TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

———————— * ————————

Báo Cáo Môn Học

HỌC SÂU

ÁP DỤNG MÔ HÌNH BIỂU DIỄN NGỮ CẢNH HAI CHIỀU TIỀN HUẤN LUYỆN CHO NGÔN NGỮ TIẾNG vIỆT

Sinh viên thực hiện : **Lương Thành Long**
MSSV :20155970
Giáo viên hướng dẫn: **TS. Trần Việt Trung**

Mục lục

Ngày 12 tháng 5 năm 2020

Chương 1

Cơ sở Lý Thuyết

1.1 Biểu diễn từ dưới dạng vector

1.1.1 giới thiệu

Có một số phương pháp vector hóa một từ đã được giới thiệu ở trên như bag of word hay tf-idf . Tuy cũng đã thu lại được những kết quả đáng kể tuy nhiên cũng như đã đề cập ở trên , các phương pháp này mang tính thống kê , vì vậy mối quan hệ giữa các từ sẽ không có ý nghĩ gì trong những phương pháp này , ví dụ như 2 câu được lấy ví dụ ở trên , mặc dù biểu diễn vector dang bow hay tf-idf đều giống nhau nhưng nghĩa lại hoàn toàn khác nhau . Do đó muốn biểu diễn được ngữ nghĩa của từ và câu ta phải sử dụng phương pháp khác.

1.1.2 Vector hóa từ sử dụng word to vec

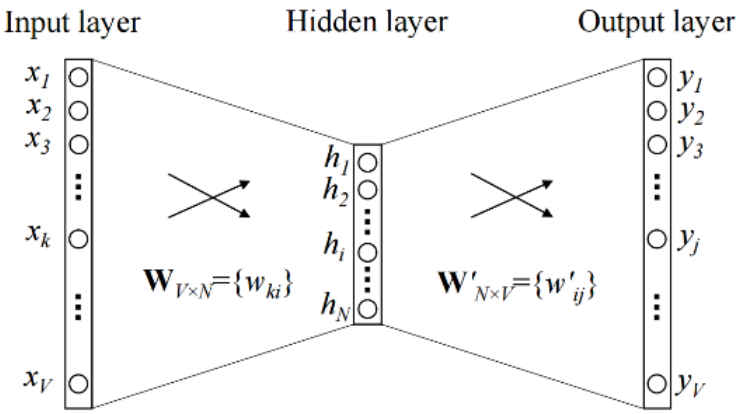
Với ý tưởng chính là :"ngữ nghĩa của một từ sẽ phụ thuộc vào ngữ cảnh của từ đó" , tức là ý nghĩa mà từ biểu thị trong văn bản sẽ phụ thuộc vào những từ xuất hiện cùng nó . Năm 2013 Tomas Mikolov thuộc viện nghiên cứu của google đã đề xuất một phương pháp biểu diễn mới thỏa mãn yêu cầu đó cho từ được gọi là word to vec [**7**]

Ví dụ với hai câu :"*Have a good day*" và :"*Have a great day*" , có thể thấy nó có rất ít sự khác biệt . Có thể thấy sự khác biệt duy nhất là hai từ "*good*" và "*greet*", hai từ này có ý nghĩa khác nhau hoặc giống nhau tùy thuộc và cách dùng nó .Mục tiêu là phải biểu diễn 2 câu này gần giống nhau trong 1 không gian vecto. Hay nói cách khác , khoảng cách cosine[**8**] của 2 vector biểu diễn 2 câu này phải gần nhau

Ý tưởng chính của word to vec chính là việc những từ hay xuất hiện cùng ngữ cảnh với

nhau sẽ có biểu diễn vector giống nhau . Word to vec có thể được học bằng hai phương pháp :

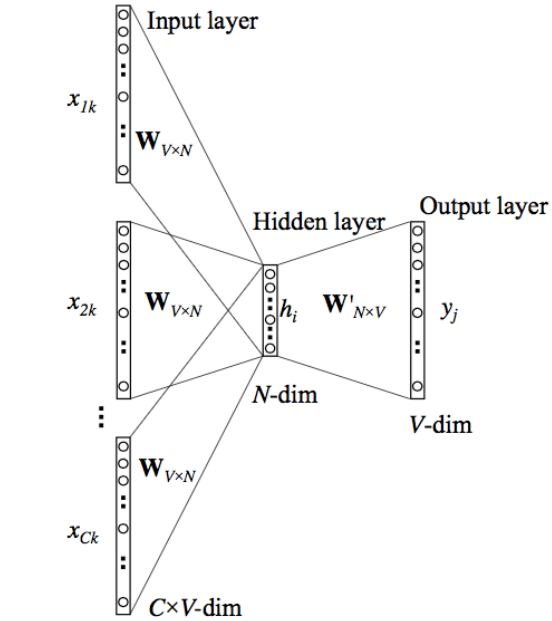
- Common Bag Of Words(CBOW)
  Mô hình này sẽ lấy đầu vào là ngữ cảnh của từ và cố gắng dự đoán ra từ đó.
  Ví dụ dữ liệu đầu vào đã có từ *good* và chúng ta đang cố gắng dự đoán từ tiếp theo là từ *day* . Trong quá trình dự đoán này , ta sẽ học được vector thể hiện của từ *day*.



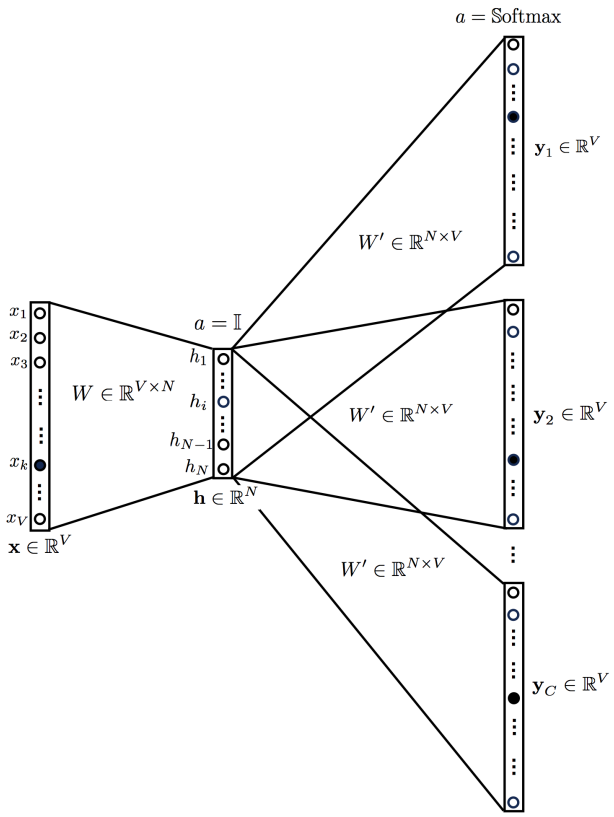
Hình 1.1: ví dụ CBOW với ngữ cảnh 1 từ

Đầu vào hoặc ngữ cảnh là một từ được biểu diễn dưới dạng one hot vector được đi qua một hidden layer của một mạng neural . Ở đây hidden layer không sử dụng các hàm kích hoạt và đầu ra thu được thông qua hàm softmax . Tuy nhiên đầu vào hoặc ngữ cảnh đầu vào như trên chỉ là một từ , trong thực tế ngữ cảnh sẽ có rất nhiều từ chứ không chỉ một , từ được dữ đoán sẽ phụ thuộc vào nhiều từ trong ngữ cảnh với một mức độ khác nhau .

- Mô hình skip-gram : Có thể nhìn thấy mô hình này có phần ngược so với mô hình ở trên , tuy nhiên nó lại có sự hợp lý theo cách khác , đó là các từ có nghĩa giống nhau thì thường sẽ có xác suất xuất hiện với những từ trong tập vocab gần bằng nhau.



Hình 1.2: CBOW với ngữ cảnh nhiều từ



Hình 1.3: Mô hình Skip-gram

Input của mô hình này là một từ và mạng neural sẽ tính ra C(window) xác suất phân tán , mỗi xác xuất là một sự dự đoán cho một từ nào đó nằm trong C từ hay xuất hiện xung quanh input .

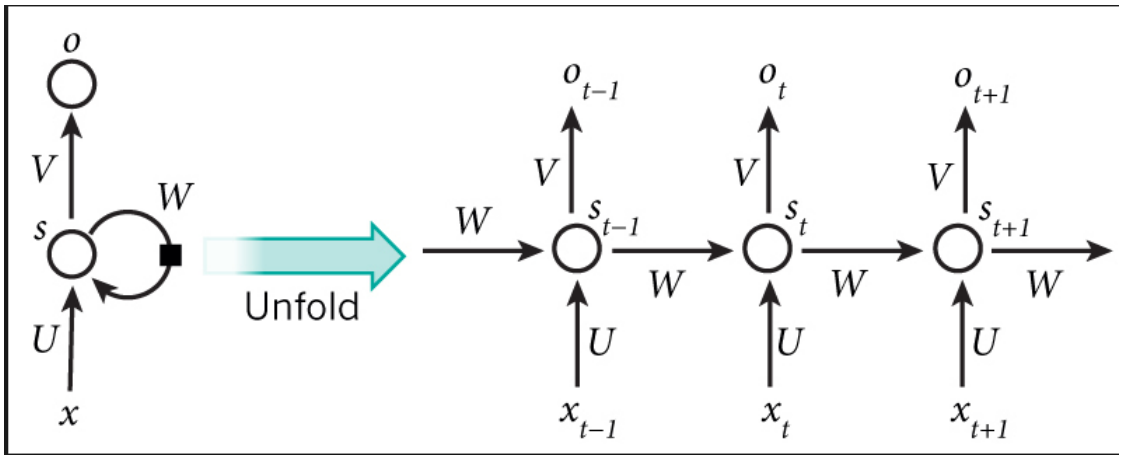
- So sánh : Mỗi mô hình đều có ưu điểm và nhược điểm riêng , tuy nhiên theo như lời tác giả Mikolov, Skip Gram sẽ làm việc tốt hơn với dữ liệu ít và CBOW sẽ làm việc tốt với dữ liệu nhiều . Tuy nhiên khi áp dụng vào tiếng việt thì sẽ có vấn đề lớn mà word to vec không giải quyết được đó là hiện tượng từ nhiều , ví dụ từ *đá* có nhiều nghĩa tùy thuộc vào ngữ cảnh sử dụng , tuy nhiên nếu word to vec được sử dụng để biểu diễn từ thì sẽ có nhiều trường hợp từ bị biểu diễn không đúng với ý nghĩa của nó .

1.2 Mạng neural hồi quy và các biến thể

1.2.1 Giới thiệu

Deep learning có 2 mô hình lớn là Convolutional Neural Network (CNN) cho bài toán có input là ảnh và Recurrent neural network (RNN) cho bài toán dữ liệu dạng chuỗi (sequence), với mạng CNN ứng dụng trong lĩnh vực xử lý ngôn ngữ tự nhiên cũng rất lớn , tuy nhiên mạng RNN mới chính là mô hình chính với các ứng dụng to lớn trong lĩnh vực này

1.2.2 Mô hình và tính toán



Hình 1.4: Mô hình RNN

Trong các mạng neural truyền thống , các đầu vào và đầu ra là độc lập với nhau , tức là nó không có sự liên kiết theo thứ tự . Trong một số bài toán việc các thành phần liên kết theo thứ tự với nhau có vai trò quan trọng , ví dụ bài toán dự đoán từ kế tiếp trong câu thì chúng ta cũng phải dựa vào thứ tự của những từ đứng trước rồi mới đưa ra được dự đoán . Cơ chế hoạt động của mạng RNN cũng như vậy , đầu vào của nó là một chuỗi có thứ tự các thành phần và đâu ra là dự đoán cho thành phần kế tiếp.
$s_t = tanh(Ux_t + Ws_{t-1})$
$\hat{y} = softmax(s_t)$
Với s_t là hidden state tại state t , x_t là biểu diễn vector của từ t (có thể sử dụng word to vec đã trình bày ở trên) và $\hat{y}$ là từ sẽ được dữ đoán ,y_t là đầu ra thực tế . Từ đó ta định nghĩa hàm loss dưới dạng cross- entropy:

$$E_t(y_t, \hat{y}_t = -y_t log\hat{y}_t$$

$$E(y, \hat{Y}) = \sum_t E_t(y_t, \hat{y}_t)$$

$$= -\sum_t y_t log(\hat{y}_t)$$

1.2.3 Lan Truyền ngược liên hồi và hiện tượng mất mát đạo hàm

Cũng như những mạng neural chuyền thẳng khác , mạng RNN cũng dùng cơ chế lan truyền ngược để cập nhật tham số trong mạng . Với hàm loss là tổng các loss của các state thì việc tính đạo hàm của hàm loss sẽ tương đương với

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

Để tính được đạo hàm ta sử dụng quy tắc chuỗi vi phân , quy tắc này được áp dụng trong giải thuật lan truyền ngược liên hồi:

$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V}$$

$$= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{V}$$

$$= (\hat{y_3} - y_3) \bigotimes s_3$$

Trong đó $z_3 = Vs_3$, có thể thấy V chỉ phụ thuộc vào giá trị hiện tại của state đó , việc tính đạo hàm cho V chỉ đơn giản là phép nhân ma trận , tuy nhiên với W và U thì lại khác:

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y_3}} \frac{\partial \hat{y_3}}{\partial s_3} \frac{\partial s_3}{\partial W}$$

Với $s_3 = tanh(Ux_3 + Ws_2)$ sẽ phụ thuộc vào s_2 , tương tự s_2 sẽ phụ thuộc vào s_1 :

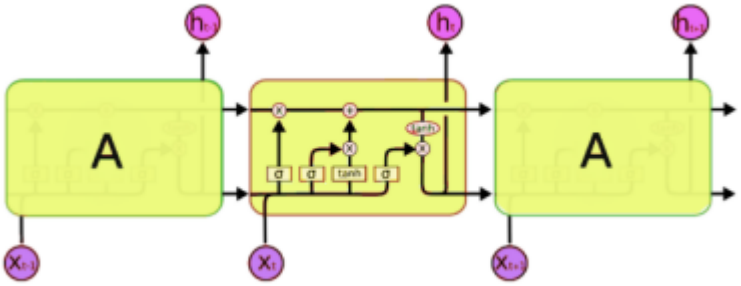
$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y_3}} \frac{\partial \hat{y_3}}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W}$$

Tương tự với E_2 và E_1 mà $E = \sum_t E_t$ nên muốn tìm đạo hàm của loss với W ta cần lan truyền ngược từ E_3 về E_1 và cộng tổng các đạo hàm đó lại . Sự khác biện giữa phương pháp này với mạng neural truyền thống là các mạng truyền thống không có sự chia sẻ tham số nên ta không cần phải *liên hồi*

Nhìn vào chuỗi chain rule , ta có thể thấy nếu số lượng state quá lớn thì số lượng tích dạng $\frac{\partial s_k}{\partial s_{k-1}} \frac{\partial s_{k-1}}{\partial s_{k-2}}$ sẽ rất lớn mà những hàm này đều là hàm tanh và có đạo hàm nhỏ hơn 1 nên đạo hàm của loss theo W sẽ tiến tới 0 và gây ra hiện tượng mất mát đạo hàm . Trên thực tế người ta thấy rằng chỉ khoảng 3 state là mạng đã bắt đầu học không tốt , từ đó các biến thể của RNN ra đời nhằm khắc phục hiện tường trên

1.2.4 Các biến thể của mạng RNN

- Mạng bộ nhớ dài ngắn (Long Short Term Memory networks)
  Thường được gọi là LSTM - là một dạng đặc biệt của RNN, nó có khả năng học được các phụ thuộc xa.LSTM được giới thiệu bởi Hochreiter Schmidhuber (1997)[**9**], và sau đó đã được cải tiến và phổ biến bởi rất nhiều người trong ngành. Chúng hoạt động cực kì hiệu quả trên nhiều bài toán khác nhau nên dần đã trở nên phổ biến như hiện nay.



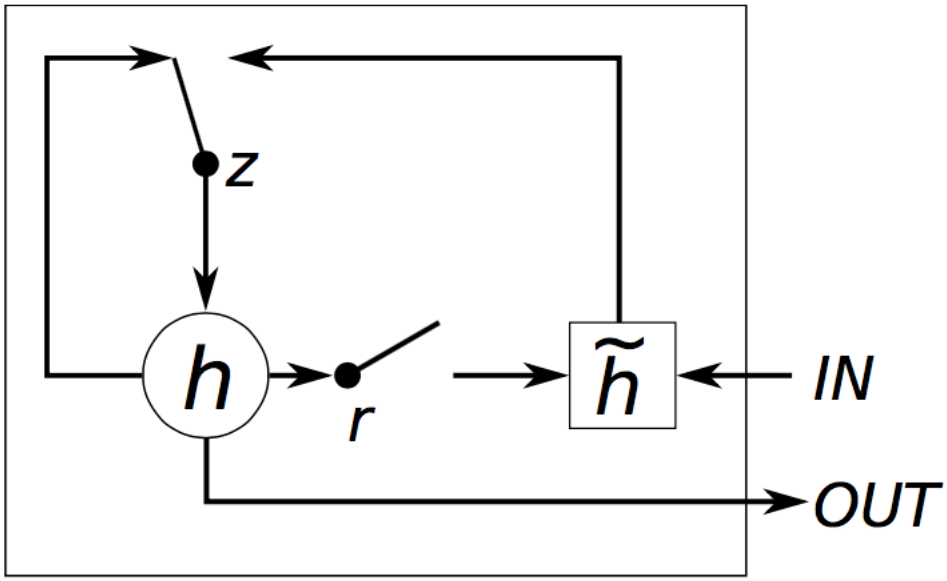
Hình 1.5: Mô hình LSTM

Chìa khóa của LSTM là trạng thái tế bào (cell state) - chính là đường chạy thông ngang phía trên của hình vẽ . Trạng thái tế bào là một dạng giống như băng truyền . Nó chath xuyên suốt tất cả các mắt xích (các nút mạng) và chỉ tương tác tuyến tính đôi chút . Vì vậy mà thông tin có thể dễ dang truyền đi thông suốt mà không sợ bị thay đổi . LSTM có khả năng bỏ đi hoặc thêm vào những thông tin cần thiết cho trạng thái tế bào và được điều chỉnh cẩn thận bởi các nhóm được gọi là cổng - nơi sàng lọc thông tin đi qua . Các cổng này bản chất là các hàm tanh hoặc sigmoid với đặc điểm : tùy thuộc vào tham số mà kết quả của hàm sẽ nằm trong khoảng [0-1] tượng trưng cho việc có ít hay nhiều dữ liệu được ghi nhớ hoặc quên đi .

Mặc dù kết quả của mô hình học sâu tương đối tốt , nhưng nhược điểm của mô hình này là yêu cầu dữ liệu huấn luyện lớn mà việc thu thập dữ liệu trong xử lý ngôn ngữ tự nhiên không hề đơn giản .

- Mạng GRU

  Ý tưởng của GRU cũng khá giống với LSTM , GRU chỉ có 2 cổng: cổng thiết lập lại r và cổng cập nhập z. Cổng thiết lập lại sẽ quyết định cách kết hợp giữa đầu vào hiện tại với bộ nhớ trước, còn cổng cập nhập sẽ chỉ định có bao nhiêu thông tin về bộ nhớ trước nên giữa lại. Như vậy RNN thuần cũng là một dạng đặc biệt của GRU, với đầu ra của cổng thiết lập lại là 1 và cổng cập nhập là 0. Cùng chung ý tưởng sử dụng cơ chế cổng điều chỉnh thông tin, nhưng chúng khác nhau ở mấy điểm sau:



Hình 1.6: Mô hình GRU

  - GRU có 2 cổng, còn LSTM có tới 3 cổng.
  - GRU không có bộ nhớ trong c_t và không có cổng ra như LSTM.
  - 2 cổng vào và cổng quên được kết hợp lại thành cổng cập nhập z và cổng thiết lập lại r r sẽ được áp dụng trực tiếp cho trạng thái ẩn trước.
  - GRU không sử dụng một hàm phi tuyến tính để tính đầu ra như LSTM

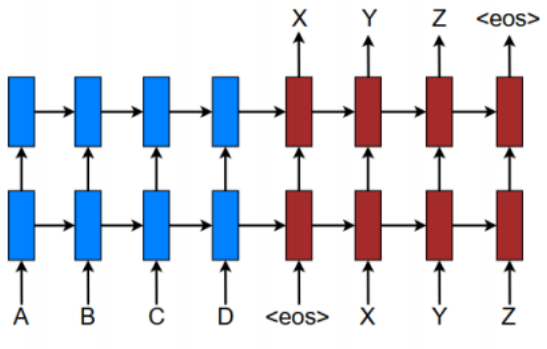
- Cả 2 kiến trúc này đều có thể giải quyết được vấn đề mất mát đạo hàm, nhưng cái nào ngon hơn cái nào? GRU còn khá trẻ tuổi (2014) so với ông chú LSTM của mình (1997) và tiềm năng của nó vẫn chưa được khám phá hết. Tuy nhiên thông qua một số đánh giá thì không cái nào thực sự là ăn được hẳn cái nào. Nhiều bài toán, việc điều chỉnh các siêu tham số (hyperparameters) như số tầng chẳng hạn lại có ý nghĩa hơn là việc chọn kiến trúc LSTM hay GRU. Nhưng cũng có những bài toán mà GRU được chọn bởi nó nhanh hơn hoặc cần ít dữ liệu hơn do GRU ít tham số hơn. Cũng có những lúc nếu bạn có đủ dữ liệu thì LSTM lại tỏ ra mạnh mẽ hơn và đạt được kết quả tốt hơn.

1.3 Cơ chế Attention

1.3.1 Mô hình seq2seq

Mô hình seq2seq gắn liền với bài toán dịch máy (Machine Translation) . Bài toán dịch máy trong quá khứ được giải quyết bằng các phương pháp thống kê hoặc so khớp ngôn ngữ , tuy nhiên với sự bùng nổ của deeplearning một phương pháp dịch máy đã ra đời với hiệu năng đáng kinh ngạc . Mô hình này về cơ bản được chia thành hai phần.

- Encode: Ở pha mã hóa , dữ liệu từ ngôn ngữ nguồn sẽ được mã hóa bằng việc sử dụng những biến thể của mạng neural liên hồi RNN . Dữ liệu sẽ được mã hóa thành một vector cố định chiều ở cuối của mạng neural , vector này sẽ nắm bắt tất cả thông tin cần thiết cho bài toán
- Decode : Ở pha giải mã , vector được mã hóa ở pha mã hóa sẽ được giải mã sang ngôn ngữ đích sử dụng cơ chế của mạng RNN , vector mã hóa đóng vai trò như một hidden state đầu tiên của mạng decode và mô hình hoạt động như một mô hình sinh ngôn ngữ



Hình 1.7: Mô hình encode-decode

1.3.2 Tổng quan cơ chế Attention

Cơ chế attention được sinh ra dựa trên nhu cầu cho việc ghi nhớ những câu dài, một trong những vấn đề cơ bản của máy dịch. Attention cho phép mô hình tập trung vào một hoặc một vài ngữ cảnh địa phương trong câu, đó cũng là nguồn gốc của tên gọi attention. Mục tiêu của attention là tính toán trạng thái tiếp theo của mô hình khi giải mã bằng

cách tính toán trọng số dựa trên trạng thái của bộ mã hóa kết hợp với trạng thái phía trước của bộ giải mã. Các khái niệm mã hóa - giải mã là các khái niệm cơ bản được sử dụng trong máy dịch. Theo đó, bộ mã hóa cho phép mã hóa một chuỗi thành một vector, trong khi bộ giải mã thực hiện việc giải mã bộ vector đó thành chuỗi tương ứng.

Cơ chế attention ban đầu được đề xuất để giải quyết bài toán sequenceto-sequence thông thường bằng cơ chế mã hóa - giải mã. Attention được đưa vào để chỉnh sửa trọng số của vector trong phiên giải mã, giúp mô hình tập trung vào những thành phần quan trọng trong chuỗi thay vì toàn bộ chuỗi. Mô tả toán học của mô hình này như sau :

- Encode

$$h_t = f(x_t, h_{t-1}) \tag{1.1}$$

$$c = qh_1, ..., h_{T_x} \tag{1.2}$$

- Decode:

$$p(y) = \prod_{t=1}^{T} p(y_t|y_1, ..y_{t-1}, c) \tag{1.3}$$

$$p(y_t|y_1, .., y_{t-1}, c) = g(y_{t-1}, s_t, c) \tag{1.4}$$

Mô hình attention đầu tiên được đề xuất bởi Bahdanau et. al. vào năm 2005 được gọi là soft-attention hay attention mềm. Mô hình toán học của phiên bản attention này được thể hiện như sau:

- Hàm phân phối xác suất của bước giải mã:

$$p(y_I|y_1, .., y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$$

  with

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

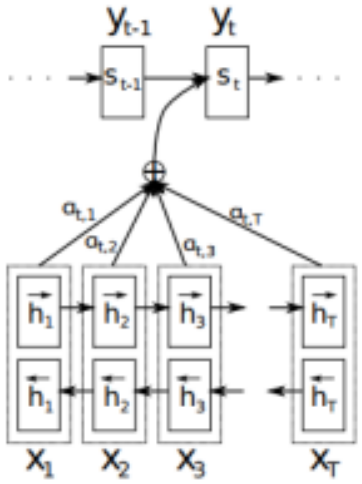
- Đưa ra một vector điều kiện c_i tương ứng với từng thành phần của chuỗi y_i

$$c_i = \sum_{j=1}^{T_x} a_{ij} h_j \tag{1.5}$$

$$a_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})} \tag{1.6}$$

$$e_{ij} = a(S_{i-1}, h_j) \tag{1.7}$$

Phương trình cuối cùng là một mô hình căn chỉnh trọng số mô tả sự tương ứng giữa giá trị đầu vào xung quanh vị trí j và đầu ra tại vị trí i.



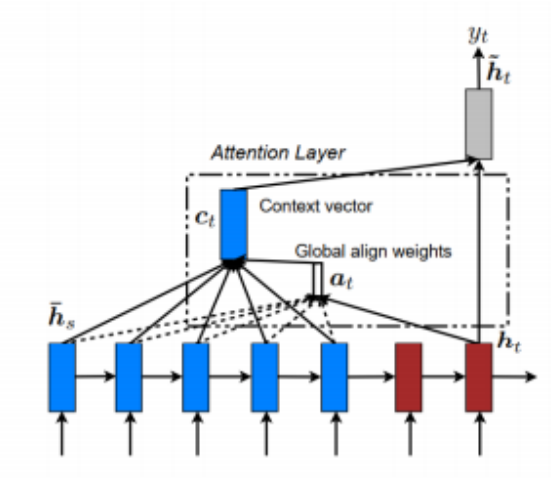
Hình 1.8: Soft attention

1.3.3 Các biến thể của cơ chế Attention

Nhận thấy tính ứng dụng cao của cơ chế attention , hàng loạt các biến thể làn lượt ra đời

- Hard attention :
  - Xem xét attention như là các biến ẩn của mô hình
  - Đưa vào một phân phối xác suất rời rạc được tham số hóa bởi a, và xem xét c_t như một biến ngẫu nhiên . Phân phối xác suất được định nghĩa bởi :$p(S_{t,i} = 1|S_{j<t}, a) = a_{t,i}$
  - $c_t = \sum_i S_{t,i} x_i$
  - $S_{t,i} \sim Multinoulli_L(a)$

  Với L là hàm mất mát $L_s = log(p(y|a))$



Hình 1.9: Global attention

- Global attention : Ý tưởng của mô hình attentional toàn cục là xem xét tất cả các trạng thái ẩn của bước mã hóa khi tính toán vector ngữ cảnh c_t

  –

  $$a_t = \frac{exp(score(h_t, \hat{h_t})}{\sum_{s'} exp(score(h_t, \hat{h_{s'}})} \tag{1.8}$$

  Với :

  * h_s : trạng thái ẩn của chuỗi nguồn
  * h_t: trạng thái ẩn của chuỗi đích

- Global attention : conten-based function có ba dạng chính :
  - Hàm nhân :$h_t^T \overline{h_s}$
  - Hàm tổng quát : $h_t^T W_a \overline{h_s}$
  - Hàm nối : $v_a^T tanh(W_a[h_t, \overline{h_s}])$
  - Tinh thần của global attention tương tự như với attention thông thường của Bahdanau et al(2015)
  - Global attention :$h_t \rightarrow a_t \rightarrow c_t \rightarrow \hat{h_t}$
  - Soft attention : $h_{t_1} \rightarrow a_t \rightarrow c_t \rightarrow h_t$

1.3.4 Self Attention

Một cách tổng quát, ta có thể coi trạng thái phía trước của bộ giải mã là một vector truy vấn - query vector, trạng thái ẩn của bộ mã hóa là khóa - key và giá trị - value vector. Kết quả của attention là giá trị trung bình có trọng số của các vector giá trị, trong đó hệ số được tính toán dựa trên hàm tương thích giữa query và key. Chú ý rằng key và value có thể là các bộ vector khác nhau

Self-attentioon là quy trình áp dụng cơ chế attention được mô tả ở phần trên vào tất cả các vị trí của chuỗi đầu vào. Điều này được thực hiện bằng cách tạo ra ba ba vector (query, key, value) cho tất cả các vị trí của chuỗi, sau đó áp dụng cơ chế attention cho mỗi vị trí x_i. Các vector key và query ở vị trí x_i được sử dụng cho tất cả các vị trí khác. Đối với mỗi chuỗi đầu vào $X = (x_1, x_2, .., x_n)$ ta có chuỗi đầu ra tương ứng $Y = (y_1, y_2, .., y_n)$ trong đó mỗi y_i kết hợp thông tin của mỗi x_i cũng như về cách thức x_i liên quan đến những vị trí khác nhau trong X . Bộ vector(query , key , value) có thể được tạo ra bằng cách sử dụng các phép chiếu tuyến tính hoặc sử dụng mạng truyền thẳng

Với một giá trị query q , các vector value $(v_1, v_2, ..., v_n)$ và các vector key $(k_1, k_2, .., k_n)$, một giá trị đầu ra z được tính toán dựa trên phương trình :

$$z = \sum_{j=1}^{n} a_j(v_j) \tag{1.9}$$

$$a_j = \frac{expf(k_j, q)}{\sum_{i=1}^{n} expf(k_i, q)} \tag{1.10}$$

với a_j được tính toán bằng hàm softmax và $f(k_i, q)$ là đặc trưng cho sự tương thích giữa k_i và q

Hàm tương thích thường được sử dụng là dot-product function :

$$f(k, q) = (k)(q)^T \tag{1.11}$$

hoặc hàm nhân scaled dot-product funtion:

$$f(k, q) = \frac{(k)(q)^T}{\sqrt{d_k}} \tag{1.12}$$

trong đó d_k là số chiều của key vectors . Việc giảm giá trị của hàm f nhằm mục đích tăng sự ổn định khi chiều của các vector key , value cũng như query tăng lên.

Việc tính toán này có thể được thể hiện một cách song song cho toàn bộ chuỗi đầu vào bằng cách nhóm các vector query , key , value tương ứng thành các ma trận Q, K , V

khi đó :

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1.13}$$

1.3.5 Multi-head attention

Thay vì chỉ sử dụng cơ chế self-attention một lần cho (Q,K,V) với số chiều d_{model} , cơ chế multi-head attention được đưa ra bằng cách tính toán attention h lần với số chiều tương ứng d_{model}/h. Với mỗi head , bộ ma trận (Q,K,V) được chiếu riêng biệt lên không gian d_{model}/h chiều và tính toán self-attention . Kết quả của mỗi head sau đó được nối lại và áp dụng một phép chiếu tuyến tính để đưa về không gian có số chiều tương ứng với (Q,K,V) ban đầu.

Mô hình được mổ tả như sau :

$$MultiHead(Q, K, V) = Concat(head_1, .., head_n)W^0 \tag{1.14}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{1.15}$$

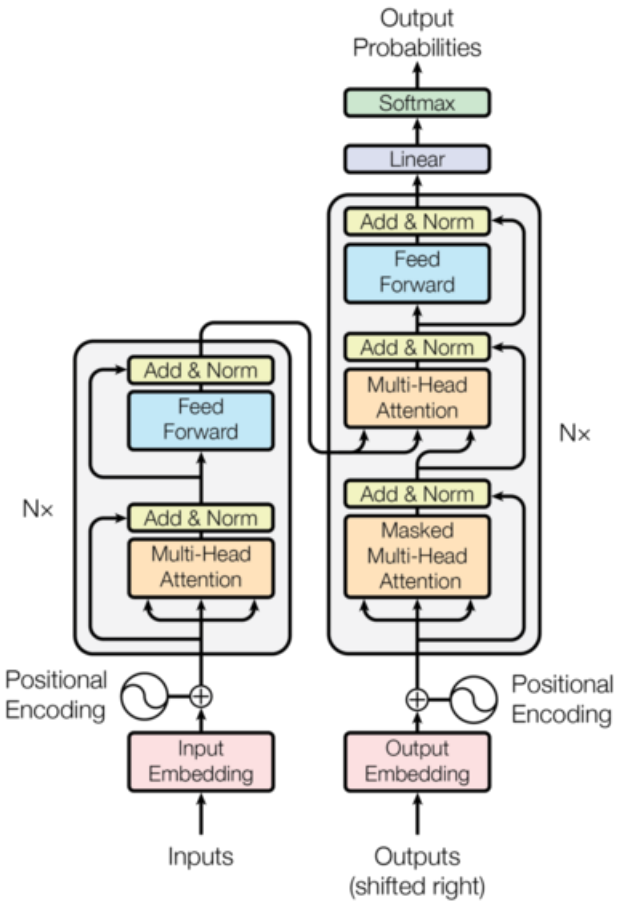
với $W_i^Q \in R^{d_{model} \times d_k}, W_i^K \in R^{d_{model} \times d_k}$,
$W_i^V \in R^{d_{model} \times d_v} v W^0 \in R^{d_{model} \times h d_v}$

1.4 Mô hình Transformer

Transformer là một mô hình không sử dụng hồi quy và dựa hoàn toàn trên cơ chế attention để thể hiện sự phụ thuộc toàn cục giữa đầu vào và đầu ra dữ liệu

1.4.1 Kiến trúc mô hình

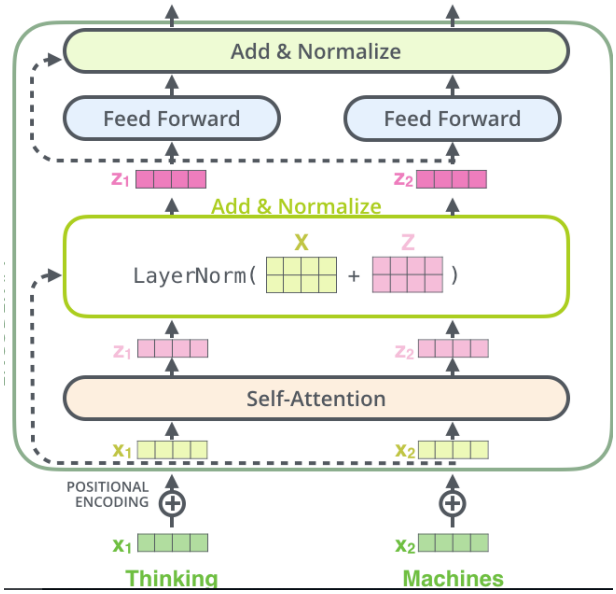
Transformer có kiến trúc mã hóa - giải mã . Ở đây , bộ mã hóa ánh xạ một chuỗi ký tự đầu vào $(x_1, x_2, ..., x_n)$ thành một chuỗi biểu diễn liên tục $z = (z_1, z_2, ..., z_n)$. Sau khi có z , bộ giải mã sẽ thực hiện sinh ra chuỗi $(y_1, y_2, .., y_n)$ tương ứng.



Hình 1.10: Mô hình Transformer

1.4.2 Lớp mã hóa và giải mã

Lớp mã hóa (encoder)



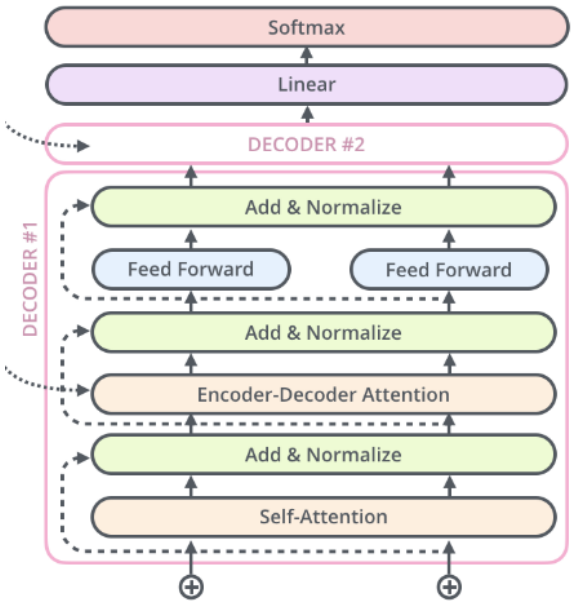
Hình 1.11: Transformer Encoder

Lớp mã hóa được hợp thành bởi 6 khối nhỏ hơn giống hệt nhau . Mỗi khối có hai lớp con , trong đó lớp đầu tiên là một multi-head self-attention như đã mô tả ở trên . Lớp thứ hai là một mạng neuron đầy đủ truyền thẳng (fully connected) . Lớp mã hóa sử dụng các kết nối dư xung quanh mỗi lớp con được đi theo một lớp chuẩn hóa (LayerNorm) , tức là đầu ra mỗi lớp con là một $LayerNorm(x + Sublayer(x))$, với $Sublayer(x)$ là hàm thực hiện bằng chính bản thân layer con đó . Đầu ra này là một vector có số chiều là $d_{model} = 512$. Tiếp tục , đầu ra của khối này sẽ là đầu vào của khối bên trên và cuối cùng ta đạt được một dạng embeddings tốt hơn embeddings ban đầu và các ma trận Q,K,V của khối cuối cùng

Lớp giải mã (decoder)

Lớp giải mã sẽ lấy 2 ma trận K và V từ lớp mã hóa rồi dự đoán từ đầu tiên , tiếp theo lớp giải mã tiếp tục lấy 2 ma trận K,V và từ vừa dự đoán để dự đoán từ tiếp theo cho tới từ cuối cùng . Lớp giải mã cũng được tạo bởi 6 khối giống hệt nhau . Tuy nhiên ở lớp giải mã , ngoài hai lớp con tương tự như các khối của lớp mã hóa , lớp giải mã còn được chèn thêm lớp con thứ ba để thực hiện multi-head attention trên đầu ra của lớp mã hóa . Tương tự như lớp mã hóa , ta sử dụng các kết nối dư xung quanh mỗi lớp con được đi theo bởi một lớp chuẩn hóa . Ở lớp giải mã này , lớp con self-attention cũng được chỉnh sửa để tránh tập trung vào các vị trí tiếp theo trong chuỗi . Điều này được thực hiện bởi một lớp mặt nạ, kết hợp với giá trị nhúng đầu ra được bù bởi một giá trị để đảm bảo rằng giá trị dự đoán cho vị trí i chỉ có thể phụ thuộc vào các giá trị đầu ra đã biết trước của

các vị trí nhỏ hơn i . Lớp con Encode-Decode Attention có tác dụng y hệt encode-decode attention tức là nhìn từ Lớp Decode sang lớp encode để thấy được giá trị nào sẽ được chú ý cho lần dự đoán tiếp theo



Hình 1.12: Transformer Decoder

1.4.3 Áp dụng attention vào Transformer

Mô hình Transformer sử dụng multi-head attention theo ba cách khác nhau :

- Trong liên kết giữa lớp mã hóa và giải mã , vector truy vấn được lấy từ lớp giải mã phía trước trong khi vector khóa và vector giá trị được lấy từ giá trị đầu ra của lớp mã hóa . Điều này cho phép mọi vị trí của lớp mã hóa sẽ được tập trung từ tất cả các vị trí của chuỗi giá trị đầu vào , tức là thực thi tư tưởng truyền thống của mô hình seq2seq.

- Lớp mã hóa sử dụng các lớp selt-attention với cả ba vector khóa , giá trị và truy vấn đều được lấy ra từ giá trị đầu ra của khối phía trước trong lớp mã hóa . Mỗi vị trí trong lớp mã hóa có thể chú ý đến tất cả các vị trí từ lớp phía trước .

- Tương tự, các lớp sefl-attention trong lớp giải mã cho phép mỗi vị trí tập trung vào tất cả vị trí của khối phía trước . Tuy nhiên ta cần phải tránh việc ảnh hưởng trái

luồng thông tin trong lớp giải mã để đảm bảo tính tự hồi quy bằng cách đánh dấu tất cả các giá trị đầu vaafo lớp soft-mã tưng ứng với các kết nối không hợp lên giá trị $-\infty$

1.5 BERT - Mô hình biểu diễn ngữ cảnh hai chiều tiền huấn luyện cho ngôn ngữ

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [**5**] được các nhà nghiên cứu của google công bố tháng 11 năm 2018 BERT được coi như là một trong những mô hình tốt nhất trong lĩnh vực xử lý ngôn ngữ tự nhiên . Mô hình này được xây dựng dựa trên kiến trúc Transformer với nền tảng là cơ chế attention. Các kết quả mới nhất cho thấy những kết quả ấn tượng của mô hình này với các bộ dữ liệu quan trọng của xử lý ngôn ngữ tự nhiên . Các reseacher của google đã nói rằng BERT có thể học được ngữ cảnh hai chiều , vậy ngữ cảnh hai chiều là gì và vì sao nó hoạt động tốt . Phần tiếp theo sẽ đi sâu vào phân tích cơ chế attention và kiến trúc Transformer để làm rõ điều này.

1.5.1 Kiếm trúc BERT

Mô hình BERT là mô hình mã hóa sử dụng nhiều lớp Transformer hai chiều , trong đó các lớp Transformer được sử dụng hoàn toàn tương tự như mô hình được đề xuất và thực thi gốc của Vasawwani et al. (2017) . Kiến trúc của Transformer được nói rất kỹ ở phần trên và việc thực thi kiến trúc này trong BERT hoàn toàn giống với cách triển khai gốc đó .

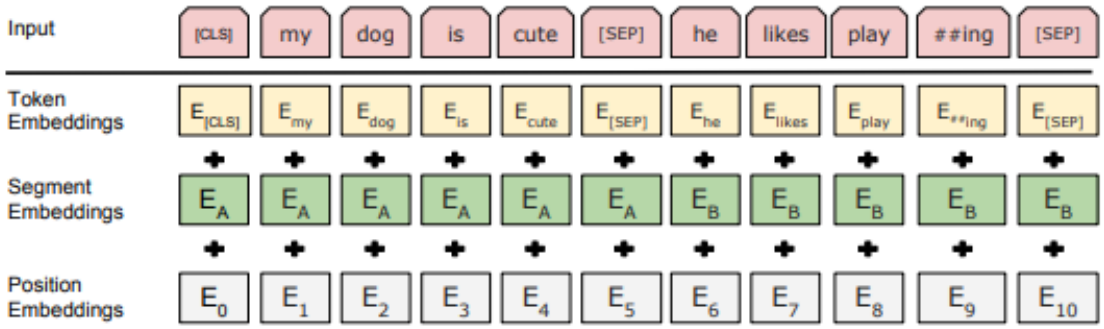
Mô hình BERT định nghĩa số lượng lớp (khối Transformer) là L , kích thước lớp ẩn là H và số lượng head của self-attention là A . In tất cả các trường hợp , BERT đặt kích thước của lớp truyền thẳng là $4H$ tức là 3072 với $H = 768$ và 4096 với $H = 1024$

BERT thực tập trung vào mô hình sau :

- $BERT_{BASE}$: $L = 12, H = 768, A = 12$ với tổng cộng 110 triệu tham số

1.5.2 Biểu diễn dữ liệu đầu vào

Cách thức biểu diễn dữ liệu đầu vào của BERT cho phép biểu diễn đồng thời một câu hoặc một cặp câu trong một chuỗi biểu diễn . Với một từ cho trước , biểu diễn của từ đó



Hình 1.13: Biểu diễn dữ liệu đầu vào của BERT

được tạo bởi tổng hợp các giá trị nhúng tưng ứng với từ ngữ , đoạn cũng như vị trí . Cụ thể :

- BERT sử dụng bộ WordPiece (Wu et al.2016) để tách các câu thành các từ nhỏ với bộ từ điển bao gồm 30000 từ.
- Lớp nhúng vị trí được sử dụng với độ dài tối đa 512.
- Ký tự đầu tiên của mọi chuối luôn luôn là ký tự đặc biệt ([CLS]) đại diện cho cả câu và sử dụng cho tác vụ phân lớp (classifi)
- Mỗi cặp chuỗi được ghép lại với nhau vào cũng một chuối đơn , phân biệt bởi hai dấu hiệu :
  - Dấu hiệu thứ nhất là sử dụng ký tự ngăn cách ([SEP])
  - Dấu hiệu thứ hai là sử dụng giá trị nhúng câu A đã được học cho chuỗi đầu tiên và giá trị nhúng B cho chuỗi thứ hai . Đối với tác vụ chỉ sử dụng một câu đơn , BERT chỉ sử sử dụng duy nhất giá trị nhứng A.
- lớp Position Embeddings có nhiệm vụ mã hóa thông tin vị trí của từ trong câu . Nếu để ý , có thể thấy toàn bộ mô hình BERT chỉ sử dụng Transformer encoder tức là chỉ có attention và các mạng neural truyền thẳng với kỹ thuật normalize , hoàn toàn không hề có thứ tự như các mang neural liên hồi . Để mô hình có thể học được sự liên hệ về thứ tự xuất hiện của các từ thì người ta đưa thêm lớp Position Embeddings mang ý nghĩa về thứ tự của từ trong chuỗi.

- Lớp Segment Embedding mang thông tin về vị trí câu mà từ đó đang phụ thuộc . Ví dụ như hình những từ có Segment là E_A tức là nó thuộc câu A và ngược lại . Trên thực tế giá trị này sẽ là 0 nếu thuộc câu đầu tiên và 1 nếu thuộc câu thứ hai . Nếu tác vụ chỉ liên quan đến một câu thì chỉ cần gán các giá trị này bằng 0 hết

- cuối cùng là token Embeddings , lớp Embeddings này có nhiệm vụ cho mô hình biết được vị trí của từ này trong tập từ điển , tương tự như mô hình word to vec nó được biểu diễn dưới dạng one hot vector với tập từ điển word piece 30000 token

-

Tác vụ tiền huấn luyện

BERT được tiền huấn luyện bởi hai tác vụ học không giám sát đó là **Mô hình ngôn ngữ đánh dấu - Masked Language Model** và **Dự đoán câu kế tiếp - Next Sentence Prediction**

Tác vụ 1 : Masked LM

Một cách trực quan , việc tin tưởng rằng một mô hình hai chiều mạnh hơn đáng kể một chiều là có ý nghĩa . Tuy nhiên không may là các mô hình ngôn ngữ thông thường chỉ huấn luyện từ trái sang phải hoặc từ phải sang trái , trong khi điều kiện hai chiều cho phép mỗi từ ngữ được xem xét chính bản thân nó trong nhiều lớp ngữ cảnh khác nhau.

Để có thể huấn luyện một biểu diễn sâu hai chiều cho ngôn ngữ , BERT sử dụng một cách tiếp cận trực tiếp đó là đánh dấu ngẫu nhiên một số lượng từ đầu vào sau đó thực hiện dự đoán những từ đã đánh dấu này . Trong trường hợp này , vector ẩn cuối cùng tương ứng với các từ được đánh dấu sẽ được truyền qua một lớp softmax trên toàn bộ từ điển tương tự như các mô hình ngôn ngữ thông thường khác . BERT thực hiện đánh dấu ngẫu nhiên 15% tất cả các từ ngữ trong WordPiece trong mỗi chuỗi đầu vào . Để tránh nhiễu , BERT thực hiện dự đoán các từ bị đánh dấu thay vì cấu trúc lại toàn bộ chuỗi đầu vào.

Cách tiếp cận này vẫn tồn tại hai nhược điểm sau khi nhận được mô hình tiền huấn luyện theo hai chiều . Nhược điểm thứ nhất là BERT tạo ra sự không phù hợp giữa quá trình tiền huấn luyện và quá trình tinh chỉnh mô hình , tức là ký tự [MASK] sẽ không bao giờ xuất hiện trong quá trình tinh chỉnh mô hình . Để giảm thiểu vấn đề này , BERT không luôn luôn đánh dấu bằng ký tự [MASK]. Thay vào đó :

- thay thế 80% số lượng từ được chọn bằng [MASK]

- Thay thế 10% số lượng từ được chọn bằng một từ bất kỳ khác

- Giữ nguyên 10% số lượng từ được chọn

Lớp Transformer mã hóa không biết trong tập hợp các từ được chọn , từ nào sẽ bị thay thế bằng những từ khác và từ nào sẽ giữ nguyên , do đó mô hình sẽ học được phân phối của tất cả các từ . Thêm vào đó , bởi vì việc thay thế ngẫu nhiên chỉ xảy ra cho 1.5% số lượng các từ nên có vẻ sẽ không ảnh hưởng đến tính phù hợp của việc thu nhận kiến thức cho mô hình ngôn ngữ.

Nhược điểm thứ hai của việc sử dụng MLM là chỉ có 15% số lượng từ ngữ được dự đoán nên mô hình sẽ hội tụ chậm hơn rất nhiều so với mô hình tuần tự từ trái sang phải .

Tác vụ 2 : Next Sentence Prediction

Rất nhiều tác vụ thực tế dựa trên kiến thức về mỗi liên hệ giữa 2 câu văn , trong khi những kiến thức này không được thu nhận trực tiếp bởi mô hình ngôn ngữ . Để có thể huấn luyện mô hình có thể hiểu được mối quan hệ giữa các từ ngữ , BERT sử dụng tác vụ *next sentence prediction* có thể được tạo ra một cách đơn giản bởi bất kỳ bộ dữ liệu nào . Ví dụ :

- *Input = [CLS] the man went to [MASK] store [SEP] he bouth a gallon [MASK] milk [SEP]*
- Label = *IsNext*
- *Input = [CLS] the man [MASK] to store [SEP] penguin [MASK] are flight less birds [SEP]*
- Label = *NotNext*

Trong đó nhãn *IsNext* thể hiện rằng đây là hai câu liên tiếp và nhãn *NotNext* thể hiện rằng đây không phải là hai câu liên tiếp nhau

1.5.3 Ngữ cảnh hai chiều và so sánh với word to vec

Thông thường , khi sử dụng word to vec người ta thường tải một bộ pre-trained $n \times 100$ chiều của google với nội dung là từ - vector $n \times 100$ biểu diễn cho từ đó . tức là vector đại diện sẽ bị fix cứng cho từ , điều này gây ra sự bất hợp lý cho tiếng việt với trường hợp các từ nhiều nghĩa , nghĩa sẽ gắn với từng trường hợp sử dụng của từ , điều này gây ra sự nhầm lẫn trong những ngôn ngữ có nhiều nhập nhằng như tiếng việt . Đối với BERT điểm yếu này được khắc phục nhờ cơ chế Transformer encoder , Mỗi từ sẽ được biểu diễn dưới dạng one hot encoder và sau khi đi qua BERT mỗi từ sẽ có một biểu diễn mới cho

mình tuy thuộc vào ngữ cảnh của nó . Có nghĩa là mỗi từ sẽ có những biểu diễn khác nhau tùy vào ngữ cảnh mà nó phụ thuộc vào

Ngữ cảnh 2 chiều của BERT lại đến từ cơ chế attention , như lý thuyết về những mạng RNN hay các mạng Liên hồi khác , các biến thể có cả các mạng 2 chiều , nhưng 2 chiều ở đây mang ý nghĩa là một chiều đi và một chiều về . Điều này tuy rằng cũng học được ngữ cảnh hai chiều nhưng về nguyên tắc nó chỉ là sự kết hợp của những ngữ cảnh một chiều mà từ phụ thuộc tức là từ chỉ nhìn thấy nó trong một ngữ cảnh nào đó tùy thuộc vào chiều hiện tại của mô hình . Còn với BERT , với cơ chế sefl Attention trong Transformer Encoder là attention 2 chiều khác với trong Transfor Decoder chỉ là attention một chiều tức là encode - decode attention , từ sẽ nhìn vào ngữ cảnh hai bên nó trong cùng một lúc .

Chương 2

Các Biến thể của Bert

2.1 XLNET

Các mô hình học ngữ cảnh của từ thông thường là những mô hình ngôn ngữ (language model) . Thông thường với những language model cũ sử dụng mạng RNN và các biến thể của RNN được gọi là autoregressive(AR) language model tức là những language sử dụng context để dự đoán những từ tiếp theo , chúng có thể sử dụng context forward kết hợp với context backward để dự đoán từ kế tiếp nhưng nó chỉ dụng được 1 trong 2 trong cùng 1 mốc thời gian , tức là trong 1 mốc thời gian thì chỉ có forward hay backward được sử dụng . AR language model rất tốt trong các task sinh ngôn ngữ vì nó có sử dụng context của từ nhưng lại có 1 nhược điểm là không thể sử dụng được cả 2 ngữ cảnh cùng 1 lúc .

Không Giống như AR language model , bert là một autoencoder(AE) language model , với mục đích dữ đoán những token [MASK] AE language sử dụng ngữ cảnh của cả 2 bên của token được mask nên trong cùng 1 thời điểm bert có thể sử dụng được cả forward và backward context . Tuy nhiên có 1 nhược điểm mà AE language model gặp phải , nếu trong 1 câu có 2 token được mask thì khi dự đoán từ được mask AE language model coi từ còn lại là 1 token ký hiệu là [MASK] , rõ ràng nếu 2 từ này ít có liên hệ và ít quan trọng thì sẽ không có vấn đề gì nhưng nếu 2 từ này có liên hệ mật thiết với nhau và có 1 vài trò quan trọng trong câu thì rõ ràng AE language model không học ra được mối quan hệ giữa 2 từ đó . XLnet ra đời để giải quyết vấn đề này .

Với kiến trúc tương đối giống bert, sự khác biệt là XLnet sử dụng một cách pre-train khác cho language model . Một câu với order :[x1,x2,x3,x4] với 4 token sẽ có 4! hoán vị

các token đó để có được 1 câu mới , ví dụ token được masked là x3 thì XLnet sẽ sử dụng những câu mà x3 lần lượt đứng ở vị trí thứ 1 , 2,3 và 4 , sử dụng những câu này để dự đoán cho token x3 . Rõ ràng đây là AR language model nhưng vì có rất nhiều hoán vị và vị trí của token x3 hoàn toàn có tất cả các khả năng , model sẽ học được cách sử dụng thông tin về vị trí của x3 và thông tin về các từ ở 2 bên nó để dự đoán x3 , điều này phần nào khắc phục được vấn đề còn tồn đọng của bert .

Chương 3

kết quả thử nghiệm

Sử dụng bộ bert multilingua cased cho mộ dữ liệu zalo question answer .
Vì bộ dữ liệu zalo question answer chỉ cần dự phân loại xem có câu trả lời của câu hỏi trong đoạn văn hay không nên có thể sử dụng task classifi senten pair cho bài toán này .
Bộ dữ liệu train được chia 7-3 với 70% cho dữ liệu train và 30% cho dữ liệu đánh giá
Với model sử dụng output của token [CLS] cho việc phân loại thì kết quả đạt được f1=0.64 .
Với quan sát nếu chỉ sử dụng output của token [CLS] thì rất có thể thông tin của vế sau (phần dữ liệu có thể chứa câu trả lời) có thể không được tận dụng hết , tiến hành add thêm token [CLS] vào đầu vế thứ 2 và sử dụng 2 token này để dự đoán đầu ra .
Kết quả : f1 = 0.68 với cùng config với mô hình đầu tiên , chứng tỏ có thể với context dài rất có thể thông tin cần thiết trong token [CLS] của câu đầu tiên là không đủ .
Vì bert multilingua được huấn huyện trên bộ wiki của nhiều ngôn ngữ nên rất có thể việc sử dụng thêm một bộ question answer của squad sẽ có kết quả cao hơn .
Thí nghiệm : sử dụng thêm một squad 1.1 thêm vào dữ liệu huấn luyện với mô hình gốc của bert clasifier thì kết quả khá khả quan với f1=0.73 . Việc này chứng tỏ nếu bộ dữ liệu tiếng việt không đủ để huấn luyện thì có thể sử dụng thêm những bộ dữ liệu ngôn ngữ khác cùng trong danh sách được hỗ trợ của bert multilingua thì kết quả có thể sẽ tốt hơn .
Sử dụng mô hình thứ 2 với bộ dữ liệu có sử dụng thêm dữ liệu squad 1.1 , kết quả : running .
Sử dụng bộ Electra mới được chia sẻ cho tiếng việt (running).
Sử dụng bộ XLnet (running)
Vì những hạn chế của google colab nên việc huấn luyện các kịch bản thử nghiệm tốn khá

nhiều thời gian . Ngoài sử dụng google colab thì việc sử dụng kaggle rất tiên lợi cho việc huấn luyện mô hình , tuy nhiên kaggle chỉ cho sử dụng GPU 30 tiếng trong 1 tuần