RNN coding 발표- 6조

강재훈, 김민선, 송영훈

목차

1. CBOW(continuous bag-of-words)

중앙 단어(center word)의 앞 뒤 단어(context)를 고려해서 중앙에 있는 단어를 예측하는 방법

1. CBOW(continuous bag-of-words)

the king loves the queen
the queen loves the king
the dwarf hates the king
the queen hates the dwarf
the dwarf poisons the king
the dwarf poisons the queen

.

.

Q. the king ______ the dwarf ?

1. CBOW(continuous bag-of-words)

V : vocabulary size(unique words의 개수)
N : hidden layer size(단어를 embedding하고 싶은 차원)
bag-of-words: unique words의 전문 용어

e.g.)
unique words : the king loves queen dwarf hates poissons

V = 7
N = 3

1. CBOW(continuous bag-of-words)

파라미터 설정

알고 있는 파라미터: one-hot vector들로 나타내어진 문장

input : context의 one-hot vector = $x^{(c)}$

output : target word = $y^{(c)}$ = y

미지의 파라미터 : $V \in R^{n \times |V|}$, $U \in R^{|V| \times n}$

1. CBOW(continuous bag-of-words)

model

1. input 컨텍스트(context)에 대한 one hot word vectors를 생성한다
   $(x^{(c-m)}, ..., x^{(c-1)}, x^{(c+1)}, ..., x^{(c+m)} \in R^{|V|})$
2. context $(v_{c-m} = Vx^{(c+m)}, v_{c-m+1} = Vx^{(c-m+1)}, ..., v_{c+m} = Vx^{(c+m)} \in R^n)$ 에 대한 embedded word vectors를 얻는다.

x	the	king	loves	queen	dwarf	hates	poissons
1	0	0	1	0	0	0	0
2	0	0	0	1	0	0	0

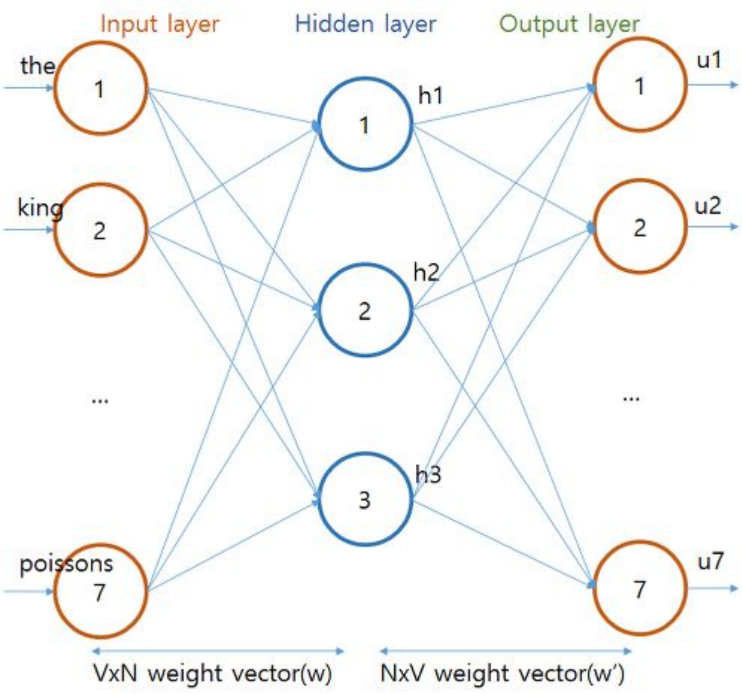
x

w	1	2	3
the	-0.05080	0.07397	0.01468
king	0.13795	-0.04257	-0.07065
loves	-0.11326	0.14482	-0.16535
queen	0.08575	-0.10088	-0.09138
dwarf	0.00959	0.12892	-0.03323
hates	0.08653	-0.05267	-0.11863
poissons	-0.06319	-0.11847	-0.12305

=

loves	-0.11326	0.14482	-0.16535
queen	0.08575	-0.10088	-0.09138

1. CBOW(continuous bag-of-words)



1. CBOW(continuous bag-of-words)

model

3. $\hat{v} = \frac{v_{c-m}+v_{c-m+1}+\ldots+v_{c+m}}{2m} \in R^n$ 를 얻기 위해 벡터들의 평균을 취한다.

4. score vector (z) 을 만든다. 비슷한 벡터들의 내적(dot product)은 높은 값을 갖고, 이는 높은 점수를 얻기 위해서 비슷한 단어들끼리 가까이 위치하도록 강제한다.

$$z = U\hat{v} \in R^{|V|}$$

5. score들을 확률로 바꾼다. $\hat{y} = softmax(z) \in R^{|V|}$

6. 우리는 우리가 만든 확률 $y \in R^{|V|}$가 실제 확률 $\hat{y} \in R^{|V|}$와 일치하고, 실제 단어의 one hot vector와 일치하기를 원한다.

1. CBOW(continuous bag-of-words)

loss function

이제 우리는 U와 V를 가지고 있을때, 우리의 모델이 어떻게 작동하는지 이해했다. 그렇다면, 이 두개의 행렬-U와V-는 어떻게 학습할까?

대중적인 distance/loss measure인 cross entropy 를 사용할 것이다.

$$H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j log(\hat{y}_j) \xrightarrow{\text{CBOW에서는 y가 one-hot vector라고 가정}} H(\hat{y}, y) = -ylog(\hat{y})$$

가정 1. 우리의 예측이 완벽해서 $\hat{y}_c = 1$ 인 상황을 가정하자.
그러면 $H(\hat{y}, y) = -1log(1) = 0$ 임을 알 수 있다. 따라서, 완벽한 예측을 위해서, 우리는 loss를 0으로 만들어야 한다.

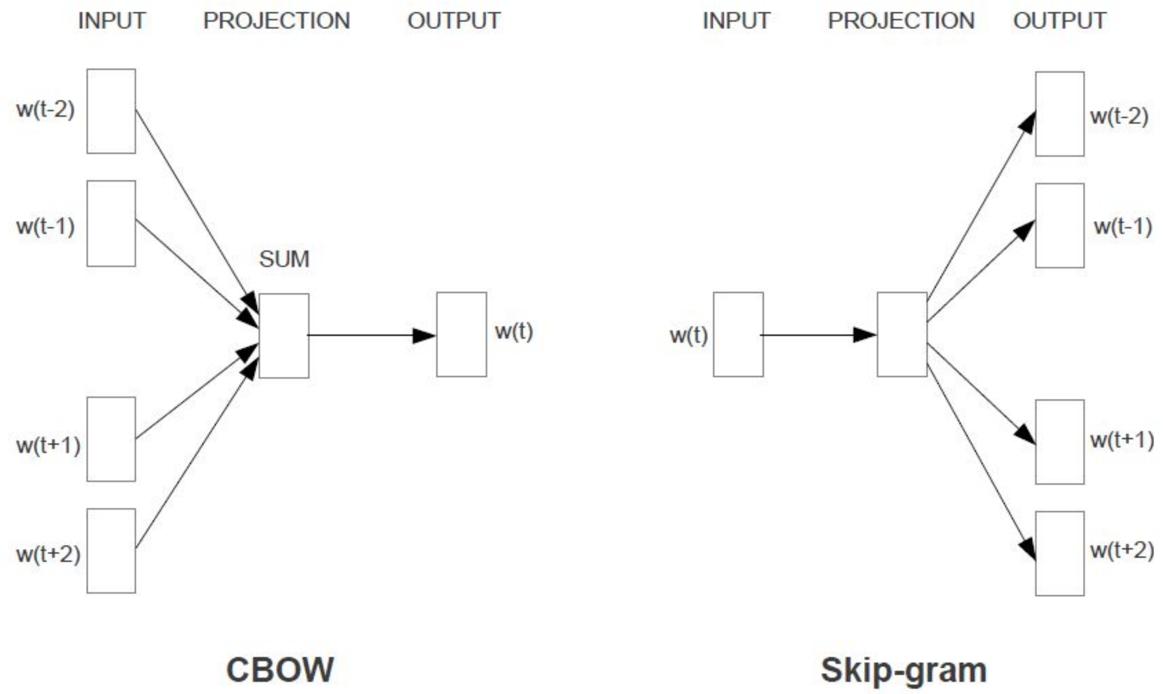
1. CBOW(continuous bag-of-words)

loss function

가정 2. 반대의 경우로, 우리의 예측이 매우 잘못되어서 $\hat{y}_c = 0.01$인 경우를 가정, loss를 계산 해보면 $H(\hat{y}, y) = -1log(0.01) \approx 4.605$이다.

$$
\begin{aligned}
minimize\ J &= -logP(w_c|w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}) \\
&= -logP(u_c|\hat{v}) \\
&= -log\frac{exp(u_c^T\hat{v})}{\sum_{j=1}^{|V|} exp(u_j^T\hat{v})} \\
&= -u_c^T\hat{v} + log\sum_{j=1}^{|V|} exp(u_j^T\hat{v})
\end{aligned}
$$

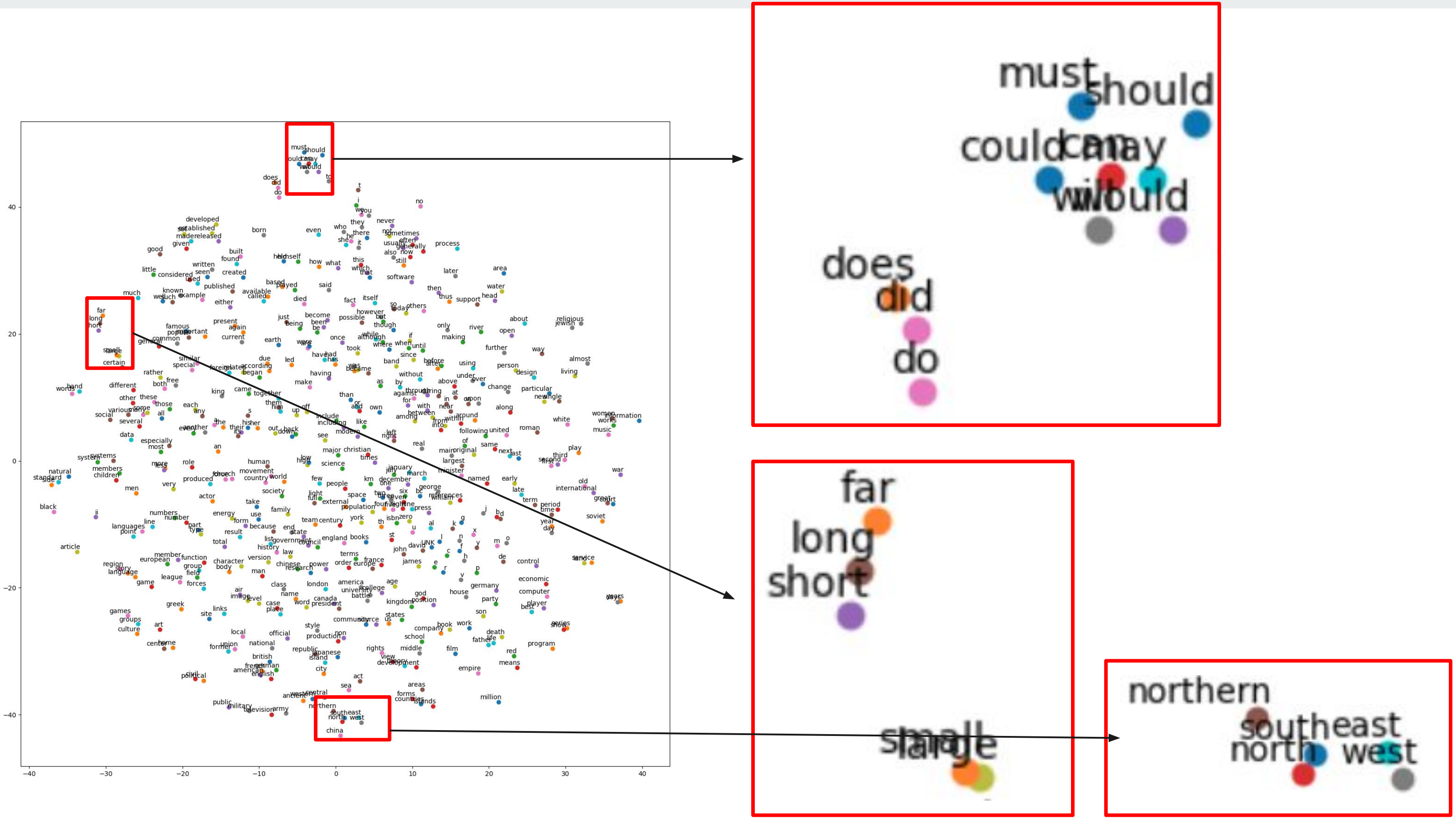
1. CBOW(continuous bag-of-words)



2. Word embedding

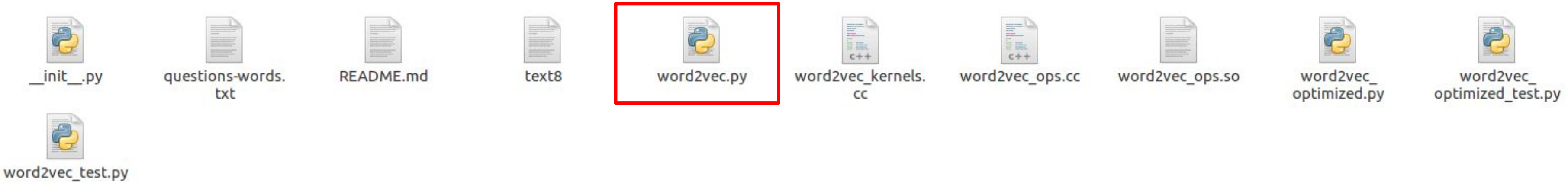
1) word2vec_basic.py

```
Average loss at step  94000 :  4.73037776172
Average loss at step  96000 :  4.68643574178
Average loss at step  98000 :  4.5888744781
Average loss at step  100000 :  4.69794555545
Nearest to people: aba, morton, microcebus, learners, mitral, faber, alkenes, concentration,
Nearest to known: used, such, amphibian, annexation, regarded, butlerian, albedo, philia,
Nearest to s: his, ursus, circ, callithrix, abet, disrespect, and, eclipse,
Nearest to there: they, it, he, often, dasyprocta, pulau, generally, which,
Nearest to at: in, ursus, on, microcebus, symbolically, during, through, under,
Nearest to called: hbox, available, cays, still, agouti, accountant, intermediates, exhibition,
Nearest to eight: seven, six, nine, five, four, three, zero, dasyprocta,
Nearest to is: was, has, are, corum, operatorname, microcebus, were, be,
Nearest to his: their, her, its, the, our, s, my, plow,
Nearest to state: basins, thaler, amalthea, thibetanus, cuauht, albury, wesleyan, golf,
Nearest to also: often, which, still, now, usually, numa, circ, abet,
Nearest to or: and, agouti, joram, dasyprocta, ursus, iit, abet, operatorname,
Nearest to however: but, that, ursus, although, while, pulau, flightless, when,
Nearest to on: in, at, iit, microcebus, dasyprocta, through, upon, within,
Nearest to b: d, j, appealed, jati, c, UNK, dist, six,
Nearest to united: shocked, study, canonization, frenchman, schott, rab, thaler, asset,
```



40
20
0
−20
−40
−40
−30
−20
−10
0
10
20
30
40
must
should
could
can
may
would
will
does
did
do
far
long
short
small
large
northern
southeast
north
west

2) word2vec.py



3. Language modeling

3. Language modeling

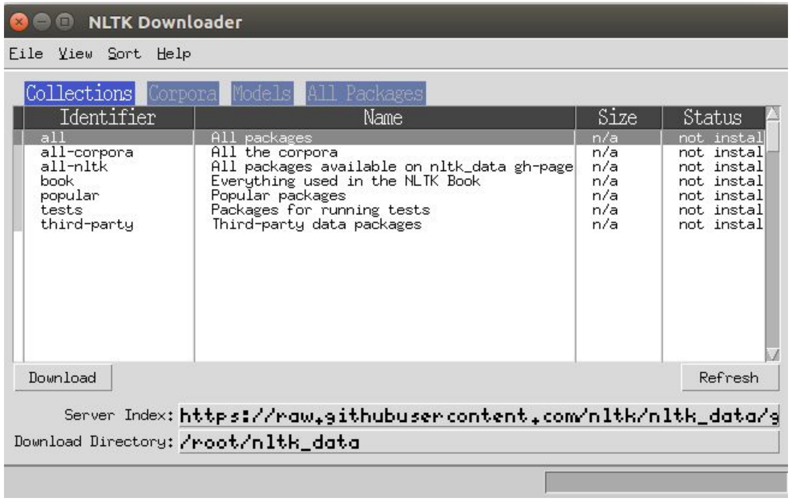
```
python ptb_word_lm.py --data_path=data/ --model=small

Epoch: 1 Learning rate: 1.000
0.004 perplexity: 6148.011 speed: 15964 wp
0.104 perplexity: 853.158 speed: 23224 wps
0.204 perplexity: 631.373 speed: 23183 wps
0.304 perplexity: 508.933 speed: 23177 wps
0.404 perplexity: 438.098 speed: 23419 wps
0.504 perplexity: 392.174 speed: 23236 wps
0.604 perplexity: 352.904 speed: 24455 wps
0.703 perplexity: 325.860 speed: 25081 wps
0.803 perplexity: 304.622 speed: 25263 wps
0.903 perplexity: 285.090 speed: 25825 wps
Epoch: 1 Train Perplexity: 270.414
Epoch: 1 Valid Perplexity: 179.309
Epoch: 2 Learning rate: 1.000
...
Epoch: 13 Learning rate: 0.002
0.004 perplexity: 61.191 speed: 25624 wps
0.104 perplexity: 45.241 speed: 23831 wps
0.204 perplexity: 49.592 speed: 24125 wps
0.304 perplexity: 47.642 speed: 24101 wps
0.404 perplexity: 46.957 speed: 24120 wps
0.504 perplexity: 46.228 speed: 24293 wps
0.604 perplexity: 44.738 speed: 24508 wps
0.703 perplexity: 44.112 speed: 24780 wps
0.803 perplexity: 43.367 speed: 25272 wps
0.903 perplexity: 41.892 speed: 25653 wps
Epoch: 13 Train Perplexity: 41.011
Epoch: 13 Valid Perplexity: 119.407
Test Perplexity: 114.413
```

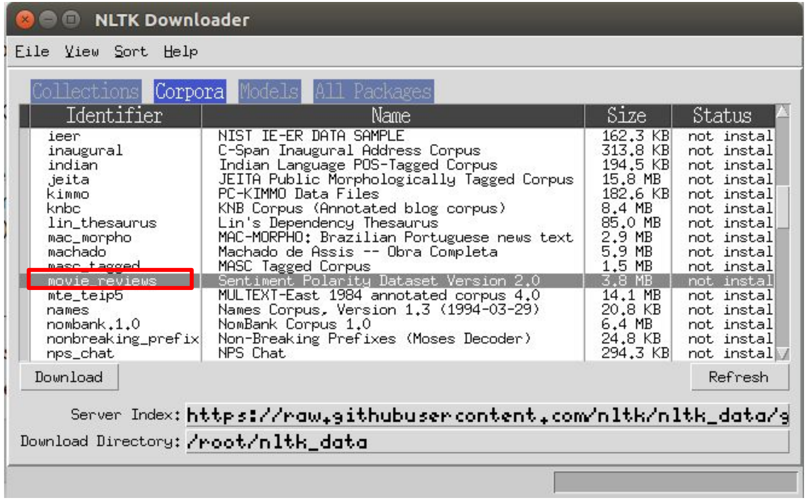
```
python ptb_word_lm.py --data_path=data/ --model=medium

Epoch: 1 Learning rate: 1.000
0.008 perplexity: 4955.667 speed: 9588 wps
0.107 perplexity: 1185.953 speed: 14875 wps
0.206 perplexity: 863.470 speed: 14998 wps
0.306 perplexity: 696.369 speed: 15161 wps
0.405 perplexity: 598.184 speed: 15206 wps
0.505 perplexity: 531.112 speed: 15212 wps
0.604 perplexity: 475.921 speed: 15293 wps
0.704 perplexity: 436.665 speed: 15303 wps
0.803 perplexity: 406.184 speed: 15353 wps
0.903 perplexity: 379.659 speed: 15358 wps
Epoch: 1 Train Perplexity: 359.130
Epoch: 1 Valid Perplexity: 205.216
Epoch: 2 Learning rate: 1.000
...
0.505 perplexity: 47.749 speed: 15410 wps
0.604 perplexity: 46.911 speed: 15429 wps
0.704 perplexity: 46.890 speed: 15439 wps
0.803 perplexity: 46.806 speed: 15444 wps
0.903 perplexity: 45.932 speed: 15421 wps
Epoch: 39 Train Perplexity: 45.602
Epoch: 39 Valid Perplexity: 88.045
Test Perplexity: 84.078
```

```
python ptb_word_lm.py --data_path=data/ --model=large

Epoch: 1 Learning rate: 1.000
0.008 perplexity: 304959.314 speed: 3878 wps
0.107 perplexity: 37387.005 speed: 4663 wps
0.206 perplexity: 8664.673 speed: 4651 wps
0.306 perplexity: 5015.499 speed: 4678 wps
0.405 perplexity: 3734.066 speed: 4646 wps
0.505 perplexity: 3150.945 speed: 4636 wps
0.604 perplexity: 2782.587 speed: 4638 wps
0.704 perplexity: 2551.124 speed: 4644 wps
0.803 perplexity: 2387.287 speed: 4650 wps
0.903 perplexity: 2263.204 speed: 4649 wps
Epoch: 1 Train Perplexity: 2161.156
Epoch: 1 Valid Perplexity: 1384.741
...
Epoch: 54 Train Perplexity: 47.415
Epoch: 54 Valid Perplexity: 87.228
Epoch: 55 Learning rate: 0.003
0.008 perplexity: 59.217 speed: 4796 wps
0.107 perplexity: 45.891 speed: 4818 wps
0.206 perplexity: 50.876 speed: 4819 wps
0.306 perplexity: 49.335 speed: 4819 wps
0.405 perplexity: 49.370 speed: 4817 wps
0.505 perplexity: 49.137 speed: 4818 wps
0.604 perplexity: 48.368 speed: 4817 wps
0.704 perplexity: 48.452 speed: 4818 wps
0.803 perplexity: 48.401 speed: 4817 wps
0.903 perplexity: 47.558 speed: 4817 wps
Epoch: 55 Train Perplexity: 47.276
Epoch: 55 Valid Perplexity: 87.181
Test Perplexity: 82.849
```

4. genism

NLTK Downloader





4. genism

Jupyter notebook