CS 224n Assignment #2: word2vec (43 Points)

Due on Tuesday Jan. 21, 2020 by **4:30pm (before class)**

1 Written: Understanding word2vec (23 points)

Let's have a quick refresher on the `word2vec` algorithm. The key insight behind `word2vec` is that *'a word is known by the company it keeps'*. Concretely, suppose we have a 'center' word c and a contextual window surrounding c. We shall refer to words that lie in this contextual window as 'outside words'. For example, in Figure 1 we see that the center word c is 'banking'. Since the context window size is 2, the outside words are 'turning', 'into', 'crises', and 'as'.

The goal of the skip-gram `word2vec` algorithm is to accurately learn the probability distribution $P(O|C)$. Given a specific word o and a specific word c, we want to calculate $P(O = o|C = c)$, which is the probability that word o is an 'outside' word for c, i.e., the probability that o falls within the contextual window of c.

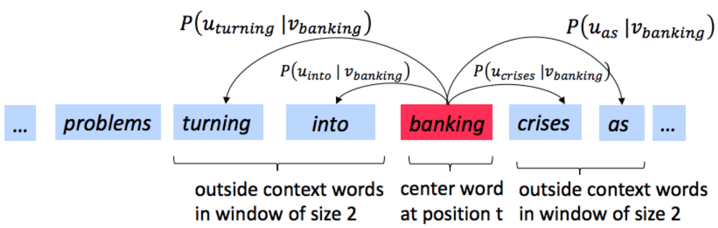


Figure 1: The word2vec skip-gram prediction model with window size 2

In `word2vec`, the conditional probability distribution is given by taking vector dot-products and applying the softmax function:

$$P(O = o \mid C = c) = \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{1}$$

= 예측값을 최대화해야
학습 잘된것

Here, $\boldsymbol{u}_o$ is the 'outside' vector representing outside word o, and $\boldsymbol{v}_c$ is the 'center' vector representing center word c. To contain these parameters, we have two matrices, $\boldsymbol{U}$ and $\boldsymbol{V}$. The columns of $\boldsymbol{U}$ are all the 'outside' vectors $\boldsymbol{u}_w$. The columns of $\boldsymbol{V}$ are all of the 'center' vectors $\boldsymbol{v}_w$. Both $\boldsymbol{U}$ and $\boldsymbol{V}$ contain a vector for every $w \in$ Vocabulary.[1]

Recall from lectures that, for a single pair of words c and o, the loss is given by:

$$\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log P(O = o|C = c). \tag{2}$$

Another way to view this loss is as the cross-entropy[2] between the true distribution $\boldsymbol{y}$ and the predicted distribution $\hat{\boldsymbol{y}}$. Here, both $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are vectors with length equal to the number of words in the vocabulary. Furthermore, the k^{th} entry in these vectors indicates the conditional probability of the k^{th} word being an 'outside word' for the given c. The true empirical distribution $\boldsymbol{y}$ is a one-hot vector with a 1 for the true outside word o, and 0 everywhere else. The predicted distribution $\hat{\boldsymbol{y}}$ is the probability distribution $P(O|C = c)$ given by our model in equation (1).

[1] Assume that every word in our vocabulary is matched to an integer number k. Bolded lowercase letters represent vectors. $\boldsymbol{u}_k$ is both the k^{th} column of $\boldsymbol{U}$ and the 'outside' word vector for the word indexed by k. $\boldsymbol{v}_k$ is both the k^{th} column of $\boldsymbol{V}$ and the 'center' word vector for the word indexed by k. **In order to simplify notation we shall interchangeably use k to refer to the word and the index-of-the-word.**

[2] The Cross Entropy Loss between the true (discrete) probability distribution p and another distribution q is $-\sum_i p_i \log(q_i)$.

$$-\sum_{w=1}^{V} y_w \log(\hat{y}_w) = -y_o \log(\hat{y}_o)$$
$$= -\log(\hat{y}_o)$$

(a) (3 points) Show that the naive-softmax loss given in Equation (2) is the same as the cross-entropy loss between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$; i.e., show that

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

Your answer should be one line. y는 원핫벡터로, y가 참인곳에서 1이고 나머지는 0임.

(b) (5 points) Compute the partial derivative of $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to $\boldsymbol{v}_c$. Please write your answer in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{U}$.

(c) (5 points) Compute the partial derivatives of $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to each of the 'outside' word vectors, $\boldsymbol{u}_w$'s. There will be two cases: when $w = o$, the true 'outside' word vector, and $w \neq o$, for all other words. Please write you answer in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{v}_c$.

(d) (3 Points) The sigmoid function is given by Equation 4:

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} \tag{4}$$

Please compute the derivative of $\sigma(x)$ with respect to x, where x is a scalar. Hint: you may want to write your answer in terms of $\sigma(x)$.

(e) (4 points) Now we shall consider the Negative Sampling loss, which is an alternative to the Naive Softmax loss. Assume that K negative samples (words) are drawn from the vocabulary. For simplicity of notation we shall refer to them as $w_1, w_2, \ldots, w_K$ and their outside vectors as $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K$. Note that $o \notin \{w_1, \ldots, w_K\}$. For a center word c and an outside word o, the negative sampling loss function is given by:

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)) \tag{5}$$

for a sample $w_1, \ldots w_K$, where $\sigma(\cdot)$ is the sigmoid function.[3]

Please repeat parts (b) and (c), computing the partial derivatives of $\boldsymbol{J}_{\text{neg-sample}}$ with respect to $\boldsymbol{v}_c$, with respect to $\boldsymbol{u}_o$, and with respect to a negative sample $\boldsymbol{u}_k$. Please write your answers in terms of the vectors $\boldsymbol{u}_o$, $\boldsymbol{v}_c$, and $\boldsymbol{u}_k$, where $k \in [1, K]$. After you've done this, describe with one sentence why this loss function is much more efficient to compute than the naive-softmax loss. Note, you should be able to use your solution to part (d) to help compute the necessary gradients here.

기존에는 단어의 확률을 모든 Vocab에 포함된 단어에 대해서 softmax를 취함. 그러나 negative sample은 정답이 아닌 단어를 몇개 선별해서 선별된 단어에 한해 확률계산을 하기때문에 이런방식같은 효과 얻어 비용을 줄임

(f) (3 points) Suppose the center word is $c = w_t$ and the context window is $[w_{t-m}, \ldots, w_{t-1}, w_t, w_{t+1}, \ldots, w_{t+m}]$, where m is the context window size. Recall that for the skip-gram version of `word2vec`, the total loss for the context window is:

$$\boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}) = \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}) \tag{6}$$

Here, $\boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$ represents an arbitrary loss term for the center word $c = w_t$ and outside word w_{t+j}. $\boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$ could be $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$ or $\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$, depending on your implementation.

Write down three partial derivatives:

[3] Note: the loss function here is the negative of what Mikolov et al. had in their original paper, because we are doing a minimization instead of maximization in our assignment code. Ultimately, this is the same objective function.

(a)

$$-\sum_{w \in vocab} y_w \log(\hat{y}_w) = -[y_1 \log(\hat{y}_1) + \cdots + y_o \log(\hat{y}_o) + \cdots y_w \log(\hat{y}_w)]$$

$$= -y_o \log(\hat{y}_o) = -\log(\hat{y}_o)$$

$$= -\log P(O=o \mid C=c)$$

y는 outside word O(오)에 대해서만 1인 원핫 벡터임.

(b)

$$J(v_c, o, U) = -\log P(O=o \mid C=c)$$

$$= -\log \frac{\exp(u_o^T v_c)}{\sum_{w \in vocab} \exp(u_w^T v_c)} = -\left(\log \exp(u_o^T v_c) - \log \sum_{w \in vocab} \exp(u_w^T v_c)\right)$$

$$= -\log \exp(u_o^T v_c) + \log \sum_{w \in vocab} \exp(u_w^T v_c)$$

$$= -u_o^T v_c + \log \sum_{w \in vocab} \exp(u_w^T v_c)$$

$$\frac{\partial J}{\partial v_c} = \frac{-u_o^T v_c}{v_c} + \frac{\sum \exp(u_w^T v_c)'}{\sum \exp(u_w^T v_c)}$$

$$\sum \exp(u_w^T v_c)' \rightarrow \frac{\left(e^{u_{w_1}^T v_c} + e^{u_{w_2}^T v_c} \cdots\right)}{\partial v_c}$$

$$= \frac{e^{u_{w_1}^T v_c}}{\partial v_c} + \frac{e^{u_{w_2}^T v_c}}{\partial v_c} \cdots$$

$$= \frac{\cancel{v_c} u_{w_1}^T \cdot e^{u_{w_1}^T v_c}}{\partial \cancel{v_c}} + \frac{u_{w_2}^T \cdot e^{u_{w_2}^T v_c}}{\partial \cancel{v_c}}$$

$$= \sum u_w \cdot e^{u_w^T v_c}$$

$$\log_a x = \frac{1}{x \ln a}$$

$$g(f(x))' = g(f(x))' \times f(x)'$$

$$= -u_o + \frac{\sum \exp(u_w^T v_c) \cdot u_w}{\sum \exp(u_w^T v_c)}$$

$$= -u_o + \sum \frac{\exp(u_w^T v_c)\, u_w}{\sum \exp(u_w^T v_c)}$$

$$= -u_o + \sum P(O=o \mid C=c) \cdot u_w$$

$$= -u_o + \sum_w \hat{y}\, u_w = U^T(\hat{y} - y)$$

(c) If. $w = o$

$$\frac{\partial J}{\partial u_w} = -v_c + \frac{\partial}{\partial u_w} \log \sum_w \exp(u_w^T v_c) = -v_c + \frac{1}{\sum_w \exp(u_w^T v_c)} \frac{\partial \sum \exp(u_w^T v_c)}{\partial u_o}$$

$$= -v_c + \frac{1}{\sum_w \exp(u_w^T v_c)} \times \frac{\partial \exp(u_o^T v_c)}{\partial u_o} = -v_c + \frac{\exp(u_o^T v_c)}{\sum_w \exp(u_w^T v_c)} \times v_c$$

w가 o이랑 같은건 1개 경우뿐!

$$= -v_c + P(O=o \mid C=c) \cdot v_c$$

$$= (P(O=o \mid C=c) - 1)\, v_c$$

$$= (\hat{y} - y)\, v_c$$

(c) If $w \neq c$ $w \neq o$ 같지 않다면, $u_o = 0$ 임

$$\frac{\partial J}{\partial u_w} = \left(-u_o v_c + \log \sum_w \exp(u_o^T v_c)\right) \times \frac{\partial}{\partial u_w}$$

$$= \frac{\partial}{\partial u_w} \log \sum_w \exp(u_o^T v_c)$$

$$= \hat{y} v_c$$

$$\partial \frac{1}{g(x)} = -\frac{g(x)'}{g(x)^2}$$

$$\frac{\partial e^x}{\partial x} = e^x \cdot x'$$

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

$$e^{-x} \cdot (-x') = e^{-x} \times -1 = -e^{-x}$$

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

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

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

$$= \sigma(x)\left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right)$$

$$= \sigma(x)\left(1 - \sigma(x)\right)$$ $\sigma(f(x)) = \sigma(f(x))(1-\sigma f(x)) f(x)$

(e) $J(v_c, o, U) = -\log(\sigma(u_o^T v_c)) - \sum_{k=1}^{K} \log(\sigma(-u_k^T v_c))$

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

$$\frac{\partial J}{\partial v_c} = -\frac{1}{\sigma(u_o^T v_c)} \times \frac{\partial \sigma(u_o^T v_c)}{\partial v_c} \times \frac{\partial u_o^T v_c}{\partial v_c} - \sum_{k=1}^{K} \frac{1}{\sigma(-u_k^T v_c)} \times \frac{\partial \sigma(-u_k^T v_c)}{\partial v_c} \times \frac{\partial -u_k^T v_c}{\partial v_c}$$

$$= -\frac{1}{\sigma(u_o^T v_c)} \times \sigma(u_o^T v_c)(1-\sigma(u_o^T v_c)) \times u_o - \sum_{k=1}^{K} \frac{1}{\sigma(-u_k^T v_c)} \times \sigma(-u_k^T v_c)(1-\sigma(-u_k^T v_c)) \times -u_k$$

$$= -(1-\sigma(u_o^T v_c))u_o - \sum_{k=1}^{K} (1-\sigma(-u_k^T v_c)) - u_k = -(1-\sigma(u_o^T v_c))u_o + \sum_{k=1}^{K} (1-\sigma(-u_k^T v_c))u_k$$

$$\frac{\partial J}{\partial u_o} = -\frac{1}{\sigma(u_o^T v_c)} \times \frac{\partial \sigma(u_o^T v_c)}{\partial u_o} \times \frac{\partial u_o^T v_c}{\partial u_o} - \sum_{k=1}^{K} \frac{\partial}{\partial u_o} \log(\sigma(-u_k^T v_c))$$

→ negative sample은 $\{w_1, w_2 \cdots w_k\} \not\ni o$ (outside)

그러므로 모든 k에 대해서 0(영)임!

$$= -\frac{1}{\sigma(u_o^T v_c)} \times \sigma(u_o^T v_c)(1-\sigma(u_o^T v_c)) \times v_c$$

$$= -(1-\sigma(u_o^T v_c))v_c$$

$(-\log \sigma(u_o^T v_c))$는 u_k에 대하여

미분시 상수취급하므로 0임

$$\frac{\partial J}{\partial u_k} = -\frac{1}{\sigma(u_o^T v_c)} \times \frac{\partial \sigma(u_o^T v_c)}{\partial u_k} \times \frac{\partial \sigma(u_o^T v_c)}{\partial u_k} - \sum_{k=1}^{K} \frac{\partial}{\partial u_k} \log(\sigma(-u_k^T v_c))$$

$$= -\sum_{k=1}^{K} \frac{1}{\sigma(-u_k^T v_c)} \times \frac{\partial \sigma(-u_k^T v_c)}{\partial u_k} \times \frac{\partial(-u_k^T v_c)}{\partial u_k}$$

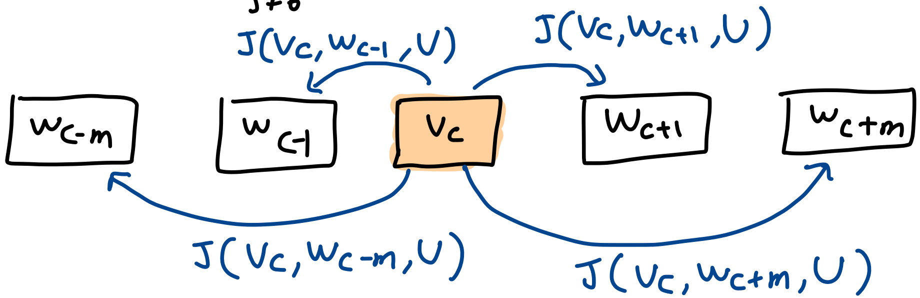
$$= -\sum_{k=1}^{K} \frac{1}{\sigma(-u_k^T v_c)} \times \sigma(-u_k^T v_c)(1-\sigma(-u_k^T v_c)) \times -v_c$$

$$= \sum_{k=1}^{K} 1-\sigma(-u_k^T v_c) \times v_c$$

(f)

$$\frac{\partial J}{\partial U} = \frac{\partial}{\partial U} \sum_{\substack{-m \le j \le m \\ j \neq 0}} J(v_c, w_{t+j}, U)$$

$$\frac{\partial J}{\partial v_c} = \frac{\partial}{\partial v_c} \sum_{\substack{-m \le j \le m \\ j \neq 0}} J(v_c, w_{t+j}, U)$$



(i) $\partial \boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U})/\partial \boldsymbol{U}$

(ii) $\partial \boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U})/\partial \boldsymbol{v}_c$

(iii) $\partial \boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U})/\partial \boldsymbol{v}_w$ when $w \neq c$

Write your answers in terms of $\partial \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})/\partial \boldsymbol{U}$ and $\partial \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})/\partial \boldsymbol{v}_c$. This is very simple – each solution should be one line.

Once you're done: *Given that you computed the derivatives of $\boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$ with respect to all the model parameters $\boldsymbol{U}$ and $\boldsymbol{V}$ in parts (a) to (c), you have now computed the derivatives of the full loss function $\boldsymbol{J}_{\textit{skip-gram}}$ with respect to all parameters. You're ready to implement `word2vec`!*

2 Coding: Implementing word2vec (20 points)

In this part you will implement the word2vec model and train your own word vectors with stochastic gradient descent (SGD). Before you begin, first run the following commands within the assignment directory in order to create the appropriate conda virtual environment. This guarantees that you have all the necessary packages to complete the assignment. Also note that you probably want to finish the previous math section before writing the code since you will be asked to implement the math functions in Python. You want to implement and test the following subsections in order since they are accumulative.

```
conda env create -f env.yml
conda activate a2
```

Once you are done with the assignment you can deactivate this environment by running:

```
conda deactivate
```

For each of the methods you need to implement, we included approximately how many lines of code our solution has in the code comments. These numbers are included to guide you. You don't have to stick to them, you can write shorter or longer code as you wish. If you think your implementation is significantly longer than ours, it is a signal that there are some `numpy` methods you could utilize to make your code both shorter and faster. `for` loops in Python take a long time to complete when used over large arrays, so we expect you to utilize `numpy` methods. We will be checking the efficiency of your code. You will be able to see the results of the autograder when you submit your code to `Gradescope`.

(a) (12 points) We will start by implementing methods in `word2vec.py`. First, implement the `sigmoid` method, which takes in a vector and applies the sigmoid function to it. Then implement the softmax loss and gradient in the `naiveSoftmaxLossAndGradient` method, and negative sampling loss and gradient in the `negSamplingLossAndGradient` method. Finally, fill in the implementation for the skip-gram model in the `skipgram` method. When you are done, test your implementation by running `python word2vec.py`.

(b) (4 points) Complete the implementation for your SGD optimizer in the `sgd` method of `sgd.py`. Test your implementation by running `python sgd.py`.

(c) (4 points) Show time! Now we are going to load some real data and train word vectors with everything you just implemented! We are going to use the Stanford Sentiment Treebank (SST) dataset to train word vectors, and later apply them to a simple sentiment analysis task. You will need to fetch the datasets first. To do this, run `sh get_datasets.sh`. There is no additional code to write for this part; just run `python run.py`.

Note: The training process may take a long time depending on the efficiency of your implementation and the compute power of your machine ***(an efficient implementation takes one to two hours)****. Plan accordingly!*

After 40,000 iterations, the script will finish and a visualization for your word vectors will appear. It will also be saved as `word_vectors.png` in your project directory. **Include the plot in your homework write up.** Briefly explain in at most three sentences what you see in the plot.

3 Submission Instructions

You shall submit this assignment on GradeScope as two submissions – one for "Assignment 2 [coding]" and another for 'Assignment 2 [written]":

(a) Run the `collect_submission.sh` script to produce your `assignment2.zip` file.

(b) Upload your `assignment2.zip` file to GradeScope to "Assignment 2 [coding]".

(c) Upload your written solutions to GradeScope to "Assignment 2 [written]".