Bài 3 - Mô hình Word2Vec

29 Apr 2019 - phamdinhkhanh

Menu

1. Giới thiệu Word Representation.

Khác với các mô hình xử lý ảnh khi các giá trị đầu vào là cường độ màu sắc đã được mã hoá thành giá trị số trong khoảng [0, 255]. Mô hình xử lý ngôn ngữ tự nhiên có đầu vào chỉ là các chữ cái kết hợp với dấu câu. Làm sao chúng ta có thể lượng hoá được những từ ngữ để làm đầu vào cho mạng nơ ron? Kĩ thuật one-hot véc tơ sẽ được áp dụng để thực hiện điều này. Trước khi đi vào phương pháp biểu diễn, chúng ta cần làm rõ một số khái niệm:

- Documents (Văn bản): Là tợp hợp các câu trong cùng một đoạn văn có mối liên hệ với nhau. Văn bản có thể được coi như một bài báo, bài văn,....
- Corpus (Bộ văn bản): Là một tợp hợp gồm nhiều văn bản thuộc các đề tài khác nhau, tạo thành một nguồn tài nguyên dạng văn bản. Một văn bản cũng có thể được coi là corpus của các câu trong văn bản. Các bộ văn bản lớn thường có từ vài nghìn đến vài trăm nghìn văn bản trong nó. Một số bộ văn bản trong tiếng việt có thể được download từ nguồn Wikipedia (https://wiki.dbpedia.org/datasets), VNCoreNLP (https://github.com/vncorenlp/VnCoreNLP).
- Character (kí tự): Là tợp hợp gồm các chữ cái (nguyên âm và phụ âm) và dấu câu. Mỗi một ngôn ngữ sẽ có một bộ các kí tự khác nhau.
- Word (từ vựng): Là các kết hợp của các kí tự tạo thành những từ biểu thị một nội dung, định nghĩa xác định, chẳng hạn `con người` có thể coi là một từ vựng. Từ vựng có thể bao gồm từ đơn có 1 âm tiết và từ ghép nhiều hơn 1 âm tiết. Khác với tiếng anh khi các từ chủ yếu là đơn âm. Tiếng việt có rất nhiều những từ ghép 2, 3 âm tiết. Do đó chúng ta cần phải có từ điển để thực hiện tách từ (tokenize) trong câu. Một số package thông dụng trong Tiếng Việt có sẵn chức năng này được sử dụng phổ biến là underthesea (https://github.com/undertheseanlp/underthesea), pyvi (https://pypi.org/project/pyvi/), VNCoreNLP (https://github.com/vncorenlp/VnCoreNLP), RDRsegmenter (https://github.com/datquocnguyen/RDRsegmenter), coccoc-tokenizer (https://github.com/coccoc/coccoc-tokenizer). Kết quả tokenize có thể khác nhau tuỳ thuộc vào cách định nghĩa từ ghép ở mỗi package. Khi xử lý ngôn ngữ tự nhiên cho một số lĩnh vực đặc biệt cần phải có từ điển chuyên ngành, vì vậy cần phải customize riêng mà không nên sử dụng từ điển từ package.
- Dictionary (từ điển): Là tợp hợp các từ vựng xuất hiện trong văn bản.
- Volcabulary (từ vựng): Tợp hợp các từ được trích xuất trong văn bản. Tương tự như từ điển.

Trước khi biểu diễn từ chúng ta cần xác định từ điển của văn bản. Số lượng từ là hữu hạn và được lặp lại trong các câu. Do đó thông qua từ điển gồm tợp hợp tất cả các từ có thể xuất hiện, ta có thể mã hoá được các câu dưới dạng ma trận mà mỗi dòng của nó là một véc tớ one-hot của từ.

Định nghĩa One-hot véc tơ của từ: Giả sử chúng ta có từ điển là tợp hợp gồm n từ vựng `{anh, em, gia đình, bạn bè,...}` . Khi đó mỗi từ sẽ được đại diện bởi một giá trị chính là index của nó. Từ `anh` có index = 0, `gia đình` có index = 2. One-hot véc tơ của từ vựng thứ i, $i \leq (n-1)$ sẽ là véc tơ $\mathbf{e_i} = [0, \ldots, 0, 1, 0, \ldots, 0] \in \mathbb{R}^n$ sao cho các phần tử e_{ij} của véc tơ thoả mãn:

$$\begin{cases} e_{ij} = 0, & \text{if } i \neq j \\ e_{ii} = 1 \end{cases}$$

$\forall i, j \in \mathbb{N}; 0 \leq i, j \leq n-1$

Hàm biểu diễn One-hot véc tơ:

Trong python chúng ta có thể biến đổi các từ sang dạng one-hot véc tơ thông qua hàm OneHotEncoder của sklearn. Nhưng trước tiên ta sẽ gán index cho các class bằng LabelEncoder:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
words = ['anh', 'em', 'gia đình', 'bạn bè', 'anh', 'em']
le.fit(words)

print('Class of words: ', le.classes_)
# Biến đổi sang dạng số
x = le.transform(words)
print('Convert to number: ', x)
# Biến đổi lại sang class
print('Invert into classes: ', le.inverse_transform(x))
```

```
Class of words:  ['anh' 'bạn bè' 'em' 'gia đình']
Convert to number:  [0 2 3 1 0 2]
Invert into classes:  ['anh' 'em' 'gia đình' 'bạn bè' 'anh' 'em']
```

Thực hiện OneHotEncoder

Top

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np

oh = OneHotEncoder()
classes_indices = list(zip(le.classes_, np.arange(len(le.classes_))))
print('Classes_indices: ', classes_indices)
oh.fit(classes_indices)
print('One-hot categories and indices:', oh.categories_)
# Biến đổi list words sang dạng one-hot
words_indices = list(zip(words, x))
print('Words and corresponding indices: ', words_indices)
one_hot = oh.transform(words_indices).toarray()
print('Transform words into one-hot matrices: \n', one_hot)
print('Inverse transform to categories from one-hot matrices: \n', oh.inverse_transform(one_hot))
```

```
Classes_indices:  [('anh', 0), ('bạn bè', 1), ('em', 2), ('gia đình', 3)]
One-hot categories and indices: [array(['anh', 'bạn bè', 'em', 'gia đình'], dtype=object), array([0, 1, 2, 3
Words and corresponding indices:  [('anh', 0), ('em', 2), ('gia đình', 3), ('bạn bè', 1), ('anh', 0), ('em',
Transform words into one-hot matrices:
 [[1. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 1. 0.]]
Inverse transform to categories from one-hot matrices:
 [['anh' 0]
 ['em' 2]
 ['gia đình' 3]
 ['bạn bè' 1]
 ['anh' 0]
 ['em' 2]]
```

2. Word Embedding

Sau khi biểu diễn từ dưới dạng one-hot véc tơ, mô hình đã có thể huấn luyện được từ dữ liệu được mã hóa. Tuy nhiên dữ liệu này chỉ đáp ứng được khả năng huấn luyện mà chưa phản ánh được mối liên hệ về mặt ngữ nghĩa của các từ. Các hạn chế đó là:

1. Mối quan hệ tương quan giữa các cặp từ bất kì luôn là không tương quan (tức bằng 0). Do đó không có tác dụng trong việc tìm mối liên hệ về nghĩa.
2. Kích thước của véc tơ sẽ phụ thuộc vào số lượng từ vựng có trong bộ văn bản dẫn đến chi phí tính toán rất lớn khi tập dữ liệu lớn.
3. Khi bổ sung thêm các từ vựng mới số chiều của véc tơ có thể thay đổi theo dẫn đến sự không ổn định trong shape.

Do đó các thuật toán nhúng từ được tạo ra nhằm mục đích tìm ra các véc tơ đại diện cho mỗi từ sao cho:

1. Một từ được biểu diễn bởi một véc tơ có số chiều xác định trước.
2. Các từ thuộc cùng 1 nhóm thì có khoảng cách gần nhau trong không gian.

Có nhiều phương pháp nhúng từ khác nhau có thể kể đến. Trong đó có 3 nhóm chính:

1. Sử dụng thống kê tần xuất: tfidf
2. Các thuật toán giảm chiều dữ liệu: SVD, PCA, auto encoder, word2vec
3. Phương pháp sử dụng mạng nơ ron: word2vec, ELMo, BERT.

Phương pháp tfidf có thể được tham khảo mục 2.1 bài viết sau Kĩ thuật feature engineering (https://phamdinhkhanh.github.io/2019/01/07/k-thu-t-feature-engineering.html). Trong bài giới thiệu này sẽ tập trung vào các phương pháp thuộc nhóm giảm chiều dữ liệu.

2.1. Phương pháp SVD

SVD là phương pháp giảm chiều dữ liệu dựa trên một phép phân tích suy biến nhằm tìm ra một ma trận gần sát với ma trận ban đầu. Về phương pháp khai triển và ứng dụng của SVD bạn đọc có thể tham khảo Singular value Decomposition (https://www.kaggle.com/phamdinhkhanh/singular-value-decomposition). Đối với word embedding theo SVD, ta sẽ áp dụng phân tích suy biến trên ma trận đồng xuất hiện của các cặp từ input và output. Trong đó input là từ hiện tại và output là các từ liền kề xung quanh nó. Chẳng hạn chúng ta có 2 câu văn như sau:

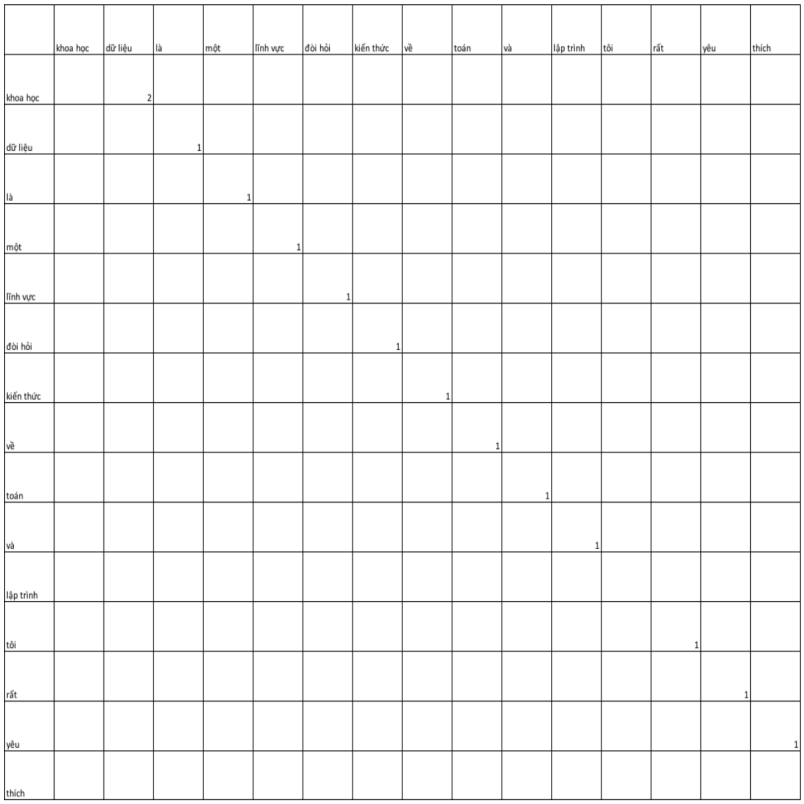
```
Khoa học dữ liệu là một lĩnh vực đòi hỏi kiến thức về toán và lập trình. Tôi rất yêu thích khoa học dữ liệu.
```

Tập từ điển sẽ bao gồm các từ sau:

```
[khoa học, dữ liệu, là, một, lĩnh vực, đòi hỏi, kiến thức, về, toán, và, lập trình, tôi, rất, yêu, thích]
```

Khi đó biểu diễn các từ trong ma trận đồng xuất hiện như bên dưới:

Top



	khoa học	dữ liệu	là	một	lĩnh vực	đòi hỏi	kiến thức	về	toán	và	lập trình	tôi	rất	yêu	thích
khoa học		2													
dữ liệu			1												
là				1											
một					1										
lĩnh vực						1									
đòi hỏi							1								
kiến thức								1							
về									1						
toán										1					
và											1				
lập trình															
tôi													1		
rất														1	
yêu															1
thích															

Hình 1: Ma trận đồng xuất hiện

Chúng ta cũng có thể tìm ra biểu diễn của mỗi từ trong từ điển bằng một véc tơ các nhân tố ẩn dựa vào việc lựa chọn một số lượng các giá trị đặc trưng.

```
import scipy.linalg as ln
import numpy as np
from underthesea import word_tokenize

sentence = 'Khoa học dữ liệu là một lĩnh vực đòi hỏi kiến thức về toán và lập trình. Tôi rất yêu thích Khoa h
token = word_tokenize(sentence)
# Tokenize câu search
print('tokenization of sentences: ', token)
```

```
tokenization of sentences:  ['Khoa học', 'dữ liệu', 'là', 'một', 'lĩnh vực', 'đòi hỏi', 'kiến thức', 'về', 't
```

```
from scipy.sparse import coo_matrix
# Tạo ma trận coherence dưới dạng sparse thông qua khai báo vị trí khác 0 của trục x và y
row = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13]
col = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14]
data =      [2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

X = coo_matrix((data, (row, col)), shape=(15, 15)).toarray()
X
```

```
 array([[0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

Top

```
# Thực hiện phân tích suy biến:
U, S_diag, V = ln.svd(X)
print('Shape of U: ', U.shape)
print('Length of diagonal: ', len(S_diag))
print('Shape of V: ', V.shape)
```

```
Shape of U:  (15, 15)
Length of diagonal:  15
Shape of V:  (15, 15)
```

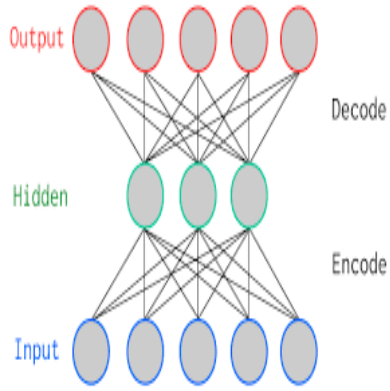
Các ma trận $\mathbf{U}, \mathbf{V}$ lần lượt là ma trận trực giao suy biến trái và phải. Ma trận $\mathbf{S}$ là ma trận đường chéo chính. Ta có: $\mathbf{U}_{15x15}\mathbf{S}_{15x15}\mathbf{V}_{15x15} = \mathbf{X}$ Đường chéo chính của ma trận $\mathbf{S}_{15x15}$ được sắp xếp theo thứ tự giảm dần. Cần lựa chọn bao nhiêu chiều dữ liệu để biểu diễn từ sẽ lấy bấy nhiêu dòng của ma trận đường chéo chính. Để véc tơ biểu diễn sát nhất chúng ta nên lấy các dòng tương ứng với các giá trị đặc trưng lớp nhất. Chẳng hạn muốn biểu diễn các từ dưới dạng véc tơ 6 chiều ta lấy tích $\mathbf{S}_{6\times15}\mathbf{V}_{15x15} = \mathbf{X}_{6\times15}$. Khi đó các cột của ma trận đầu ra $\mathbf{X}_{6\times15}$ sẽ là một véc tơ nhúng của từ tại vị trí tương ứng trong từ điển.

```
import numpy as np
S_truncate = np.zeros(shape = (6, 15))
np.fill_diagonal(S_truncate, S_diag[:6])
print('S truncate: \n', S_truncate)
print('Word Embedding 6 dimensionality: \n', np.dot(S_truncate, V))
```

```
 S truncate:
  [[2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
 Word Embedding 6 dimensionality:
  [[0. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
```

2.2. Phương pháp auto encoder

Auto encoder được xây dựng trên một mạng nơ ron có 3 layer: input, hidden layer và output. Trong đó số units ở input và output là bằng nhau. Số units ở hidden layer sẽ qui định số chiều của véc tơ biểu diễn từ và thông thường sẽ nhỏ hơn số units ở đầu vào.



Hình 2: phương pháp auto encoder với số units ở đầu vào bằng đầu ra.

Bên dưới chúng ta sẽ tiến hành nhúng từ thông qua auto encoder

Top

```
from keras.layers import Dense, Input
from keras.models import Model, Sequential
from keras.optimizers import RMSprop, Adam

def autoencoder(input_unit, hidden_unit):
    model = Sequential()
    model.add(Dense(input_unit, input_shape = (15,), activation = 'relu'))
    model.add(Dense(hidden_unit, activation = 'relu'))
    model.add(Dense(input_unit, activation = 'softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer = Adam(),
                  metrics = ['accuracy'])
    model.summary()
    return model

model_auto = autoencoder(input_unit = 15, hidden_unit = 6)

model_auto.fit(X, X, epochs = 5, batch_size = 3)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 15)                240
_________________________________________________________________
dense_8 (Dense)              (None, 6)                 96
_________________________________________________________________
dense_9 (Dense)              (None, 15)                105
=================================================================
Total params: 441
Trainable params: 441
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
15/15 [==============================] - 0s 20ms/step - loss: 2.5388 - acc: 0.1333
Epoch 2/5
15/15 [==============================] - 0s 585us/step - loss: 2.5290 - acc: 0.0667
Epoch 3/5
15/15 [==============================] - 0s 629us/step - loss: 2.5188 - acc: 0.0667
Epoch 4/5
15/15 [==============================] - 0s 645us/step - loss: 2.5114 - acc: 0.0667
Epoch 5/5
15/15 [==============================] - 0s 719us/step - loss: 2.5023 - acc: 0.0667



<keras.callbacks.History at 0x7ff579fdd518>
```

Mỗi một từ sẽ được biểu diễn bởi véc tơ nhúng có các thành phần là hệ số kết nối hidden units tới output unit tương ứng. Trích xuất layers cuối cùng ta sẽ thu được ma trận nhúng:

```
embedding_matrix = model_auto.layers[2].get_weights()[0]
bias = model_auto.layers[2].get_weights()[1]

print('Shape of embedding_matrix: ', embedding_matrix.shape)
print('Embedding_matrix: \n', embedding_matrix)
```

```
 Shape of embedding_matrix:  (6, 15)
 Embedding_matrix:
  [[ 0.38889918  0.5211232  -0.35681784 -0.29142842  0.25496536  0.47015667
    0.12295379  0.34093136  0.36910903  0.09683032 -0.41072607 -0.07050186
    0.28118226  0.14136976 -0.398313  ]
  [ 0.18342797 -0.14228119 -0.29116338  0.40031028  0.47284338  0.5166124
   -0.47880676  0.49956253  0.36308518  0.07943692  0.46039233 -0.04482159
    0.14367305  0.46219113 -0.37292722]
  [ 0.4906134  -0.00613014 -0.09216617  0.3174584   0.08535323  0.03718374
   -0.0576647   0.13673814 -0.0192671   0.16489299 -0.3544627  -0.4466407
   -0.46152878  0.35548216  0.19229826]
  [-0.04221632 -0.2623642  -0.2671243  -0.14902063 -0.08061455  0.08999895
    0.22966935 -0.54198337 -0.2509707   0.46091208 -0.06831685 -0.5284586
   -0.21089761 -0.13299096  0.36479107]
  [ 0.09093584  0.38861293  0.24202171  0.20458116 -0.25571942  0.05853903
   -0.267772   -0.12935235  0.27599117 -0.25800633  0.2633568  -0.25931272
   -0.03536293 -0.29268453 -0.4267695 ]
  [ 0.26897088  0.24455284 -0.27629155  0.4157534  -0.27802745  0.12034645
    0.47979772  0.5275412   0.00355813  0.26329502 -0.18948056  0.00509128
    0.4196368   0.4636546   0.08472057]]
```

Top

```
from sklearn.metrics.pairwise import cosine_similarity
from numpy.linalg import norm

def cosine(x, y):
    cos_sim = np.dot(x, y)/(norm(x)*norm(y))
    return cos_sim
# Véc tơ biểu diễn từ khoa học
e0 = list(embedding_matrix[:, 0])
# Véc tơ biểu diễn từ dữ liệu
e1 = list(embedding_matrix[:, 1])
# Quan hệ tương quan ngữ nghĩa giữa từ khoa học và dữ liệu
cosine(e0, e1)
```

```
0.5303333
```

Tìm từ tương quan nhất với một từ thông qua khoảng cách cosine_similarity.

```
# Từ có khoảng cách lớn nhất với từ khoa học theo thứ tự
cosines = [cosine(e0, embedding_matrix[:, i]) for i in np.arange(15)]
print('cosines: ', cosines)
np.argsort([cosine(e0, embedding_matrix[:, i]) for i in np.arange(15)])[::-1]
```

```
cosines:  [1.0, 0.5303333, -0.59790766, 0.46414658, 0.27999142, 0.6444732, 0.04827654, 0.63190365, 0.52158606
```

```
array([ 0, 13,  5,  7,  1,  8,  3,  9,  4, 12,  6, 14, 11, 10,  2])
```

như vậy 2 từ ở vị trí thứ 13 và 1 tương ứng với `yêu` và `dữ liệu` là 2 từ có mối liên hệ gần nhất với từ `khoa học` . Xét với bối cảnh của 2 câu văn trên cho thấy khá phù hợp bởi 2 cụm từ: `yêu khoa_học` và `khoa_học dữ_liệu` .

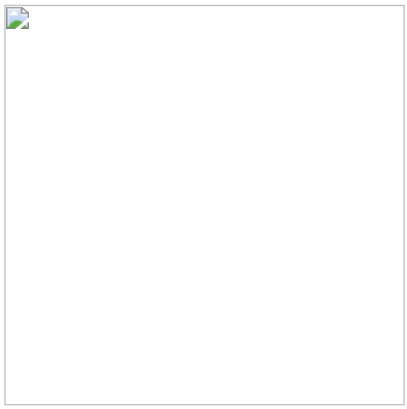
2.3. Mô hình word2vec

Mô hình word2vec có 2 phương pháp chính là skip-grams và CBOW như sau:

skip-grams: Giả sử chúng ta có một câu văn như sau: `Tôi muốn một chiếc cốc màu_xanh đựng hoa quả dầm` . Để thu được một phép nhúng từ tốt hơn chúng ta sẽ lựa chọn ra ngẫu nhiên các từ làm bối cảnh (context). Dựa trên từ bối cảnh, các từ mục tiêu (target) sẽ được xác định nằm trong phạm vi xung quanh từ bối cảnh. Chẳng hạn ta với việc lựa chọn từ `cốc` làm bối cảnh nếu lấy từ tiếp theo, từ liền trước, từ cách đó liền trước 2, 3 từ ta sẽ lần lượt thu được các từ mục tiêu như sau:

Bối cảnh (context)	Mục tiêu (target)
cốc	màu_xanh
cốc	chiếc
cốc	một
cốc	muốn

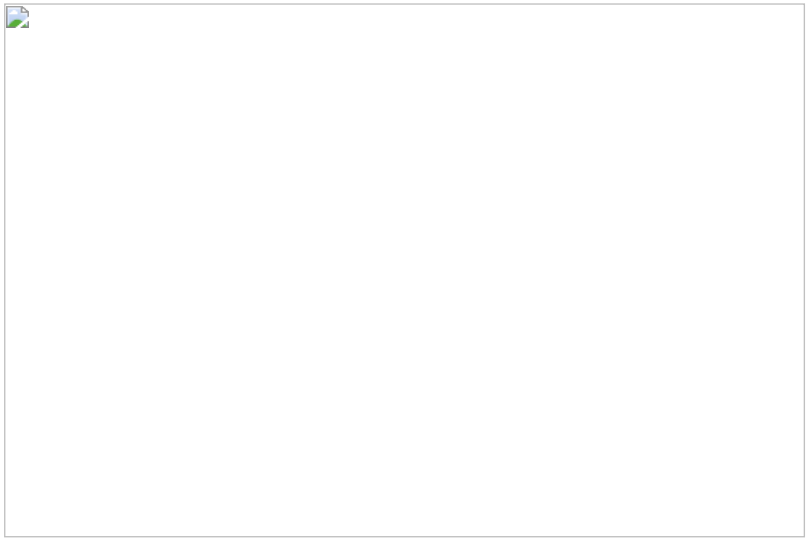
Các nghiên cứu cho thấy từ mục tiêu sẽ được giải thích tốt hơn nếu được học theo các từ bối cảnh. Do đó mô hình skip-grams tìm cách xây dựng một thuật toán học có giám sát có đầu vào là các từ bối cảnh –> đầu ra là từ mục tiêu:



Hình 3: Kiến trúc mô hình skip-grams. $\mathbf{w_t}$ là từ bối cảnh, $\mathbf{w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}}$ là các từ mục tiêu.

- Mục tiêu: Từ từ bối cảnh c ta muốn dự báo từ mục tiêu t. $\mathrm{Context\text{-}c}\ (\text{"cốc"}) \rightarrow \mathrm{Target\text{-}t}\ (\text{"màu\_xanh"})$
- Mô hình:

Top



Hình 4: Kiến trúc mạng nơ ron trong mô hình skip-grams.

Cũng giống như các cách tiếp cận thông thường khác, mô hình sẽ biểu diễn một từ bối cảnh dưới dạng one-hot véc tơ $\mathbf{o_c}$. Véc tơ này sẽ trở thành đầu vào cho một mạng nơ ron có tầng ẩn gồm 300 units. Kết quả ở output layer là một hàm softmax tính xác xuất để các từ mục tiêu phân bố vào những từ trong vocabulary (10000 từ). Dựa trên quá trình feed forward và back propagation mô hình sẽ tìm ra tham số tối ưu để kết quả dự báo từ mục tiêu là chuẩn xác nhất. Khi đó quay trở lại tầng hidden layer ta sẽ thu được đầu ra tại tầng này là ma trận nhúng $\mathbf{E} \in \mathbb{R}^{n \times 300}$.

$$\mathbf{o_c} \rightarrow \mathbf{E} \rightarrow \mathbf{e_c} \rightarrow \text{softmax} \rightarrow \hat{\mathbf{y}}$$

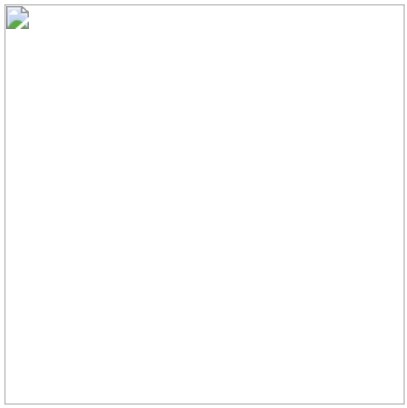
$\mathbf{e_c} \in \mathbb{R}^{300}$ là véc tơ nhúng trích xuất từ ma trận $\mathbf{E}$ tương ứng với từ bối cảnh $\mathbf{c}$. $\hat{\mathbf{y}}$ là xác xuất được dự báo của từ mục tiêu.

Khi áp dụng hàm softmax, xác xuất ở đầu ra có dạng: $\mathbf{P(t = v_i|c)} = \frac{e^{\theta_i{}^T \mathbf{e_c}}}{\sum_{j=1}^{10000} e^{\theta_j{}^T \mathbf{e_c}}}$

trong đó $\theta_\mathbf{i} \in \mathbb{R}^{300}$ là các véc tơ tham số thể hiện sự liên kết giữa các units ở hidden layer với output layer.

Kết quả dự báo mô hình mạng nơ ron càng chuẩn xác thì véc tơ nhúng sẽ càng thể hiện được mối liên hệ trên thực tế giữa từ bối cảnh và mục tiêu chuẩn xác. Kết quả cuối cùng ta quan tâm chính là các dòng của ma trận $\mathbf{E}$. Chúng là các véc tơ nhúng $\mathbf{e_c}$ đại diện cho một từ bối cảnh $\mathbf{c}$.

CBOW: Chúng ta nhận thấy rằng mô hình skip-grams sẽ rất tốn chi phí để tính toán vì mẫu số xác xuất là tổng của rất nhiều số mũ cơ số tự nhiên. Để hạn chế chi phí tính toán mô hình CBOW (continueos backward model) được áp dụng. Về cơ bản thì CBOW là một quá trình ngược lại của skip-grams. Khi đó input của skip-grams sẽ được sử dụng làm output trong CBOW và ngược lại.



Hình 5: Kiến trúc CBOW

Kiến trúc mạng nơ ron của CBOW sẽ gồm 3 layers:

1. Input layers: Là các từ bối cảnh xung quanh từ mục tiêu.
2. Projection layer: Lấy trung bình véc tơ biểu diễn của toàn bộ các từ input để tạo ra một véc tơ đặc trưng.
3. Output layer: Là một dense layers áp dụng hàm softmax để dự báo xác xuất của từ mục tiêu.

Bên dưới chúng ta cùng sử dụng mô hình word2vec theo phương pháp CBOW để nhúng các từ bối cảnh thành những véc tơ có 300 chiều bằng `keras`. Dữ liệu input là các câu trong kinh thánh được lấy từ bible-kjv.txt (http://www.gutenberg.org/ebooks/10). Để xây dựng mô hình sẽ đi qua các bước sau đây:

1. Tạo bộ từ điển cho toàn bộ các câu trong kinh thánh sao cho mỗi từ được gán giá trị bởi 1 số index.
2. Mã hoá toàn bộ các câu văn bằng index.
3. Xác định các cặp `Context --> Target` tương ứng với input và output của mô hình. Trong đó từ `Target` là từ hiện tại ở vị trí `index`, các từ `Context` nằm ở khoảng `[index - window_size, index + window_size]`. Padding giá trị 0 tại những context không đủ độ dài là `2*window_size`.
4. Xây dựng mạng nơ ron.
5. Huấn luyện mô hình.
6. Trích xuất ma trận nhúng tại đầu ra của hidden layer.

Top

Bước 1: Tạo từ điển

```
from keras.preprocessing import text
from keras.utils import np_utils
from keras.preprocessing import sequence
from nltk.corpus import gutenberg
from string import punctuation
import nltk
nltk.download('gutenberg')
nltk.download('punkt')
norm_bible = gutenberg.sents('bible-kjv.txt')
norm_bible = [' '.join(doc) for doc in norm_bible]
tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(norm_bible)
word2id = tokenizer.word_index

# build vocabulary of unique words
word2id['PAD'] = 0
id2word = {v:k for k, v in word2id.items()}
vocab_size = len(word2id)

print('Vocabulary Size:', vocab_size)
print('Vocabulary Sample:', list(word2id.items())[:10])
```

```
Using TensorFlow backend.


[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Vocabulary Size: 12746
Vocabulary Sample: [('the', 1), ('and', 2), ('of', 3), ('to', 4), ('that', 5), ('in', 6), ('he', 7), ('shall'
```

Bước 2: Mã hoá toàn bộ các câu văn bằng index.

```
wids = [[word2id[w] for w in text.text_to_word_sequence(doc)] for doc in norm_bible]
print('Embedding sentence by index: ', wids[:5])
```

```
Embedding sentence by index:  [[1, 53, 1342, 6058], [1, 280, 2678, 3, 1, 53, 1342, 6058], [1, 254, 448, 3, 16
```

Bước 3: Xác định `Context --> Target`.

```
import numpy as np
def generate_context_word_pairs(corpus, window_size, vocab_size):
    context_length = window_size*2
    for words in corpus:
        sentence_length = len(words)
        # print('words: ', words)
        for index, word in enumerate(words):
            context_words = []
            label_word   = []
            # Start index of context
            start = index - window_size
            # End index of context
            end = index + window_size + 1
            # List of context_words
            context_words.append([words[i] for i in range(start, end) if 0 <= i < sentence_length and i != i
            # List of label_word (also is target word).
            # print('context words {}: {}'.format(context_words, index))
            label_word.append(word)
            # Padding the input 0 in the left in case it does not satisfy number of context_words = 2*window_
            x = sequence.pad_sequences(context_words, maxlen=context_length)
            # print('context words padded: ', x)
            # Convert label_word into one-hot vector corresponding with its index
            y = np_utils.to_categorical(label_word, vocab_size)
            yield (x, y)


# Test this out for some samples
i = 0
window_size = 2 # context window size
for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, vocab_size=vocab_size):
    if 0 not in x[0]:
        print('Context (X):', [id2word[w] for w in x[0]], '-> Target (Y):', id2word[np.argwhere(y[0])[0][0]]

        if i == 10:
            break
        i += 1
```

Top

```
Context (X): ['the', 'old', 'of', 'the'] -> Target (Y): testament
Context (X): ['old', 'testament', 'the', 'king'] -> Target (Y): of
Context (X): ['testament', 'of', 'king', 'james'] -> Target (Y): the
Context (X): ['of', 'the', 'james', 'bible'] -> Target (Y): king
Context (X): ['the', 'first', 'of', 'moses'] -> Target (Y): book
Context (X): ['first', 'book', 'moses', 'called'] -> Target (Y): of
Context (X): ['book', 'of', 'called', 'genesis'] -> Target (Y): moses
Context (X): ['1', '1', 'the', 'beginning'] -> Target (Y): in
Context (X): ['1', 'in', 'beginning', 'god'] -> Target (Y): the
Context (X): ['in', 'the', 'god', 'created'] -> Target (Y): beginning
Context (X): ['the', 'beginning', 'created', 'the'] -> Target (Y): god
```

Bước 4: Xây dựng mạng nơ ron gồm 3 layers chính:

1. Embedding layer: dùng để mã hoá đầu vào thành các one-hot véc tơ. Số lượng từ ở đầu vào chính là `2*window_size` . Sau khi mã hoá, qua quá trình training mỗi một từ vựng sẽ được biểu diễn bởi một véc tơ nhúng 100 chiều tương ứng với `embed_size` .
2. Mean layer: Tính véc tơ trung bình của các véc tơ đầu ra ở Embedding layer. Số lượng véc tơ là `2*window_size` .
3. Dense layer: Tính phân phối xác xuất của từ `Target` dựa vào hàm softmax.

```
import keras.backend as K
from keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda
embed_size = 100

# build CBOW architecture
cbow = Sequential()
cbow.add(Embedding(input_dim=vocab_size, output_dim=embed_size, input_length=window_size*2))
cbow.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(embed_size,)))
cbow.add(Dense(vocab_size, activation='softmax'))
cbow.compile(loss='categorical_crossentropy', optimizer='rmsprop')

# view model summary
print(cbow.summary())
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op_def_library.py
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 4, 100)            1274600
_________________________________________________________________
lambda_1 (Lambda)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 12746)             1287346
=================================================================
Total params: 2,561,946
Trainable params: 2,561,946
Non-trainable params: 0
_________________________________________________________________
None
```

```
print('number of window: ', len(wids))
```

```
number of window:  30103
```

Bước 5: Huấn luyện mô hình. Chúng ta sẽ huấn luyện mô hình dựa trên 100 câu văn đầu tiên và trải qua 5 epochs.

```
for epoch in range(1, 6):
    loss = 0.
    i = 0
    for x, y in generate_context_word_pairs(corpus=wids[:100], window_size=window_size, vocab_size=vocab_siz
        i += 1
        loss += cbow.train_on_batch(x, y)
        if i % 500 == 0:
            print('Processed {} (context, word) pairs'.format(i))

    print('Epoch:', epoch, '\tLoss:', loss)
```

Top

```
Processed 500 (context, word) pairs
Processed 1000 (context, word) pairs
Processed 1500 (context, word) pairs
Processed 2000 (context, word) pairs
Processed 2500 (context, word) pairs
Epoch: 1        Loss: 16144.638676483184
Processed 500 (context, word) pairs
Processed 1000 (context, word) pairs
Processed 1500 (context, word) pairs
Processed 2000 (context, word) pairs
Processed 2500 (context, word) pairs
Epoch: 2        Loss: 15855.159716077149
Processed 500 (context, word) pairs
Processed 1000 (context, word) pairs
Processed 1500 (context, word) pairs
Processed 2000 (context, word) pairs
Processed 2500 (context, word) pairs
Epoch: 3        Loss: 16312.521473242901
Processed 500 (context, word) pairs
Processed 1000 (context, word) pairs
Processed 1500 (context, word) pairs
Processed 2000 (context, word) pairs
Processed 2500 (context, word) pairs
Epoch: 4        Loss: 16708.009846252855
Processed 500 (context, word) pairs
Processed 1000 (context, word) pairs
Processed 1500 (context, word) pairs
Processed 2000 (context, word) pairs
Processed 2500 (context, word) pairs
Epoch: 5        Loss: 16937.563758765813
```

Bước 6: Trích xuất ma trận nhúng của các từ.

```
import pandas as pd
weights = cbow.get_weights()[0]
weights = weights[1:]
print(weights.shape)

pd.DataFrame(weights, index=list(id2word.values())[1:]).head()
```

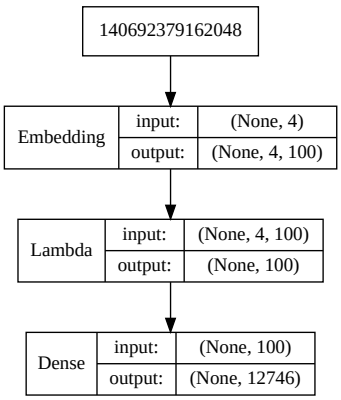
```
(12745, 100)
```

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94
and	0.566605	0.193528	0.580568	0.399872	-0.477950	0.058564	0.136201	-0.180622	0.332103	-0.126461	...	-0.066192	0.102113	0.221867	0.187878	-0.256676
of	-0.006290	1.063426	-0.064591	0.273369	0.005391	-0.099132	0.092056	0.334700	-0.223147	-0.510919	...	-0.263550	0.419170	0.212709	0.760192	-0.473723
to	0.212710	1.351286	0.431805	0.586412	-0.079169	-0.097280	-0.117581	0.064991	0.095262	-0.399057	...	0.089253	-0.047217	0.033623	-0.407661	0.051037
that	0.105932	0.584896	0.032046	0.090305	0.009700	0.017799	-0.115047	-0.002097	0.204439	-0.182319	...	-0.117154	0.324759	0.126172	-0.197954	-0.247685
in	0.167913	0.545757	-0.136884	0.033220	-0.025752	0.112379	-0.253199	0.116862	0.254202	0.242693	...	-0.087999	-0.037836	0.077510	-0.073683	-0.367073

5 rows × 100 columns

```
# visualize model structure
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(cbow, show_shapes=True, show_layer_names=False,
                 rankdir='TB').create(prog='dot', format='svg'))
```



Hoàn toàn tương tự như kiến trúc của **CBOW**, ta xây dựng model **skip-grams** như sau:

Bước 1: Chuẩn bị dữ liệu là các cặp [context, target]

Top

```
from keras.preprocessing.sequence import skipgrams

# generate skip-grams
skip_grams = [skipgrams(wid, vocabulary_size=vocab_size, window_size=window_size) for wid in wids[:100]]

# view sample skip-grams
pairs, labels = skip_grams[0][0], skip_grams[0][1]
for i in range(10):
    print("({:s} ({:d}), {:s} ({:d})) -> {:d}".format(
          id2word[pairs[i][0]], pairs[i][0],
          id2word[pairs[i][1]], pairs[i][1],
          labels[i]))
```

```
(the (1), harmless (6878)) -> 0
(king (53), ramoth (4038)) -> 0
(james (1342), bible (6058)) -> 1
(king (53), bible (6058)) -> 1
(james (1342), moist (9056)) -> 0
(james (1342), coffer (6377)) -> 0
(bible (6058), james (1342)) -> 1
(james (1342), lintels (11682)) -> 0
(king (53), give (155)) -> 0
(the (1), james (1342)) -> 1
```

Bước 2: Xây dựng mạng nơ ron

```
from keras.layers import Input, Dot, dot, concatenate
# from keras.engine.input_layer import Input
from keras.layers.core import Dense, Reshape
from keras.layers.embeddings import Embedding
from keras.models import Sequential, Model, Input

# build skip-gram architecture
word_input = Input(shape = (1,))
word_embed = Embedding(vocab_size, embed_size,
                       embeddings_initializer="glorot_uniform",
                       input_length=1, name = 'word_embedding')(word_input)
word_output = Reshape((embed_size, ))(word_embed)
word_model = Model(word_input, word_output)

print('word_model: \n', word_model.summary())
context_input = Input(shape = (1,))
context_embed = Embedding(vocab_size, embed_size,
                  embeddings_initializer="glorot_uniform",
                  input_length=1, name = 'context_embedding')(context_input)
context_output = Reshape((embed_size,))(context_embed)
context_model = Model(context_input, context_output)
print('context_model: \n', context_model.summary())

concate = dot([word_output, context_output], axes = -1)
dense = Dense(1, kernel_initializer="glorot_uniform", activation="sigmoid")(concate)
model = Model(inputs = [word_input, context_input], outputs = dense)
model.compile(loss="mean_squared_error", optimizer="rmsprop")

# view model summary
print('model merge word and context: \n', model.summary())
```

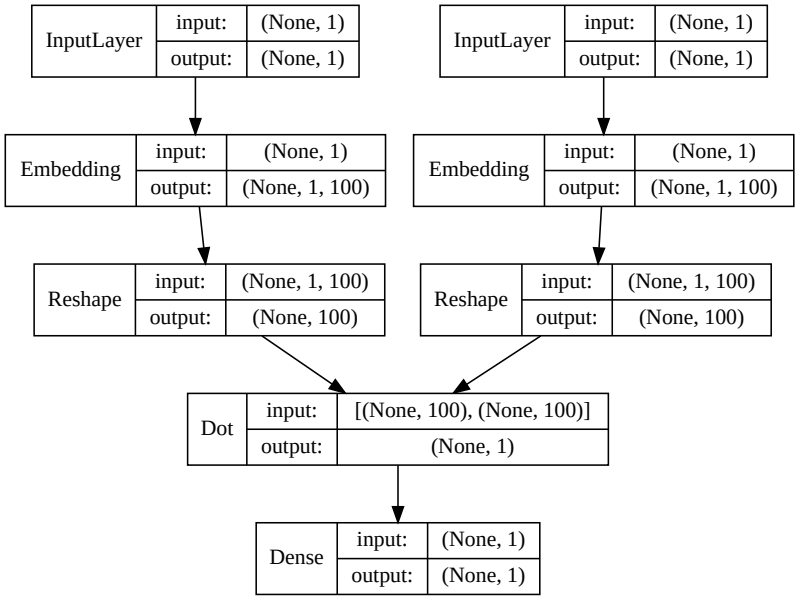
Top

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_20 (InputLayer)        (None, 1)                 0         
_________________________________________________________________
word_embedding (Embedding)   (None, 1, 100)            1274600   
_________________________________________________________________
reshape_21 (Reshape)         (None, 100)               0         
=================================================================
Total params: 1,274,600
Trainable params: 1,274,600
Non-trainable params: 0
_________________________________________________________________
word_model:
 None
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_21 (InputLayer)        (None, 1)                 0         
_________________________________________________________________
context_embedding (Embedding (None, 1, 100)            1274600   
_________________________________________________________________
reshape_22 (Reshape)         (None, 100)               0         
=================================================================
Total params: 1,274,600
Trainable params: 1,274,600
Non-trainable params: 0
_________________________________________________________________
context_model:
 None
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to                     
==================================================================================================
input_20 (InputLayer)           (None, 1)            0                                            
__________________________________________________________________________________________________
input_21 (InputLayer)           (None, 1)            0                                            
__________________________________________________________________________________________________
word_embedding (Embedding)      (None, 1, 100)       1274600     input_20[0][0]                   
__________________________________________________________________________________________________
context_embedding (Embedding)   (None, 1, 100)       1274600     input_21[0][0]                   
__________________________________________________________________________________________________
reshape_21 (Reshape)            (None, 100)          0           word_embedding[0][0]             
__________________________________________________________________________________________________
reshape_22 (Reshape)            (None, 100)          0           context_embedding[0][0]          
__________________________________________________________________________________________________
dot_5 (Dot)                     (None, 1)            0           reshape_21[0][0]                 
                                                                 reshape_22[0][0]                 
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 1)            2           dot_5[0][0]                      
==================================================================================================
Total params: 2,549,202
Trainable params: 2,549,202
Non-trainable params: 0
__________________________________________________________________________________________________
model merge word and context:
 None
```

```
# visualize model structure
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(model, show_shapes=True, show_layer_names=False,
                 rankdir='TB').create(prog='dot', format='svg'))
```

Top



Bước 3: Huấn luyện mô hình.

Để cho nhanh thì mình sẽ training trên 100 skip_grams đầu tiên.

```
for epoch in range(1, 6):
    loss = 0
    for i, elem in enumerate(skip_grams[:100]):
        pair_first_elem = np.array(list(zip(*elem[0]))[0], dtype='int32')
        pair_second_elem = np.array(list(zip(*elem[0]))[1], dtype='int32')
        labels = np.array(elem[1], dtype='int32')
        X = [pair_first_elem, pair_second_elem]
        Y = labels
        if i % 500 == 0:
            print('Processed {} (skip_first, skip_second, relevance) pairs'.format(i))
        loss += model.train_on_batch(X,Y)

    print('Epoch:', epoch, 'Loss:', loss)

```

```
Processed 0 (skip_first, skip_second, relevance) pairs
Epoch: 1 Loss: 24.676736623048782
Processed 0 (skip_first, skip_second, relevance) pairs
Epoch: 2 Loss: 22.21561288833618
Processed 0 (skip_first, skip_second, relevance) pairs
Epoch: 3 Loss: 18.663180768489838
Processed 0 (skip_first, skip_second, relevance) pairs
Epoch: 4 Loss: 15.21924028545618
Processed 0 (skip_first, skip_second, relevance) pairs
Epoch: 5 Loss: 12.227664299309254
```

Bước 4: Trích xuất ra véc tơ nhúng ở layer đầu tiên.

```
import pandas as pd

word_embedding_layer = model.get_layer('word_embedding')
weights = word_embedding_layer.get_weights()[0]

print(weights.shape)
pd.DataFrame(weights, index=id2word.values()).head()
```

```
(12746, 100)
```

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	
the	0.017877	-0.006345	0.010881	0.013126	-0.012344	0.001429	0.013889	0.003133	-0.021061	-0.018586	...	-0.020651	0.013076	-0.021328	0.013153	0.007915	0
and	0.382729	0.396258	-0.364188	0.380298	-0.391238	0.368729	0.358361	0.386061	0.325783	0.334060	...	0.363881	0.367605	-0.358629	-0.358335	-0.375458	0
of	0.349388	0.410817	-0.408293	0.396892	-0.386039	0.360959	0.393057	0.292898	0.354751	-0.366920	...	0.378525	0.407809	-0.411903	0.257437	-0.376674	0
to	0.272330	0.324286	-0.283027	0.296941	-0.275317	0.268769	0.288633	0.217401	0.276515	-0.203013	...	0.265409	0.290523	-0.281284	0.086492	-0.265537	0
that	0.198596	0.185280	-0.195971	0.194342	-0.172201	0.198094	0.193203	0.124202	0.142453	-0.162316	...	0.179012	0.151464	-0.189423	-0.135934	-0.167137	0

5 rows × 100 columns

top

Tìm các từ gần nghĩa nhất với 2 từ `['egypt', 'king']` dựa trên khoảng cách euclidean.

```
from sklearn.metrics.pairwise import euclidean_distances

distance_matrix = euclidean_distances(weights)
print(distance_matrix.shape)

similar_words = {search_term: [id2word[idx] for idx in distance_matrix[word2id[search_term]-1].argsort()[1:6]
                   for search_term in ['egypt', 'king']}

similar_words
```

```
(12746, 12746)



{'egypt': ['ethiopia', 'earth', 'dwell', 'go', 'from'],
 'king': ['out', 'these', 'by', 'lord', 'son']}
```

2.3.1. Biểu diễn t-SNE

t-SNE là một thuật toán giảm chiều dữ liệu `dimensionality reduction` rất hiệu quả. Thông thường đối với những véc tơ nhiều hơn 3 chiều chúng ta sẽ tìm cách giảm chúng về 2 hoặc 3 chiều bằng thuật toán t-SNE và biểu diễn chúng trong không gian để nhận biết mối liên hệ, tính chất.

Tiếp theo chúng ta sẽ biểu diễn các từ trong không gian 2 chiều dựa trên thuật toán t-SNE.

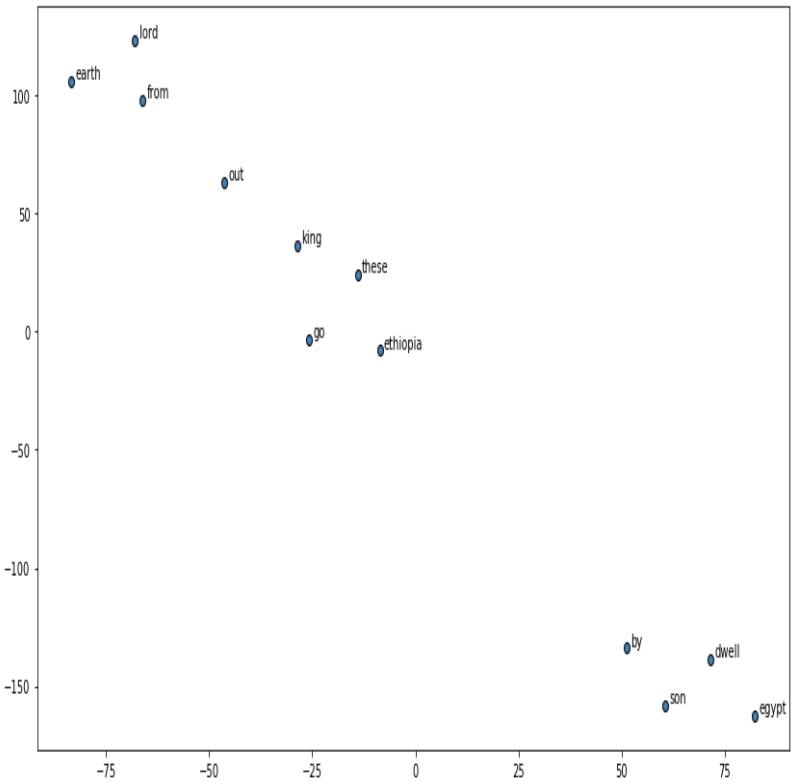
```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

words = sum([[k] + v for k, v in similar_words.items()], [])
words_ids = [word2id[w] for w in words]
word_vectors = np.array([weights[idx] for idx in words_ids])
print('Total words:', len(words), '\tWord Embedding shapes:', word_vectors.shape)

tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=3)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(word_vectors)
labels = words

plt.figure(figsize=(14, 8))
plt.scatter(T[:, 0], T[:, 1], c='steelblue', edgecolors='k')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points')
```

```
Total words: 12         Word Embedding shapes: (12, 100)
```



Top

Để tiết kiệm thời gian, tôi chỉ training với 100 skip-grams đầu tiên nên mô hình chưa phản ánh được chuẩn xác mối quan hệ của từ. Bạn đọc có thể tăng số lượng skip-grams để các từ có mối liên hệ gần sẽ được nhóm vào 1 nhóm trên biểu đồ TSNE.

2.4. Sử dụng gensim cho mô hình word2vec

Cách training trên chỉ sử dụng để chúng ta hiểu rõ cơ chế hoạt động của 2 phương pháp **skip-grams** và **CBOW** trong mô hình word2vec. Trên thực tế mô hình có thể được training trên gensim với chỉ 1 vài dòng rất đơn giản như sau:

```
from gensim.models import Word2Vec
# Training model với 1000 câu đầu tiên trong kinh thánh
sentences = [[item.lower() for item in doc.split()] for doc in norm_bible[:1000]]
model = Word2Vec(sentences, min_count = 1, size = 150, window = 10, sg = 1, workers = 8)
model.train(sentences, total_examples = model.corpus_count, epochs = 10)
```

```
(210070, 336740)
```

Trong đó có một số tham số quan trọng trong Word2Vec như sau:

- size: Kích thước của ma trận nhúng.
- window: Kích thước cửa sổ được sử dụng để khởi tạo các n-gram.
- sg: Nhận 2 giá trị {0, 1}. Nếu là 0: phương pháp CBOW, nếu là 1: skip-grams.
- wokers: Số core CPU được huy động để huấn luyện. Càng nhiều core tốc độ huấn luyện càng nhanh.

```
# Lấy véc tơ biểu diễn của từ king
print('embedding vector shape: ', model.wv['king'].shape)
model.wv['king']
```

```
 embedding vector shape:  (150,)

 array([-0.00468112,  0.16854957,  0.01438654, -0.05265754,  0.14835362,
         0.16904514,  0.13904604, -0.389859  , -0.27150822, -0.20627081,
         0.14461713,  0.14430152, -0.6332871 ,  0.26111943, -0.62677157,
         0.31035894, -0.06819729, -0.09760765,  0.03894788,  0.08955805,
         0.37997362, -0.11426175, -0.24091758,  0.21360792,  0.21109544,
        -0.35530874, -0.11317078,  0.32211563, -0.20230685,  0.13549906,
         0.35079992,  0.12786317,  0.37597153,  0.23084798, -0.26415083,
         0.26244414,  0.07653711, -0.50538695,  0.2834227 ,  0.20041615,
         0.0674964 ,  0.01574622,  0.42599007, -0.16902669,  0.4619288 ,
        -0.30663815, -0.27341986, -0.02219926,  0.63796794, -0.05939501,
        -0.2685611 ,  0.05930207,  0.14947902,  0.12269587, -0.13594696,
         0.07239573,  0.43372607,  0.05574725,  0.47558722,  0.01881623,
        -0.67344916,  0.02950857,  0.25267097,  0.34665427, -0.2924466 ,
        -0.3019795 , -0.40723747,  0.22149928,  0.09181835, -0.2102407 ,
         0.3960522 ,  0.33556274, -0.35339063, -0.06665646,  0.03615884,
        -0.04388156,  0.78695637,  0.07246866, -0.10199204,  0.0916383 ,
         0.21444249, -0.12521476,  0.21644261,  0.313953  ,  0.09498119,
         0.09211312, -0.32217   ,  0.00767796,  0.10209975,  0.42178214,
         0.2544956 ,  0.22292465,  0.40680042,  0.33036977,  0.01546835,
         0.58035815,  0.02209221,  0.13864015, -0.29937524, -0.14904518,
        -0.23794968,  0.42327195, -0.18905397,  0.27455658,  0.02095251,
         0.17467256, -0.10094242,  0.12557817, -0.07476169, -0.12560274,
        -0.23021477,  0.20215885,  0.03653349, -0.14345853, -0.09200411,
         0.23576148,  0.4421827 ,  0.32885996,  0.01603066,  0.20421034,
        -0.17228857,  0.08368498,  0.22233133, -0.03762142, -0.30013585,
         0.2022897 , -0.26879194,  0.20945235,  0.3739482 , -0.41301957,
        -0.17121448, -0.49887335,  0.15468772, -0.42403707, -0.40717396,
        -0.2646839 ,  0.30112094,  0.16615865, -0.44990897,  0.17940831,
        -0.06671996,  0.1638959 ,  0.4423822 ,  0.24692418, -0.09863947,
        -0.06495735, -0.5664116 ,  0.52329963,  0.01605448,  0.33879682],
       dtype=float32)
```

```
# Lấy các từ có mối liên hệ gần nhất với 1 từ dựa trên khoảng cách
model.most_similar('king')
```

```
 [('admah', 0.8957317471504211),
  ('tidal', 0.872719407081604),
  ('zeboiim', 0.8709798455238342),
  ('shinar', 0.870228111743927),
  ('elam', 0.8701319098472595),
  ('ellasar', 0.8675274848937988),
  ('arioch', 0.8656346201896667),
  ('chedorlaomer', 0.8627975583076477),
  ('amraphel', 0.861689031124115),
  ('bela', 0.8449435830116272)]
```

3. Tài liệu tham khảo

Top

1. SVD - phamdinhkhanh (https://www.kaggle.com/phamdinhkhanh/singular-value-decomposition)
2. Auto Encoder - standford university (http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/)

3. word2vec gensim package (https://radimrehurek.com/gensim/models/word2vec.html)
4. Efficient Estimation of Word Representations in Vector Space - Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean (https://arxiv.org/abs/1301.3781)
5. Vector Representations of Words - tensorflow (https://www.tensorflow.org/tutorials/representation/word2vec)
6. The Skip-gram Model - kdnuggets.com (https://www.kdnuggets.com/2018/04/implementing-deep-learning-methods-feature-engineering-text-data-skip-gram.html)
7. The Continuous Bag of Words (CBOW) - kdnuggets.com (https://www.kdnuggets.com/2018/04/implementing-deep-learning-methods-feature-engineering-text-data-cbow.html)

Top