô hình, mỗi đầu có bộ tham số riêng của nó.

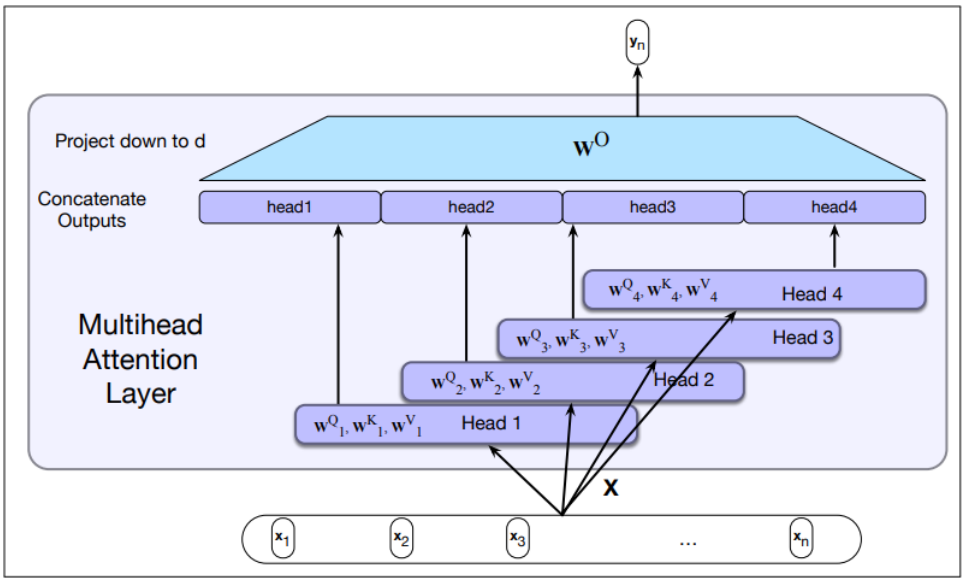
Với các tập hợp tham số riêng biệt này, mỗi đầu có thể học các khía cạnh khác nhau của các mối quan hệ tồn tại giữa các đầu vào cùng một cấp độ trừu tượng.

Để triển khai ý tưởng này, mỗi đầu, i, trong một lớp self-attention được cung cấp một tập hợp riêng biệt các ma trận key, query và value: W_i^K, $W_i^Q$and W_i^V. Chúng được sử dụng để chiếu các đầu vào thành embeddings key, value và query riêng biệt cho mỗi đầu, trong đó các Phép tính self-attention còn lại không thay đổi. Trong chú ý đa đầu vào, thay vì sử dụng kích thước mô hình d được sử dụng cho đầu vào và đầu ra từ mô hình, embeddings key và query có kích thước d_k, và embeddings value có kích thước d_v (trong bài báo gốc về transformer) d_k=d_v=64. Do đó, đối với mỗi đầu i, chúng ta có các lớp trọng số $W_i^Q \in \mathbb{R}^{d\times d_k}$, $W_i^K \in \mathbb{R}^{d\times d_k}$ and $W_i^V \in \mathbb{R}^{d\times d_v}$ và chúng được nhân với các đầu vào được đóng gói thành X để tạo ra $Q \in \mathbb{R}^{N\times d_k}$, $K \in \mathbb{R}^{N\times d_k}$,$V \in \mathbb{R}^{N\times d_v}$. Đầu ra của mỗi đầu trong đầu có hình dạng $N \times d_v$, do đó đầu ra của lớp đa đầu với h đầu bao gồm h vector có hình dạng $N \times d_v$. Để sử dụng những vector này trong việc xử lý tiếp theo, chúng được kết hợp và sau đó giảm xuống kích thước đầu vào ban đầu d . Điều này được thực hiện bằng cách nối các đầu ra từ mỗi đầu và sau đó sử dụng một phép chiếu tuyến tính khác, $W^O \in \mathbb{R}^{hd_v\times d}$, để giảm nó xuống kích thước đầu ra ban đầu cho mỗi token, hoặc tổng cộng $N \times d$ đầu ra.

$$\text{MultiHeadAttention}(X) = (head_1 \oplus head_2 \ldots \oplus head_h)W^O \qquad (10.17)$$

$$Q = XW_i^Q; K = XW_i^K; V = XW_i^V \qquad (10.18)$$

$$head_i = SelfAttention(Q, K, V) \qquad (10.19)$$



Hình 5. Chú ý đa đầu vào: Mỗi lớp chú ý đa đầu vào được cung cấp một tập hợp riêng biệt các ma trận trọng số key, query và value.

Các đầu ra từ mỗi lớp được nối lại và sau đó được chiếu xuống kích thước d , từ đó tạo ra một đầu ra có cùng kích thước với đầu vào để có thể xếp chồng các lớp.

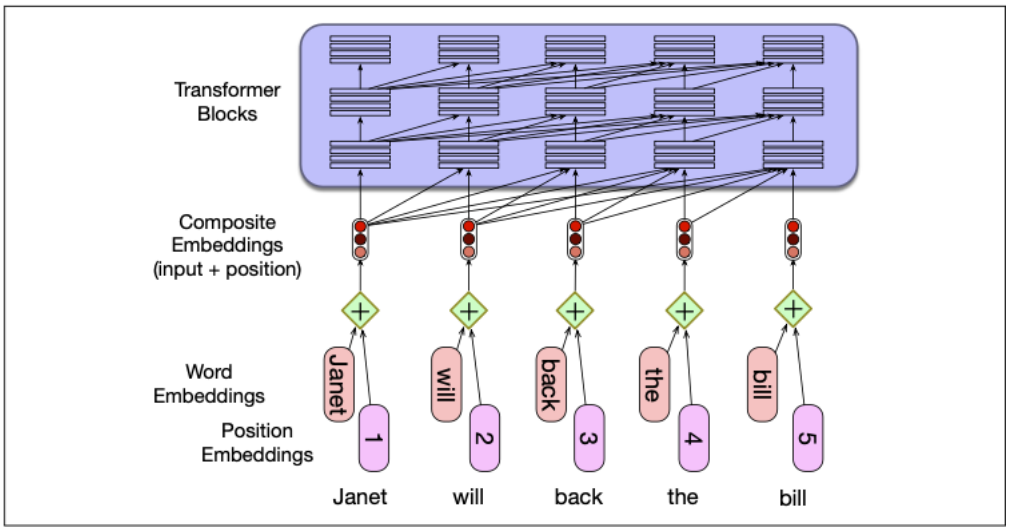
1.1.3. Modeling word order: positional embeddings

Làm thế nào transformer mô hình hóa vị trí của mỗi thành phần trong chuỗi đầu vào? Với RNN, thông tin về thứ tự của các đầu vào được tích hợp vào cấu trúc của mô hình. Rất tiếc, điều này không đúng với transformers; như cách mà chúng ta đã mô tả cho đến nay, các mô hình không có khái niệm về vị trí tương đối hoặc tuyệt đối của các thành phần trong đầu vào. Điều này có thể thấy từ việc nếu bạn xáo trộn thứ tự các đầu vào trong quá trình tính toán chú ý theo hình 10.2, bạn sẽ nhận được chính xác cùng một câu trả lời.

Một giải pháp đơn giản là sửa đổi các input embedding bằng cách kết hợp chúng với các Position Embeddings cụ thể cho mỗi vị trí trong một chuỗi đầu vào.

Chúng ta lấy position Embeddings này từ đâu? Phương pháp đơn giản nhất là bắt đầu với embeddings được khởi tạo ngẫu nhiên tương ứng với mỗi vị trí đầu vào khả dĩ lên đến một độ dài tối đa nào đó. Ví dụ, giống như chúng ta có một embedding cho từ "fish", chúng ta sẽ có embedding cho vị trí 3. Tương tự như word embeddings, các position Embeddings này được học cùng với các tham số khác trong quá trình huấn

luyện. Để tạo ra một input embedding mà chứa thông tin về vị trí, chúng ta chỉ cần thêm word embeddingsng từ đầu vào vào position Embeddings tương ứng của nó. (Chúng ta không nối hai embedding này lại, chúng ta chỉ cộng chúng để tạo ra một vector mới cùng chiều dài.). Embedding mới này được sử dụng làm đầu vào cho các bước xử lý tiếp theo. Hình 10.6 cho thấy ý tưởng này.



Hình 6. Một cách đơn giản để mô hình vị trí là đơn giản là thêm một biểu diễn position Embeddings tuyệt đối vào input embedding để tạo ra embedding mới có cùng số chiều.

Một vấn đề tiềm năng với phương pháp Position Embeddings tuyệt đối đơn giản là có rất nhiều ví dụ huấn luyện cho các vị trí ban đầu trong đầu vào và ít hơn ở các giới hạn độ dài ngoại cùng. Những position Embeddings sau này có thể được huấn luyện kém và có thể không tổng quát tốt trong quá trình kiểm tra. Một phương pháp thay thế cho position Embeddings là chọn một hàm tĩnh mà ánh xạ các đầu vào số nguyên thành các vectơ có giá trị thực sao cho nắm bắt được mối quan hệ bên trong giữa các vị trí. Đó là, nó nắm bắt được việc vị trí 4 trong đầu vào có mối quan hệ gần hơn với vị trí 5 hơn là với vị trí 17. Một kết hợp của các hàm sin và cos với các tần số khác nhau đã được sử dụng trong công trình gốc của transformer. Phát triển các biểu diễn vị trí tốt hơn là một chủ đề nghiên cứu tiếp tục.

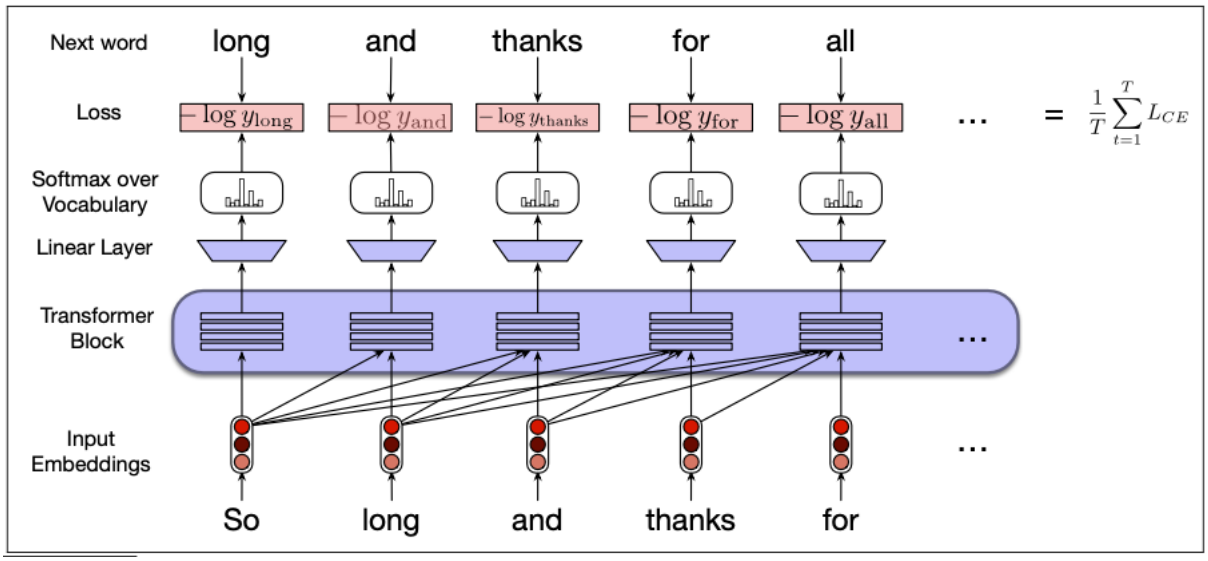
1.2. Transformers as Language Models

Sau khi đã xem xét tất cả các thành phần chính của transformers, chúng ta hãy xem xét cách triển khai chúng như các mô hình ngôn ngữ thông qua việc tự giám sát học. Để làm điều này, chúng ta sẽ sử dụng cùng mô hình tự giám sát mà chúng ta đã sử dụng để huấn luyện mô hình ngôn ngữ RNN trong Chương 9. Cho một tập dữ liệu huấn luyện gồm văn bản đơn thuần, chúng ta sẽ huấn luyện mô hình theo cách tự động dự đoán từ tiếp theo trong một chuỗi yt, bằng cách sử dụng hàm mất mát cross-entropy. Nhớ lại từ Phương trình ?? rằng hàm mất mát cross-entropy cho mô hình ngôn ngữ được xác định bởi xác suất mà mô hình gán cho từ tiếp theo chính xác. Vì vậy, tại thời điểm t, hàm mất mát CE là âm log xác suất mà mô hình gán cho từ tiếp theo trong chuỗi huấn luyện:

$$L_{CE}(\hat{y}_t, y_t) = -log\hat{y}_t[w_{t+1}] \quad (10.20)$$

Giống như trong trường hợp đó, chúng ta sử dụng phương pháp "teacher forcing". Hãy nhớ lại rằng trong teacher forcing, ở mỗi bước thời gian trong quá trình decoding, chúng ta buộc hệ thống sử dụng từ mục tiêu chính xác từ huấn luyện làm đầu vào tiếp theo xt+1, thay vì cho phép nó dựa vào đầu ra của bộ decoder (có thể sai lầm) $\hat{y}$.

Hình 10.7 minh họa phương pháp huấn luyện tổng quát. Ở mỗi bước, dựa trên tất cả các từ trước đó, lớp transformer cuối cùng tạo ra phân phối đầu ra trên toàn bộ từ vựng. Trong quá trình huấn luyện, xác suất được gán cho từ chính xác được sử dụng để tính toán hàm mất mát cross-entropy cho mỗi mục trong chuỗi. Tương tự như RNN, hàm mất mát cho một chuỗi huấn luyện là trung bình của hàm mất mát cross-entropy trên toàn bộ chuỗi.



Hình 7. Huấn luyện một transformer như một mô hình ngôn ngữ.

Lưu ý sự khác biệt chính giữa hình này và phiên bản dựa trên RNN được hiển thị trong Hình ??. Ở đó, tính toán đầu ra và hàm mất mát tại mỗi bước là tuần tự theo tính chất tái diễn trong tính toán các trạng thái ẩn. Với transformers, mỗi mục huấn luyện có thể được xử lý song song vì đầu ra cho mỗi phần tử trong chuỗi được tính toán riêng biệt.

Sau khi được huấn luyện, chúng ta có thể tạo ra văn bản mới theo cách tự động như với các mô hình dựa trên RNN. Hãy nhớ lại từ phần ?? rằng việc sử dụng một mô hình ngôn ngữ để tạo ra từng từ một cách tăng dần bằng cách lấy mẫu từ tiếp theo dựa trên những lựa chọn trước đó của chúng ta được gọi là tạo ra tự động hoặc tạo ra LM theo hướng tự động.

Hãy nhớ lại rằng trong Chương 3, chúng ta đã thấy cách tạo ra văn bản từ một mô hình ngôn ngữ n-gram bằng cách thích nghi một kỹ thuật lấy mẫu được đề xuất vào khoảng thời gian đó bởi Claude Shannon (Shannon, 1951) và các nhà tâm lý học George Miller và Jennifer Selfridge (Miller và Selfridge, 1950). Đầu tiên, chúng ta lấy mẫu một từ ngẫu nhiên để bắt đầu một chuỗi dựa trên tính phù hợp của nó như là một điểm bắt đầu của chuỗi. Sau đó, chúng ta tiếp tục lấy mẫu các từ dựa trên những lựa chọn trước đó cho đến khi đạt đến một độ dài đã được xác định trước, hoặc một từ kết thúc chuỗi được tạo ra.

Quy trình tạo ra từ các mô hình ngôn ngữ transformer cơ bản giống như được mô tả trên trang ??, nhưng được thích nghi với ngữ cảnh của mạng neural:

- Lấy mẫu một từ trong đầu ra từ phân phối softmax kết quả từ việc sử dụng từ bắt đầu câu, <s>, làm đầu vào đầu tiên.

- Sử dụng word embeddings cho từ đầu tiên đó làm đầu vào cho mạng tại bước thời gian tiếp theo, sau đó lấy mẫu từ tiếp theo theo cùng cách.

- Tiếp tục tạo ra từ cho đến khi từ kết thúc câu, </s>, được lấy mẫu hoặc đạt đến giới hạn độ dài cố định.

Kỹ thuật mô hình tự động hóa kỹ thuật này được gọi là tạo ra tự động vì từ được tạo ra ở mỗi bước thời gian được điều kiện trên từ được chọn bởi mạng từ bước trước.

Việc sử dụng một mô hình ngôn ngữ để tạo ra văn bản là một trong những lĩnh vực mà tác động của các mô hình ngôn ngữ neural đối với NLP là lớn nhất. Việc tạo ra văn bản, cùng với việc tạo ra hình ảnh và mã code, tạo thành một lĩnh vực mới của trí tuệ nhân tạo thường được gọi là trí tuệ nhân tạo sinh.

Một cách chính thức hơn, để tạo ra từ một mô hình ngôn ngữ đã được huấn luyện, ở mỗi bước thời gian trong quá trình decoding, đầu ra yt được chọn bằng cách tính toán softmax trên tập các đầu ra có thể có (từ vựng) và sau đó chọn mã thông tin có xác suất cao nhất (argmax).

$$\hat{y} = argmax\, P(w|y \dots y)$$

Việc chọn mã thông tin có xác suất cao nhất để tạo ra ở mỗi bước được gọi là decoding tham lam (greedy decoding); thuật toán tham lam là thuật toán chọn một lựa chọn tối ưu cục bộ, dù cho sau này có thể sẽ không phải là lựa chọn tốt nhất. Chúng ta sẽ thấy trong các phần tiếp theo rằng có các lựa chọn khác cho decoding tham lam.

1.3. Sampling

Trong ngữ cảnh của transformer language models, sampling (lấy mẫu) đề cập đến quá trình tạo ra các đầu ra văn bản từ mô hình. Mô hình transformer thường được sử dụng để sinh văn bản tự nhiên, dịch máy, hoặc hoàn thiện câu văn dở dang.

Có một số phương pháp sampling khác nhau được sử dụng trong transformer language models, bao gồm:

- **Greedy Sampling**: Phương pháp này chọn từ có xác suất cao nhất làm từ tiếp theo trong quá trình tạo ra văn bản. Nó đơn giản và nhanh, nhưng có thể dẫn đến sự không nhất quán và lặp lại trong đầu ra.
- **Random Sampling**: Trong phương pháp này, từ tiếp theo được chọn ngẫu nhiên dựa trên phân phối xác suất của các từ. Điều này tạo ra đầu ra đa dạng hơn so với greedy sampling, nhưng có thể dẫn đến những câu văn không rõ ràng hoặc không liên quan.
- **Top-k Sampling**: Phương pháp này giới hạn việc chọn từ tiếp theo chỉ trong một tập hợp kích thước nhất định của các từ có xác suất cao nhất. Các từ được chọn ngẫu nhiên trong tập hợp này theo phân phối xác suất. Top-k sampling giúp kiểm soát độ đa dạng của đầu ra.
- **Top-p Sampling (hay còn gọi là "nucleus sampling" hoặc "truncated softmax")**: Phương pháp này giới hạn việc chọn từ tiếp theo trong tập hợp của các từ có tổng xác suất lớn hơn một ngưỡng xác định p. Các từ được chọn ngẫu nhiên trong tập hợp này theo phân phối xác suất. Top-p sampling giúp kiểm soát độ đa dạng và độ chắc chắn của đầu ra.

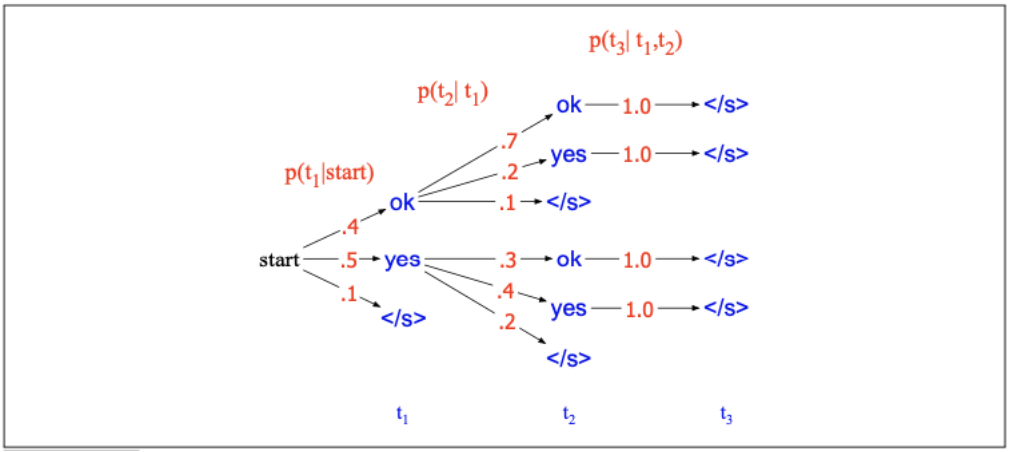
Cả hai phương pháp top-k sampling và top-p sampling được sử dụng để giới hạn sự không chắc chắn và tăng tính đa dạng của đầu ra trong quá trình sinh văn bản.

Thông qua việc lựa chọn phương pháp sampling phù hợp, người dùng và nhà nghiên cứu có thể kiểm soát độ chắc chắn, độ đa dạng và tính sáng tạo của đầu ra từ transformer language models.

1.4. Beam Search

Tìm kiếm tham lam không phải là tối ưu và có thể không tìm ra bản dịch có xác suất cao nhất. Vấn đề là mã thông tin mà trông tốt đối với bộ decoder hiện tại có thể sau này sẽ chứng tỏ là lựa chọn sai!

Hãy xem điều này bằng cách nhìn vào cây tìm kiếm, một biểu đồ đại diện cho các lựa chọn mà bộ decoder thực hiện để tạo ra mã thông tin tiếp theo. Trong đó, chúng ta xem xét vấn đề decoding như một quá trình tìm kiếm không gian trạng thái dựa trên các heuristics và khám phá một cách có hệ thống không gian các đầu ra có thể có. Trong cây tìm kiếm như vậy, các nhánh đại diện cho các hành động, trong trường hợp này là hành động tạo ra một mã thông tin, và các nút đại diện cho các trạng thái, trong trường hợp này là trạng thái đã tạo ra một tiền tố cụ thể. Chúng ta đang tìm kiếm chuỗi hành động tốt nhất, tức là chuỗi chuỗi mục tiêu có xác suất cao nhất. Hình 10.8 mô tả vấn đề này bằng một ví dụ được tạo ra. Lưu ý rằng chuỗi có xác suất cao nhất là ok ok </s> (với xác suất .4 * .7 * 1.0), nhưng một thuật toán tìm kiếm tham lam sẽ không thể tìm ra nó, vì nó lựa chọn sai từ "yes" làm từ đầu tiên vì nó có xác suất cục bộ cao nhất.



Hình 8. Một cây tìm kiếm để tạo ra chuỗi mục tiêu T = t1, t2, ... từ từ vựng V = {yes, ok, <s>}, hiển thị xác suất của việc tạo ra mỗi mã thông tin từ trạng thái đó

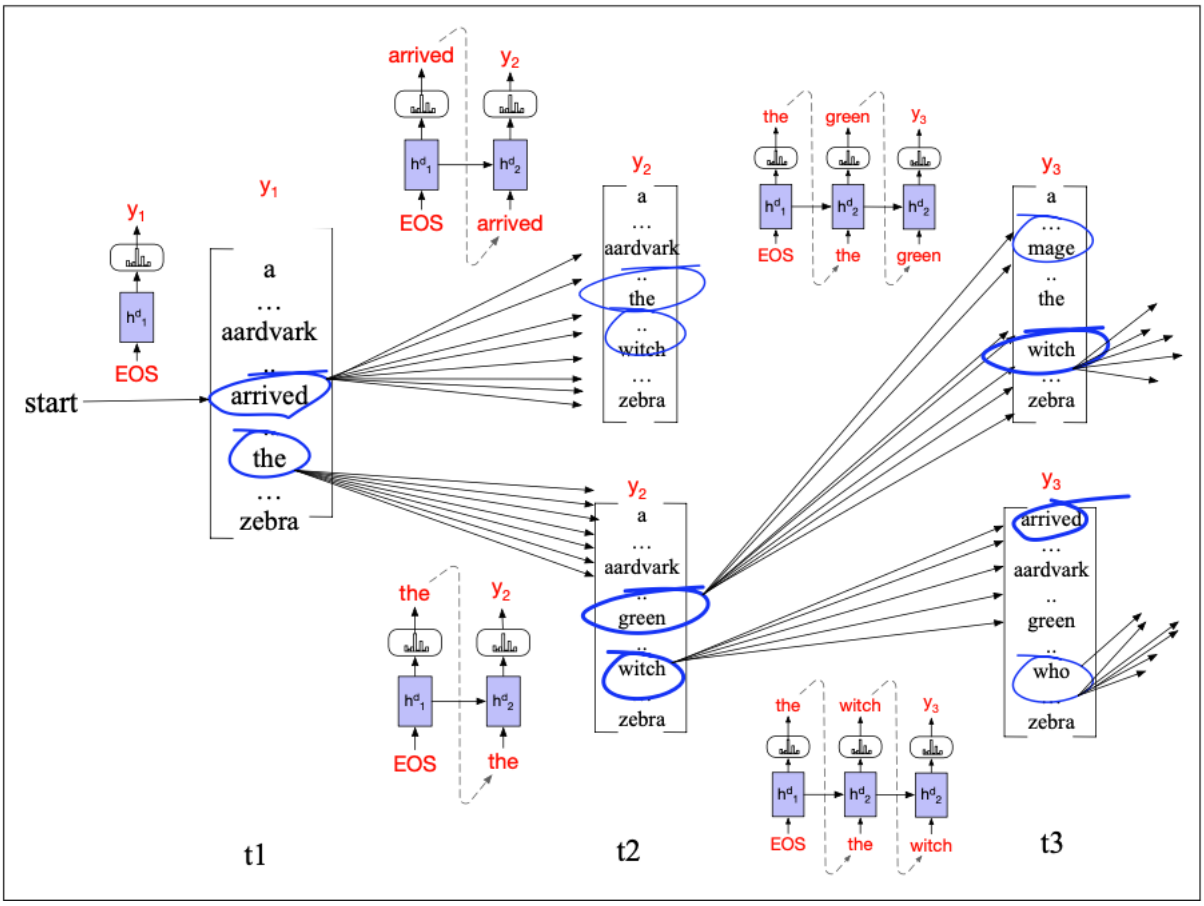
Tìm kiếm tham lam sẽ chọn "yes" tại bước thời gian đầu tiên, tiếp theo là "yes", thay vì chuỗi toàn cục có xác suất cao nhất là ok ok.

Nhớ lại từ Chương 8, chúng ta đã sử dụng tìm kiếm động (thuật toán Viterbi) để giải quyết vấn đề gán nhãn phần từ (part-of-speech tagging). Thật không may, tìm kiếm động không áp dụng được cho các vấn đề tạo ra mã thông tin có phụ thuộc xa giữa các quyết định đầu ra. Phương pháp duy nhất đảm bảo tìm ra giải pháp tốt nhất là tìm kiếm toàn diện: tính toán xác suất của tất cả V^T câu có thể (với một giá trị chiều dài T) điều này rõ ràng quá chậm.

Thay vào đó, trong các vấn đề tạo ra mã thông tin theo chuỗi, thông thường sử dụng một phương pháp gọi là tìm kiếm beam (beam search). Trong tìm kiếm beam, thay vì chọn mã thông tin tốt nhất để tạo ra ở mỗi bước thời gian, chúng ta giữ lại k mã thông tin có thể tại mỗi bước. Kích thước bộ nhớ cố định này k được gọi là độ rộng của "beam" (beam width), theo ẩn dụ của một tia sáng đèn pin có thể điều chỉnh để rộng hoặc hẹp.

Do đó, ở bước đầu tiên của quá trình decoding, chúng ta tính toán softmax trên toàn bộ từ vựng, gán một xác suất cho mỗi từ. Sau đó, chúng ta chọn ra k lựa chọn tốt nhất từ đầu ra softmax này. Kết quả k đầu tiên này là biên giới tìm kiếm và các từ ban đầu này được gọi là giả thuyết (hypotheses). Một giả thuyết là một chuỗi đầu ra, một dịch đến hiện tại, kèm theo xác suất của nó.

Ở các bước tiếp theo, mỗi trong k giả thuyết tốt nhất được mở rộng dần dần bằng cách được chuyển đến các bộ decoder riêng biệt, mỗi bộ sẽ tạo ra một softmax trên toàn bộ từ vựng để mở rộng giả thuyết đến mọi mã thông tin tiếp theo có thể có. Mỗi trong k*V giả thuyết này được đánh điểm bằng P(yi|x,y<i): tích của xác suất lựa chọn từ hiện tại nhân với xác suất của chuỗi đã dẫn đến nó. Sau đó, chúng ta cắt tỉa k*V giả thuyết xuống còn k giả thuyết tốt nhất, vì vậy không bao giờ có nhiều hơn k giả thuyết ở biên giới tìm kiếm và không bao giờ có nhiều hơn k bộ decoder.



Hình 9. Decoding tìm kiếm beam với độ rộng beam k = 2.

Ở mỗi bước thời gian, chúng ta chọn k giả thuyết tốt nhất, tính toán V mở rộng có thể của mỗi giả thuyết, đánh điểm cho k*V giả thuyết có thể có và chọn ra k giả thuyết tốt nhất để tiếp tục. Tại thời điểm 1, biên giới được điền bằng 2 lựa chọn tốt nhất từ trạng thái ban đầu của bộ decoder: "arrived" và "the". Sau đó, chúng ta mở rộng từng giả thuyết đó, tính toán xác suất của tất cả các giả thuyết cho đến nay (arrived the, arrived aardvark, the green, the witch) và tính toán 2 giả thuyết tốt nhất (trong trường hợp này là "the green" và "the witch") để trở thành biên giới tìm kiếm để mở rộng ở bước tiếp theo. Trên các cung, chúng ta hiển thị các bộ decoder mà chúng ta chạy để đánh điểm cho các từ mở rộng (mặc dù vì đơn giản, chúng tôi chưa hiển thị giá trị ngữ cảnh ci đầu vào ở mỗi bước). Hình 10.9 minh họa quá trình này với độ rộng beam là 2 cho câu khởi đầu mà chúng ta đã sử dụng trong Chương 9 và sẽ tiếp tục sử dụng trong Chương 13 để giới thiệu dịch máy, ("The green witch arrived").

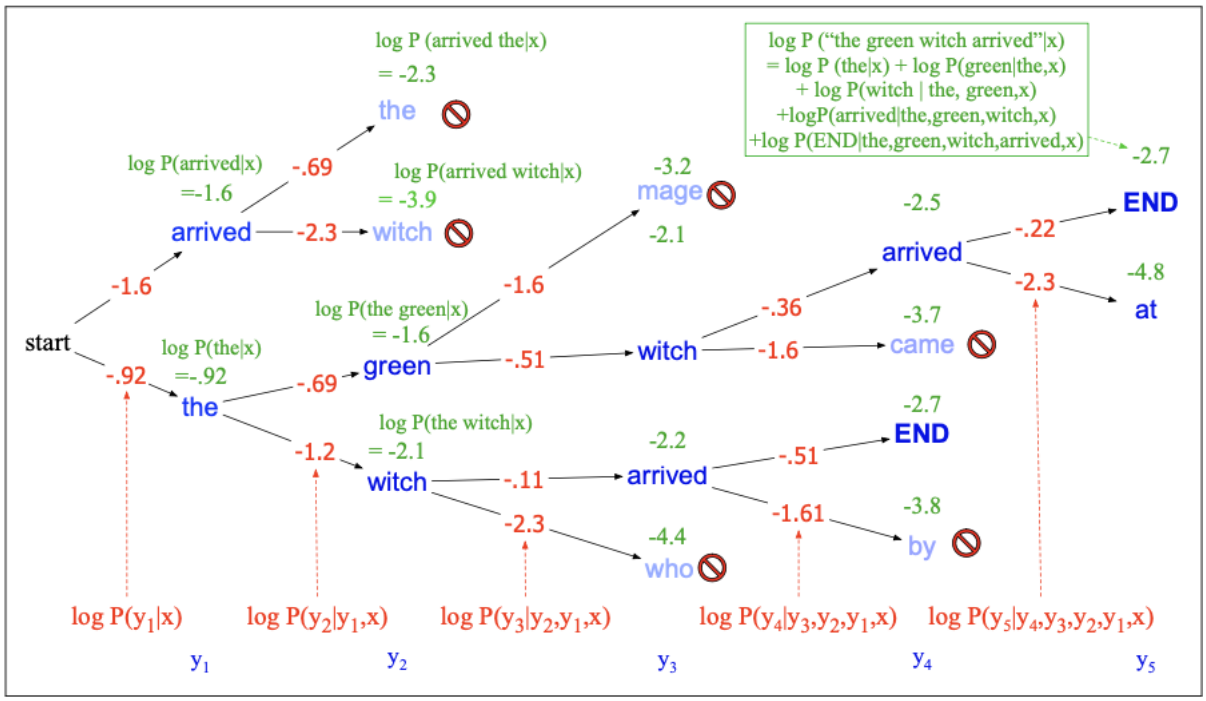
Quá trình này tiếp tục cho đến khi được tạo ra một </s> cho thấy đã tìm thấy một đầu ra ứng viên hoàn chỉnh. Tại điểm này, giả thuyết đã hoàn thành được loại bỏ khỏi biên giới tìm kiếm và kích thước của beam được giảm đi một. Tìm kiếm tiếp tục cho đến khi beam đã được giảm xuống còn 0. Kết quả sẽ là k giả thuyết.

Hãy xem cách tính điểm (scoring) hoạt động chi tiết, tính điểm cho mỗi nút dựa trên xác suất log của nó. Nhớ lại từ Công thức ?? rằng chúng ta có thể sử dụng quy tắc chuỗi xác suất để phân rã p(y|x) thành tích của xác suất của mỗi từ cho trước ngữ cảnh trước đó của nó, chúng ta có thể chuyển đổi thành tổng của log (cho một chuỗi đầu ra có độ dài t):

(Note: The translation of Eq. ?? represents a placeholder for an equation reference that needs to be filled in with the correct equation number.)

$$\begin{aligned} score(y) &= logP(y|x) \\ &= log(P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)\dots P(y_t|y_1,\dots,y_{t\_1},x)) \\ &= \sum_{i=1}^{t} logP(y_i|y_1,\dots,y_{i\_1},x) \end{aligned} \tag{10.22}$$

Do đó, ở mỗi bước, để tính toán xác suất của một câu không hoàn chỉnh, chúng ta đơn giản chỉ cần cộng thêm xác suất log của câu tiền tố cho đến nay với xác suất log của việc tạo ra mã thông tin tiếp theo. Hình 10.10 cho thấy cách tính điểm cho câu ví dụ được hiển thị trong Hình 10.9, sử dụng một số xác suất đơn giản được tạo ra. Xác suất log là âm hoặc bằng 0, và giá trị lớn nhất của hai xác suất log là xác suất có giá trị lớn hơn (gần với 0 hơn).



Hình 10. Điểm số cho decoding tìm kiếm beam với độ rộng beam k = 2.

Chúng tôi duy trì xác suất log của mỗi giả thuyết trong beam bằng cách tăng dần thêm log xác suất của việc tạo ra mỗi mã thông tin tiếp theo. Chỉ có các đường dẫn hàng đầu k được mở rộng vào bước tiếp theo.

Hình 10.11 cung cấp thuật toán.

Một vấn đề phát sinh là giả thuyết đã hoàn thành có thể có độ dài khác nhau. Vì các mô hình thông thường gán xác suất thấp hơn cho chuỗi dài hơn, một thuật toán ngây thơ sẽ chọn các chuỗi ngắn hơn cho y. Điều này không phải là một vấn đề trong những bước đầu của việc decoder; do tính chất của tìm kiếm beam theo chiều rộng, tất cả các giả thuyết được so sánh có cùng độ dài. Giải pháp thông thường cho vấn đề này là áp dụng một hình thức chuẩn hóa độ dài cho mỗi giả thuyết, ví dụ như đơn giản là chia xác suất log âm cho số từ:

$$score(y) = -logP(y|x) = \frac{1}{T}\sum_{i=1}^{t} -logP(y_i|y_1, \ldots, y_{i-1}, x) \qquad (10.23)$$

Tìm kiếm beam là phổ biến trong các nhiệm vụ tạo mô hình ngôn ngữ. Đối với nhiệm vụ như dịch máy (MT), mà chúng ta sẽ trở lại trong Chương 13, chúng ta thường sử dụng độ rộng beam k từ 5 đến 10. Nhưng chúng ta làm gì với k giả thuyết kết quả? Trong

một số trường hợp, điều chúng ta cần từ thuật toán tạo mô hình ngôn ngữ của chúng ta chỉ là giả thuyết tốt nhất duy nhất, vì vậy chúng ta có thể trả về kết quả đó. Trong các trường hợp khác, ứng dụng phía dưới của chúng ta có thể muốn xem tất cả k giả thuyết, vì vậy chúng ta có thể truyền tất cả chúng (hoặc một phần) cho ứng dụng phía dưới với các điểm số tương ứng của chúng.

```
function BEAMDECODE(c, beam_width) returns best paths

  y_0, h_0 ← 0
  path ← ()
  complete_paths ← ()
  state ← (c, y_0, h_0, path)          ;initial state
  frontier ← ⟨state⟩        ;initial frontier

  while frontier contains incomplete paths and beamwidth > 0
    extended_frontier ← ⟨⟩
    for each state ∈ frontier do
      y ← DECODE(state)
      for each word i ∈ Vocabulary do
        successor ← NEWSTATE(state, i, y_i)
        extended_frontier ← ADDTOBEAM(successor, extended_frontier,
                                      beam_width)

    for each state in extended_frontier do
      if state is complete do
        complete_paths ← APPEND(complete_paths, state)
        extended_frontier ← REMOVE(extended_frontier, state)
        beam_width ← beam_width - 1
    frontier ← extended_frontier

  return completed_paths

function NEWSTATE(state, word, word_prob) returns new state

function ADDTOBEAM(state, frontier, width) returns updated frontier

  if LENGTH(frontier) < width then
    frontier ← INSERT(state, frontier)
  else if SCORE(state) > SCORE(WORSTOF(frontier))
    frontier ← REMOVE(WORSTOF(frontier))
    frontier ← INSERT(state, frontier)
  return frontier
```

Hình 11. Decoding tìm kiếm beam.

1.5. Pretraining Large Language models

Pretraining large language models là một phương pháp quan trọng trong transformer language models, nhằm huấn luyện một mô hình ngôn ngữ lớn trước khi áp dụng cho các tác vụ cụ thể như dịch máy, hoàn thiện câu văn, hoặc trả lời câu hỏi.

Trong quá trình này, một mô hình transformer được huấn luyện trên một lượng lớn dữ liệu ngôn ngữ không giám sát từ các nguồn khác nhau trên Internet. Thông thường, mô hình được huấn luyện trên một tập dữ liệu lớn, chẳng hạn như toàn bộ Wikipedia hoặc các văn bản trên Internet.

Quá trình huấn luyện này thường bao gồm hai giai đoạn chính:

Một là, Masked Language Model (MLM): Trong giai đoạn này, một phần ngẫu nhiên các từ trong văn bản đầu vào được che đi (mask). Mô hình cố gắng dự đoán lại các từ bị che đi dựa trên ngữ cảnh xung quanh. Điều này giúp mô hình hiểu được ngữ nghĩa và cấu trúc của câu.

Hai là, Next Sentence Prediction (NSP): Giai đoạn này nhằm giúp mô hình hiểu được mối quan hệ giữa các câu trong văn bản. Hai câu ngẫu nhiên được chọn và mô hình được huấn luyện để dự đoán xem chúng có liên quan nhau (ví dụ: câu thứ hai là câu tiếp theo của câu thứ nhất) hay không.

Trong quá trình huấn luyện, mô hình transformer học cách biểu diễn ngôn ngữ và nắm bắt các mẫu, ngữ cảnh, và kiến thức ngôn ngữ phổ quát từ dữ liệu không giám sát. Sau đó, mô hình này có thể được sử dụng cho nhiều tác vụ khác nhau bằng cách tinh chỉnh (fine-tune) trên dữ liệu có nhãn như dịch máy, phân loại văn bản, hoặc sinh văn bản tự nhiên.

Pretraining large language models đã đạt được những thành công đáng kể trong việc cải thiện khả năng hiểu ngôn ngữ tự nhiên và thực hiện các tác vụ ngôn ngữ phức tạp. Tuy nhiên, việc huấn luyện các mô hình lớn như vậy đòi hỏi một lượng lớn dữ liệu và tài nguyên tính toán.

1.6. Language Models for Zero-Shot Learning

Language models for zero-shot learning là một khái niệm trong transformer language models, mô tả khả năng của mô hình ngôn ngữ để thực hiện các tác vụ mà nó chưa được huấn luyện trực tiếp.

Trong mô hình transformer, khi được huấn luyện trên một lượng lớn dữ liệu không giám sát, mô hình học được cấu trúc ngôn ngữ tổng quát và các mẫu ngôn ngữ phổ biến. Vì vậy, một mô hình transformer huấn luyện tốt có thể áp dụng cho các tác vụ zero-shot, tức là các tác vụ mà không có dữ liệu huấn luyện cụ thể.

Ví dụ, nếu một mô hình transformer được huấn luyện trên dữ liệu tiếng Anh, nó có thể thực hiện dịch máy cho một ngôn ngữ khác mà không cần dữ liệu huấn luyện tiếng Anh-đích cụ thể. Bằng cách sử dụng các phương pháp chuyển đổi ngôn ngữ hoặc kỹ thuật đối ngẫu, mô hình có thể áp dụng các kiến thức ngôn ngữ tổng quát đã học được từ tiếng Anh sang tác vụ dịch máy cho ngôn ngữ mới.

Tương tự, mô hình có thể được sử dụng cho các tác vụ như phân loại văn bản, phát hiện thực thể, hoặc điền từ còn thiếu trong câu mà không cần huấn luyện trực tiếp trên dữ liệu của tác vụ đó. Mô hình sử dụng kiến thức ngôn ngữ tổng quát và khả năng khái quát hóa để áp dụng cho các tác vụ mới mà không yêu cầu dữ liệu huấn luyện cụ thể.

Tuy nhiên, trong zero-shot learning, hiệu suất của mô hình có thể bị hạn chế so với khi huấn luyện trên dữ liệu cụ thể của tác vụ. Điều này có thể được cải thiện bằng cách sử dụng phương pháp tinh chỉnh (fine-tuning) hoặc kết hợp dữ liệu huấn luyện tác vụ cụ thể vào quá trình huấn luyện mô hình.

1.7. Potential Harms from Language Models

Các mô hình ngôn ngữ mạng thần kinh đã được tiền huấn luyện có thể gây nhiều vấn đề tiềm ẩn như đã được thảo luận trong Chương 4 và Chương 6. Nhiều vấn đề này trở nên rõ ràng khi các mô hình ngôn ngữ được điều chỉnh lại cho các nhiệm vụ cụ thể, đặc biệt là những nhiệm vụ liên quan đến việc tạo ra văn bản, chẳng hạn như trong các công nghệ hỗ trợ như hoàn thiện truy vấn tìm kiếm web hoặc gõ tiên đoán cho email (Olteanu et al., 2020).

Ví dụ, các mô hình ngôn ngữ có thể tạo ra ngôn ngữ độc hại. Gehman et al. (2020) cho thấy nhiều loại gợi ý hoàn toàn không độc hại vẫn có thể khiến các mô hình ngôn ngữ lớn tạo ra lời lẽ phân biệt chủng tộc và lạm dụng. Brown et al. (2020) và Sheng et al. (2019) đã chỉ ra rằng các mô hình ngôn ngữ lớn tạo ra các câu thể hiện thái độ tiêu cực đối với các nhóm thiểu số như người da đen hoặc người đồng tính.

Thật vậy, các mô hình ngôn ngữ bị thiên vị theo nhiều cách khác nhau do phân phối dữ liệu huấn luyện của chúng. Gehman et al. (2020) chỉ ra rằng các bộ dữ liệu huấn luyện mô hình ngôn ngữ lớn bao gồm văn bản độc hại được thu thập từ các trang web bị cấm, chẳng hạn như cộng đồng Reddit đã bị đóng cửa nhưng dữ liệu của chúng có thể vẫn còn tồn tại. Ngoài vấn đề về tính độc hại, dữ liệu trên internet được tạo ra một cách chênh lệch bởi các tác giả từ các quốc gia phát triển, và nhiều mô hình ngôn ngữ lớn được huấn luyện trên dữ liệu từ Reddit, trong đó tác giả có xu hướng là nam và trẻ tuổi. Các mẫu dân số bị thiên vị như vậy có thể làm sai lệch việc tạo ra văn bản kết quả khỏi quan điểm hoặc chủ đề của các nhóm dân số thiểu số. Hơn nữa, các mô hình ngôn ngữ có thể làm tăng thiên vị dân số và các thiên vị khác trong dữ liệu huấn luyện, giống như chúng ta đã thấy cho các mô hình Embedding trong Chương 6.

Các mô hình ngôn ngữ cũng có thể là công cụ để tạo ra văn bản sai thông tin, lừa đảo, cực đoan và các hoạt động xã hội gây hại khác (Brown et al., 2020). McGuffie và Newhouse (2020) cho thấy cách các mô hình ngôn ngữ lớn tạo ra văn bản giống như các nhóm cực đoan trực tuyến, có nguy cơ tăng cường các phong trào cực đoan và nỗ lực của chúng để làm cực đoan và tuyển mộ.

Cuối cùng, có những vấn đề quan trọng vềquyền riêng tư. Các mô hình ngôn ngữ, giống như các mô hình học máy khác, có thể rò rỉ thông tin về dữ liệu huấn luyện của chúng. Do đó, một kẻ tấn công có thể trích xuất các cụm từ dữ liệu huấn luyện cá nhân từ một mô hình ngôn ngữ, chẳng hạn như tên, số điện thoại và địa chỉ của một cá nhân cụ thể (Henderson et al., 2017, Carlini et al., 2020). Điều này trở thành một vấn đề nếu các mô hình ngôn ngữ lớn được huấn luyện trên các bộ dữ liệu riêng tư như hồ sơ sức khỏe điện tử (EHRs).

Giảm thiểu tất cả những vấn đề này là một câu hỏi nghiên cứu quan trọng nhưng chưa được giải quyết trong lĩnh vực NLP. Tiền huấn luyện bổ sung (Gururangan et al., 2020) trên các tập con không độc hại có vẻ giảm đôi chút xu hướng tạo ra ngôn ngữ độc hại của mô hình ngôn ngữ (Gehman et al., 2020). Và phân tích dữ liệu được sử dụng để tiền huấn luyện các mô hình ngôn ngữ lớn là quan trọng để hiểu tính độc hại và thiên vị trong việc tạo ra văn bản, cũng như quyền riêng tư. Do đó, việc mô hình ngôn ngữ cung cấp bảng thông tin dữ liệu (trang ??) hoặc thẻ mô hình (trang ??) cung cấp thông tin đáng tin cậy về các bộ dữ liệu được sử dụng để huấn luyện chúng.

CHƯƠNG 2. THỰC NGHIỆM VÀ ĐÁNH GIÁ

2.1. Môi trường thực nghiệm

Đề tài sử dụng dịch vụ cung cấp môi trường trực tuyến miễn phí của Google – Google Colab (viết tắt của Google Colaboratory) với T4 GPU và ngôn ngữ lập trình là Python 3 cùng với một số gói thư viện hỗ trợ xử lý như Numpy, Pandas, Torch, ...

2.2. Cơ sở dữ liệu

Dataset: English-Vietnamese translation [5]

Bộ dữ liệu này cung cấp một bộ gồm 254.090 bộ chứa một câu nguồn tiếng Anh, bản dịch tiếng Việt của nó và chúng tôi lấy 170000 bộ để đào tạo và đánh giá, chia thành 2 tập tin là en_sents và vi_sents.

Một số câu trong tập tin en_sents

```
Please put the dustpan in the broom closet
Be quiet for a moment.
Read this
Tom persuaded the store manager to give him back his money.
Friendship consists of mutual understanding
Are you going to come tomorrow?
See to this matter right away, will you?
I showed my friends these picture postcards.
Mary is the youngest of the three sisters
He has two aunts on his mother's side.
That is what I want to know
Onions can be used in many dishes.
The rumor is not true as far as I know
I told Tom to clean his room.
Has anybody seen my beer mug?
```

Một số câu trong tập tin vi_sents

```
xin vui lòng đặt người quét rác trong tủ chổi
im lặng một lát
đọc này
tom thuyết phục người quản lý cửa hàng trả lại tiền cho anh ta.
tình bạn bao gồm sự hiểu biết lẫn nhau
ngày mai bạn có đến không
nhìn thấy vấn đề này ngay lập tức, bạn sẽ?
tôi đã cho bạn bè của tôi xem những tấm bưu thiếp hình ảnh.
mary là em út trong ba chị em
anh ấy có hai người dì ở bên mẹ.
đó là những gì tôi muốn biết
hành tây có thể được sử dụng trong nhiều món ăn.
tin đồn không đúng như tôi biết
tôi bảo tom dọn phòng.
có ai nhìn thấy cốc bia của tôi không?
```

2.3. Thực nghiệm

Bước 1. Cài đặt môi trường chạy thực nghiệm trên Google Colab.

Xem chi tiết trong Link

- Import các gói thư viện hỗ trợ.

```
import pandas as pd
import numpy as np
import re,string
from underthesea import word_tokenize
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator
from typing import Iterable, List
from gensim.models import KeyedVectors
from torch import Tensor
import torch
import torch.nn as nn
from torch.nn import Transformer
from torch.nn.utils.rnn import pad_sequence
from timeit import default_timer as timer
import math
import warnings
warnings.filterwarnings('ignore')
```

```
# Install the dependencies.
!pip install underthesea
```

- Kết nối thư mục lưu trữ bộ dataset.

```
sfrom google.colab import drive
drive.mount('/content/drive')
```

- Load dữ liệu để đọc từ 2 tập tin en_sents và vi_sents, trong 254090 bộ câu Tiếng Anh và nghĩa Tiếng Việt tương ứng chỉ lấy ra 170000 câu.

Bước 2. Tiền xử lý dữ liệu

```
df.isna().sum()
```

```
en    0
vi    0
dtype: int64
```

```
def preprocessing(df):
  df["en"] = df["en"].apply(lambda ele: ele.translate(str.maketrans('', '', string.punctuation))) # Remove punctuation
  df["vi"] = df["vi"].apply(lambda ele: ele.translate(str.maketrans('', '', string.punctuation)))
  df["en"] = df["en"].apply(lambda ele: ele.lower()) # convert text to lowercase
  df["vi"] = df["vi"].apply(lambda ele: ele.lower())
  df["en"] = df["en"].apply(lambda ele: ele.strip())
  df["vi"] = df["vi"].apply(lambda ele: ele.strip())
  df["en"] = df["en"].apply(lambda ele: re.sub("\s+", " ", ele))
  df["vi"] = df["vi"].apply(lambda ele: re.sub("\s+", " ", ele))

  return df

df = preprocessing(df)
df.head()
```

Tạo token tương ứng

```
# Create source and target language tokenizer.
SRC_LANGUAGE = 'en'
TGT_LANGUAGE = 'vi'

# Place-holders
token_transform = {}
vocab_transform = {}

# Tokenize for vietnames by underthesea
def vi_tokenizer(sentence):
    tokens = word_tokenize(sentence)
    return tokens

token_transform[SRC_LANGUAGE] = get_tokenizer('basic_english')
token_transform[TGT_LANGUAGE] = get_tokenizer(vi_tokenizer)

# helper function to yield list of tokens
def yield_tokens(data_iter: Iterable, language: str) -> List[str]:
    for index,data_sample in data_iter:
        yield token_transform[language](data_sample[language])

# Define special symbols and indices
UNK_IDX, PAD_IDX, BOS_IDX, EOS_IDX = 0, 1, 2, 3
# Make sure the tokens are in order of their indices to properly insert them in vocab
special_symbols = ['<unk>', '<pad>', '<bos>', '<eos>']

for ln in [SRC_LANGUAGE, TGT_LANGUAGE]:
    # Training data Iterator
    train_iter = df.iterrows()
    # Create torchtext's Vocab object
    vocab_transform[ln] = build_vocab_from_iterator(yield_tokens(train_iter, ln),
                                                    min_freq=1,
                                                    specials=special_symbols,
                                                    special_first=True)

# Set UNK_IDX as the default index. This index is returned when the token is not found.
# If not set, it throws RuntimeError when the queried token is not found in the Vocabulary.
for ln in [SRC_LANGUAGE, TGT_LANGUAGE]:
  vocab_transform[ln].set_default_index(UNK_IDX)
```

Bước 3. Định nghĩa model transformer

- Positional Encoding

```
class PositionalEncoding(nn.Module):
    def __init__(self,
                 emb_size: int,
                 dropout: float = 0.1,
                 maxlen: int = 5000):
        super(PositionalEncoding, self).__init__()

        den = torch.exp(- torch.arange(0, emb_size, 2)* math.log(10000) / emb_size)
        pos = torch.arange(0, maxlen).reshape(maxlen, 1)
        pos_embedding = torch.zeros((maxlen, emb_size))
        pos_embedding[:, 0::2] = torch.sin(pos * den)
        pos_embedding[:, 1::2] = torch.cos(pos * den)
        pos_embedding = pos_embedding.unsqueeze(-2)

        self.dropout = nn.Dropout(dropout)
        self.register_buffer('pos_embedding', pos_embedding)

    def forward(self, token_embedding: Tensor):
        return self.dropout(token_embedding + self.pos_embedding[:token_embedding.size(0), :])
```

- Chuyển đổi các tensor đầu vào thành các tensor tương ứng cho token embedding

```
class TokenEmbedding(nn.Module):
    def __init__(self, vocab_size: int, emb_size):
        super(TokenEmbedding, self).__init__()
        self.embedding = nn.Embedding(vocab_size, emb_size)
        self.emb_size = emb_size

    def forward(self, tokens: Tensor):
        return self.embedding(tokens.long()) * math.sqrt(self.emb_size)
```

- Seq2seq Network

```
class Seq2SeqTransformer(nn.Module):
    def __init__(self,
                 num_encoder_layers: int,
                 num_decoder_layers: int,
                 emb_size: int,
                 nhead: int,
                 src_vocab_size: int,
                 tgt_vocab_size: int,
                 dim_feedforward: int = 512,
                 dropout: float = 0.1):
        super(Seq2SeqTransformer, self).__init__()
        self.transformer = Transformer(d_model=emb_size,
                                       nhead=nhead,
                                       num_encoder_layers=num_encoder_layers,
                                       num_decoder_layers=num_decoder_layers,
                                       dim_feedforward=dim_feedforward,
                                       dropout=dropout)
        self.generator = nn.Linear(emb_size, tgt_vocab_size)
        self.src_tok_emb = TokenEmbedding(src_vocab_size, emb_size)
        self.tgt_tok_emb = TokenEmbedding(tgt_vocab_size, emb_size)
        self.positional_encoding = PositionalEncoding(
            emb_size, dropout=dropout)
```

```
def forward(self,
            src: Tensor,
            trg: Tensor,
            src_mask: Tensor,
            tgt_mask: Tensor,
            src_padding_mask: Tensor,
            tgt_padding_mask: Tensor,
            memory_key_padding_mask: Tensor):
    src_emb = self.positional_encoding(self.src_tok_emb(src))
    tgt_emb = self.positional_encoding(self.tgt_tok_emb(trg))
    outs = self.transformer(src_emb, tgt_emb, src_mask, tgt_mask, None,
                            src_padding_mask, tgt_padding_mask, memory_key_padding_mask)
    return self.generator(outs)

def encode(self, src: Tensor, src_mask: Tensor):
    return self.transformer.encoder(self.positional_encoding(
                        self.src_tok_emb(src)), src_mask)

def decode(self, tgt: Tensor, memory: Tensor, tgt_mask: Tensor):
    return self.transformer.decoder(self.positional_encoding(
                      self.tgt_tok_emb(tgt)), memory,
                      tgt_mask)
```

```
def generate_square_subsequent_mask(sz):
    mask = (torch.triu(torch.ones((sz, sz), device=DEVICE)) == 1).transpose(0, 1)
    mask = mask.float().masked_fill(mask == 0, float('-inf')).masked_fill(mask == 1, float(0.0))
    return mask


def create_mask(src, tgt):
    src_seq_len = src.shape[0]
    tgt_seq_len = tgt.shape[0]

    tgt_mask = generate_square_subsequent_mask(tgt_seq_len)
    src_mask = torch.zeros((src_seq_len, src_seq_len),device=DEVICE).type(torch.bool)

    src_padding_mask = (src == PAD_IDX).transpose(0, 1)
    tgt_padding_mask = (tgt == PAD_IDX).transpose(0, 1)
    return src_mask, tgt_mask, src_padding_mask, tgt_padding_mask
```

…

Bước 4. Mô hình huấn luyện

- Mô hình huấn luyện

```
early_stopping = EarlyStopping(tolerance=5, min_delta=0.1)
NUM_EPOCHS = 30
history = {
        "loss": [],
        "val_los": []
        }

for epoch in range(1, NUM_EPOCHS+1):
    start_time = timer()
    train_loss = train_epoch(transformer, optimizer)
    end_time = timer()
    val_loss = evaluate(transformer)
    history['loss'].append(train_loss)
    history['val_los'].append(val_loss)
    print((f"Epoch: {epoch}, Train loss: {train_loss:.3f}, Val loss: {val_loss:.3f}, "f"Epoch time = {(end_time - start_time):.3f}s"))
    # Early Stopping
    early_stopping(train_loss, val_loss)
    if early_stopping.early_stop:
        print("We are at epoch:", epoch)
        break
```

- Lưu mô hình

```
torch.save(transformer.state_dict(), "viEn_transformer.pth")
```

2.4. Kết quả thực nghiệm

Thử nghiệm bằng cách lấy ngẫu nhiên 10 mẫu từ phần còn lại của tập dữ liệu.

```
from random import randint
rand = [randint(170000,200000) for i in range(10)]
test_set = [
        [en_sents[i] for i in rand],
        [vi_sents[i] for i in rand]]

for i in range(len(test_set[0])):
    print('Input English sentence:', test_set[0][i])
    print('Actual Vietnamese Translation:', test_set[1][i])
    print('Predicted Vietnamese Translation:', translate(transformer, test_set[0][i]))
    print("\n")
```

Kết quả như sau:

```
Input English sentence: You were absolutely right
Actual Vietnamese Translation: bạn đã hoàn toàn đúng
Predicted Vietnamese Translation:  bạn hoàn toàn đúng


Input English sentence: Sing along
Actual Vietnamese Translation: hát theo
Predicted Vietnamese Translation:  hát hay


Input English sentence: Tom started shaking uncontrollably.
Actual Vietnamese Translation: tom bắt đầu run rẩy không kiểm soát.
Predicted Vietnamese Translation:  tom bắt đầu run lên cách hỗ trợ


Input English sentence: Tom told me
Actual Vietnamese Translation: tom nói với tôi
Predicted Vietnamese Translation:  tom nói với tôi


Input English sentence: When's Tom due back?
Actual Vietnamese Translation: Khi nào tom trở lại?
Predicted Vietnamese Translation:  khi nào cách xin cách tom do hỗ trợ xin vui lòng
```

```
Input English sentence: It's time you got back to work
Actual Vietnamese Translation: đã đến lúc bạn trở lại làm việc
Predicted Vietnamese Translation:  nó làm một thời gian bạn quay trở lại làm việc


Input English sentence: I eat lunch in that restaurant every day.
Actual Vietnamese Translation: Tôi ăn trưa ở nhà hàng đó mỗi ngày.
Predicted Vietnamese Translation:  tôi ăn trưa ở nhà hàng đó mỗi ngày để được thả ra


Input English sentence: We were attacked by swarms of bees.
Actual Vietnamese Translation: chúng tôi đã bị tấn công bởi bầy ong.
Predicted Vietnamese Translation:  chúng tôi bị tấn công bởi cách ong trạm ong


Input English sentence: Nobody lives with me.
Actual Vietnamese Translation: không ai sống với tôi
Predicted Vietnamese Translation:  không ai sống với tôi cách con tin


Input English sentence: I have some English books.
Actual Vietnamese Translation: tôi có một số sách tiếng anh
Predicted Vietnamese Translation:  tôi có một số sách tiếng anh chính thức
```

CHƯƠNG 3. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

3.1. Kết luận

Chương này đã giới thiệu về transformer và cách áp dụng nó vào các vấn đề ngôn ngữ. Dưới đây là tóm tắt những điểm chính chúng ta đã bàn:

- Transformer là mạng không tuần hoàn dựa trên self-attention (self-attention). Một lớp self-attention ánh xạ các chuỗi đầu vào thành các chuỗi đầu ra cùng chiều dài, sử dụng các đầu self-attention để mô hình hóa cách các từ xung quanh liên quan đến việc xử lý từ hiện tại.

- Một khối transformer bao gồm một lớp chú ý đầu vào tiếp theo là một lớp feedforward, với các kết nối dư và layer normalization sau mỗi lớp. Các khối transformer có thể được xếp chồng để tạo ra mạng sâu và mạnh mẽ hơn.

- Các ứng dụng thông thường của RNN và transformer trong ngôn ngữ bao gồm:

  - Xác suất hóa ngôn ngữ: gán xác suất cho một chuỗi hoặc cho phần tử tiếp theo của một chuỗi dựa trên các từ trước đó.

  - Tạo sinh tự động sử dụng mô hình ngôn ngữ đã được huấn luyện.

  - Gán nhãn chuỗi như gán nhãn phần từ loại, trong đó mỗi phần tử của một chuỗi được gán một nhãn.

  - Phân loại chuỗi, trong đó một văn bản đầy đủ được gán vào một danh mục, như phát hiện thư rác, phân tích cảm xúc hoặc phân loại chủ đề.

Bibliographical and Historical Notes

Mô hình Transformer (Vaswani et al., 2017) được phát triển dựa trên hai dòng nghiên cứu trước đó: self-attention và mạng lưu trữ. Sự chú ý giữa encoder - decoder, ý tưởng sử dụng trọng số mềm trên các encoding của các từ đầu vào để thông báo cho bộ decoding tạo ra (xem Chương 13), được phát triển bởi Graves (2013) trong ngữ cảnh sinh chữ viết tay, và Bahdanau et al. (2015) cho máy dịch. Ý tưởng này được mở rộng thành self-attention bằng cách loại bỏ nhu cầu cho các chuỗi encoding và decoding riêng

biệt và thay vào đó coi sự chú ý là cách để xác định trọng số các mã thông báo trong việc thu thập thông tin từ các tầng thấp đến các tầng cao hơn (Ling et al., 2015; Cheng et al., 2016; Liu et al., 2016). Các khía cạnh khác của Transformer, bao gồm thuật ngữ "key" (khóa), "query" (truy vấn) và "value" (giá trị), được lấy từ mạng lưu trữ, một cơ chế để thêm bộ nhớ đọc-viết bên ngoài vào mạng, bằng cách sử dụng embedding của truy vấn để khớp với các khóa đại diện cho nội dung trong một bộ nhớ kết hợp (Sukhbaatar et al., 2015; Weston et al., 2015; Graves et al., 2014).

3.2. Hướng phát triển

Tiếp tục nghiên cứu và phát triển mô hình transformer đối với bài toán translation để đạt được độ chính xác cao hơn.

TÀI LIỆU THAM KHẢO

[1] Dan Jurafsky and James H. Martin, Speech and Language Processing, 3. e. draft, Ed., 2023.

[2] "Geeksforgeeks," [Online]. Available: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/.

[3] Zhao Zhang, Feng Feng, Tingting Huang, "An Effective Feedforward Neural Network Scheme with Random Weights for Processing Large-Scale Datasets," *ResearchGate ,* 2022.

[4] "Geeksforgeeks," [Online]. Available: https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/. [Accessed 21 12 2023].

[5] "Kaggle," [Online]. Available: https://www.kaggle.com/datasets/hungnm/englishvietnamese-translation/data. [Accessed 19 12 2023].