XCS224N Assignment #2: word2vec (20 Points + 5 Point Extra Credit Challenge)

1 Understanding word2vec (Refresher)

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.

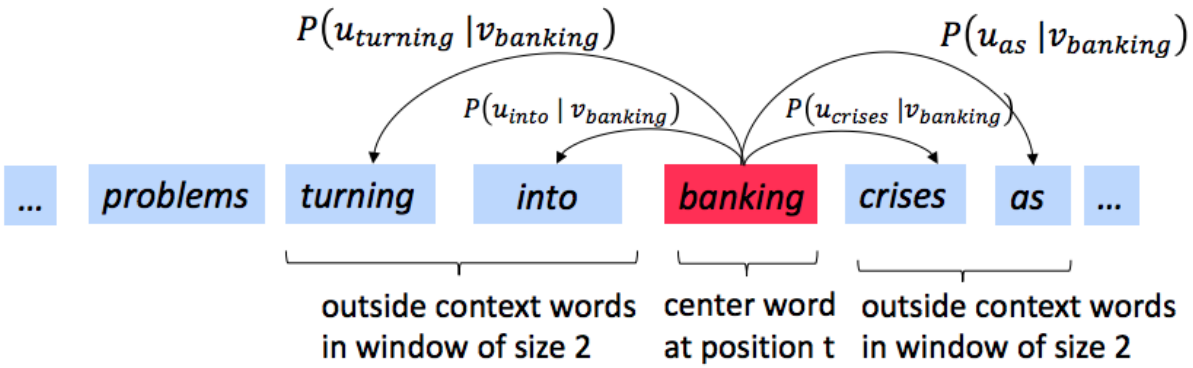

Figure 1: The word2vec skip-gram prediction model with window size 2

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

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

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]

Both U and V are (Word Embedding Dim x Vocab Size)

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

[1] Assume that every word in our vocabulary is matched to an integer number k. $\boldsymbol{u}_k$ 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 $-\sum_i p_i \log(q_i)$.

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). There are 2 optional questions below which you may attempt (*Note: Bonus points will be awarded for the optional assignments*)

2 Extra Credit Challenge I (2.5 Points)

The partial derivative of $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to $\boldsymbol{v}_c$ in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{U}$ is given below:

$$\frac{\partial J}{\partial \boldsymbol{v}_c} = \boldsymbol{U}(\hat{\boldsymbol{y}} - \boldsymbol{y}) \tag{3}$$

or equivalently,

$$\frac{\partial J}{\partial \boldsymbol{v}_c} = -\boldsymbol{u}_o + \sum_{w=1}^{V} \hat{y}_w \boldsymbol{u}_w \tag{4}$$

The naive-softmax loss given in Equation (2) is the same as the cross-entropy loss between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$; This is equivalent to:

$$-\sum_{w \in Vocab} y_w \log(\hat{y}_w) = -\log(\hat{y}_o). \tag{5}$$

Reasoning: Since $\boldsymbol{y}$ is a one-hot vector, all $\boldsymbol{y}_k = 0$ where $k \neq o$. $\boldsymbol{y}_o = 1$, so we are left with $-\log(\hat{\boldsymbol{y}}_o)$.

Write the steps to arrive at equation 3 or 4, the partial derivative of $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to $\boldsymbol{v}_c$, starting from equation 5. Please write your answer in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{U}$. The first few steps have been provided below (loss function $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$) and the rest of the proof may take 4 or 5 steps.

$$\begin{aligned}
J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) &= -\log(\hat{y}_o) && \text{(refer to equation 5)} \\
&= -\log\left(\frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}\right) \\
&= -\left(\log\left(\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\right) - \log\left(\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)\right)\right) \\
&= -\boldsymbol{u}_o^\top \boldsymbol{v}_c + \log\left(\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)\right)
\end{aligned}$$

3 Extra Credit Challenge II (2.5 Points)

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 is given below:

$$\frac{\partial J}{\partial \boldsymbol{U}} = \boldsymbol{v}_c(\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \tag{6}$$

or equivalently:

$$\frac{\partial J}{\partial \boldsymbol{u}_w} = \begin{cases} (\hat{y}_w - 1)\boldsymbol{v}_c & \text{if } w = o \\ \hat{y}_w \boldsymbol{v}_c & \text{otherwise} \end{cases} \tag{7}$$

Write the steps required to arrive at the partial derivative 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 are two cases you need to consider: 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 $\boldsymbol{v}_c$. The proof may take 4 or 5 steps. The loss function $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ is:

$$J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\boldsymbol{u}_o^\top \boldsymbol{v}_c + \log\left(\sum_{w' \in \text{Vocab}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_c)\right)$$

4 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 virtual environment. This guarantees that you have all the necessary packages to complete the assignment. Make sure to take a look at the equations presented in section 2 and 3, especially **equations (3) for gradient (of loss function $J_{\textbf{naive-softmax}}(v_c, o, U)$) with respect to v_c and (6) for gradient with respect to each of the 'outside' word vectors, u_w's** which we'll be using in our gradient calculation code

```
source venv/bin/activate
```

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

```
deactivate
```

(a) (12 points) First, implement the `sigmoid` function in `word2vec.py` to apply the sigmoid function to an input vector. In the same file, fill in the implementation for the naive softmax loss and gradient function. Then, fill in the implementation of the loss and gradient functions for the skip-gram model. When you are done, test your implementation by running `python word2vec.py`.
To complete these steps you will need to refer to Sections 2 and 3, Equations (3) and (6).

Pseudo code for skip-gram

Algorithm 1 Skipgram

procedure SKIPGRAM($outsideWords, lossAndGradientFunc, **kwargs$)
  $loss \leftarrow 0$
  $gradCenter \leftarrow np.zeros(*args)$
  $gradOutside \leftarrow np.zeros(*args)$
  for `word in outsideWords` **do** ▷ Iterate over outside words
    $lossCurrent, gradc, grado \leftarrow lossAndGradientFunc(**kwargs)$
    $loss \leftarrow loss + lossCurrent$ ▷ Loss is accumulated
    $gradCenter \leftarrow gradCenter + gradc$
    $gradOutside \leftarrow gradOutside + grado$
  end for
  return $loss, gradCenter, gradOutside$ ▷ Return loss and gradients
end procedure

(b) (5 points) Complete the implementation for your SGD optimizer in `sgd.py`. Test your implementation by running `python sgd.py`.

(c) (3 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. There is no additional code to write for this part; just run `python run.py`.

*Note: **Please** do not use external python modules which are not specified in the `requirements.txt` file. This will cause the `autograder` to fail.*

*Note: The training process may take a long time depending on the efficiency of your implementation **(an efficient implementation takes approximately an hour)**. Plan accordingly!*

*Additional Note: Although you have implemented the naiveSoftmaxLossAndGradient function, to make sure that the code converges faster do not forget to switch the loss function to negSamplingLossAndGradient (the implementation for this has been provided in **word2vec.py**) in the function call for **sgd(**kwargs)** in the file **run.py***

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` and the corresponding wordvectors as `sampleVectors.json` in your project directory.

Submission Instructions

1. **Please** do not use external python modules which are not specified in the `requirements.txt` file. This will cause the `autograder` to fail.
2. Run the `collect_submission.sh` script to produce your `assignment2.zip` file.
3. Upload your `assignment2.zip` file via the **Assignment 2 Coding Submission Link** on the Assignment 2 page of your SCPD learning portal.
4. If you completed the extra credit derivation prompts, upload your responses as a PDF via the **Assignment 2 Extra Credit Submission Link** on the Assignment 2 page of your SCPD learning portal.

5 Appendix

Negative sampling is briefly introduced in Lecture 2: Word2Vec: Model Variants and an implementation is provided in the Assignment 2 coding assignment. For detailed notes on the math behind negative sampling, see below.

1. Negative Sampling loss 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:

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)) \tag{8}$$

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

Below we compute 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 $\boldsymbol{u}_k$.

Solution: Firstly:

$$\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{v}_c} &= -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \times \frac{\partial}{\partial \boldsymbol{v}_c}\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - \sum_{k=1}^{K} \frac{1}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} \times \frac{\partial}{\partial \boldsymbol{v}_c}\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c) && \text{(chain rule on log)} \\
&= -\frac{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\boldsymbol{u}_o}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} - \sum_{k=1}^{K} -\frac{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)(1-\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\boldsymbol{u}_k}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} && \text{(chain rule on } \sigma\text{)} \\
&= -(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\boldsymbol{u}_o - \sum_{k=1}^{K} -(1-\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\boldsymbol{u}_k && \text{(cancel)} \\
&= (\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)-1)\boldsymbol{u}_o - \sum_{k=1}^{K} (\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)-1)\boldsymbol{u}_k && \text{(rearrange)}
\end{aligned}$$

Secondly:

$$\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{u}_o} &= \frac{\partial}{\partial \boldsymbol{u}_o}\Big(-\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\Big) \\
&= -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \times \frac{\partial}{\partial \boldsymbol{u}_o}\Big(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\Big) && \text{(chain rule on log)} \\
&= -\frac{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\boldsymbol{v}_c}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} && \text{(chain rule on } \sigma\text{)} \\
&= -(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\boldsymbol{v}_c && \text{(cancel)} \\
&= (\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)-1)\boldsymbol{v}_c && \text{(rearrange)}
\end{aligned}$$

[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.

Thirdly, for all $k = 1, 2, \ldots, K$:

$$\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{u}_k} &= \frac{\partial}{\partial \boldsymbol{u}_k}\Big(-\log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\Big) \\
&= \frac{1}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} \times \frac{\partial}{\partial \boldsymbol{u}_k}\Big(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\Big) && \text{(chain rule on log)} \\
&= \frac{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)(1-\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\boldsymbol{v}_c}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} && \text{(chain rule on } \sigma\text{)} \\
&= (1-\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\boldsymbol{v}_c && \text{(cancel)}
\end{aligned}$$

The naive-softmax loss contains a summation over the entire vocabulary as part of computing the $P(O = o \mid C = c)$ term. Here, we don't do that calculation, approximating it with K samples (where K is much smaller than the vocabulary size).

2. 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:

$$\boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}) = \sum_{\substack{-m \le j \le m \\ j \neq 0}} \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}) \tag{9}$$

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 the implementation.

The proofs for the three partial derivatives are given below:

(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 $w \neq c$

Note that the final derivatives of $\boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$ with respect to all the model parameters $\boldsymbol{U}$ and $\boldsymbol{V}$ are provided in the appendix part 1 derivation

Solution: Given a loss function J, we already know how to obtain the following derivatives

$$\frac{\partial J(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})}{\partial \boldsymbol{U}} \text{ and } \frac{\partial J(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})}{\partial \boldsymbol{v}_c}$$

Therefore, for skip-gram, the gradients for the loss of one context window can be expressed in terms of these:

$$\begin{aligned}
\frac{\partial J_{\text{skip-gram}}(w_{t-m} \ldots w_{t+m})}{\partial \boldsymbol{U}} &= \sum_{-m \le j \le m, j \neq 0} \frac{\partial J(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})}{\partial \boldsymbol{U}}, \\
\frac{\partial J_{\text{skip-gram}}(w_{t-m} \ldots w_{t+m})}{\partial \boldsymbol{v}_c} &= \sum_{-m \le j \le m, j \neq 0} \frac{\partial J(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})}{\partial \boldsymbol{v}_c}, \\
\frac{\partial J_{\text{skip-gram}}(w_{t-m} \ldots w_{t+m})}{\partial \boldsymbol{v}_w} &= \boldsymbol{0}, \text{when } w \neq c.
\end{aligned}$$