CS 224n Assignment #2: word2vec (44 Points)

Due on Tuesday Jan. 26, 2021 by **4:30pm (before class)**

1 Written: Understanding word2vec (26 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.

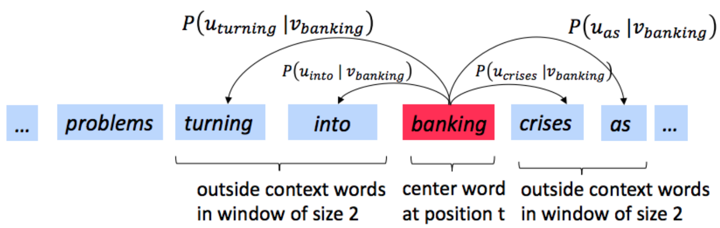

Figure 1: The word2vec skip-gram prediction model with window size 2

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

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

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:

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

We can view this loss 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 $-\sum_i p_i \log(q_i)$.

(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.

Answer: We know y is a one-hot vector, which means $y_w = 1$ when $w = o$ and $y_w = 0$ when $w \neq o$, then we could find:

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

(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 $\boldsymbol{U}$. Note that in this course, we expect your final answers to follow the shape convention.[3] This means that the partial derivative of any function $f(x)$ with respect to x should have the same shape as x. For this subpart, please present your answer in vectorized form. In particular, you may not refer to specific elements of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{U}$ in your final answer (such as $\boldsymbol{y}_1$, $\boldsymbol{y}_2$, $\ldots$).

Answer: Let's define intput vector as $\theta = U^\intercal v_c$ and prediction function be $\hat{y} = softmax(\theta)$. From the equation (7) in the course gradient notes we see,

$$\begin{aligned} J &= CrossEntropy(y, \hat{y}) \\ \frac{\partial J}{\partial \theta} &= (\hat{y} - y)^\intercal \end{aligned}$$

After applying chain rule, we could see:

$$\begin{aligned} \frac{\partial J}{\partial v_c} &= \frac{\partial J}{\partial \theta} \frac{\partial \theta}{\partial v_c} \\ &= (\hat{y} - y)^\intercal \frac{\partial U^\intercal v_c}{\partial v_c} \\ &= U(\hat{y} - y) \end{aligned}$$

[3] This allows us to efficiently minimize a function using gradient descent without worrying about reshaping or dimension mismatching. While following the shape convention, we're guaranteed that $\theta := \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$ is a well-defined update rule.

(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 your answer in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{v}_c$. In this subpart, you may use specific elements within these terms as well, such as $(\boldsymbol{y}_1, \boldsymbol{y}_2, \dots)$.

Answer: Let's denote $\theta_w = u_w^\intercal v_c$ and $w \in$ Vocab, then we could have the partial derivatives in below:

$$\begin{aligned}\frac{\partial J}{\partial u_w} &= \frac{\partial J}{\partial \theta_w}\frac{\partial \theta_w}{\partial u_w} \\ &= (\hat{y}_w - y_w)\frac{\partial u_w^\intercal v_c}{\partial u_w} \\ &= (\hat{y}_w - y_w)v_c\end{aligned}$$

Then we could see $\frac{\partial J}{\partial u_w} = (\hat{y}_o - 1)v_c$ when $w = o$, and $\frac{\partial J}{\partial u_w} = (\hat{y}_w)v_c$ when $w \neq o$.

(d) (1 point) Compute the partial derivative of $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to U. Please write your answer in terms of $\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_1}, \frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_2}, \dots, \frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_{|Vocab|}}$. The solution should be one or two lines long.

Answer: From (c), we could easily come out:

$$\begin{aligned}\frac{\partial J}{\partial U} &= \frac{\partial J}{\partial \theta_1}\frac{\partial \theta_1}{\partial u_1}, \frac{\partial J}{\partial \theta_2}\frac{\partial \theta_2}{\partial u_2} \cdots \frac{\partial J}{\partial \theta_{|Vocab|}}\frac{\partial \theta_{|Vocab|}}{\partial u_{|Vocab|}} \\ &= (\hat{y}_1 - y_1)v_c, (\hat{y}_2 - y_2)v_c \dots (\hat{y}_{|Vocab|} - y_{|Vocab|})v_c\end{aligned}$$

(e) (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 $\sigma(x)$.

Answer:

$$\begin{aligned}\frac{\partial \sigma(x)}{\partial x} &= \frac{\partial (e^{-x} + 1)^{-1}}{\partial x} \\ &= -(e^{-x} + 1)^{-2} \cdot \frac{\partial (e^{-x} + 1)}{\partial x} \\ &= -(1 + e^{-x})^{-2} \cdot (e^{-x} \cdot -1) \\ &= (1 + e^{-x})^{-2} \cdot (e^{-x}) \\ &= \frac{1}{(1 + e^{-x})} \cdot (1 - \frac{1}{(1 + e^{-x})}) \\ &= \sigma(x)(1 - \sigma(x))\end{aligned}$$

(f) (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 $w_1, w_2, \dots, w_K$ and their outside vectors as $\boldsymbol{u}_1, \dots, \boldsymbol{u}_K$. For this question, assume that the K negative samples are distinct. In other words, $i \neq j$ implies $w_i \neq w_j$ for $i, j \in \{1, \dots, K\}$. Note that $o \notin \{w_1, \dots, w_K\}$. 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 $w_1, \dots w_K$, where $\sigma(\cdot)$ is the sigmoid function.[4]

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 (e) to help compute the necessary gradients here.

Answer: Let's denote $x_o = u_o^\top v_c$, $p_{u_o} = \sigma(x_o)$, $z_k = -u_k^\top v_c$, $p_{u_k} = \sigma(z_k)$, then we could convert the negative sampling loss from (5) to

$$J_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(p_{u_o}) - \sum_{k=1}^{K} \log(p_{u_k})$$

Then we could see, for partial derivative to v_c

$$\begin{aligned}
\frac{\partial J}{\partial v_c} &= \frac{\partial J}{\partial p_{u_o}} \frac{\partial p_{u_o}}{\partial x_o} \frac{\partial x_o}{\partial v_c} + \sum_{k=1}^{K} \frac{\partial J}{\partial p_{u_k}} \frac{\partial p_{u_k}}{\partial z_k} \frac{\partial z_k}{\partial v_c} \\
&= -(1 - p_{u_o}) u_o + \sum_{k=1}^{K} (1 - p_{u_k}) u_k \\
&= -(1 - \sigma(u_o^\intercal v_c)) u_o + \sum_{k=1}^{K} (1 - \sigma(-u_k^\intercal v_c)) u_k
\end{aligned}$$

For partial derivative to u_o, u_k, let's first add a new index j to represent $u_j \in U$, similarly, we could get a general presentation on partial derivative on arbitrary vector $u_j$in U:

$$\frac{\partial J}{\partial u_j} = \frac{\partial J}{\partial p_{u_j}} \frac{\partial p_{u_j}}{\partial x_j} \frac{\partial x_j}{\partial u_j} + \sum_{k=1}^{K} \frac{\partial J}{\partial p_{u_k}} \frac{\partial p_{u_k}}{\partial z_k} \frac{\partial z_k}{\partial u_j}$$

Since we know $o \notin \{w_1, \dots, w_K\}$, with above formula and when $o = j$, we have

$$\begin{aligned}
\frac{\partial J}{\partial u_o} &= -(1 - p_{u_o}) v_c = -(1 - \sigma(u_o^\intercal v_c)) v_c \\
\frac{\partial J}{\partial u_k} &= (1 - p_{u_k}) v_c = (1 - \sigma(-u_k^\intercal v_c)) v_c, \forall k \in [1, K]
\end{aligned}$$

Answer: The loss computation of Negative Sampling is much more efficient to compute the naive-softmax loss is because it only calculates the probability of K vectors from vocabulary U, but naive-softmax loss has to compute all of them.

[4]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.

(g) (2 point) Now we will repeat the previous exercise, but without the assumption that the K sampled words are distinct. 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 $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K$. In this question, you may not assume that the words are distinct. In other words, $w_i = w_j$ may be true when $i \neq j$ is true. 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:

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)) \tag{6}$$

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

Compute the partial derivative of $\boldsymbol{J}_{\text{neg-sample}}$ with respect to a negative sample $\boldsymbol{u}_k$. Please write your answers in terms of the vectors $\boldsymbol{v}_c$ and $\boldsymbol{u}_k$, where $k \in [1, K]$. Hint: break up the sum in the loss function into two sums: a sum over all sampled words equal to u_k and a sum over all sampled words not equal to u_k.

Answer: Let's assume there is m duplicate word w_k in the K negative samples. $k \in [1, K]$

$$\begin{aligned}
\frac{\partial J}{\partial u_k} &= \frac{\partial(-\sum_{i=1}^{K} \log(p_{u_i}))}{\partial u_k} \\
&= \frac{\partial(-\sum_{i=1, w_i \neq w_k}^{K} \log(p_{u_i}) - \sum_{i=1, w_i = w_k}^{K} \log(p_{u_k}))}{\partial u_k} \\
&= m(1 - p_{u_k})v_c \\
&= m(1 - \sigma(-u_k^\intercal v_c))v_c, \forall k \in [1, K]
\end{aligned}$$

(h) (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:

$$\boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}) = \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}) \tag{7}$$

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:

(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`*!*

Answer:

$$\partial J_{skip-gram}(v_c, w_{t-m}, \cdots, w_{t+m})/\partial U = \sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \frac{J(v_c, w_{t+j}, U)}{\partial U}$$

$$\partial J_{skip-gram}(v_c, w_{t-m}, \cdots, w_{t+m})/\partial v_c = \sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \frac{J(v_c, w_{t+j}, U)}{\partial v_c}$$

$$\partial J_{skip-gram}(v_c, w_{t-m}, \cdots, w_{t+m})/\partial w_c (w \neq c) = 0$$

2 Coding: Implementing word2vec (18 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`, we recommend submitting early and often.

(a) (12 points) We will start by implementing methods in `word2vec.py`. You can test a particular method by running `python word2vec.py m` where `m` is the method you would like to test. For example, you can test the sigmoid method by running `python word2vec.py sigmoid`.

   (i) Implement the `sigmoid` method, which takes in a vector and applies the sigmoid function to it.

   (ii) Implement the softmax loss and gradient in the `naiveSoftmaxLossAndGradient` method.

   (iii) Implement the negative sampling loss and gradient in the `negSamplingLossAndGradient` method.

   (iv) Implement the skip-gram model in the `skipgram` method.

   When you are done, test your entire 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) (2 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.

Answer: Below picture visualizes list of words' center word vectors' covariance after SVD, from the image we could see how these words' similarity to each other, thinking by our new-trained word2vec matrix.

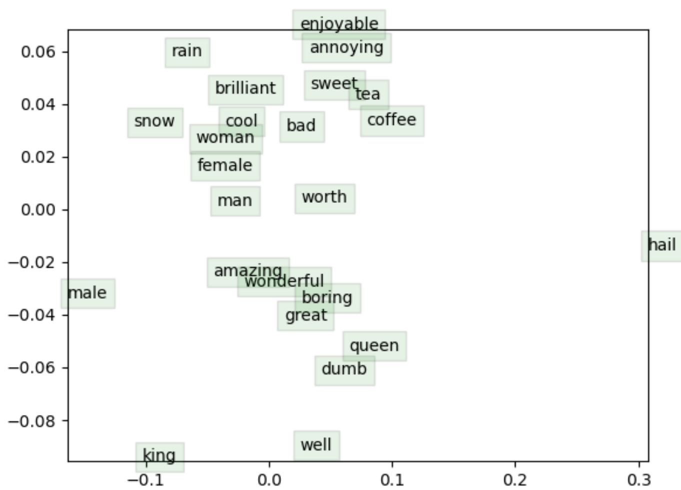

Figure 2: The word$_vectorsgraphontestwords$

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]".