# **Sentence Vectors: Các phương pháp mô hình hóa câu văn lên không gian vector**

[Editors' Choice](https://viblo.asia/editors-choice)

Bài đăng này đã không được cập nhật trong 3 năm

Trong một bài viết trước đây [về biểu diễn không gian vector cho tiếng Việt](https://viblo.asia/p/xay-dung-mo-hinh-khong-gian-vector-cho-tieng-viet-GrLZDXr2Zk0), mình đã giới thiệu về một số phương pháp biểu diễn vector cho dữ liệu dạng văn bản với đơn vị biểu diễn là từ. Tuy nhiên, trong quá trình thực hiện các nhiệm vụ thực tế liên quan tới [xử lý ngôn ngữ tự nhiên](https://viblo.asia/p/lecture-1-introduction-to-nlp-and-deep-learning-bJzKmXzk59N), dạng biễu diễn cho dữ liệu dạng text mà ta thường xuyên phải làm việc cùng lại không chỉ là từ mà còn là các câu văn, đoạn văn. Vậy câu hỏi đặt ra là:

* Để có thể làm đầu vào cho các mô hình học máy, phương pháp nào giúp ta mô hình hóa các câu văn, đoạn văn thô đó thành biểu diễn vector?
* Có thể sử dụng các phương pháp biểu diễn vector cho từ để biểu diễn vector cho câu hay không?

Trong bài viết này, chúng ta sẽ tìm hiểu điều đó, cùng mình điểm lại một số phương pháp phổ biến để mô hình hóa câu văn lên không gian vector hay còn gọi là [Sentence2vec](https://en.wikipedia.org/wiki/Sentence_embedding). Chúng ta bắt đầu từ phương pháp đơn giản nhất: *Sử dụng word2vec để biểu diễn vector cho câu*.

# **Sử dụng word2vec để biểu diễn vector cho câu**

Như một lẽ tự nhiên, các bạn sẽ thấy ngay một điều là các câu văn về cơ bản là tập hợp của các từ, khi ta đã có biểu diễn vector cho các từ cấu thành lên câu rồi thì liệu có cần thiết phải tìm một phương pháp khác để tìm biểu diễn vector cho câu không.

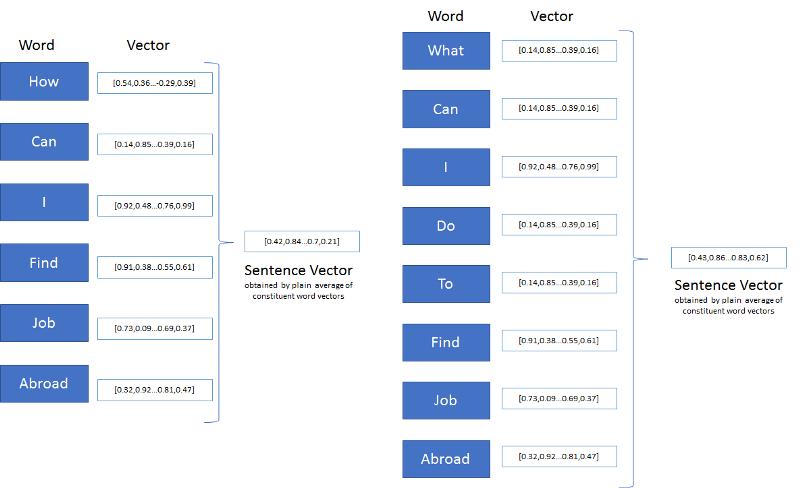


Đúng vậy, suy nghĩ của bạn hoàn toàn chính xác và *hợp tự nhiên*. Chúng ta hoàn toàn có thể lấy tổng hoặc lấy tổng trung bình vector các từ cấu thành lên câu để làm vector biểu diễn cho câu. Một ví dụ đơn giản cho bạn dễ thấy sự *hợp tự nhiên* mà mình nói tới ở đây.



Mình có 5 loại cocktail khác nhau, mỗi loại được được phân tích thành phần theo 300 loại phân tử phổ biến ngày này. Và khi mình trộn 5 loại này với nhau mình sẽ được một loại cocktail mới, ly cocktail thứ 6 chứa đủ các thành phần của 5 loại cockltail kia cũng như có đủ vị của 5 loại (Điều này chỉ đúng khi các thành phần này không phản ứng hóa học với nhau).

Tương tự, chúng ta biết Word2vec là phương pháp biểu diễn vector cho từ, mỗi từ cũng được chúng ta biểu diễn thành vector với trọng số của 100, 150 hay 300 chiều tương tự như các phân tử hóa học kia. Việc tìm biểu diễn vector cho câu tương tự như khi ta trộn cocktail, ta kết hợp các thành phần vector của từ để làm biểu diễn cho câu.



Vậy nếu các thành phần đó phản ứng với nhau như việc các phân tử có thể phản ứng hóa học với nhau thì sao? Chính xác điều đó sẽ xảy ra vì các từ cấu thành nên câu có thể xây dựng lên các câu với nghĩa khác nhau tùy thuộc vào cách sắp xếp của chúng. Vì vậy, phương pháp này tồn tại những điểm hạn chế nghiêm trọng.

* Nó bỏ qua thứ tự của các từ trong câu
* Nó hoàn toàn bỏ qua ngữ nghĩa, ngữ cảnh của câu

Ví dụ như 2 câu sau có cùng thành phần nhưng mang hai ý nghĩa hoàn toàn trái ngược:

1. Vì quá yêu bạn gái, chàng trai quyết định từ bỏ game.
2. Vì quá yêu game, chàng trai quyết định từ bỏ bạn gái.

Để ví dụ minh họa cho phương pháp này, mình sẽ giới thiệu một thư viện vô cùng quen thuộc trong xử lý ngôn ngữ tự nhiên đó là [Spacy](https://spacy.io/). Sử dụng Spacy, bạn có thể dễ dàng lấy ra vector đại diện của một từ có sẵn đã được huấn luyện trước đó. Spacy cũng cho phép lấy luôn ra vector đại diện cho câu theo phương pháp trung bình này.

Để cài Spacy, thực hiện lệnh:

pip install spacy

python -m spacy download en\_core\_web\_md

import spacy

nlp = spacy.load('en\_core\_web\_md')

text = nlp("I am Iron man")

v1\_vector = 0

v2\_vector = text.vector # Lấy vector của cả câu

for t in text:

v1\_vector += t.vector/len(text) # Lấy vector của cả từ chia trung bình

print(v1\_vector == v2\_vector)

Kết quả là True, Spacy chính là đang sử dụng phương pháp này để lấy vector đại diện cho câu.

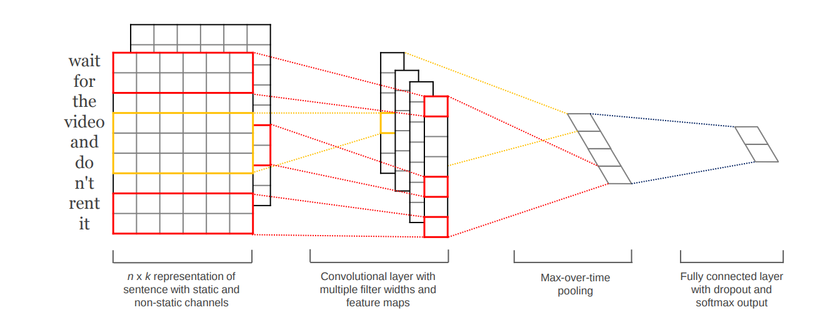
Các phương pháp tiếp cận dựa trên vector của từ khác như lấy trung bình có trọng số các vector từ hay kết hợp các từ theo thứ tự của cây phân tích ngữ pháp của câu đều rơi vào điểm hạn chế tương tự và không hiểu quả với các trường hợp câu khó phân tích như các câu thiếu, câu không hoàn chỉnh dẫn đến việc phân tích cây ngữ pháp bị sai.

Để khắc phục những điểm hạn chế kể trên, các phương pháp Sentence2vec mới đã được ra đời, sử dụng các tác vụ trung gian để tìm vector biểu diễn cho câu như classification, language modeling, autoencoder, encoder-decoder,... Trong phần tiếp theo, chúng ta cùng tiếp tục điểm qua một vài phương pháp nổi bật trong số đó.

# **Sử dụng pre-trained từ các bài toán Classification**

Trong thực tế, khi sử dụng các mô hình phân loại văn bản, các vector ở layer cận cuối cùng, trước khi đưa qua layer cuối cùng sử dụng hàm kích hoạt softmax để phân loại đã có đủ tính tổng quát để đại diện cho câu, đoạn văn bản đó. Do vậy, ở phương pháp này, ta có thể sử dụng lại model đã được huấn luyện từ các tác vụ phân loại văn bản trước đó ví dụ như Sentiment analysis để sinh ra vector đại diện cho một câu văn mới.

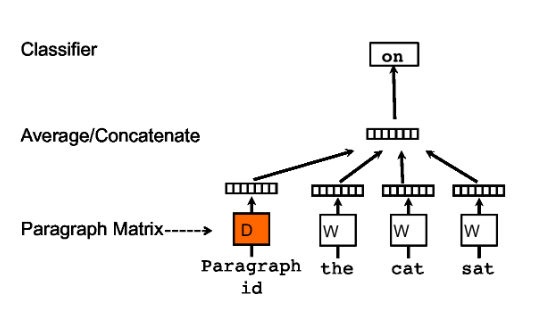
Như trong bài báo [Convolutional Neural Networks for Sentence Classification](https://arxiv.org/pdf/1408.5882.pdf), chúng ta ánh xạ các câu vào trong mạng bằng cách sử dụng một pre-trained word embbeding. Sau đó sử dụng một kiến trúc CNN để học ra các đặc trưng được sử dụng cho quá trình phân loại trước khi cho qua một fully connected layer. Vector cuối cùng thu được bởi kiến trúc chính là vector đại diện cho câu.



Nhược điểm của phương pháp này là các biểu diễn thông qua các nhiệm vụ trung gian như vậy thường thu được các vector không tổng quát mà gắn chặt với nhiệm vụ trước đó. Như vậy, các vector này có thể không sử dụng được cho các nhiệm vụ khác mà chỉ được sử dụng cho các nhiệm vụ tương tự hoặc ta phải là tìm vector biển diễn cho câu mà các vector thu được đủ tổng quát cho các nhiệm vụ khác.

# **Paragraph Vectors: Doc2vec**

Trong bài báo [Distributed Representations of Sentences and Documents](http://proceedings.mlr.press/v32/le14.pdf) của Quốc Lê và Tomas Mikolov năm 2015, các tác giả đã giới thiệu một phương pháp mới có khả năng tìm vector biểu diễn tốt hơn cho các câu văn/đoạn văn thông qua một mô hình tương tự như mô hình word2vec được giới thiệu trước đó cũng bởi Mikolov. Phương pháp được mô tả đơn giản như hình dưới đây.



Nhìn vào hình, bạn có thể thấy mô hình này chỉ khác mô hình word2vec 1 điểm duy nhất là ngoài input là các từ ngữ cảnh, chúng ta còn có thêm 1 giá trị mới là ID của các câu văn. Về cơ bản, mỗi câu văn được ánh xạ tới một vector duy nhất và mô hình sử dụng các vector của câu văn này kết hợp với các vector từ cấu thành lên câu đó để dự đoán từ tiếp theo cho ngữ cảnh. Thông qua quá trình đào tạo như vậy, các vector của câu sẽ có thể lưu trữ thông tin về ngữ cảnh, ý nghĩa của câu, những thông tin mà bản thân ý nghĩa của từng từ trong câu không thể thể hiện được. Nó hoạt động như một bộ nhớ lưu trữ của câu. Cũng chính vì vậy, phương pháp này được gọi là Distributed Memory model (PV-DM).

Để tìm được vector cho một câu mới, chúng ta sử dụng mô hình trên để dự đoán bằng cách khởi tạo một vector ngẫu nhiên cho câu văn đó. Trọng số của tất cả các phần còn lại trong mạng được giữ nguyên, sau một vài epochs cho quá trình inference, ta thu được vector trọng số đại diện cho câu đầu vào mới.

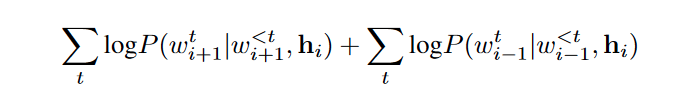
Ngoài ra, tương tự như word2vec, chúng ta cũng có một mô hình khác, dự đoán các từ ngẫu nhiên trong ngữ cảnh và được gọi là Distributed BOW (PV-DBOW).

Việc sử dụng và huấn luyện Doc2vec giờ đây đã khá đơn giản, việc duy nhất bạn phải làm đó là tìm hiểu về thư viên [Gensim](https://radimrehurek.com/gensim/) và học cách sử dụng nó. Việc sử dụng Gensim để huẩn luyện Doc2vec khá đơn giản, các bạn xem qua tại [Doc2vec paragraph embeddings](https://radimrehurek.com/gensim/models/doc2vec.html), mình sẽ không đi vào chi tiết.

# **Skip-thoughts**

Trong khi Doc2vec là một mô hình intra-sentence, tức là ta tìm ra vector đại diện của một câu chỉ dựa vào bản thân câu đấy mà không dựa vào các câu xung quanh thì Skip-thoughts lại là một mô hình inter-sentence. Phương pháp sử dụng tính liên tục của văn bản để dự đoán câu tiếp theo dựa vào câu đã cho. Tư tưởng của điều này khá giống với thuật toán Skip-gram trong mô hình word2vec, sử dụng thông tin của câu hiện tại để dự đoán ngữ cảnh xung quanh nó, chỉ khác là word2vec sử dụng trừu tượng hóa ở mức từ thì ở skip-thoughts chúng ta trừu tượng hóa ở mức câu.

Phương pháp được giới thiệu trong bài báo [Skip-Thought Vectors](https://arxiv.org/pdf/1506.06726.pdf) của đại học Toronto đầu năm 2015. Trong bài báo này, các tác giả đề xuất một mô hình encoder-decoder cho quá trình huấn luyện, sử dụng kiến trúc RNN cho cả phần encoding và decoding. Ngoài sinh ra một vector đại diện cho câu, phương pháp này cũng sinh ra các vector cho các từ trong kho từ vựng. Hàm mục tiêu được tối ưu hóa như sau:



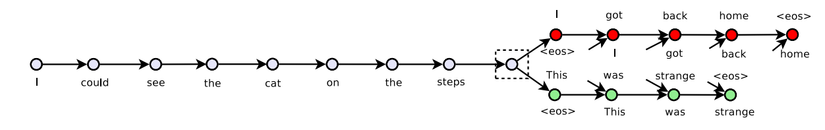
Trong đó, chỉ số

i+1

*i*+1 và

i-1

*i*−1 tương ứng thể hiện là các câu tiếp theo và các câu trước đó của câu đầu vào. Nhìn chung, hàm này đại diện cho tổng logarithm xác suất dự đoán đúng câu tiếp theo và câu trước đó của câu hiện tại.



Như trong ví dụ này, input là bộ 3 câu:

I got back home. I could see the cat on the steps. This was strange.

* Câu trước: I got back home
* Câu hiện tại: I could see the cat on the steps
* Câu sau: This was strange.

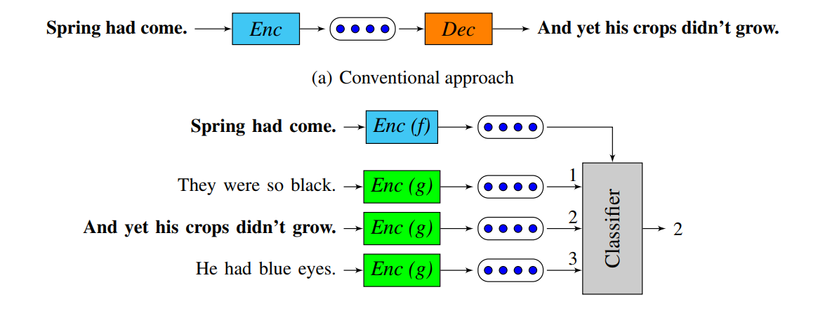
Encoder: Đầu vào là chuỗi các vector từ của các từ có trong câu hiện tại, sau đó được chuyển tiếp vào GRU hoặc LSTM. Decoder: Kiến trúc tương tự như kiến trúc encoder nhưng thay vì chỉ có một, chúng ta có tới 2 decoder cho câu trước và câu sau.

Chúng ta có thể tham khảo 1 số source code đã implement skip-thought sau:

* [Skip-Thoughts Tensorflow github](https://github.com/tensorflow/models/tree/master/research/skip_thoughts)
* [Skip-Thoughts Pytorch github](https://github.com/Cadene/skip-thoughts.torch/tree/master/pytorch)

# **Quick-thoughts**

Quick thought vector là phương pháp được phát triển dựa trên Skip-thoughts vectors, được giới thiệu trong bài báo [AN EFFICIENT FRAMEWORK FOR LEARNING SENTENCEREPRESENTATIONS](https://openreview.net/pdf?id=rJvJXZb0W) đầu năm 2018. Tuy nhiên, thay vì sử dụng một lớp decoder để dự đoán các câu tiếp theo của câu hiện tại, Quick thought sử dụng một nhiệm vụ phân loại: Trong những câu được đưa ra, câu nào là câu kế tiếp của câu hiện tại.



Sự thay đổi này tạo nên cho nó một lợi thế cạnh tranh mạnh mẽ so với Skip-thought đó là thời gian huấn luyện mô hình. Tốc độ huấn luyện giảm đáng kể khiến nó sử dụng tốt khi ra cần sử dụng với những bộ dữ liệu lớn và cực lớn.

# **FastSent và Denoising autoencoders**

FastSent được giới thiệu trong bài báo [Learning Distributed Representations of Sentences from Unlabelled Data](https://arxiv.org/pdf/1602.03483.pdf) vào đầu năm 2016 bởi nhóm tác giả Felix Hill, Kyunghyun Cho và Anna Korhonen.

Mô hình này được đề xuất là một kỹ thuật inter-sentence, về mặt tư tưởng khá giống với mô hình Skip-thoughts trước đó. Điểm khác biệt duy nhất là nó sử dụng biểu diễn BOW(bag of word) của câu để dự đoán các câu xung quanh, điều này làm cho nó tính toán hiệu quả hơn so với Skip-thoughts, dẫn đến quá trình training được diễn ra nhanh hơn nhiều. Giả thuyết đào tạo vẫn giữ nguyên, tức là ngữ nghĩa của câu có thể được suy ra từ nội dung của các câu liền kề.

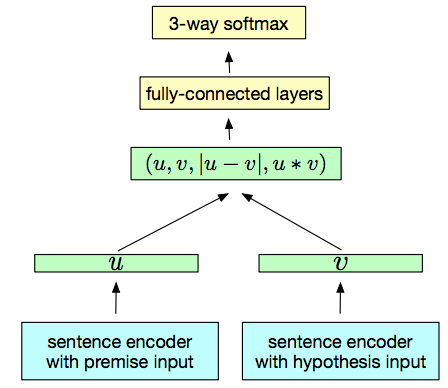
Cũng trong bài báo này, một phiên bản khác của mô hình FastSent cũng được đề xuất được gọi là Denoising autoencoders. Về cơ bản, nó là một phương pháp intra-sentence trong đó mục tiêu là tái tạo lại một câu từ một phiên bản lỗi/nhiễu của chính nó. Trong phiên bản sửa đổi của một câu gốc, có 2 cách để tạo nhiễu để sinh ra nó:

* Đối với mỗi từ w trong câu S, xóa từ đó đi theo xác suất xóa là
* p\_0
* *p*
* 0
* ​
* Đối với mỗi bigram không chồng chéo nhau, đổi chỗ các phần tử trong bigram với một xác suất là
* p\_x
* *p*
* *x*
* ​
* .

Vì các chi tiết của mô hình khá giống với Skip-thought nên chúng ta sẽ không nói quá nhiều về nó nữa. Chi tiết bạn có thể xem tại bào báo gốc được giới thiệu phần bên trên.

# **InferSent**

Không giống như các phương pháp trên sử dụng hướng tiếp cận học máy không giám sát, InferSent sử dụng Stanford Natural Language Inference (SNLI) với 570k cặp câu với 3 nhãn: neutral, contradiction và entailment để huấn luyện một bộ phân loại. Cả hai câu đều được vector hóa bởi một kiến trúc encoder duy nhất. Sau đó, các vector đại diện cho 2 câu được kết hợp theo 3 cách(Concatenation, Element-wise product, Absolute element-wise) và đưa qua một fully connected layers để thực hiện phân loại 3 lớp. Kiến trúc được mô tả chi tiết trong bài báo [Supervised Learning of Universal Sentence Representations from Natural Language Inference Data](https://arxiv.org/pdf/1705.02364.pdf) của Facebook AI Research và được mô tả bằng hình vẽ dưới đây.



InferSent được Facebook viết bằng Pytorch, bạn có thể tham khảo và sử dụng tại [Facebookresearch/InferSent](https://github.com/facebookresearch/InferSent).

Trên đây là những phương pháp mình cảm thấy phổ biến và thường hay được sử dụng nhất cho tới thời điểm hiện tại. Ngoài ra, còn có rất nhiều biến thể và đề xuất khác được đưa ra, tuy nhiên, mình sẽ dựa vào mức độ phổ biến và hiệu quả của nó để quyết định bổ sung vào bài sau.

Cảm ơn các bạn vì đã đọc bài. ) )