TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN I

Nhận diện thực thể có tên trong khai báo hải quan

HOÀNG PHI LONG

philonghust@gmail.com

Ngành: Toán Tin

Giảng viên hướng dẫn: TS. Trần Ngọc Thăng _____
Chữ kí của GVHD

Viện: Toán ứng dụng và Tin học

HÀ NỘI, 6/2021

Lời cảm ơn

Tác giả gửi lời cảm ơn sâu sắc tới **TS. Trần Ngọc Thăng**, người thầy đã tận tình chỉ bảo, luôn theo dõi sát sao và giúp đỡ tác giả trong quá trình nghiên cứu. Không có những lời động viên và hướng dẫn của thầy, đồ án sẽ không thể hoàn thiện.

Tác giả xin gửi lời cảm ơn đến Viện Toán ứng dụng và Tin học, Trường Đại học Bách Khoa Hà Nội đã cung cấp những kiến thức, cơ quan hải quan đã cung cấp dữ liệu tờ khai hàng hóa, anh Lương Khắc Mạnh đã cũng cấp bộ từ điển phương tiện để chuẩn hóa dữ liệu, tạo cho tác giả những điều kiện thuận lợi để hoàn thành đồ án này.

Tác giả đặc biệt cảm ơn gia đình, người thân, bạn bè luôn ở bên và hỗ trợ tác giả những gì tốt nhất.

Tác giả xin chân thành cảm ơn!

Tóm tắt nội dung luận văn

Nhận dạng thực thể có tên (Named Entity Recognition – NER) nhằm nhận biết các chuỗi từ trong văn bản là tên của một đối tượng nào đó, điển hình như tên người, tên tổ chức, tên địa danh, thời gian v.v. NER là nhiệm vụ đóng vai trò quan trọng trong các ứng dụng trích xuất thông tin, đã được quan tâm nghiên cứu trên thế giới từ đầu những năm 1990. Từ năm 1995, hội thảo quốc tế chuyên đề Hiểu thông điệp (Message Understanding Conference - MUC) lần thứ 6 đã bắt đầu tổ chức đánh giá các hệ thống NER cho tiếng Anh.

Các hệ thống NER thường được tạo ra bằng các kỹ thuật truyền thống như ngữ pháp, thống kê đến các phương pháp hiện đại như học máy (Machine Learning), học sâu (Deep Learning). Các hệ thống xây dựng dựa trên ngữ pháp bằng tay thường có precision cao nhưng lại kém recall và tốn nhiều tháng làm việc của các chuyên gia ngôn ngữ. Các hệ thống xây dựng dựa trên thống kê, ML, DL thường cần một lượng lớn dữ liệu tốn kem. Trong đồ án này, tác giả sẽ sử dụng phương pháp Transfer Learning để khắc phục những nhược điểm trên và giải quyết bài toán nhận diện thực thể có tên trong khai báo hải quan.

Từ khóa: *Transfer learning, Deep transfer learning, Deep learning, Named entity recognition, Pretraining model, BERT, PhoBERT.*

Hà Nội, ngày 25 tháng 06 năm 2021

Học viên thực hiện

Hoàng Phi Long

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

Từ viết tắt	**Ý nghĩa**
BERT	Bidirectional Encoder Representation from Transformers
RNN	Recurrent Neural Network
seq2seq	sequence to sequence
NEs	Named entities
NER	Named-Entity Recognition
ML	Machine Learning
DF	Deep Learning

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

> After supervised learning, Transfer Learning will be the next driver of Machine Learning commercial success.
>
> Andrew Ng

1.1 Giới thiệu Transfer Learning

Con người không được dạy mọi nhiệm vụ hoặc vấn đề để có thể xử lý , tất cả đều sẽ phải đối mặt với những tình huống chưa bao giờ gặp phải. Khi đó, họ sẽ sử dụng những kinh nghiệm trong quá khứ và vận dụng nó để giải quyết những nhiệm vụ mới. Ví dụ:

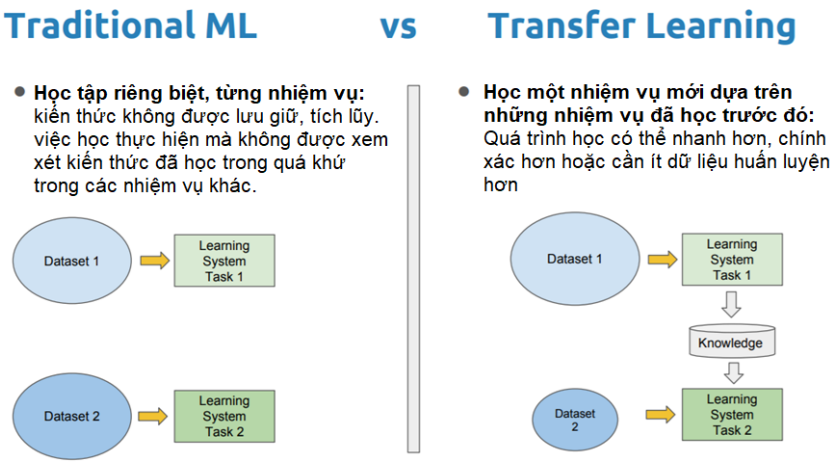
- Biết chơi cajon -> Học chơi trống dàn dễ hơn.
- Biết toán học, thống kê, lập trình -> Học Machine Learning dễ hơn.
- Biết đi xe đạp -> Học đi xe máy dễ hơn.

Khả năng học hỏi từ một số lượng lớn kinh nghiệm và áp dụng 'kiến thức' sang môi trường mới chính là điều mà Transfer Learning hướng đến.

1.1.1 Định nghĩa Transfer Learning

Transfer Learning là việc ứng dụng kĩ năng (tri thức) mình học được vấn đề nguồn (source domain - $\mathcal{D}_{\mathcal{S}}$), với nhiệm vụ nguồn (source task - $\mathcal{T}_{\mathcal{S}}$), sang vấn đề mục tiêu (target domain - $\mathcal{D}_{\mathcal{T}}$) với nhiệm vụ mục tiêu (target task - $\mathcal{T}_{\mathcal{T}}$) có liên quan. Transfer Learning nhằm cải thiện việc học hàm $\mathcal{F}_{\mathcal{T}}(.)$ cho nhiêm

vụ $\mathcal{T}_\mathcal{T}$ trên miền $\mathcal{D}_\mathcal{T}$ [1]. Thông qua transfer learning, mình không phải học lại từ con số không nữa, mà dựa trên những tri thức mình đã biết mà học lên tiếp. Từ đó giảm thời gian và công sức học nhiệm vụ mới.



Hình 1.1 Mô hình Machine Learning truyền thống (bên trái) và mô hình sử dụng Transfer Learning (bên phải).

Các mô hình Machine Learning truyền thống được tách biệt và hoàn toàn dựa trên các nhiệm vụ cụ thể, bộ dữ liệu và đào tạo các mô hình biệt lập riêng biệt trên chúng. Không có kiến thức nào được giữ lại hoặc được tận dụng mà có thể được chuyển từ mô hình này sang mô hình khác. Trong Transfer Learning, bạn có thể tận dụng kiến thức (tính năng, trọng số, ...) từ các mô hình đã được đào tạo trước đó để đào tạo các mô hình mới hơn và thậm chí giải quyết các vấn đề như có ít dữ liệu hơn cho nhiệm vụ mới hơn [2]. Trong suốt quá trình học chuyển giao chúng ta cần trả lời những câu hỏi sau:

- **Chuyển giao cái gì?:** Vì một số tri thức có tính đặc thù riêng của nhiệm vụ, và có các tri thức chung giữa các nhiệm vụ khác nhau. Về cơ bản, tri thức chung thì có thể chuyển giao được.
- **Chuyển giao khi nào?:** Transfer Learning có thể sẽ không đem lại lợi ích nếu nhiệm vụ nguồn và nhiệm vụ mục tiêu không có mối liên hệ, tương quan (Negative Transfer). Do đó cần thận trọng trong việc áp dụng Transfer Learning.
- **Chuyển giao như thế nào?:** Sau khi trả lời được câu hỏi trên, chúng ta cần xác định cách thức thực hiện chuyển giao cho hiệu quả.

1.1.2 Phân loại Transfer Learning

Phân loại theo chủ đề nghiên cứu

Dựa theo chủ đề nghiên cứu, ta chia Transfer Learning thành 3 loại:

- **Học chuyển giao đệ quy (Inductive Transfer Learning):** Miền nguồn và miền mục tiêu giống nhau (cả hai miền được có sẵn nhãn), nhưng tác vụ nguồn và tác vụ mục tiêu khác nhau.
- **Học chuyển giao không giám sát (Unsupervised Transfer Learning):** Miền nguồn và miền mục tiêu tương tự nhau (một trong hai miền không được gán nhãn), nhiệm vụ nguồn và nhiệm vụ mục tiêu khác nhau.
- **Học chuyển giao chuyển đổi (Transductive Transfer Learnig):** Nhiệm vụ nguồn và nhiệm vụ mục tiêu khác nhau nhưng miền nguồn và miền mục tiêu giống nhau.

Phân loại theo hướng tiếp cận

Dựa theo hướng tiếp cận, ta chia Transfer Learning thành 4 loại:

- **Chuyển giao phiên bản (Instance Transfer):** Dùng lại tri thức từ miền nguồn cho nhiệm vụ mục tiêu.
- **Chuyển giao đại diện tính năng (Feature Representation Transfer):** Giảm thiểu sự khác biệt của hai miền và tỉ lệ lỗi bằng cách xác định biểu diễn tính năng tốt của miền nguồn có thể sử dụng lên miền mục tiêu.
- **Chuyển giao tham số (Parameter Transfer):** Cách tiếp cận này hoạt động dựa trên giả thuyết các nhiệm vụ liên hệ với nhau có cùng phân bố về bộ các siêu tham số (hyper parameter).
- **Chuyển giao kiến thức - quan hệ (Relational - Knowledge Transfer):** Hướng tới giải quyết vấn đề khi dữ liệu không độc lập và phân bố ngẫu nhiên, thường thấy ở ứng dụng liên quan tới mạng xã hội.

Liên hệ giữa phân loại theo hướng tiếp cận và domain được thể hiện như sau:

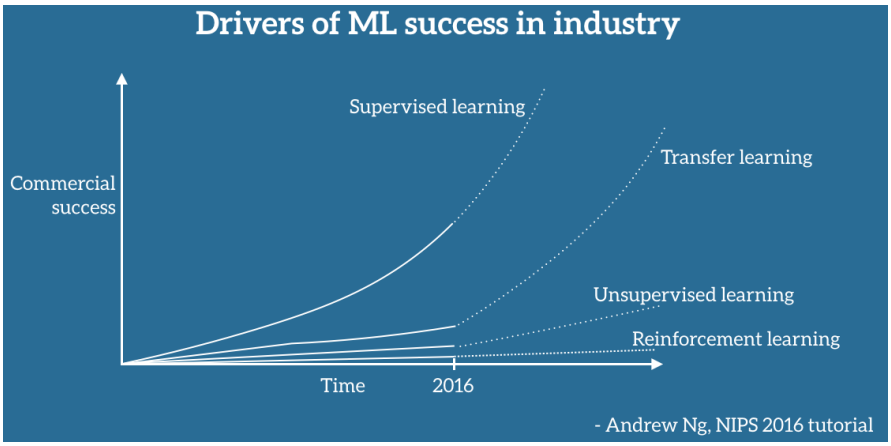
	Inductive Transfer Learning	Transductive Transfer Learning	Unsupervised Transfer Learning
Instance Transfer	✓	✓	
Feature Representation Transfer	✓	✓	✓
Parameter Transfer	✓		
Relational - Knowledge Transfer	✓		

Bảng 1.1 Liên hệ giữa phân loại theo hướng tiếp cận và domain

1.1.3 Tại sao nên sử dụng Transfer Learning trong Deep Learning?

Áp dụng Transfer Learning trong Deep Learning gọi là Deep Transfer Learning, việc này mang lại nhiều lợi ích:

- Không đủ dữ liệu: Deep Learning cần rất nhiều dữ liệu, huấn luyện các mô hình Deep Learning thường không hiệu quả. Mặt khác, việc có được một bộ dữ liệu lớn là rất tốn kém. Transfer Learning là một giải pháp cho vấn đề này.
- Không đủ tài nguyên: Huấn luyện mô hình Deep Learning với tập dữ liệu khổng lô cần điều chỉnh tới hàng triệu đến hàng trăm triệu tham số. Transfer Learning sẽ góp phần làm giảm thời gian huấn luyện.
- Cải thiện chất lượng: Rất nhiều mô hình Transfer Learning cho kết quả tốt hơn thay vì huấn luyện lại từ đầu.



Hình 1.2 Xu hướng của Machine Learning.

Như trong bài diễn thuyết của giáo sư **Andrew Ng** tại hội nghị NIPS 2016 [3], Transfer Learning sẽ là cầu nối về chất lượng của Machine Learning giữa Un-Supervised Learning và Supervised Learning. Transfer Learning giúp giải quyết các vấn đề mà không có dữ liệu, các vấn đề mới chưa được học trước đó.

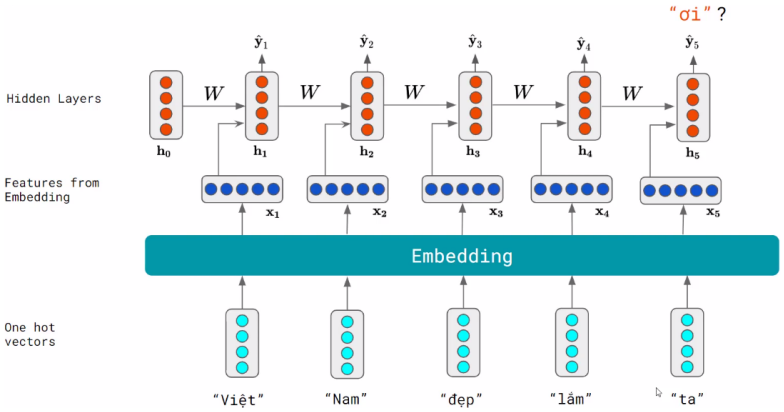
1.2 Cơ chế Attention

1.2.1 Giới thiệu về mạng nơ ron hồi quy (RNN)

Kiến trúc mạng nơ ron hồi quy (RNN)

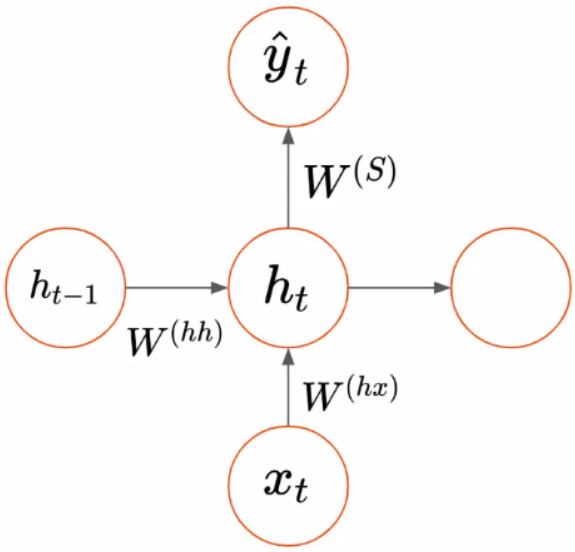
RNN là mô hình thuộc lớp seq2seq[1]. Các từ trong câu sẽ đi qua Embedding layer (hình 1.3), tạo thành các vector từ $x_1, x_2, ..., x_n$. Các h_t là hidden state thứ t, hay còn gọi là vector lịch sử (history), lưu trữ thông tin lịch sử của các từ trong từ điển. Các $\hat{y}_t$ là dự đoán đầu ra của mô hình. W là ma trận trọng số, ma trận này là giống nhau ở tất cả các time step t. Ma trận trọng số W và các vector h_t sẽ là những hệ số cần học.

[1]seq2seq chỉ một lớp các bài toán hoặc các kiến trúc mô hình có đầu vào và đầu ra là một chuỗi các phần tử.



Hình 1.3 Kiến trúc mạng RNN.

Tính toán mạng nơ ron hồi quy (RNN)



Hình 1.4 Tính toán mạng RNN.

Các ma trận và vector được tính lần lượt theo công thức [4]:

$$h_t = sigmoid(W^{hh}h_{t-1} + W^{hx}x_t) \quad (1.2.1)$$

$$\hat{y}_t = softmax(W^S h_t) \quad (1.2.2)$$

$$\hat{P}(x_{t+1} = v_j|x_t, ..., x1) = \hat{y}_(t, j) \quad (1.2.3)$$

với $x \in \mathbb{R}$, $W^{hh} \in \mathbb{R}^{D_h * D_h}$, $W^{hx} \in \mathbb{R}^{D_h * d}$, $W^s \in \mathbb{R}^{|V| * D_h}$, $\hat{y} \in \mathbb{R}^{|V|}$, $h_t \in \mathbb{R}^{D_h}$ ($|V|$ là kích thước của từ điển).

Hàm mất mát

- Hàm mất mát tại bước t:

$$J^{(t)}(W) = -\sum_{j=1}^{|V|} y_{t,j} log(\hat{y}_{t,j}) \quad (1.2.4)$$

- Hàm mất mát toàn bộ chuỗi độ dài T:

$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j} log(\hat{y}_{t,j}) \quad (1.2.5)$$

Nhược điểm của RNN

Mạng nơ ron hồi quy (RNN) có 2 nhược điểm lớn:

1. Quá trình huấn luyện mạng nơ ron hồi quy (RNN) là tuần tự, vì thể không thể tận dụng được khả năng tính toán song song của GPU.

2. Khi tính toán trên chuỗi dài truy hồi thì kết quả thường không tốt vì xảy ra hiện tượng Vanishing/Exploding Gradient.

1.2.2 Attention layer

Máy tính không thể học được từ các dữ liệu thô như bức ảnh, file text, file âm thanh, đoạn video. Do đó nó cần đến quá trình mã hóa thông tin sang dạng số và từ dạng số giải mã kết quả đầu ra. Đó chính là 2 quá trình **encoder** và **decoder**:

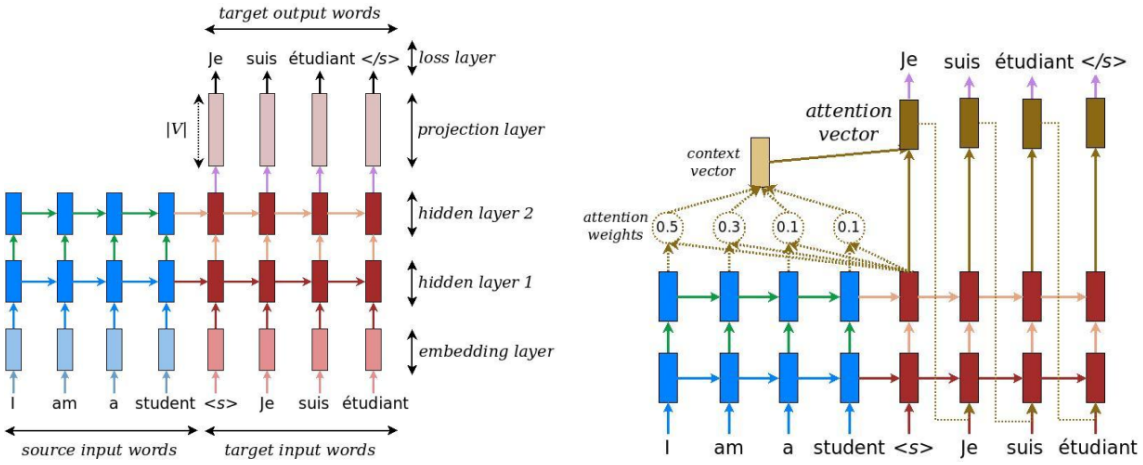
- **Encoder**: Chuyển input thành những features learning có khả năng học tập các task. Trong mô hình RNN, quá trình Encoder chính là các Embedding layers và Recurrent Neral Network.

- **Encoder**: Đầu ra của encoder chính là đầu vào của các decoder. Mục đích của decoder là tìm ra phân phối xác xuất từ các features learning ở Encoder từ đó xác định đâu là nhãn của đầu ra. Kết quả có thể là một nhãn đối với các model phân loại hoặc một chuỗi các nhãn theo tứ tự thời gian đối với model seq2seq.

Mô hình seq2seq là mô hình chuỗi nên có thứ tự về thời gian. Trong một tác vụ dịch máy, các từ ở input sẽ có mối liên hệ lớn hơn đối với từ ở output cùng vị trí. Do đó attention hiểu một cách đơn giản sẽ giúp thuật toán điều chỉnh sự tập trung lớn hơn ở các cặp từ (input, output) nếu chúng có vị trí tương đương hoặc gần tương đương,

Như trong hình 1.5, Từ 'I' trong tiếng anh tương ứng với 'Je' trong tiếng Pháp. Do đó attention layer điều chỉnh một trọng số α lớn hơn ở context vector so với các từ khác.

Màu xanh đại diện cho encoder và màu đỏ đại diện cho decoder. Chúng ta thấy context vector chính là tổ hợp tuyến tính của các output theo trọng số attention. Ở vị trí thứ nhất của phase decoder thì context véc tơ sẽ phân bố trọng số attention cao hơn so với các vị trí còn lại. Điều đó thể hiện rằng véc tơ context tại mỗi time step sẽ ưu tiên bằng cách đánh trọng số cao hơn cho các từ



(a) Mô hình seq2seq khi chưa có attention layer. **(b)** Mô hình seq2seq khi đã có attention layer.

Hình 1.5 Attention layer.

ở cùng vị trí time step. Ưu điểm khi sử dụng attention đó là mô hình lấy được toàn bộ bối cảnh của câu thay vì chỉ một từ input so với model ở hình 1 khi không có attention layer. Cơ chế xây dựng attention layer là một qui trình khá đơn giản. Chúng ta sẽ trải qua các bước: [5]

1. Đầu tiên tại time step t ta tính ra danh sách các điểm số, mỗi điểm tương ứng với mỗi cặp input t và các vị trí còn lại theo công thức bến dưới:

$$score(h_t, \overline{h}_s) \tag{1.2.6}$$

   Ở đây h_t là cố định tại time step t và là hidden state của từ mục tiêu thứ t ở decoder, $\overline{h}_s$ là hidden state của từ thứ s trong encoder. Công thức tính score có thể là `dot product` hoặc `cosine similarity` tùy vào lựa chọn.

2. Các scores sau bước 1 chưa được chuẩn hóa. Để tạo thành một phân phối xác xuất chúng ta đi qua hàm softmax khi đó ta sẽ thu được các trọng số attention weight.

$$\alpha_{ts} = \frac{exp(score(h_t, \overline{h}_s))}{\sum_{s'=1}^{S} exp(score(h_t, \overline{h}_{s'}))} \tag{1.2.7}$$

   $\alpha_t s$ là phân phối attention (attention weight) của các từ trong input với các từ ở vị trí trong output hoặc target.

3. Kết hợp vector phân phối xác suất $\alpha_t s$ với các hidden state để thu được context vector:

$$c_t = \sum_{s'=1}^{S} \alpha_t s \overline{h}_{s'} \tag{1.2.8}$$

4. Tính attention vector để decode ra từ tương ứng ở ngôn ngữ đích. Attention vector sẽ là kết hợp của context vector và các hidden state ở decoder. Theo cách này attention vector sẽ không chỉ được học từ chỉ hidden state ở unit cuối cùng như hình 1.5a mà còn được học từ toàn bộ các từ ở vị trí khác thông qua context vector.

$$a_t = f(c_t, h_t) = tanh(W_c[c_t, h_t]) \tag{1.2.9}$$

Kí hiệu $[c_t, h_t]$ là phép concatenate 2 vector c_t, h_t theo chiều dài. Giả sử $c_t \in \mathbb{R}^c$, $h_t \in \mathbb{R}^h$ thì vector $[c_t, h_t] \in \mathbb{R}^{c+h}$. $W_c \in \mathbb{R}^{a*(c+h)}$ trong đó a là độ dài của attention vector. Ma trận mà chúng ta cần huấn luyện chính là W_c.

1.2.3 Scale dot product attention

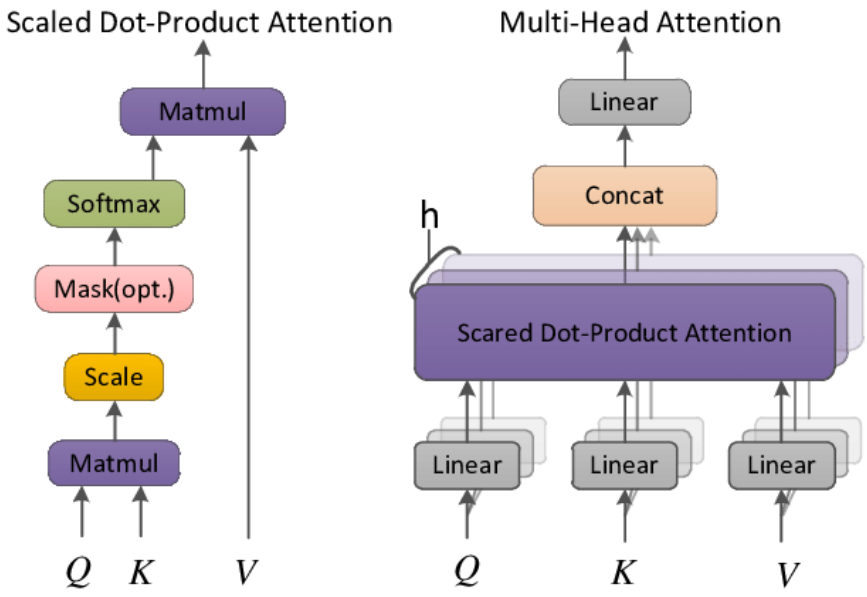
Đây chính là một cơ chế self-attention khi mỗi từ có thể điều chỉnh trọng số của nó cho các từ khác trong câu sao cho từ ở vị trí càng gần nó nhất thì trọng số càng lớn và càng xa thì càng nhỏ dần. Sau bước nhúng từ (đi qua embeding layer) ta có đầu vào của encoder và decoder là ma trận X kích thước $m * n$, m, n lần lượt là độ dài câu và số chiều của một Embedding vector. Để tính attention vector, ta cần ba ma trận Q, K, V (Query, Key, Value), chính là những hệ số mà mô hình cần huấn luyện. Ma trận Query và Key có tác dụng tính toán ra phân phối *score* cho các cặp từ. Ma trận Value sẽ dựa trên phân phối *score* để tính ra phân phối xác suất output. Ta có phương trình Attention:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1.2.10}$$

với d_k là số chiều của vector Key nhằm mục đích tránh tràn số.

1.2.4 Multi-head Attention

Sau quá trình Scale dot production chúng ta sẽ thu được 1 ma trận attention. Mỗi quá trình như vậy được gọi là 1 head của attention. Khi lặp lại quá trình này nhiều lần (như trong bài báo [6] là 3 heads) ta thu được quá trình Multi-head Attention. Quá trình này sẽ được tính toán song song. Sau khi thu được các ma trận attention, chúng ta sẽ nối các ma trận này theo cột để thu được ma trận tổng hợp Multi-head:



Hình 1.6 Scale dot product attention (trái) và Multi-head Attention (phải).

$$MultiHead(Q, K, V) = concatenate(head_1, head_2, ..., head_h)W_0 \tag{1.2.11}$$

trong đó

$$head_i = Attention(Q_i, K_i, V_i) \tag{1.2.12}$$

và W_0 là ma trận có chiều rộng bằng với chiều rộng của ma trận input.

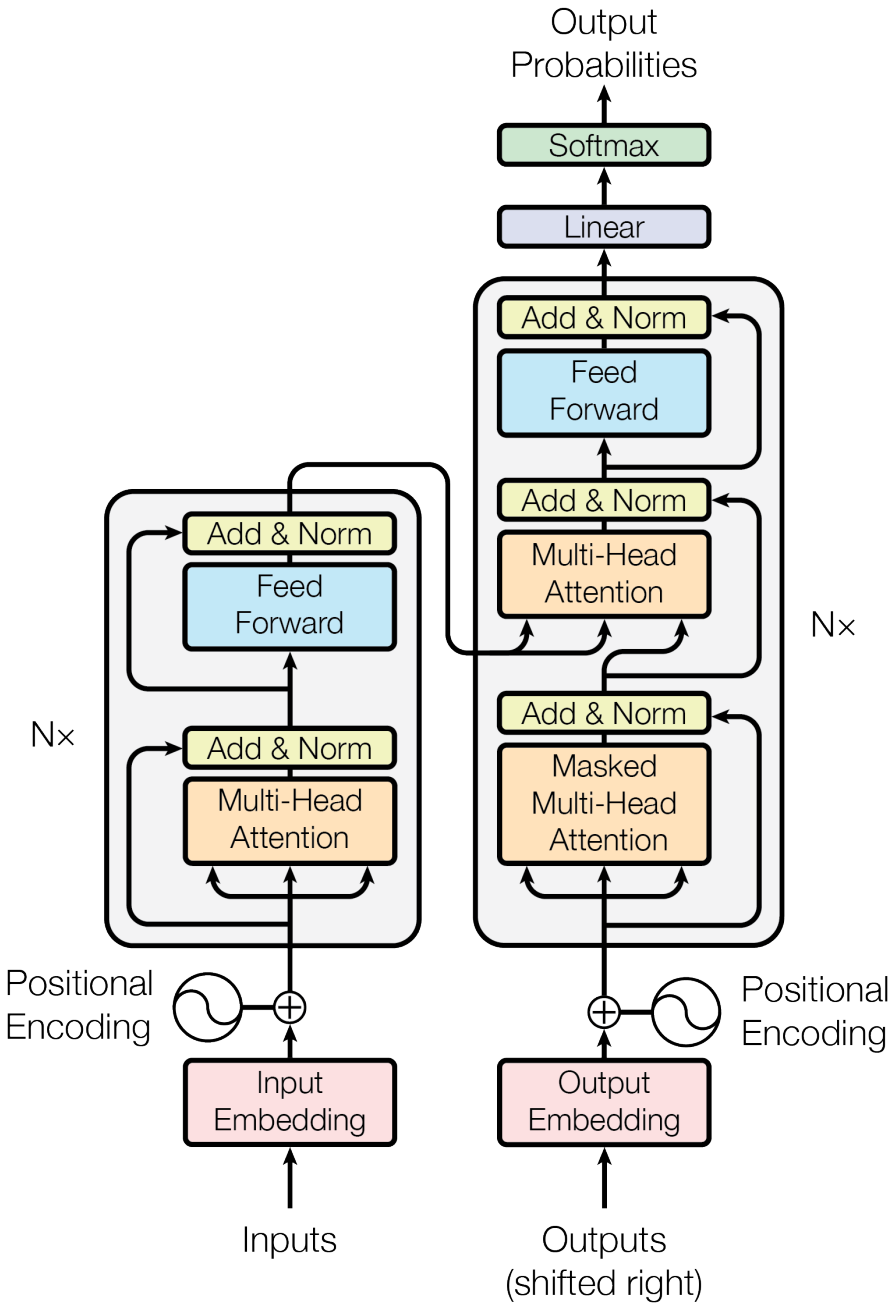
1.3 Mô hình Transformer

Bài báo [6] đưa ra kiến trúc Transfomer hoàn toàn khác với các kiến trúc RNN trước đây, mặc dù cả 2 đều thuộc lớp model seq2seq nhằm chuyển 1 câu văn input ở ngôn ngữ A sang 1 câu văn output ở ngôn ngữ B. Tuy nhiên mô hình hoàn toàn không sử dụng các kiến trúc Recurrent Neural Network của RNN mà chỉ sử dụng các attention layers để embedding các từ trong câu.

1.3.1 Kiến trúc mô hình

Transformer cũng có kiến trúc gồm bộ mã hóa và bộ giải mã giống như mô hình seq2seq. Trong mô hình Transformer, bộ mã hóa ánh xạ một chuỗi ký tự

đầu vào $(x_1, ..., x_n)$ thành một chuỗi biểu diễn liên tiếp $\mathbf{z} = (z_1, ..., z_n)$. Sau khi có $\mathbf{z}$, bộ giải mã thực hiện việc sinh ra một chuỗi đầu ra $(y_1, ..., y_m)$ tương ứng.



Hình 1.7 Kiến trúc Transformer.

1.3.2 Bộ mã hóa (Encoder)

Bộ mã hóa của Transformer được cấu tạo bởi 6 lớp giống nhau ghép chồng lên nhau. Mỗi lớp gồm hai lớp con, trong đó lớp đầu tiên là một Multi-head Attention như mô tả ở trên. Lớp con thứ hai là một mạng neuron đầy đủ truyền

thẳng. Bộ mã hóa sử dụng các kết nối dư xung quanh mỗi lớp con được đi theo bởi một lớp chuẩn hóa, tức là đầu ra của mỗi lớp con là một $LayerNorm(x + Sublayer(x))$ với $Sublayer(x)$ là hàm được thực hiện bởi chính layer con đó. Đầu ra có số chiều là $d_{model} = 512$.

1.3.3 Bộ giải mã (Decoder)

Bộ giải mã cũng giống bộ mã hóa, cũng được tạo bởi 6 lớp giống nhau. Tuy nhiên ngoài các lớp con như bộ mã hóa, lớp giải mã có thêm một lớp con nữa để thực hiện việc 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.

1.3.4 Multi-head Attention trong Transformer

Mô hình Transformer sử dụng multi-head attention theo ba cách khác nhau[2]:

- Trong *Encoder-decoder attention*, vector truy vấn được lấy từ lớp giải mã phía trước trong khi vector key, và vector value 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 mỗi chuỗi giá trị đầu vào, tức là thực thi ý tưởng truyền thống của mô hình sequence to sequence.

- Bộ mã hóa sử dụng các lớp attention với cả ba vector key, value, query đều được lấy từ giá trị đầu ra của khối phía trước trong bộ mã hóa. Mỗi vị trí trong lớp mã hóa có thể chú ý đến tất cả các vị trí của lớp phía trước.

- Tương tự, các lớp attention trong bộ 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 bộ giải mã để đảm bảo đặt tính tự hồi quy bằng cách đánh dấu tất cả các trí trị đầu vào lớp softmax tương ứng với các kết nối không hợp lệ (giá trị $-\infty$).

1.4 Mô hình BERT

BERT là viết tắt của cụm từ *Bidirectional Encoder Representation from Transformer* có nghĩa là mô hình biểu diễn từ theo 2 chiều ứng dụng kỹ thuật

Transformer. BERT được thiết kế để huấn luyện trước các biểu diễn từ (pre-train word embedding). Điểm đặc biệt ở BERT đó là nó có thể điều hòa cân bằng bối cảnh theo cả 2 chiều trái và phải. BERT được coi là một trong những mô hình tốt nhất, giải pháp để giải quyết với khó khăn về sự thiếu hụt dữ liệu đào tạo trong lĩnh vực xử lý ngôn ngữ tự nhiên ([7]).

1.4.1 Kiến trúc mô hình BERT

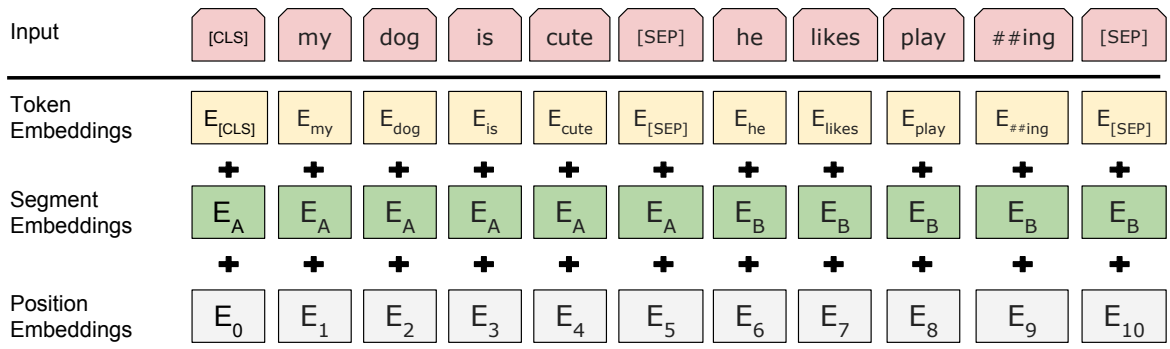
Mô hình BERT là mô hình mã hóa sử dụng nhiều lớp Transformer. Hiện tại có nhiều phiên bản khác nhau của model BERT. Các phiên bản đều dựa trên việc thay đổi kiến trúc của Transformer tập trung ở 3 tham số[7]:

- **L**: số lượng các block sub-layers trong Transformer.
- **H**: kích thước của embedding vector (hay còn gọi là hidden size).
- **A**: Số lượng head trong multi-head layer, mỗi một head sẽ thực hiện một self-attention.

BERT có 2 mô hình với kích thước:

- $BERT_{BASE}(\mathbf{L} = 12, \mathbf{H} = 768, \mathbf{A} = 12$: Tổng tham số 110 triệu.
- $BERT_{LARGE}(\mathbf{L} = 24, \mathbf{H} = 1024, \mathbf{A} = 16$: Tổng tham số 340 triệu.

1.4.2 Biểu diễn dữ liệu đầu vào



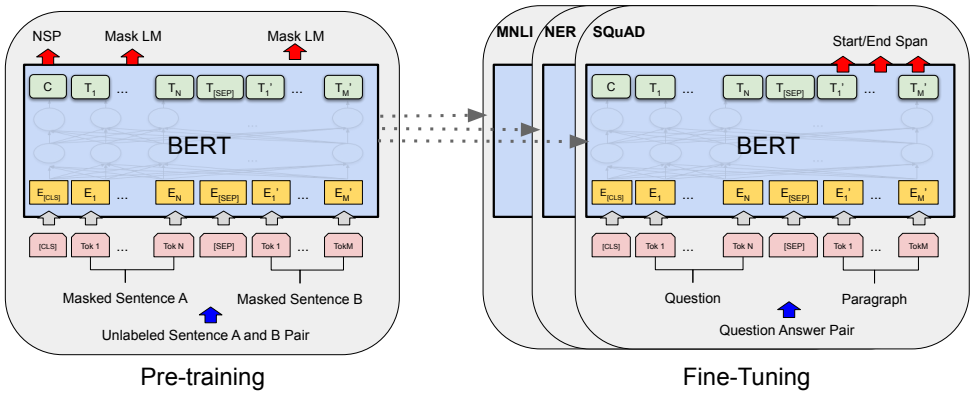
Hình 1.8 Biểu diễn dữ liệu đầu vào của BERT.

Cách biểu diễn dữ liệu đầu vào của BERT: Với mỗi từ cho trước, biểu diễn của từ đó được tạo bởi tổng hợp của các giá trị nhúng tương ứng với từ, đoạn cũng như vị trí. Cụ thể:

- BERT sử dụng WordPiece [8] để tách câu thành các từ nhỏ.
- Lớp nhúng vị trí được sử dụng với độ dài tối đa là 512.
- Ký tự đầu tiên của mỗi chuỗi luôn là ký tự đặc biệt `[CLS]` đại diện cho cả câu và sử dụng cho tác vụ phân lớp.
- Một cặp câu A và B được ghép với nhau phục vụ cho bài toán như Question and Answer được phân biệt bằng cách ngăn cách giữa hai câu bởi ký tự `[SEP]`.

1.4.3 Fine-tuning mô hình BERT

Một điểm đặc biệt ở BERT mà các embedding model trước đây chưa từng có đó là kết quả huấn luyện có thể fine-tuning được. Chúng ta sẽ thêm vào kiến trúc model một output layer để tùy biến theo tác vụ huấn luyện.



Hình 1.9 Tổng quan quá trình pre-training và fine-tuning của BERT.

Hình 1.9 cho thấy mô hình pre-training và mô hình fine-tuning cùng sử dụng một kiến trúc tương tự. Chúng ta sử dụng cùng một tham số pretrain để khởi tạo mô hình cho các tác vụ down stream khác nhau. Trong suốt quá trình fine-tuning thì toàn bộ các tham số của layers học chuyển giao sẽ được fine-tune. Đối với các tác vụ sử dụng input là một cặp sequence (pair-sequence) thì ta sẽ thêm token khởi tạo là `[CLS]` ở đầu câu, token `[SEP]` ở giữa để ngăn cách 2 câu (ví dụ như `question and answering`). Tiến trình áp dụng fine-tuning sẽ như sau [9]:

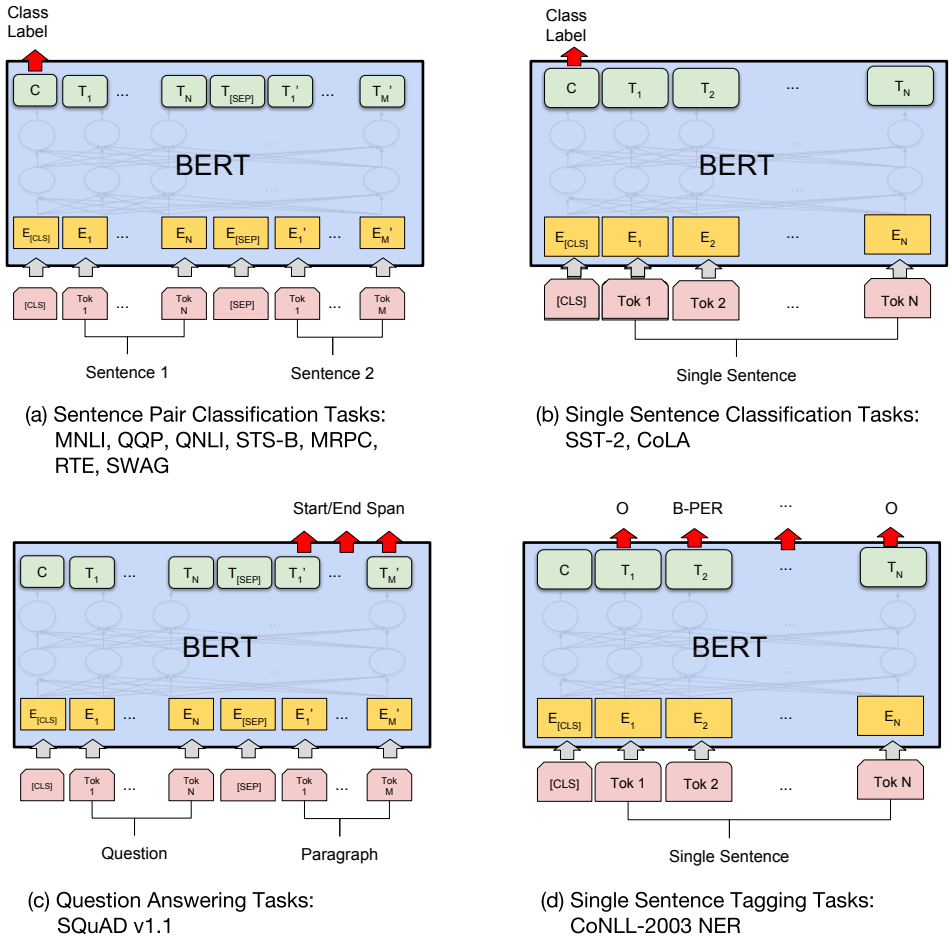
1. Embedding toàn bộ các token của cặp câu bằng các véc tơ nhúng từ pretrain model. Các token embedding bao gồm cả 2 token là `[CLS]` và `[SEP]` để đánh

dấu vị trí bắt đầu của câu hỏi và vị trí ngăn cách giữa 2 câu. 2 token này sẽ được dự báo ở output để xác định các phần `Start/End Spand` của câu output.

2. Các embedding véc tơ sau đó sẽ được truyền vào kiến trúc multi-head attention với nhiều block code (thường là 6, 12 hoặc 24 blocks tùy theo kiến trúc BERT). Ta thu được một véc tơ output ở encode.

3. Để dự báo phân phối xác suất cho từng vị trí từ ở decoder, ở mỗi time step chúng ta sẽ truyền vào decoder véc tơ output của encoder và véc tơ embedding input của decoder để tính encoder-decoder attention. Sau đó projection qua liner layer và softmax để thu được phân phối xác suất cho output tương ứng ở time step t.

4. Trong kết quả trả ra ở output của transformer ta sẽ cố định kết quả sao cho trùng với trùng với kết quả mong muốn ở input.



Hình 1.10 Chi tiết quá trình fine-tuning trong từng tác vụ của BERT.

Lưu ý quá trình huấn luyện chúng ta sẽ fine-tune lại toàn bộ các tham số của model BERT đã bỏ đi top linear layer và huấn luyện lại từ đầu các tham số của linear layer mà chúng ta thêm vào kiến trúc model BERT để customize lại phù hợp với bài toán.

1.4.4 Tiền huấn luyện mô hình

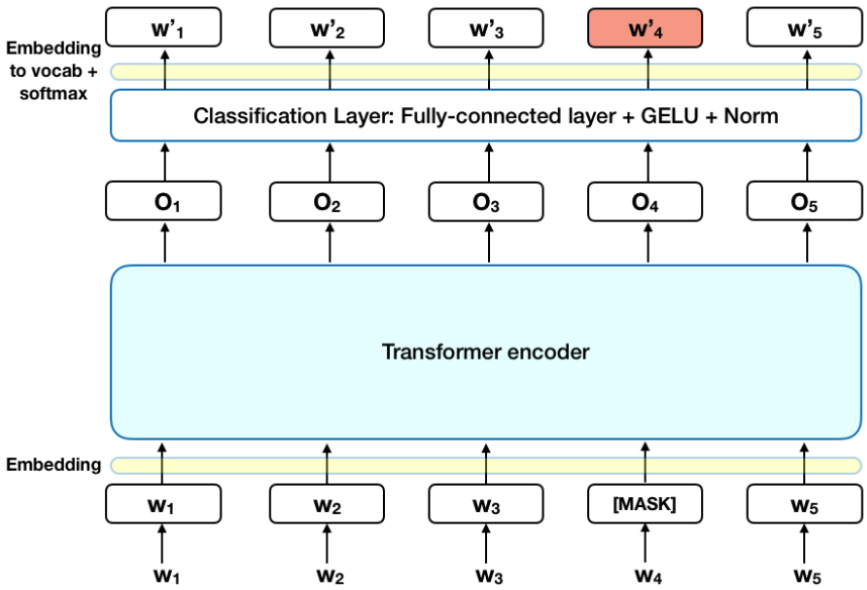
BERT được tiền huấn luyện (pre-train) bởi hai tác vụ học tự giám sát đó là *Masked LM (Mô hình ngôn ngữ đánh dấu)* và *Next Sentence Prediction (Dự đoán câu tiếp theo)*. Quá trình này thực hiện song song cả 2 tác vụ.

Masked ML (MLM)

Masked ML là một tác vụ cho phép chúng ta fine-tuning lại các biểu diễn từ trên các bộ dữ liệu unsupervised-text bất kỳ. Chúng ta có thể áp dụng Masked ML cho những ngôn ngữ khác nhau để tạo ra biểu diễn embedding cho chúng. Các bộ dữ liệu của tiếng anh có kích thước lên tới vài vài trăm tới vài nghìn GB được huấn luyện trên BERT đã tạo ra những kết quả khá ấn tượng. Khoảng 15% các token của câu input sẽ được thay thế bởi `[MASK]` token trước khi truyền vào model đại diện cho những từ bị đánh dấu (masked). Mô hình sẽ dựa trên các từ không được đánh dấu (non-masked) xung quanh `[MASK]` và đồng thời là bối cảnh của `[MASK]` để dự đoán giá trị gốc của từ bị đánh dấu. Về mặt kỹ thuật, việc dự đoán các từ đầu ra yêu cầu:

1. Thêm một lớp phân loại ở trên các output vectors O_i.
2. Nhân các output vectors với ma trận Embedding, chuyển đổi chúng về không gian từ điển.
3. Tính toán xác xuất của mỗi từ bằng hàm $softmax$.

Hàm loss function của BERT sẽ bỏ qua mất mát từ những từ không bị che dấu và chỉ đưa vào mất mát của những từ bị che dấu.



Hình 1.11 Sơ đồ kiến trúc BERT cho tác vụ Masked ML.

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 hai 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. Để giảm thiểu vấn đề này, BERT không phải lúc nào cũng đánh dấu ký tự bằng `[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.

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 (hoặc từ phải sang trái). Nhưng đây là đặc tính bù trừ cho sự gia tăng ý thức về bối cảnh. Việc lựa chọn ngẫu nhiên 15% số lượng các từ bị che dấu cũng tạo ra vô số các kịch bản input cho mô hình huấn luyện nên mô hình sẽ cần phải huấn luyện rất lâu mới học được toàn diện các khả năng.

Next Sentence Prediction

Rất nhiều tác vụ thực tế dựa trên kiến thức về mối liên hệ giữa hai câu văn, trong khi 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 đâu là mối quan hệ giữa các từ, BERT sử dụng tác vụ *next sentence prediction* có thể được tạo ra một cách đơn giản bởi bất kỳ bộ dữ liệu đơn ngôn ngữ nào.Đây là một bài toán phân loại học có giám sát với 2 nhãn (hay còn gọi là phân loại nhị phân). Input đầu vào của mô hình là một cặp câu (pair-sequence) sao cho 50% câu thứ 2 được lựa chọn là câu tiếp theo của câu thứ nhất và 50% được lựa chọn một cách ngẫu nhiên từ bộ văn bản mà không có mối liên hệ gì với câu thứ nhất. Nhãn của mô hình sẽ tương ứng với `IsNext` khi cặp câu là liên tiếp hoặc `NotNext` nếu cặp câu không liên tiếp. Chúng ta cần đánh dấu các vị trí đầu câu thứ nhất bằng token `[CLS]` và vị trí cuối câu bằng token `[SEP]`. Các token này có tác dụng nhận biết các vị trí bắt đầu và kết thúc của từng câu thứ nhất và thứ hai. Ví dụ:

1.
   - Input = `[CLS]` the man went to [MASK] store `[MASK]` he bought a gallon [MASK] milk `[SEP]`.
   - Label = `IsNext`.
2.
   - Input = `[CLS]` the man [MASK] to the store `[MASK]` penguin [MASK] are ##flight [MASK] less bird `[SEP]`.
   - Label = `NotNext`.

CHƯƠNG 2. NHẬN DIỆN THỰC THỂ CÓ TÊN TRONG KHAI BÁO HẢI QUAN

2.1 Bài toán nhận diện thực thể có tên (NER)

2.1.1 Định nghĩa

Thực thể có tên (Named entities) là những cụm danh từ để chỉ định các loại cụ thể, chẳng hạn tổ chức, người, ngày tháng,... Mục đích của bài toán *Nhận diện thực thể có tên (NER)* là xác định tất cả các NEs được nhắc đến trong văn bản. Việc này có thể chia thành 2 nhiệm vụ con là:

1. Xác định NEs.
2. Xác định loại của các NEs.

Một số vấn đề của bài toán:

- Các NEs thường mập mờ, không rõ ràng về loại của nó. VD: *Luật* vừa có thể là tên người thuộc loại PERSON, vừa có thể là tên của một ngành học thuộc loại MAJOR.
- Các NEs gồm nhiều từ hợp thành như: Trường Đại học Bách khoa Hà Nội, Viện Toán ứng dụng và Tin học => Cần có khả năng xác định điểm bắt đầu và kết thúc của ngôn ngữ.

2.1.2 Quy ước gắn thẻ BIO

Với mỗi entity có nhãn T, ta có hai nhãn B-T và I-T. B-T là begin type T, I-T là inside type T. Ngoài ra ta còn có nhãn O cho biết outside của thực thể. Ví dụ: Thầy Trần Ngọc Thăng là giảng viên ngành Toán Tin của Viện Toán ứng dụng và Tin học.

Word	Thầy	Trần	Ngọc	Thăng
Tag	O	B-PERSON	I-PERSON	I-PERSON
Word	là	giảng	viên	ngành
Tag	O	B-JOB	I-JOB	O
Word	Toán	Tin	của	Viện
Tag	B-MAJOR	I-MAJOR	O	B-ORG
Word	Toán	ứng	dụng	và
Tag	I-ORG	I-ORG	I-ORG	I-ORG
Word	Tin	học		
Tag	I-ORG	I-ORG		

Bảng 2.1 Ví dụ gán nhãn BIO.

2.2 Dữ liệu bài toán nhận diện thực thể có tên trong khai báo hải quan

2.2.1 Mô tả dữ liệu

Dữ liệu được lấy từ mô tả tờ khai hải quan về ô tô và phương tiện không phải chạy trên đường sắt đường thủy bao gồm các bản ghi bằng tiếng Việt. Cần nói thêm dữ liệu này được lấy từ các bản ghi trong năm 2015 và 2017 và một vài tháng đầu năm 2020 và mặt hàng chủ yếu các phương tiện là xe ô tô. Dữ liệu đã được gán nhãn như hình 2.1, **Word** là từ, **Tag** là nhãn thực thể tương ứng với **Word**, **Sentence #** là vị trí của word nằm trong câu nào. Dữ liệu gồm 9420 câu ở file train và 2354 câu ở file test.

	Word	Tag	Sentence #
58	xe	O	Sentence: 3
59	ô	B-TYPE	Sentence: 3
60	tô	I-TYPE	Sentence: 3
61	tải	I-TYPE	Sentence: 3
62	van	O	Sentence: 3
63	02	O	Sentence: 3
64	chỗ	O	Sentence: 3
65	,	O	Sentence: 3

Hình 2.1 Một phần dữ liệu bài toán.

2.2.2 Mô tả thực thể cần xác định

Với dữ liệu mô tả hàng hóa trong tờ khai hải quan ở mục 2.2.1, chúng ta cần xác định các thực thể sau:

Nhãn thực thể	**Mô tả**
BRAND	Tên thương hiệu xe Ví dụ: *Mercedes*,*Honda*
TYPE	Loại xe Ví dụ: *ô tô con*,*ô tô tải*
NAME	Tên của xe, hay tên mô den (model) *GLS400 4Matic*, *maybach s600*
ENGINE FULE TYPE	loại động cơ và nhiên liệu trong việc lựa chọn xe Ví dụ: *xăng*, textitdiezel
YEAR	Năm phát hành của xe Ví dụ: Xe CRV 2021 thì tag YEAR là *2021*
STATUS	Tình trạng xe Ví dụ: *đã qua sử dụng* *chưa qua sử dụng* *mới 100%*

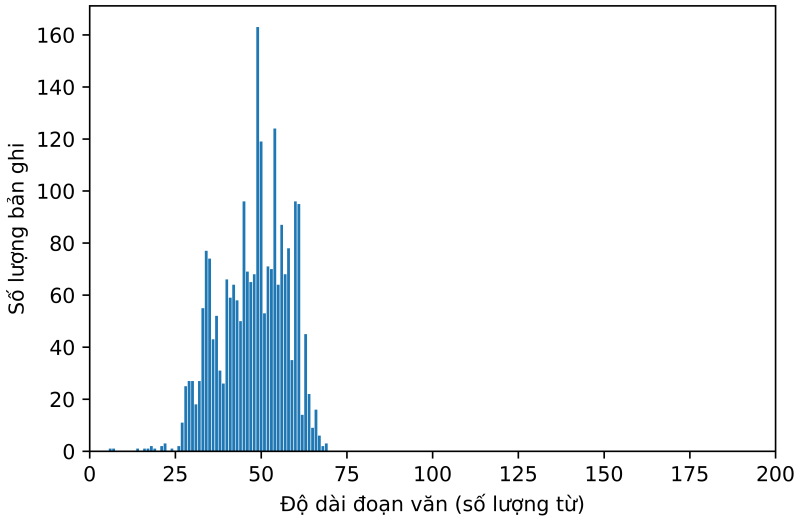
Bảng 2.2 Bảng mô tả nhãn dữ liệu

2.3 Áp dụng mô hình BERT để giải quyết bài toán

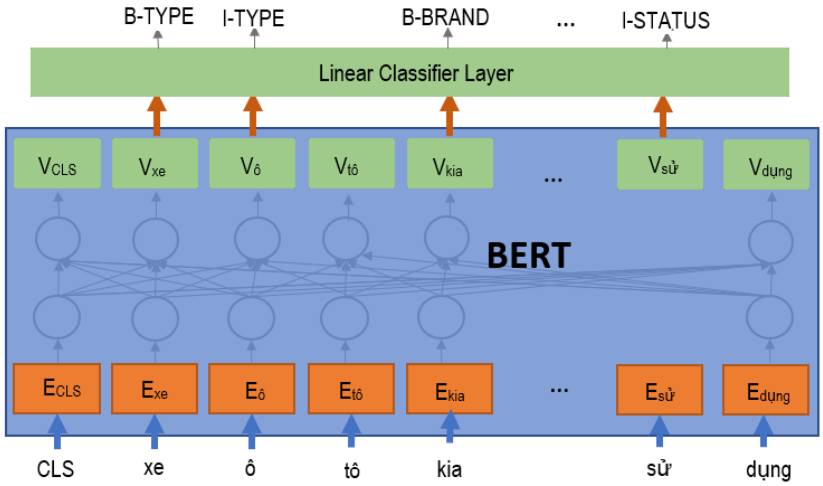
Hình 2.2 cho thấy đa số các câu có độ dài trong khoảng [30, 60]. Sử dụng mô hình RNN truyền thống để giải quyết bài toán sẽ không phù hợp vì những nhược điểm đã đề cập trong phần 1.2.1.

Mặt khác, bộ dữ liệu của bài toán là khá nhỏ (9420 câu ở file train và 2354 câu ở file test), nếu tạo một mô hình từ đầu và chỉ huấn luyện trên bộ dữ liệu đăc thù này sẽ khó có thể khái quát hóa. Vì vậy, tác giả quyết định sử dụng

BERT - mô hình đã được huấn luyện từ trước với bộ dữ liệu tiếng Việt. Về cơ bản mô hình sẽ có kiến trúc như hình 2.3.



Hình 2.2 Thống kê độ dài của một câu.



Hình 2.3 Kiến trúc mô hình BERT-NER.

CHƯƠNG 3. Cài đặt và đánh giá kết quả

3.1 Kiểm tra dữ liệu

Sau quá trình kiểm tra, tác giả nhận thấy dữ liệu bị gán nhãn sai khá nhiều. Ví dụ:

- Dữ liệu bị gán nhãn sai nghiêm trọng đối với xe lăn ở hầu hết các trường như TYPE, BRAND, NAME,...

	Word	Tag	Sentence #		Word	Tag	Sentence #
0	xe	O	Sentence: 1	**8**	.	O	Sentence: 1
1	có	O	Sentence: 1	**9**	hiệu	O	Sentence: 1
2	bô	O	Sentence: 1	**10**	lucass	O	Sentence: 1
3	cho	O	Sentence: 1	**11**	,	O	Sentence: 1
4	người	O	Sentence: 1	**12**	hàng	O	Sentence: 1
5	khuyết	O	Sentence: 1	**13**	mới	B-STATUS	Sentence: 1
6	tật	O	Sentence: 1	**14**	100	I-STATUS	Sentence: 1
7	gk-96	O	Sentence: 1	**15**	%	I-STATUS	Sentence: 1

Hình 3.1 Dữ liệu bị lỗi với xe lăn.

- Trường NAME bị gán nhãn sai rất nhiều.

	Word	Tag	Sentence #
11857	hiệubmw	O	Sentence: 322
11858	x6	B-NAME	Sentence: 322
11859	xdrive35i	I-NAME	Sentence: 322
11860	(	O	Sentence: 322
11861	bmw	B-BRAND	Sentence: 322
11862	x6	O	Sentence: 322
11863	xdrive35i	O	Sentence: 322
11864	)	O	Sentence: 322

Hình 3.2 Từ x6 xdrive35i bị gán 2 nhãn khác nhau trong cùng một câu.

	Word	Tag	Sentence #		Word	Tag	Sentence #
3974	audi	B-BRAND	Sentence: 104	**4290**	audi	B-BRAND	Sentence: 112
3975	q2	B-NAME	Sentence: 104	**4291**	a4	O	Sentence: 112
3976	design	I-NAME	Sentence: 104	**4292**	2.0	O	Sentence: 112
3977	35	I-NAME	Sentence: 104	**4293**	tfsi	O	Sentence: 112
3978	tfsi	I-NAME	Sentence: 104	**4294**	ultra	O	Sentence: 112

Hình 3.3 Trường NAME bị gán nhãn sai trong nhiều câu.

- Trường TYPE bị gán nhãn sai rất nhiều.

	Word	Tag	Sentence #		Word	Tag	Sentence #		Word	Tag	Sentence #
58	xe	O	Sentence: 3	**346**	xe	B-TYPE	Sentence: 11	**247**	ô	O	Sentence: 9
59	ô	B-TYPE	Sentence: 3	**347**	ô	I-TYPE	Sentence: 11	**248**	tô	O	Sentence: 9
60	tô	I-TYPE	Sentence: 3	**348**	tô	I-TYPE	Sentence: 11	**249**	con5	O	Sentence: 9
61	tải	I-TYPE	Sentence: 3	**349**	bmw	B-BRAND	Sentence: 11	**250**	chỗ	O	Sentence: 9
62	van	O	Sentence: 3	**350**	x1	B-NAME	Sentence: 11	**251**	5	O	Sentence: 9

Hình 3.4 Trường TYPE bị gán nhãn sai.

- Ngoài ra trường BRAND, YEAR đôi khi cũng bị gán nhãn sai.

3.2 Gán nhãn lại cho dữ liệu

Những thao tác dưới đây mới chỉ thực hiện với file train, với file test chúng ta làm tương tự.

Load dữ liệu.

```
import numpy as np
import pandas as pd
train = pd.read_csv("/content/train.csv")
train['Word'] = train['Word'].astype(str)
```

Mã 3.1: Load data

Xóa những hàng chứa missing values.

```
train.dropna(inplace=True)
train.reset_index(inplace=True, drop=True)
```

Mã 3.2: Xử lý missing values

Chuẩn hóa các từ dạng **-word-** thành **word**. Ví dụ từ **-ky160-a-46-** (câu 3941).

```
def chuanhoaword(x): #nhung tu thuoc dang -abc- se thanh abc
  if len(x) <= 2:
    return x
  if x[0] == "-":
    x = x.replace("-", "")
  if x[-1] == "-":
    x = x[:-1]
  return x
train["Word"] = train.Word.apply(chuanhoaword)
```

Mã 3.3: Xử lý missing values

Những từ kiểu **model word** hay **model : word** thì word sẽ có nhãn là B-NAME. Ví dụ **model : postar** (câu 24) thì postar có nhãn là B-NAME.

```
for i in range(train.shape[0]-2):
  if train.at[i, "Word"] == "model":
    if (train.at[i+1, "Word"] != ":") and (train.at[i+1, "Tag"]=="O"):
      train.at[i+1, "Tag"] = "B-NAME"
    elif (train.at[i+2, "Word"] != ":") and (train.at[i+2, "Tag"]=="O"):
```

```
6       train.at[i+2, "Tag"] = "B-NAME"
```

Mã 3.4: Gán nhãn B-NAME theo quy tắc model

Rất nhiều dấu phẩy (",") bị gán nhãn sai, nên gán nhãn lại cho dấulà O.

```
1 for i in range(train.shape[0]):
2   if train.at[i, "Word"] == ",":
3     train.at[i, "Tag"] = "O"
```

Mã 3.5: Gán nhãn dấu phẩy là O

3.2.1 Một số dữ liệu cần gán nhãn lại

Sau khi xem xét những câu mô tả hàng hóa cho xe lăn (dữ liệu bị gán nhãn sai nghiêm trọng), tác giả đã rút ra một số quy luật và tạo ra các file **Brand.txt**, **Name.txt**, để lưu thông tin về BRAND, NAME, TYPE của xe lăn.

Những từ bắt đầu mảng rule (trong code bên dưới) sẽ là B-NAME.

```
1 rule = ["x-", "w-", "sc-", "s-", "uck-", "ucf-", "xk-", "kt-", "ms", "ky", "h0",
      "gk-"]
2 for i in range(train.shape[0]):
3   for j in rule:
4     if train.at[i, "Word"].startswith(j):
5       train.at[i, "Tag"] = "B-NAME"
6       break
```

Mã 3.6: Gán nhãn B-NAME xe lăn theo quy tắc

Những từ **full throttle**, **ec02 18**, **kjt802 chrome** là tên của một số loại xe lăn.

```
1  for i in range(train.shape[0]-1):
2    if (train.at[i, "Word"] == "full") and (train.at[i+1, "Word"] == "throttle"):
3      train.at[i, "Tag"] == "B-NAME"
4      train.at[i+1, "Tag"] == "I-NAME"
5
6  for i in range(train.shape[0]-1):
7    if (train.at[i, "Word"] == "ec02") and (train.at[i+1, "Word"] == "18"):
8      train.at[i, "Tag"] == "B-NAME"
9      train.at[i+1, "Tag"] == "I-NAME"
10
11 for i in range(train.shape[0]-1):
```

```
  if (train.at[i, "Word"] == "kjt802") and (train.at[i+1, "Word"] == "chrome"):
    train.at[i, "Tag"] == "B-NAME"
    train.at[i+1, "Tag"] == "I-NAME"
```

Mã 3.7: Gán nhãn một số từ thành NAME của xe lăn.

Sử dụng file **Name.txt** để gán nhãn B-NAME cho xe lăn.

```
f = open("Name.txt", "r")

kt = kt = f.readline().replace("\n","")
Rule2 = []
while kt !="":
  Rule2.append(kt)
  kt = f.readline().replace("\n","")
f.close()
print(Rule2)
```

Mã 3.8: Đọc file Name.txt

```
['fhq011q', 'w65cmxl103cmxh86', 'ky608lgcj-46', 'ms-809ij', 'ms-607gcj', 'w06-
    t3j00-53-am06',...]
```

```
for i in range(len(Rule2)):
  Rule2[i] = chuanhoaword(Rule2[i])
  Rule2[i] = Rule2[i].replace("\t","")
Rule2 = set(Rule2)
for i in range(train.shape[0]):
  if train.at[i, "Word"] in Rule2:
    train.at[i, "Tag"] = "B-NAME"
```

Mã 3.9: Gán nhãn B-NAME cho xe lăn

Gán nhãn BRAND cho những từ trong file **Brand.txt**

```
f = open("Brand.txt", "r")

kt = f.readline()
Brand = []
while kt !="":
  if kt.endswith(" "):
    kt = kt.replace(" ", "")
  kt = kt.replace("\n","")
  kt = kt.split(" ")
  Brand.append(kt)
  kt = f.readline()
```

```
12
13 f.close()
14 print(Brand)
```

Mã 3.10: Đọc file Brand.txt

```
[['ducati', 'scrambler'], ['lucass'], ['jerry', 'medical'], ['kawamura'], ['
    lucass'], ['foshan', 'dongfang'],...]
```

```
1 for i in range(train.shape[0]):
2   for j in Brand:
3     if (i + len(j) < train.shape[0]) and (train.at[i, "Word"] == j[0])\
4       and (train.at[i+len(j)-1, "Word"] == j[-1]):
5       for k in range(len(j)):
6         if k == 0:
7           train.at[i+k, "Tag"] = "B-BRAND"
8         else:
9           train.at[i+k, "Tag"] = "I-BRAND"
```

Mã 3.11: Gán nhãn BRAND cho xe lăn

3.2.2 Gán nhãn NAME và BRAND cho toàn bộ dữ liệu

Sử dụng từ điển phương tiện[1], duyệt từng tù trong câu nếu có từ giống brand trong từ điển thì gán nhãn BRAND, duyệt tiếp có từ nào thuộc name của BRAND thì gán nhãn NAME.

```
1 import json
2 with open("/content/drive/MyDrive/FLAIR_NER/TuDien/dict_car_v3 (1).json") as f:
3   tudien = json.load(f)
4
5 for key in tudien.keys():
6   brand = key.lower().split(" ")
7   name = []
8   for i in tudien[key]:
9     name.append(i.lower().split(" "))
10  for i in range(train.shape[0]):
11    for j in range(len(brand)):
12      if train.at[i+j, "Word"] != brand[j]:
13        break
14      else:
15        pass
```

[1]Từ điển này được anh Lương Khắc Mạnh cung cấp

```
    else:
      for k in range(len(brand)):
        if k == 0:
          train.at[i+k, "Tag"] = "B-BRAND"
        else:
          train.at[i+k, "Tag"] = "I-BRAND"
      for start in range(i, 0, -1):
        if train.at[start, "Sentence #"] != train.at[i, "Sentence #"]:
          break
      for end in range(i, train.shape[0]):
        if train.at[end, "Sentence #"] != train.at[i, "Sentence #"]:
          break
      for k in range(start, end+1):
        for x in range(len(name)):
          for z in range(len(name[x])):
            if train.at[k+z, "Word"] != name[x][z]:
              break
            else:
              pass
          else:
            for y in range(len(name[x])):
              if y == 0:
                train.at[k+y, "Tag"] = "B-NAME"
              else:
                train.at[k+y, "Tag"] = "I-NAME"
```

Mã 3.12: Sử dụng từ điển để gán nhãn BRAND và NAME

Gán nhãn BRAND

Những từ nào có dạng là **hiệu word** hoặc **hiệu : word** thì đánh nhãn là B-BRAND. Ví dụ: hiệu bmw thì bmw là B-BRAND.

```
#Từ tiếp theo từ "hieu" neu co length >= 2 se la B-BRAND, neu length <=2 thi se
    la tu ke tiep, tu dang truoc tu hieu khac chu "ky".
for i in range(1, train.shape[0]-1):
  if (train.at[i, "Word"] =="hiệu") and ((train.at[i-1, "Word"] != "ký") or (
      train.at[i-1, "Word"] != "kí")):
    if len(train.at[i+1, "Word"])>2:
      train.at[i+1, "Tag"] = "B-BRAND"
    elif i+2<train.shape[0]:
      train.at[i+2, "Tag"] = "B-BRAND"
```

Mã 3.13: Gán nhãn B-BRAND theo quy tắc

Vì BRAND là những danh từ riêng, vì vậy những từ mà được gán nhãn là BRAND thì toàn bộ từ điển sẽ gán nhãn là BRAND.

```
tudien_BBRAND = {}
tudien_IBRAND = {}
for i in range(train.shape[0]):
  if train.at[i, "Tag"] == "B-BRAND":
    if train.at[i, "Word"] in tudien_BBRAND.keys():
      tudien_BBRAND[train.at[i, "Word"]] += 1
    else:
      tudien_BBRAND[train.at[i, "Word"]] = 1
  elif train.at[i, "Tag"] == "I-BRAND":
    if train.at[i, "Word"] in tudien_IBRAND.keys():
      tudien_IBRAND[train.at[i, "Word"]] += 1
    else:
      tudien_IBRAND[train.at[i, "Word"]] = 1

#xoa nhung tu bi danh nhan sai
del tudien_BBRAND["chữ"]
del tudien_BBRAND["."]
del tudien_BBRAND["cần"]

#Nhung tu duoc gan nhan la B-BRAND > 5 lan se duoc su dung de danh nhan B-BRAND
    cho toan bo du lieu
BBRAND = []
for i in tudien_BBRAND.keys():
  if tudien_BBRAND[i] > 5:
    BBRAND.append(i)
for i in range(train.shape[0]):
  if train.at[i, "Word"] in BBRAND:
    train.at[i, "Tag"] = "B-BRAND"

#"-" bi gan nhan sai nen gan lai thanh O
for i in range(train.shape[0]):
  if train.at[i, "Word"] == "-":
    train.at[i, "Tag"] = "O"

#xoa dau tru khoi tu dien I-BRAND
del tudien_IBRAND['-']

#Nhung tu duoc gan nhan la I-BRAND > 2 lan se duoc su dung de danh nhan I-BRAND
    cho toan bo du lieu
IBRAND = []
for i in tudien_IBRAND.keys():
```

```
  if tudien_IBRAND[i] > 2:
    IBRAND.append(i)
for i in range(train.shape[0]):
  if train.at[i, "Word"] in IBRAND:
    train.at[i, "Tag"] = "I-BRAND"
```

Mã 3.14: Gán nhãn BRAND cho toàn bộ dữ liệu

Những từ cùng câu đứng trước I-BRAND sẽ được gán nhãn là B-BRAND.

```
for i in range(train.shape[0]-1):
  if train.at[i+1, "Tag"] == "I-BRAND" and train.at[i, "Tag"] == "O"\
  and train.at[i+1, "Sentence #"] == train.at[i, "Sentence #"]:
    train.at[i, "Tag"] = "B-BRAND"
```

Mã 3.15: B-BRAND đứng trước I-BRAND

Các từ ['&', ',', '.', ':', 'bơm', 'chỗ', 'hiệu', 'hàng', 'hãng', 'kích', 'mới', 'tay', 'đạp'] ở B-BRAND, ta sẽ gán lai nhãn O cho chúng.

```
loai_bo_BBRAND = ['&', ',', '.', ':', 'bơm', 'chỗ', 'hiệu', 'hàng', 'hãng', 'kích
    ', 'mới', 'tay', 'đạp']
for i in range(train.shape[0]):
  if (train.at[i, "Word"] in loai_bo_BBRAND) and (train.at[i, "Tag"] == "B-BRAND"
      ):
    train.at[i, "Tag"] = "O"
```

Mã 3.16: Sửa lỗi một số từ bị gán nhầm nhãn B-BRAND

Gán nhãn NAME

Vì NAME là những danh từ riêng, vì vậy những từ mà được gán nhãn là NAME thì toàn bộ từ điển sẽ gán nhãn là NAME.

```
tudien_BNAME = {}
tudien_INAME = {}
for i in range(train.shape[0]):
  if train.at[i, "Tag"] == "B-NAME":
    if train.at[i, "Word"] in tudien_BNAME.keys():
      tudien_BNAME[train.at[i, "Word"]] += 1
    else:
      tudien_BNAME[train.at[i, "Word"]] = 1
  elif train.at[i, "Tag"] == "I-NAME":
    if train.at[i, "Word"] in tudien_INAME.keys():
      tudien_INAME[train.at[i, "Word"]] += 1
    else:
```

```
      tudien_INAME[train.at[i, "Word"]] = 1

#"s" và ")" bi gan nhan sai
del tudien_BNAME[")"]
del tudien_BNAME["s"]

#Nhung tu duoc gan nhan la B-NAME > 10 lan se duoc su dung de danh nhan B-NAME
    cho toan bo du lieu
BNAME = []
for i in tudien_BNAME.keys():
  if tudien_BNAME[i] > 10:
    BNAME.append(i)
for i in range(train.shape[0]):
  if train.at[i, "Word"] in BNAME:
    train.at[i, "Tag"] = "B-NAME"

#Nhung tu duoc gan nhan la I-NAME > 3 lan se duoc su dung de danh nhan I-NAME cho
     toan bo du lieu
INAME = []
for i in tudien_INAME.keys():
  if tudien_INAME[i] > 3:
    INAME.append(i)
for i in range(train.shape[0]):
  if train.at[i, "Word"] in INAME:
    train.at[i, "Tag"] = "I-NAME"
```

Mã 3.17: Gán nhãn NAME cho toàn bộ dữ liệu

Những từ cùng câu đứng trước I-NAME sẽ được gán là B-NAME.

```
#Nhung tu nao dung truoc I-NAME se la B-NAME
for i in range(train.shape[0]-1):
  if train.at[i+1, "Tag"] == "I-NAME" and train.at[i, "Tag"] == "O"\
  and train.at[i+1, "Sentence #"] == train.at[i, "Sentence #"]:
    train.at[i, "Tag"] = "B-NAME"
```

Mã 3.18: B-NAME đứng trước I-NAME

Các từ ['dùng', 'lượng', 'mâm', 'năm', 'sản', 'sức', 'trọng', 'xe', 'đ', 'đc', 'đầu', 'để', 'động'] ở B-NAME, ta sẽ gán lai nhãn O cho chúng.

```
loai_bo_BNAME = ['dùng', 'lượng', 'mâm', 'năm', 'sản', 'sức', 'trọng', 'xe', 'đ',
     'đc', 'đầu', 'để', 'động']
for i in range(train.shape[0]):
  if (train.at[i, "Word"] in loai_bo_BNAME) and (train.at[i, "Tag"] == "B-NAME"):
```

```
4     train.at[i, "Tag"] = "O"
```

Mã 3.19: Sửa lỗi một số từ bị gán nhầm nhãn B-NAME

3.2.3 Gán nhãn TYPE cho toàn bộ dữ liệu

Xử lý những từ dạng "m", "ô"thành chữ "mô"(mô tô).

```
drop_list = []
for i in range(train.shape[0]-1):
  if (train.at[i, "Word"] == "m") and (train.at[i+1, "Word"] == "ô"):
    train.at[i+1, "Word"] = "mô"
    drop_list.append(i)
train.drop(drop_list, inplace=True)
train.reset_index(inplace=True, drop=True)
```

Mã 3.20: Sửa lỗi từ mô tô

Tác giả đã tạo file **Type.txt** gồm những loại xe như xe thể thao, xe đẩy hàng, xe chở quân nghiệp vụ,... để gán nhãn TYPE cho dữ liệu.

```
f = open("Type.txt", "r")

kt = f.readline()
Type = []
while kt !="":
  if kt.endswith(" "):
    kt = kt.replace(" ", "")
  kt = kt.replace("\n","")
  kt = kt.split(" ")
  Type.append(kt)
  kt = f.readline()

f.close()
print(Type)
```

Mã 3.21: Đọc file Type.txt

```
[['xe', 'lăn'], ['xe', 'lăn', 'bô'], ['xe', 'lăn', 'ghế', 'bô'], ['xe', 'lăn', 'c
    ó', 'bô'], ['wheel', 'chair'],...]
```

```
for i in range(train.shape[0]):
  for j in Type:
    if (i + len(j) < train.shape[0]) and (train.at[i, "Word"] == j[0])\
```

```
      and (train.at[i+len(j)-1, "Word"] == j[-1]) and (train.at[i+len(j)-2, "Word
          "] == j[-2]):
      for k in range(len(j)):
        if k == 0:
          train.at[i+k, "Tag"] = "B-TYPE"
        else:
          train.at[i+k, "Tag"] = "I-TYPE"
```

Mã 3.22: Gán nhãn TYPE cho dữ liệu

Những từ ô tô, ô tô con, ô tô đầu kéo, cần trục bánh lốp, ô tô x chỗ, ô tô con x chỗ => gán nhãn I-TYPE

```
for i in range(train.shape[0] - 1):
  if (train.at[i, "Word"] == "ô") and (train.at[i+1, "Word"] == "tô"):
    train.at[i, "Tag"] = "I-TYPE"
    train.at[i+1, "Tag"] = "I-TYPE"

for i in range(train.shape[0] - 2):
  if (train.at[i, "Word"] == "ô")and (train.at[i+1, "Word"] == "tô") and (train.
      at[i+2, "Word"] == "tải"):
    train.at[i, "Tag"] = "I-TYPE"
    train.at[i+1, "Tag"] = "I-TYPE"
    train.at[i+2, "Tag"] = "I-TYPE"

for i in range(train.shape[0] - 3):
  if (train.at[i, "Word"] == "ô")and (train.at[i+1, "Word"] == "tô") and (train.
      at[i+2, "Word"] == "đầu") and (train.at[i+2, "Word"] == "kéo"):
    train.at[i, "Tag"] = "I-TYPE"
    train.at[i+1, "Tag"] = "I-TYPE"
    train.at[i+2, "Tag"] = "I-TYPE"
    train.at[i+3, "Tag"] = "I-TYPE"

for i in range(train.shape[0] - 3):
  if (train.at[i, "Word"] == "cần")and (train.at[i+1, "Word"] == "trục") and (
      train.at[i+2, "Word"] == "bánh") \
  and (train.at[i+2, "Word"] == "lốp"):
    train.at[i, "Tag"] = "I-TYPE"
    train.at[i+1, "Tag"] = "I-TYPE"
    train.at[i+2, "Tag"] = "I-TYPE"
    train.at[i+3, "Tag"] = "I-TYPE"

for i in range(train.shape[0] - 3):
  if (train.at[i, "Word"] == "ô")and (train.at[i+1, "Word"] == "tô") and (train.
      at[i+2, "Word"].isdigit()) and (train.at[i+2, "Word"] == "chỗ"):
```

```
    train.at[i, "Tag"] = "I-TYPE"
    train.at[i+1, "Tag"] = "I-TYPE"
    train.at[i+2, "Tag"] = "I-TYPE"
    train.at[i+3, "Tag"] = "I-TYPE"

for i in range(train.shape[0] - 4):
  if (train.at[i, "Word"] == "ô")and (train.at[i+1, "Word"] == "tô") and (train.
      at[i+2, "Word"] == "con")\
  and (train.at[i+3, "Word"].isdigit()) and (train.at[i+4, "Word"] == "chỗ"):
    train.at[i, "Tag"] = "I-TYPE"
    train.at[i+1, "Tag"] = "I-TYPE"
    train.at[i+2, "Tag"] = "I-TYPE"
    train.at[i+3, "Tag"] = "I-TYPE"
    train.at[i+4, "Tag"] = "I-TYPE"
```

Mã 3.23: Gán nhãn I-TYPE cho một số từ

Những từ cùng câu đứng trước I-TYPE sẽ được gán nhãn là B-TYPE.

```
for i in range(train.shape[0]-1):
  if train.at[i+1, "Tag"] == "I-TYPE" and train.at[i, "Tag"] == "O"\
  and train.at[i+1, "Sentence #"] == train.at[i, "Sentence #"]:
    train.at[i, "Tag"] = "B-TYPE"
```

Mã 3.24: B-TYPE đứng trước I-TYPE

Một số kí hiệu bị gán nhãn sai, ta sẽ gán nhãn cho nó là O.

```
loai_bo =['(', ')', ',', ';', '?', ':', '.', '[', ']', '{', '}']
for i in range(train.shape[0]):
  if (train.at[i, "Word"] in loai_bo):
    train.at[i, "Tag"] = "O"
```

Mã 3.25: Gán nhãn O cho một số ký hiệu

3.2.4 Gán nhãn YEAR cho toàn bộ dữ liệu

Những số >1900 và <2022 sẽ được gán nhãn là B-YEAR.

```
def check_year(x):
  if x.isdigit():
    x = int(x)
    if (x>1900) and (x<2021):
      return True
  return False
for i in range(train.shape[0]):
```

```
  if check_year(train.at[i, "Word"]) == True:
    train.at[i, "Tag"] = "B-YEAR"
```

Mã 3.26: Gán nhãn B-YEAR cho toàn bộ dữ liệu

Cách gán nhãn trên có một số lỗi, ví dụ dung tích 1980 cc thì 1980 phải có nhãn là O. Ta sẽ khắc phục lỗi của quy tắc trên.

Những B-YEAR có từ đứng trước là ['dt','*', 'x', 'dtxl', 'lanh', 'xilanh'] sẽ được gán nhãn là O.

```
B_list = ['dt','*', 'x', 'dtxl', 'lanh', 'xilanh']
for i in range(1, data.shape[0]):
  if (data.at[i-1, "Word"] in B_list) and (data.at[i, "Tag"] == "B-YEAR"):
    data.at[i, "Tag"] = "O"
```

Mã 3.27: Sửa các từ bị gán nhầm nhãn B-YEAR

Những B-YEAR có từ đứng sau là ['cc','*', 'cm3', 'mm', 'x'] sẽ được gán nhãn là O.

```
A_list = ['cc','*', 'cm3', 'mm', 'x']
for i in range(data.shape[0]-1):
  if (data.at[i+1, "Word"] in A_list) and (data.at[i, "Tag"] == "B-YEAR"):
    data.at[i, "Tag"] = "O"
```

Mã 3.28: Sửa các từ bị gán nhầm nhãn B-YEAR (tiếp)

Số 2064 bị gán nhầm thành nhãn B-YEAR nên sẽ sửa lại thành nhãn O.

```
for i in range(data.shape[0]):
  if (data.at[i, "Word"] == "2064") and (data.at[i, "Tag"] == "B-YEAR"):
    data.at[i, "Tag"] = "O"
```

Mã 3.29: Sửa các từ bị gán nhầm nhãn B-YEAR (tiếp)

3.3 Cài đặt mô hình

3.3.1 Tải dữ liệu

Chúng ta sẽ sử dụng dữ liệu sau khi đã được gán lại nhãn để xây dựng mô hình.

```
import pandas as pd
```

```
import numpy as np
from tqdm import tqdm, trange

!pip install transformers

data = pd.read_csv("/content/drive/MyDrive/Project 1/train_official_ver2.csv",
                   encoding="utf-8").fillna(method="ffill")
data.head()
```

Mã 3.30: Tải dữ liệu huấn luyện

	Word	Tag	Sentence #
0	xe	B-TYPE	Sentence: 1
1	lăn	I-TYPE	Sentence: 1
2	dùng	O	Sentence: 1
3	cho	O	Sentence: 1
4	bệnh	O	Sentence: 1

Hình 3.5 5 dòng đầu của dữ liệu train

Chúng ta sẽ tạo class `SentenceGetter` để lấy những từ trong cột của dữ liệu và tạo thành các câu hoàn chỉnh.

```
class SentenceGetter(object):

    def __init__(self, data):
        self.n_sent = 1
        self.data = data
        self.empty = False
        agg_func = lambda s: [(w, p, t) for w, p, t in zip(s["Word"].values.
            tolist(), s["POS"].values.tolist(), s["Tag"].values.tolist())]
        self.grouped = self.data.groupby("Sentence #").apply(agg_func)
        self.sentences = [s for s in self.grouped]

```

```
    def get_next(self):
        try:
            s = self.grouped["Sentence: {}".format(self.n_sent)]
            self.n_sent += 1
            return s
        except:
            return None

getter = SentenceGetter(data)
```

Mã 3.31: SentenceGetter class

Một câu ở trong tập dữ liệu sẽ như thế này.

```
sentences = [[word[0] for word in sentence] for sentence in getter.sentences]
sentences[0]
```

Mã 3.32: Minh họa sentences

```
['xe', 'lăn', 'dùng', 'cho', 'bệnh', 'nhân', ',', 'người', 'tàn', 'tật', 'h011q',
    '16', "''", 'blue', '.', 'mới', '100', '%']
```

Và đây là các nhãn của câu trên.

```
labels = [[s[1] for s in sentence] for sentence in getter.sentences]
print(labels[0])
```

Mã 3.33: Minh họa labels

```
['B-TYPE', 'I-TYPE', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-NAME', 'O', 'O',
   'O', 'O', 'B-STATUS', 'I-STATUS', 'I-STATUS']
```

Tác giả sẽ tạo từ điển `tag2idx` để chuyển đổi nhãn thành số.

```
tag_values = list(set(data["Tag"].values))
tag_values.append("PAD")
tag2idx = {t: i for i, t in enumerate(tag_values)}

#luu lai tudien
import pickle
tag_values_file = open("/content/drive/MyDrive/Project 1/tag_values.pkl", "wb")
pickle.dump(tag_values, tag_values_file)
tag_values_file.close()
```

Mã 3.34: Từ điển tag2idx

3.3.2 Cài đặt thông số và dữ liệu để huấn luyện

Trước khi có thể bắt đầu tinh chỉnh mô hình, tác giả sẽ chuẩn bị tập dữ liệu để sử dụng với pytorch và BERT. Đầu tiên, ta sẽ `import` một vài thư viện cần thiết.

```
import torch
from torch.utils.data import TensorDataset, DataLoader, RandomSampler,
    SequentialSampler
from transformers import BertTokenizer, BertConfig

from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
```

Mã 3.35: Import thư viện

Tác giả sẽ cố định một vài cấu hình. Tác giả giới hạn độ dài tối đa của sequence là 75 tokens (xem hình 2.2 để rõ hơn vì sự lựa chọn này), và sử dụng `batch size = 32` theo khuyến nghị của bài báo [7].

```
MAX_LEN = 75
bs = 32
```

Mã 3.36: Max length và batch size

Thiết lập môi trường để huấn luyện.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
n_gpu = torch.cuda.device_count()
```

Mã 3.37: Thiết lập môi trường huấn luyện

Tác giả sẽ sử dụng tokenizer của BERT đã được tiền huấn luyện.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-cased', do_lower_case=False)
```

Mã 3.38: Bert Tokenizer

Giờ chúng ta sẽ dùng tokenizer để mã hóa tất cả các câu. Vì BERT tokenizer dựa trên **Wordpiece Tokenizer** [8], nó sẽ chia nhỏ các tokens thành các subwords. Ví dụ: "benz"sẽ bị chia thành 2 tokens là "ben"và "##z". Vì vậy chúng ta phải xử lý phần nhãn để sao cho cả 2 tokens này đều mang nhãn "I-BRAND".

```
def tokenize_and_preserve_labels(sentence, text_labels):
    tokenized_sentence = []
    labels = []
```

```

    for word, label in zip(sentence, text_labels):

        # Tokenize the word and count # of subwords the word is broken into
        tokenized_word = tokenizer.tokenize(word)
        n_subwords = len(tokenized_word)

        # Add the tokenized word to the final tokenized word list
        tokenized_sentence.extend(tokenized_word)

        # Add the same label to the new list of labels `n_subwords` times
        labels.extend([label] * n_subwords)

    return tokenized_sentence, labels

tokenized_texts_and_labels = [
    tokenize_and_preserve_labels(sent, labs)
    for sent, labs in zip(sentences, labels)
]

tokenized_texts = [token_label_pair[0] for token_label_pair in
    tokenized_texts_and_labels]
labels = [token_label_pair[1] for token_label_pair in tokenized_texts_and_labels]
```

Mã 3.39: Mã hóa và đồng nhất với nhãn

Cắt và pad các chuỗi token và nhãn để tất cả các câu đều có độ dài `MAX_LEN`.

```
input_ids = pad_sequences([tokenizer.convert_tokens_to_ids(txt) for txt in
    tokenized_texts],
                          maxlen=MAX_LEN, dtype="long", value=0.0,
                          truncating="post", padding="post")

tags = pad_sequences([[tag2idx.get(l) for l in lab] for lab in labels],
                     maxlen=MAX_LEN, value=tag2idx["PAD"], padding="post",
                     dtype="long", truncating="post")
```

Mã 3.40: Cut and Pad

Tác giả sẽ tạo mảng `attention_masks` đánh dâu những từ `pad`, để hàm loss function của BERT bỏ qua mất mát của những từ này.

```
attention_masks = [[float(i != 0.0) for i in ii] for ii in input_ids]
```

Mã 3.41: Attention masks

10% của tập dữ liệu được sử dụng để thẩm định.

```
tr_inputs, val_inputs, tr_tags, val_tags = train_test_split(input_ids, tags,
                                                            random_state=2018,
                                                                test_size=0.1)
tr_masks, val_masks, _, _ = train_test_split(attention_masks, input_ids,
                                             random_state=2018, test_size=0.1)
```

Mã 3.42: Chia dữ liệu

Vì chương trình tính toán trên pytorch, nên cần phải chuyển đổi dữ liệu sang torch tensors.

```
tr_inputs = torch.tensor(tr_inputs)
val_inputs = torch.tensor(val_inputs)
tr_tags = torch.tensor(tr_tags)
val_tags = torch.tensor(val_tags)
tr_masks = torch.tensor(tr_masks)
val_masks = torch.tensor(val_masks)
```

Mã 3.43: Torch tensors

Cuối cùng là định nghĩa dataloaders. Tác giả sẽ xáo trộn dữ liệu trong thời điểm huấn luyện với `RandomSampler` và tại thời điểm test tác giả sẽ đưa dữ liệu vào tuần tự với `SequentialSampler`.

```
train_data = TensorDataset(tr_inputs, tr_masks, tr_tags)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=bs)

valid_data = TensorDataset(val_inputs, val_masks, val_tags)
valid_sampler = SequentialSampler(valid_data)
valid_dataloader = DataLoader(valid_data, sampler=valid_sampler, batch_size=bs)
```

Mã 3.44: Torch tensors

3.3.3 Cài đặt mô hình BERT để fine-tuning

Transformer package cung cấp `BertForTokenClassification` class để dự đoán ở mức độ token-level. `BertForTokenClassification` là mô hình fine-tuning được gói trong `BERT model` và top linear layer (xem hình 1.7) là token-level classifier với input là sequence của hidden state cuối cùng. Tác giả sẽ load `bert-base-cased` pre-trained model và cung cấp `num_labels = len(tag2idx)`.

```
import transformers
from transformers import BertForTokenClassification, AdamW
```

```
3
4 model = BertForTokenClassification.from_pretrained(
5     "bert-base-cased",
6     num_labels=len(tag2idx),
7     output_attentions = False,
8     output_hidden_states = False
9 )
```

Mã 3.45: Load BERT model

Truyền các tham số của mô hình tới GPU.

```
1 model.cuda();
```

Mã 3.46: Truyền các tham số của mô hình tới GPU

Tác giả sẽ cài đặt optimizer và các tham số mà nó cần cập nhật, ở đây tác giả lựa chọn `AdamW` optimizer. Ngoài ra, tác giả cũng sẽ thêm một số `weigth_decay` cho các ma trận trọng số.

```
1 FULL_FINETUNING = True
2 if FULL_FINETUNING:
3     param_optimizer = list(model.named_parameters())
4     no_decay = ['bias', 'gamma', 'beta']
5     optimizer_grouped_parameters = [
6         {'params': [p for n, p in param_optimizer if not any(nd in n for nd in
              no_decay)],
7          'weight_decay_rate': 0.01},
8         {'params': [p for n, p in param_optimizer if any(nd in n for nd in
              no_decay)],
9          'weight_decay_rate': 0.0}
10    ]
11 else:
12    param_optimizer = list(model.classifier.named_parameters())
13    optimizer_grouped_parameters = [{"params": [p for n, p in param_optimizer]}]
14
15 optimizer = AdamW(
16    optimizer_grouped_parameters,
17    lr=3e-5,
18    eps=1e-8
19 )
```

Mã 3.47: Setup optimizer

Tạo scheduler cho optimizer.

```
1 from transformers import get_linear_schedule_with_warmup
```

```
2
3 epochs = 5
4 max_grad_norm = 1.0
5
6 # Total number of training steps is number of batches * number of epochs.
7 total_steps = len(train_dataloader) * epochs
8
9 # Create the learning rate scheduler.
10 scheduler = get_linear_schedule_with_warmup(
11     optimizer,
12     num_warmup_steps=0,
13     num_training_steps=total_steps
14 )
```

Mã 3.48: Scheduler

3.3.4 Tinh chỉnh mô hình với bài toán yêu cầu

Mô hình hội tụ sau 5 epochs.

```
1 from seqeval.metrics import f1_score, accuracy_score
2
3 ## Store the average loss after each epoch so we can plot them.
4 loss_values, validation_loss_values = [], []
5
6 for _ in trange(epochs, desc="Epoch"):
7     # ========================================
8     #               Training
9     # ========================================
10    # Perform one full pass over the training set.
11
12    # Put the model into training mode.
13    model.train()
14    # Reset the total loss for this epoch.
15    total_loss = 0
16
17    # Training loop
18    for step, batch in enumerate(train_dataloader):
19        # add batch to gpu
20        batch = tuple(t.to(device) for t in batch)
21        b_input_ids, b_input_mask, b_labels = batch
22        # Always clear any previously calculated gradients before performing a
              backward pass.
```

```
        model.zero_grad()
        # forward pass
        # This will return the loss (rather than the model output)
        # because we have provided the `labels`.
        outputs = model(b_input_ids, token_type_ids=None,
                        attention_mask=b_input_mask, labels=b_labels)
        # get the loss
        loss = outputs[0]
        # Perform a backward pass to calculate the gradients.
        loss.backward()
        # track train loss
        total_loss += loss.item()
        # Clip the norm of the gradient
        # This is to help prevent the "exploding gradients" problem.
        torch.nn.utils.clip_grad_norm_(parameters=model.parameters(), max_norm=
            max_grad_norm)
        # update parameters
        optimizer.step()
        # Update the learning rate.
        scheduler.step()

    # Calculate the average loss over the training data.
    avg_train_loss = total_loss / len(train_dataloader)
    print("Average train loss: {}".format(avg_train_loss))

    # Store the loss value for plotting the learning curve.
    loss_values.append(avg_train_loss)


    # ========================================
    #               Validation
    # ========================================
    # After the completion of each training epoch, measure our performance on
    # our validation set.

    # Put the model into evaluation mode
    model.eval()
    # Reset the validation loss for this epoch.
    eval_loss, eval_accuracy = 0, 0
    nb_eval_steps, nb_eval_examples = 0, 0
    predictions , true_labels = [], []
    for batch in valid_dataloader:
        batch = tuple(t.to(device) for t in batch)
```

```
        b_input_ids, b_input_mask, b_labels = batch

        # Telling the model not to compute or store gradients,
        # saving memory and speeding up validation
        with torch.no_grad():
            # Forward pass, calculate logit predictions.
            # This will return the logits rather than the loss because we have
                not provided labels.
            outputs = model(b_input_ids, token_type_ids=None,
                            attention_mask=b_input_mask, labels=b_labels)
        # Move logits and labels to CPU
        logits = outputs[1].detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()

        # Calculate the accuracy for this batch of test sentences.
        eval_loss += outputs[0].mean().item()
        predictions.extend([list(p) for p in np.argmax(logits, axis=2)])
        true_labels.extend(label_ids)

    eval_loss = eval_loss / len(valid_dataloader)
    validation_loss_values.append(eval_loss)
    print("Validation loss: {}".format(eval_loss))
    pred_tags = [[tag_values[p_i] for p, l in zip(predictions, true_labels)
                                  for p_i, l_i in zip(p, l) if tag_values[l_i] !=
                                      "PAD"]]
    valid_tags = [[tag_values[l_i] for l in true_labels
                                   for l_i in l if tag_values[l_i] != "PAD"]]
    print("Validation Accuracy: {}".format(accuracy_score(pred_tags, valid_tags))
        )
    print("Validation F1-Score: {}".format(f1_score(pred_tags, valid_tags)))
    print()
```

Mã 3.49: Fine-tune mô hình

3.4 Thẩm định mô hình

3.4.1 Kiểm tra mô hình với câu xác định

```
test_sentence = "xe ô tô mercedes benz sản xuất năm 1956 sử dụng xăng , đã qua sử
    dụng"
tokenized_sentence = tokenizer.encode(test_sentence)
```

```
input_ids = torch.tensor([tokenized_sentence]).cuda()

with torch.no_grad():
    output = model(input_ids)
label_indices = np.argmax(output[0].to('cpu').numpy(), axis=2)

# join bpe split tokens
tokens = tokenizer.convert_ids_to_tokens(input_ids.to('cpu').numpy()[0])
new_tokens, new_labels = [], []
for token, label_idx in zip(tokens, label_indices[0]):
    if token.startswith("##"):
        new_tokens[-1] = new_tokens[-1] + token[2:]
    else:
        new_labels.append(tag_values[label_idx])
        new_tokens.append(token)

for token, label in zip(new_tokens, new_labels):
    print("{}\t{}".format(label, token))
```

Mã 3.50: Kiểm tra mô hình với một câu xác định

```
B-TYPE  [CLS]
B-TYPE  xe
I-TYPE  ô
I-TYPE  tô
B-BRAND mercedes
I-BRAND benz
O       sản
O       xuất
O       năm
B-YEAR  1956
O       sử
O       dụng
B-ENGINE_FULE_TYPE      xăng
O       ,
B-STATUS        đã
I-STATUS        qua
I-STATUS        sử
I-STATUS        dụng
O       [SEP]
```

Hình 3.6 Kết quả kiểm tra mô hình với câu xác định.

Ta có thể thấy mô hình hoạt động khá chính xác. Tiếp theo, ta sẽ sử dụng file test để đánh giá độ chính xác của mô hình.

3.4.2 Kiểm tra mô hình với tập test

```
data_test = pd.read_csv("/content/drive/MyDrive/Project 1/test_official_ver2.csv",
                        encoding="utf-8")
data_test.head(20)
data_test["Word"] = data_test.Word.apply(str)

def predict(test_sentence):
    tokenized_sentence = tokenizer.encode(test_sentence)
    input_ids = torch.tensor([tokenized_sentence]).cuda()

    with torch.no_grad():
        output = model(input_ids)
    label_indices = np.argmax(output[0].to('cpu').numpy(), axis=2)

    # join bpe split tokens
    tokens = tokenizer.convert_ids_to_tokens(input_ids.to('cpu').numpy()[0])
    new_tokens, new_labels = [], []
    for token, label_idx in zip(tokens, label_indices[0]):
        if token.startswith("##"):
            new_tokens[-1] = new_tokens[-1] + token[2:]
        else:
            new_labels.append(tag_values[label_idx])
            new_tokens.append(token)
    return new_labels[1:-1]

getter_test = SentenceGetter(data_test)
sentences_test = [[word[0] for word in sentence] for sentence in getter_test.sentences]
sentences_test[0]
labels_test = [[s[1] for s in sentence] for sentence in getter_test.sentences]

A, B = [], []
for sent, labs in zip(sentences_test, labels_test):
    c, d = tokenize_and_preserve_labels(sent, labs)
    A.append(c)
    B.append(d)

M, N = [], []
for i in range(len(A)):
    new_tokens, new_labels = [], []
    for token, label_idx in zip(A[i], B[i]):
        if token.startswith("##"):
            new_tokens[-1] = new_tokens[-1] + token[2:]
```

```
        else:
            new_labels.append(label_idx)
            new_tokens.append(token)
    M.append(new_tokens)
    N.append(new_labels)

X = []
for i in sentences_test:
    X.append(' '.join(i))

#ket qua du doan se duoc luu vao y_pred
y_pred = []
for i in X:
  y_pred.append(predict(i))

from sklearn.metrics import classification_report
from sklearn.preprocessing import MultiLabelBinarizer
mlb = MultiLabelBinarizer()
tag_values.remove("PAD")
mlb.fit([tag_values])
actual = mlb.transform(N)
pred = mlb.transform(y_pred)
print(classification_report(actual,pred, target_names=mlb.classes_))
```

Mã 3.51: Kiểm tra mô hình với tập test

	precision	recall	f1-score	support
B-BRAND	1.00	0.98	0.99	1707
B-ENGINE_FULE_TYPE	0.98	0.99	0.98	864
B-NAME	0.97	0.98	0.97	1188
B-STATUS	1.00	0.99	0.99	1899
B-TYPE	1.00	0.95	0.97	1611
B-YEAR	0.96	1.00	0.98	1174
I-BRAND	0.99	0.95	0.97	153
I-NAME	0.99	0.99	0.99	470
I-STATUS	0.99	0.99	0.99	1816
I-TYPE	1.00	0.96	0.98	2137
O	1.00	1.00	1.00	2341
micro avg	0.99	0.98	0.99	15360
macro avg	0.99	0.98	0.98	15360
weighted avg	0.99	0.98	0.99	15360
samples avg	0.99	0.98	0.98	15360

Hình 3.7 Kết quả kiểm tra mô hình với tập test.

Như vậy kết quả mô hình thu được là rất tốt.

3.5 So sánh mô hình BERT-NER và PhoBERT

3.5.1 Giới thiệu về mô hình PhoBERT

PhoBERT [10] là một pre-trained được huấn luyện monolingual language, tức là chỉ huấn luyện dành riêng cho tiếng Việt. Việc huấn luyện dựa trên kiến trúc và cách tiếp cận giống RoBERTa của Facebook được Facebook giới thiệu giữa năm 2019. Đây là một cái tiến so với BERT trước đây.
PhoBERT được train trên khoảng 20GB dữ liệu bao gồm khoảng 1GB Vietnamese Wikipedia corpus và 19GB còn lại lấy từ Vietnamese news corpus. Đây là một lượng dữ liệu khả ổn để train một mô hình như BERT.

3.5.2 Kết quả thu được với mô hình PhoBERT

Hình 3.8 cho kết quả rất tốt, có vẻ nhỉnh hơn so với mô hình BERT-NER hơn một chút. Đây là kết quả không bất ngờ vì PhoBERT là mô hình đặc thù được đặc biệt tiền huấn luyện với dữ liệu tiếng Việt. Tuy nhiên mô hình PhoBERT cần đến 10 epochs để hội tụ, gấp đôi so với mô hình BERT-NER (5 epochs).

B-BRAND	1.00	1.00	1.00	1707
B-ENGINE_FULE_TYPE	1.00	1.00	1.00	864
B-NAME	0.98	0.99	0.99	1188
B-STATUS	0.99	1.00	1.00	1899
B-TYPE	0.96	0.97	0.96	1611
B-YEAR	1.00	1.00	1.00	1174
I-BRAND	0.99	1.00	0.99	153
I-NAME	0.99	1.00	0.99	470
I-STATUS	0.99	1.00	1.00	1816
I-TYPE	1.00	1.00	1.00	2137
O	1.00	1.00	1.00	2341
PAD	0.00	0.00	0.00	0
micro avg	0.99	1.00	0.99	15360
macro avg	0.91	0.91	0.91	15360
weighted avg	0.99	1.00	0.99	15360
samples avg	0.99	0.99	0.99	15360

Hình 3.8 Kết quả thu được với PhoBERT.

Tài liệu tham khảo

[1] N. Thuong, "Tổng hợp transfer learning," *Machine Learning cơ bản*, 2019.

[2] N. T. Binh, *Trích rút thông tin văn bản sử dụng Deep Transfer Learning.* 2021.

[3] N. Andrew, "Nuts and bolts of building ai applications using deep learning," 2016.

[4] N. B. Ngoc, "Mô hình ngôn ngữ và mạng nơ ron hồi quy (language model and recurrent neural network)," *ProtonX*, 2020.

[5] P. D. Khanh, "Attention is all you need," *Pham Dinh Khanh Blog*, 2019.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[7] D. Jacob, C. Ming-Wei, L. Kenton, and T. Kristina, "Pre-training of deep bidirectional transformers for language understanding," 2018.

[8] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016.

[9] P. D. Khanh, "Bert model," *Pham Dinh Khanh Blog*, 2020.

[10] N. Q. Dat and N. T. Anh, "Phobert: Pre-trained language models for vietnamese," 2020.