Group 8 Midterm Project : word2vec

Anton Eklund, Eric Wang, Donovan Thaing, Kevin Xu, Clément Veyssière

April 30, 2019

1 Written : Understanding word2vec

$$P(O = o|C = c) = \frac{e^{\mathbf{u}_o^T \mathbf{v_c}}}{\sum_w e^{\mathbf{u}_w^T \mathbf{v}_c}} \tag{1}$$

$$\mathbf{J}_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = -logP(O = o|C = c) \tag{2}$$

a. Since $y_w = 0, \forall w \in Vocab \setminus \{o\}$ and $y_o = 1$:

$$-\sum_{w \in Vocab} y_w log(\hat{y}_w) = -\sum_{w \in Vocab \setminus \{o\}} y_w log(\hat{y}_w) - y_o log(\hat{y}_o) = 0 - log(\hat{y}_o) \tag{3}$$

b. $\frac{\partial}{\partial v_c} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = \frac{\partial}{\partial v_c}(-logP(O = o|C = c))$
$\frac{\partial}{\partial v_c} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = \frac{\partial}{\partial v_c}(-u_o^T v_c + log(\sum_z e^{u_z^T v_c}))$
$\frac{\partial}{\partial v_c} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = -u_o + \frac{\sum_z u_z e^{u_z^T v_c}}{\sum_w e^{u_w^T v_c}}$
$\frac{\partial}{\partial v_c} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = -u_o + \sum_z u_z P(O = z|C = c)$
$\frac{\partial}{\partial v_c} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = -u_o + \sum_z u_z \hat{y}_z$
$\frac{\partial}{\partial v_c} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = U(\hat{y} - y)$

c. Case w = o :
$\frac{\partial}{\partial u_o} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = \frac{\partial}{\partial u_o}(-logP(O = o|C = c))$
$\frac{\partial}{\partial u_o} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = \frac{\partial}{\partial u_o}(-u_o^T v_c + log(\sum_z e^{u_z^T v_c}))$
$\frac{\partial}{\partial u_o} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = -v_c + v_c \frac{e^{u_0^T v_c}}{\sum_z e^{u_z^T v_c}}$
$\frac{\partial}{\partial u_o} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = -v_c + v_c P(O = o|C = c)$
$\frac{\partial}{\partial u_o} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = v_c(\hat{y}_o - y_o)$

Case w $\neq$ o :
$\frac{\partial}{\partial u_w} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = \frac{\partial}{\partial u_w}(-logP(O = o|C = c))$
$\frac{\partial}{\partial u_w} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = \frac{\partial}{\partial u_w}(-u_o^T v_c + log(\sum_z e^{u_z^T v_c}))$
$\frac{\partial}{\partial u_w} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = v_c \frac{e^{u_w^T v_c}}{\sum_z e^{u_z^T v_c}}$
$\frac{\partial}{\partial u_w} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = v_c P(O = w|C = c)$
$\frac{\partial}{\partial u_w} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = v_c \hat{y}_w$

Thus, for every word w $\in$ Vocab, we have $\frac{\partial}{\partial u_w} J_{naive-softmax}(\mathbf{v}_c, o, \mathbf{U}) = v_c(\hat{y}_w - y_w)$

d. $\sigma(x) = \frac{e^x}{e^x+1}$

$\sigma'(x) = \frac{e^x(e^x+1)-e^x e^x}{(e^x+1)^2} = \frac{e^x}{(e^x+1)^2}$

e. (i) $\frac{\partial}{\partial \mathbf{U}} J(v_c, w_{t-m}, ..., w_{t+m}, \mathbf{U}) = \sum_{\substack{-m \leq j \leq m \\ j \neq o}} \frac{\partial}{\partial \mathbf{U}} J(v_c, w_{t+j}, \mathbf{U})$

(ii) $\frac{\partial}{\partial \mathbf{v_c}} J(v_c, w_{t-m}, ..., w_{t+m}, \mathbf{U}) = \sum_{\substack{-m \leq j \leq m \\ j \neq o}} \frac{\partial}{\partial \mathbf{v_c}} J(v_c, w_{t+j}, \mathbf{U})$

(iii) $\frac{\partial}{\partial \mathbf{v_w}} J(v_c, w_{t-m}, ..., w_{t+m}, \mathbf{U}) = \sum_{\substack{-m \leq j \leq m \\ j \neq o}} \frac{\partial}{\partial \mathbf{v_w}} J(v_c, w_{t+j}, \mathbf{U}) = O$

2 Coding : Implementing word2vec

(a).1 The sigmoid function is relatively easy to implement.

(a).2 Many auxiliaries functions were written first to ease the clarity of the code. For example the partial derivative with respect to its arguments. Every function was checked in the form of the result (size of the matrix, size of the vector) and judged by the value it is comprised of (if the values were not extravagant).

Our stochastic gradient descent (SGD in short) uses mini-batches. One mini-batch is composed of pairs [center word, outside word] where center word is always the same for one batch. The SGD needs attention on 3 elements :

- Initalisation of the the set of parameters θ
- Choice of the learning rate α
- Computing the gradient of the cost function J

θ is a matrix of 2*n rows and n columns where n is the size of the vocabulary. It concatenates the matrix of centers V and the matrix of probability of the outside words U both of size n*n whence the '2'. To initialize this matrix, we randomly affected a value between 0 and 1 to all the elements of the matrix.

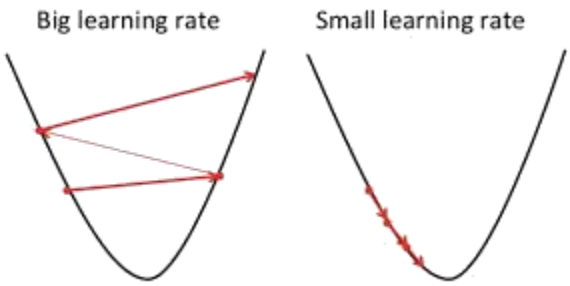

Figure 1: α big vs small

α is not to be too low, else the algorithm will progress very slowly. It cannot be too high neither because of the threat of a divergent algorithm not attaining the local minimum.

Finally, we compute the gradient of the cost function J easily thanks to the theoretical pre-work in the first part. More precisely, the gradient is the sum of the partial derivative which is equal to a sum of matrix product of U, V, $\hat{y}$ and y. Once these four matrix/vector are calculated, the gradient is mostly done.

When we first ran our code, we obtained an math overflow error due to the computation of $\hat{y}_w = \frac{e^{\mathbf{u}_w^T \mathbf{v_c}}}{\sum_z e^{\mathbf{u}_z^T \mathbf{v}_c}}$

Obviously the product $u_w^T v_c$ became too big to compute the exponential.

So we decided to compute $\frac{e^{\mathbf{u}_w^T \mathbf{v_c} - max(\mathbf{Uv_c})}}{\sum_z e^{\mathbf{u}_z^T \mathbf{v}_c - max(\mathbf{Uv_c})}} = \frac{e^{\mathbf{u}_w^T \mathbf{v_c}} e^{-max(\mathbf{Uv_c})}}{\sum_z e^{\mathbf{u}_z^T \mathbf{v}_c} e^{-max(\mathbf{Uv_c})}} = \hat{y}_w$

Thus, all the values inside the exponentials are inferior or equal to 0 and the computation is possible.

(b)

- We have decided to crawl Korean newspaper on the website https://kr.investing.com/news/most-popular-news, choosing several articles. The corpus is processed to remove all unwanted characters like symbols [!, ?, *, ;, etc.],filtering out digits and Korean particles like [는,은,과,이]]. After removing all symbols the text is split into sentences represented as a list and with each word as an entry in the list. A dictionary is created for each unique word in the whole corpus which makes it possible to represent words as numbers. With these sentences, a training batch can be generated by iterating through all words in all sentences and saving the center word together with its outside word for the whole window size. E.g. "The King sits on the throne" gives the batch [["King", "The"], ["King", "sits"], ...]. The words are here represented as integers.

  When we plot the word vectors, there is an encoding issue.

- Korean words appeared as rectangles. This issue has been fixed thanks to a .ttf file.

Illustrations are constructed thanks to pyplot, using a PCA with two components.

The closer two words are on the graph, the more chance they have to be used next to each other in a sentence. Inversely, the further two words are, the less chance they are close in a sentence. The plots seem to scatter words and group them together as intended for the assignment. However, no one in the group has sufficient knowledge in Korean to be able to determine whether these results are good or bad groupings. Following are the main plot, the 'head' of the shape, its 'body' and a dense area zoomed.

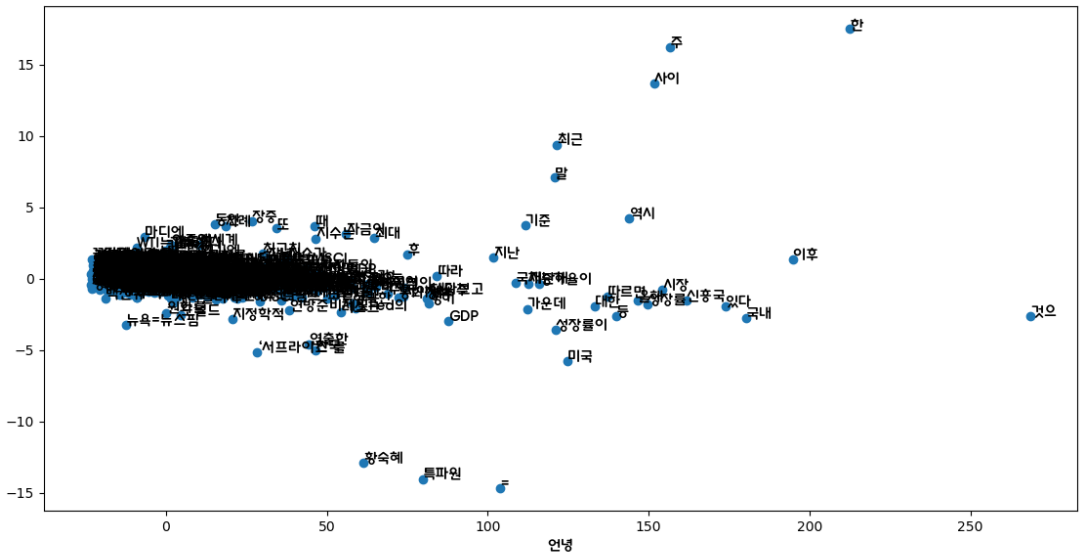

Figure 2: Plot of words in Korean

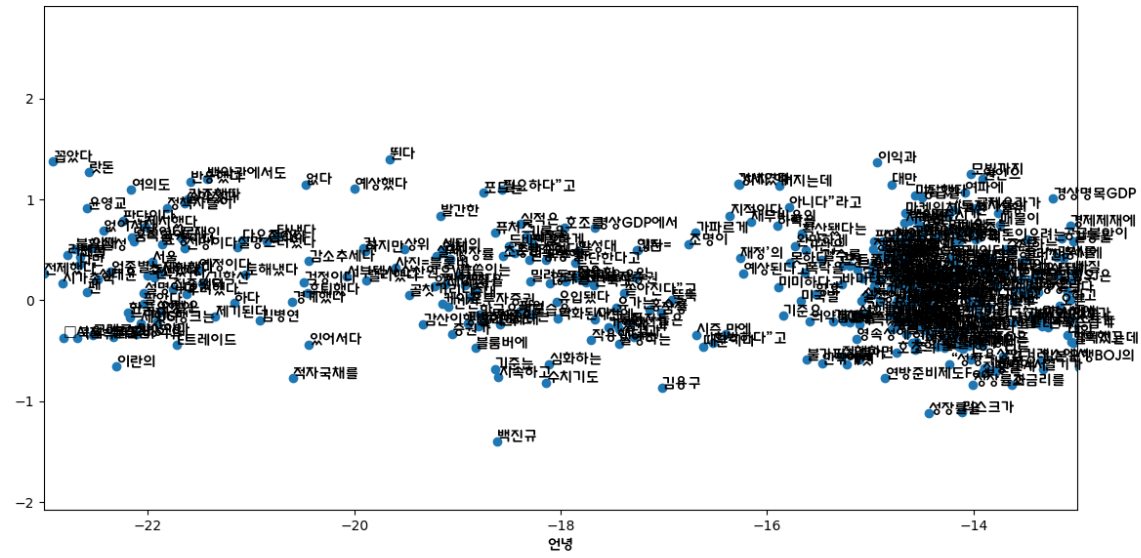

Figure 3: Head of the shape

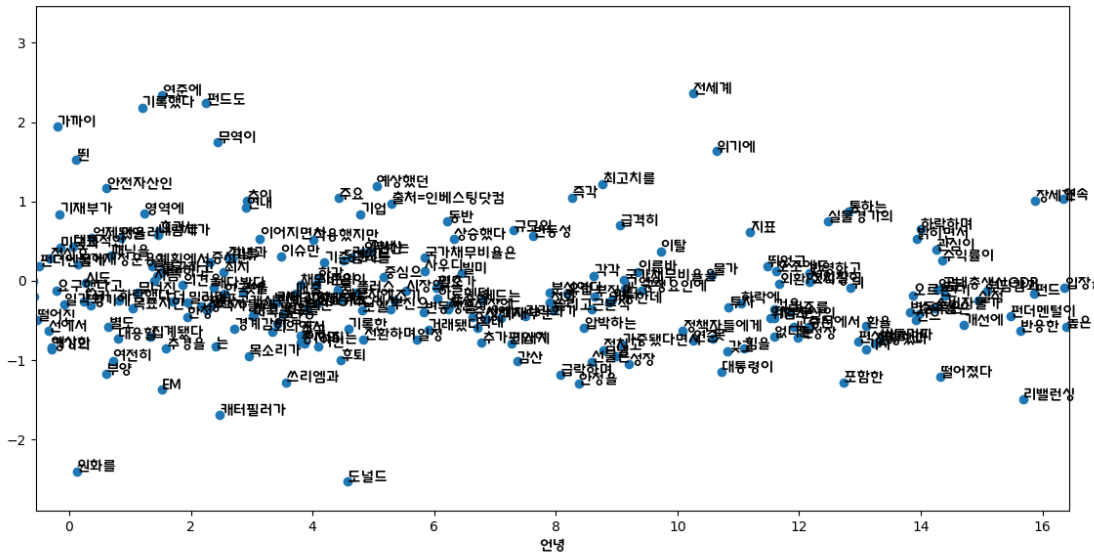

Figure 4: The body

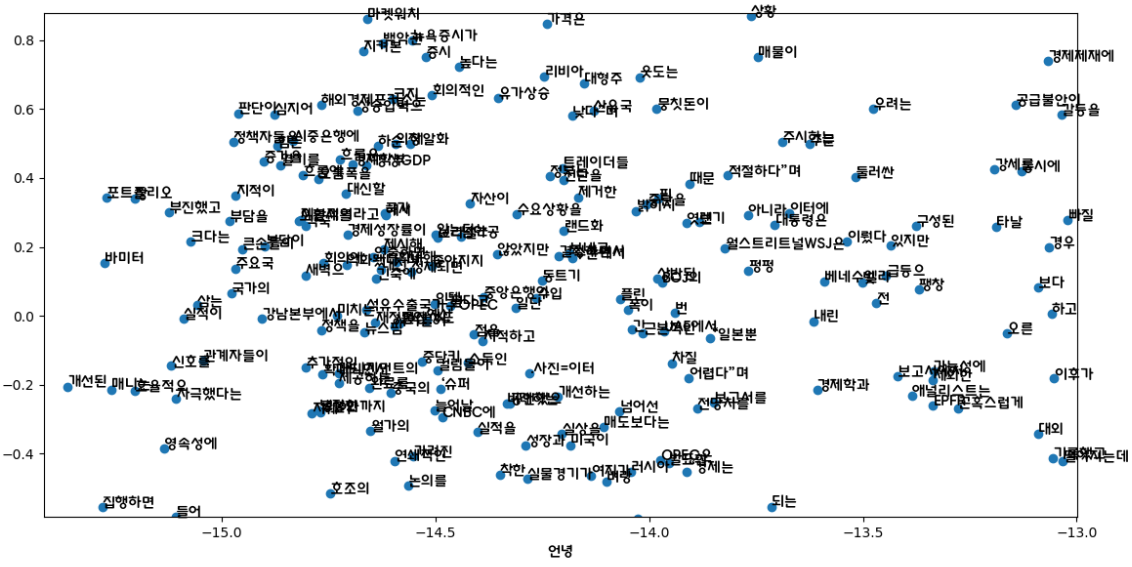

Figure 5: The dense area