CS 224n Assignment #2: word2vec (49 Points)

Due on Tuesday Jan. 24, 2023 by **4:30pm (before class)**

1 Written: Understanding word2vec (31 points)

Recall that the key insight behind `word2vec` is that *'a word is known by the company it keeps'*. Concretely, consider a 'center' word c surrounded before and after by a context of a certain length. We term words in this contextual window 'outside words' (O). For example, in Figure 1, the context window length is 2, the center word c is 'banking', and the outside words are 'turning', 'into', 'crises', and 'as':

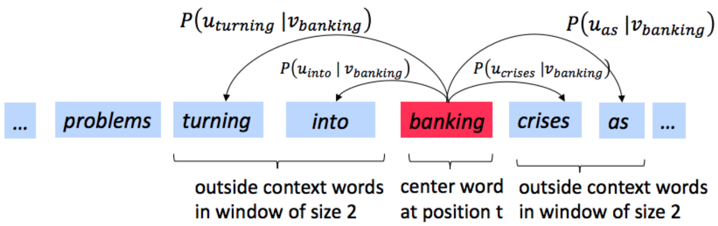

Figure 1: The word2vec skip-gram prediction model with window size 2

Skip-gram `word2vec` aims to learn the probability distribution $P(O|C)$. Specifically, given a specific word o and a specific word c, we want to predict $P(O = o|C = c)$: the probability that word o is an 'outside' word for c (i.e., that it falls within the contextual window of c). We model this probability by taking the softmax function over a series of vector dot-products:

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

For each word, we learn vectors u and v, where $\boldsymbol{u}_o$ is the 'outside' vector representing outside word o, and $\boldsymbol{v}_c$ is the 'center' vector representing center word c. We store these parameters in 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 $\hat{\boldsymbol{y}}$, for a particular center word c and a particular outside word o. 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, for this particular example of center word c and outside word o.[3] The predicted distribution $\hat{\boldsymbol{y}}$ is the probability distribution $P(O|C = c)$ given by our model in equation (1).

Note: Throughout this homework, when computing derivatives, please use the method reviewed during the lecture (i.e. no Taylor Series Approximations).

[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 word k and the index of word k.**

[2] The **cross-entropy loss** between the true (discrete) probability distribution p and another distribution q is $-\sum_i p_i \log(q_i)$.

[3] Note that the true conditional probability distribution of context words for the entire training dataset would not be one-hot.

(a) (2 points) Prove that the naive-softmax loss (Equation 2) is the same as the cross-entropy loss between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$, i.e. (note that $\boldsymbol{y}, \hat{\boldsymbol{y}}$ are vectors and $\hat{\boldsymbol{y}}_o$ is a scalar):

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

Your answer should be one line. You may describe your answer in words.

Answer: Remember from above, "The true empirical distribution y is a one-hot vector with a 1 for the true outside word o, and 0 everywhere else"
Therefore $\forall w \neq o : -0 \cdot \log(\hat{\boldsymbol{y}}_w)$ and we get $-\sum_{w \in \text{Vocab}} \boldsymbol{y}_w \log(\hat{\boldsymbol{y}}_w) = -1 \cdot \log(\hat{\boldsymbol{y}}_o) = -\log(\hat{\boldsymbol{y}}_o)$

(b) (7 points)

(i) 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 $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, $\boldsymbol{U}$, and show your work to receive full credit.

- **Note**: Your final answers for the partial derivative should follow the shape convention: the partial derivative of any function $f(x)$ with respect to x should have the **same shape** as x.[4]
- Please provide your answers for the partial derivative in vectorized form. For example, when we ask you to write your answers in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{U}$, you may not refer to specific elements of these terms in your final answer (such as $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots$).

(ii) When is the gradient you computed equal to zero?
Hint: You may wish to review and use some introductory linear algebra concepts.

(iii) The gradient you found is the difference between two terms. Provide an interpretation of how each of these terms improves the word vector when this gradient is subtracted from the word vector v_c.

(iv) In many downstream applications using word embeddings, L2 normalized vectors (e.g. $\mathbf{u}/||\mathbf{u}||_2$ where $||\mathbf{u}||_2 = \sqrt{\sum_i u_i^2}$) are used instead of their raw forms (e.g. $\mathbf{u}$). Now, suppose you would like to classify phrases as being positive or negative. When would L2 normalization take away useful information for the downstream task? When would it not? Hint: Consider the case where $\mathbf{u}_x = \alpha \mathbf{u}_y$ for some words $x \neq y$ and some scalar α.

Answer:

(i) $\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = \frac{\partial -\log P(O=o|C=c)}{\partial \boldsymbol{v}_c} = \frac{\partial -\log \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}}{\partial \boldsymbol{v}_c} = \frac{\partial -\log \exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - \log \sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}{\partial \boldsymbol{v}_c}$
$= \frac{\partial -\boldsymbol{u}_o^\top \boldsymbol{v}_c}{\partial \boldsymbol{v}_c} - \frac{\log \sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}{\partial \boldsymbol{v}_c} = -\boldsymbol{u}_o - \frac{1}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \frac{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}{\partial \boldsymbol{v}_c}$
$= -\boldsymbol{u}_o - \frac{1}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c) \boldsymbol{u}_w = \sum_{w \in \text{Vocab}} (-\boldsymbol{y}_w \boldsymbol{u}_w + \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \boldsymbol{u}_w)$
$= \sum_{w \in \text{Vocab}} (-\boldsymbol{y}_w \boldsymbol{u}_w + \hat{\boldsymbol{y}}_w \boldsymbol{u}_w) = U^\top(-\boldsymbol{y}_w + \hat{\boldsymbol{y}}_w)$

(ii) The gradient is equal to zero when $\boldsymbol{y} = \hat{\boldsymbol{y}}$

(iii) The first term in the gradient is the weighted sum of all the other word vectors in the vocabulary. The weights are determined by the probability of each word in the vocabulary given the current context. The second term is simply the current word vector.
When this gradient is subtracted from the current word vector, it has the effect of pushing the vector

[4] 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.

away from the other words in the vocabulary that are contextually similar to it and pulling it towards words that are dissimilar. The first term of the gradient improves the word vector by incorporating information about the surrounding context and adjusting the vector to be more representative of that context. The second term ensures that the vector is still anchored to the original word and doesn't stray too far from its original meaning.
Overall, subtracting the gradient of the naive softmax function from the word vector helps to improve the vector's representation of the current context while still preserving its core meaning.

(iv) Firstly, in most cases, L2 normalization does not take away useful information for the downstream task and is common practice in most downstream applications. In sentiment analysis, the sentiment of a phrase is usually determined by the relative positions of the word embeddings in the feature space rather than their magnitudes. In such cases, L2 normalization can actually improve the discriminative power of the embeddings by making them more invariant to differences in scale.
However, there is a case where L2 normalization can take away useful information for the downstream task. Following the hint, consider the case where $\boldsymbol{u}_x = \alpha \boldsymbol{u}_y$ for some words $x \neq y$ and some scalar α. In this case, L2 normalization would make the embeddings of these words indistinguishable from each other, as they would have the same direction and hence the same normalized form. This can be problematic for downstream tasks that require differentiation between words with similar directions but different magnitudes.

(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 $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots$). Note that $\boldsymbol{u}_w$ is a vector while $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots$ are scalars. Show your work to receive full credit.

Answer: $\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_w} = \frac{\partial -\boldsymbol{u}_o^\top v_c}{\partial \boldsymbol{u}_w} + \frac{\log \sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top v_c)}{\partial \boldsymbol{u}_w} = -\boldsymbol{y}_w v_c + \frac{\exp(\boldsymbol{u}_w^\top v_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top v_c)} v_c = -\boldsymbol{y}_w v_c + \hat{\boldsymbol{y}}_w v_c = (\hat{\boldsymbol{y}}_w - \boldsymbol{y}_w)^\top v_c$

where $y_w = \begin{cases} 1 & \text{if } w = o \\ 0 & \text{else} \end{cases}$

(d) (1 point) Write down the partial derivative of $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to $\boldsymbol{U}$. Please break down your answer in terms of the column vectors $\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_1}, \frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_2}, \cdots, \frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_{|\text{Vocab}|}}$. No derivations are necessary, just an answer in the form of a matrix.

Answer: From (c) we know $\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_w} = (\hat{\boldsymbol{y}}_w - \boldsymbol{y}_w)^\top v_c$ for $w = 1, 2, ..., Vocab$
Therefore $(\hat{\boldsymbol{y}} - \boldsymbol{y})^\top v_c = \frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{U}} = [\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_1}, \frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_2}, \cdots, \frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_{|\text{Vocab}|}}]$

(e) (2 points) The Leaky ReLU (Leaky Rectified Linear Unit) activation function is given by Equation 4 and Figure 2:

$$f(x) = \max(\alpha x, x) \tag{4}$$

Where x is a scalar and $0 < \alpha < 1$, please compute the derivative of $f(x)$ with respect to x. You may ignore the case where the derivative is not defined at 0.[5]

[5] If you're interested in how to handle the derivative at this point, you can read more about the notion of subderivatives.

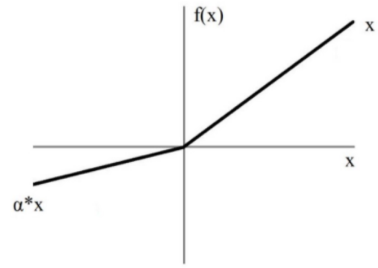

Figure 2: Leaky ReLU

Answer: $\frac{\partial f(x)}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$

(f) (3 points) The sigmoid function is given by Equation 5:

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{5}$$

Please compute the derivative of $\sigma(x)$ with respect to x, where x is a scalar. Please write your answer in terms of $\sigma(x)$. Show your work to receive full credit.

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

(g) (6 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 $\boldsymbol{u}_{w_1}, \boldsymbol{u}_{w_2}, \dots, \boldsymbol{u}_{w_K}$.[6] 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:

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

for a sample $w_1, \dots w_K$, where $\sigma(\cdot)$ is the sigmoid function.[7]

(i) 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 the s^{th} negative sample $\boldsymbol{u}_{w_s}$. Please write your answers in terms of the vectors $\boldsymbol{v}_c$, $\boldsymbol{u}_o$, and $\boldsymbol{u}_{w_s}$, where $s \in [1, K]$. Show your work to receive full credit. **Note:** you should be able to use your solution to part (f) to help compute the necessary gradients here.

(ii) In lecture, we learned that an efficient implementation of backpropagation leverages the re-use of previously-computed partial derivatives. Which quantity could you reuse amongst the three

[6]Note: In the notation for parts (g) and (h), we are using words, not word indices, as subscripts for the outside word vectors.
[7]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.

partial derivatives calculated above to minimize duplicate computation? Write your answer in terms of
$\boldsymbol{U}_{o,\{w_1,\ldots,w_K\}} = \left[\boldsymbol{u}_o, -\boldsymbol{u}_{w_1}, \ldots, -\boldsymbol{u}_{w_K}\right]$, a matrix with the outside vectors stacked as columns, and $\mathbf{1}$, a $(K+1)\times 1$ vector of 1's.[8] Additional terms and functions (other than $\boldsymbol{U}_{o,\{w_1,\ldots,w_K\}}$ and $\mathbf{1}$) can be used in your solution.

(iii) Describe with one sentence why this loss function is much more efficient to compute than the naive-softmax loss.

Caveat: So far we have looked at re-using quantities and approximating softmax with sampling for faster gradient descent. Do note that some of these optimizations might not be necessary on modern GPUs and are, to some extent, artifacts of the limited compute resources available at the time when these algorithms were developed.

Answer:

(i) (i) $\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{v}_c} = \frac{\partial(-\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{s=1}^K \log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)))}{\partial \boldsymbol{v}_c}$

$= \frac{\partial(-\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))}{\partial \boldsymbol{v}_c} - \frac{\partial \sum_{s=1}^K \log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)))}{\partial \boldsymbol{v}_c}$

$= -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \frac{\partial \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\partial \boldsymbol{v}_c} - \sum_{s=1}^K \frac{1}{\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)} \frac{\partial \sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c))}{\partial \boldsymbol{v}_c}$

$= -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} (\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)))\boldsymbol{u}_o + \sum_{s=1}^K \frac{1}{\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)} (\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)(1-\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)))\boldsymbol{u}_{w_s}$

$= -(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\boldsymbol{u}_o + \sum_{s=1}^K (1-\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c))\boldsymbol{u}_{w_s}$

(ii) similar to above, the second term can be dropped because $o \notin \{w_1, \ldots, w_K\}$

$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_o} = -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \frac{\partial \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\partial \boldsymbol{u}_o} = -(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\boldsymbol{v}_c$

(iii) similar to (i) again, but the first term can be dropped

$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_{w_s}} = -\sum_{k=1}^K \frac{1}{\sigma(-\boldsymbol{u}_{w_k}^\top \boldsymbol{v}_c)} \frac{\partial \sigma(-\boldsymbol{u}_{w_k}^\top \boldsymbol{v}_c))}{\partial \boldsymbol{u}_{w_s}}$

$= (1-\sigma(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c))\boldsymbol{v}_c$

(ii) The embeddings of negative samples $\boldsymbol{u}_{w_s}$ are shared across multiple terms of the loss function. Therefore, we can reuse the partial derivative of the loss with respect to each negative sample's embedding across all the terms it appears in. Therefore, we reuse the quantity $\left[\mathbf{1}, -\boldsymbol{u}_{w_1}, \ldots, -\boldsymbol{u}_{w_K}\right]$

(iii) The negative sampling loss function is more efficient than the naive-softmax loss because it replaces the computation of the softmax function over the entire vocabulary with a smaller computation over a small number of negative samples, resulting in faster training and lower computational cost.

(h) (2 points) 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 $\boldsymbol{u}_{w_1}, \ldots, \boldsymbol{u}_{w_K}$. 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:

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{s=1}^{K} \log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)) \tag{7}$$

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

[8]Note: NumPy will automatically broadcast 1 to a vector of 1's if the computation requires it, so you generally don't have to construct $\mathbf{1}$ on your own during implementation.

Compute the partial derivative of $\boldsymbol{J}_{\text{neg-sample}}$ with respect to a negative sample $\boldsymbol{u}_{w_s}$. Please write your answers in terms of the vectors $\boldsymbol{v}_c$ and $\boldsymbol{u}_{w_s}$, where $s \in [1, K]$. Show your work to receive full credit. Hint: break up the sum in the loss function into two sums: a sum over all sampled words equal to w_s and a sum over all sampled words not equal to w_s. Notation-wise, you may write 'equal' and 'not equal' conditions below the summation symbols, such as in Equation 8.

Answer:

Firstly breaking up the sum, we get

$\boldsymbol{J}$neg-sample$(\boldsymbol{v_c}, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u_o}^\top \boldsymbol{v_c})) - \sum_{\substack{1\leq j\leq k\\ w_j = w_s}} \log(\sigma(-\boldsymbol{u}_{\boldsymbol{w_i}}^\top \boldsymbol{v_c})) - \sum_{\substack{1\leq j\leq k\\ w_i \neq w_s}} \log(\sigma(-\boldsymbol{u}_{\boldsymbol{w_j}}^\top \boldsymbol{v_c}))$

Now it is obvious that the first and last term vanish by taking the derivative and we are left with

$\frac{\partial \boldsymbol{J}\text{neg-sample}(\boldsymbol{v_c}, o, \boldsymbol{U})}{\partial \boldsymbol{u_{w_s}}} = -\sum_{\substack{1\leq j\leq k\\ w_j = w_s}} (\sigma(-\boldsymbol{u}_{\boldsymbol{w_s}}^\top \boldsymbol{v_c}) - 1)\boldsymbol{v_c}$

(i) (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:

$$\boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}) = \sum_{\substack{-m\leq j\leq m\\ j\neq 0}} \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}) \tag{8}$$

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

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

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

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

Answer:

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

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

(iii) $\frac{\partial \boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U})}{\partial \boldsymbol{v}_w} = 0$ when $w \neq c$

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 $\boldsymbol{J}_{\text{skip-gram}}$ with respect to all parameters. You're ready to implement* `word2vec`*!*

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. **Windows users** may wish to install the Linux Windows Subsystem[9]. 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'll probably want to implement and test each part of this section in order, since the questions are cumulative.
Once you are done with the assignment you can deactivate this environment by running:
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.
Note: If you are using Windows and have trouble running the .sh scripts used in this part, we recommend trying Gow or manually running commands in the scripts.

(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.** In at most three sentences, briefly explain what you see in the plot. This may include, but is not limited to, observations on clusters and words that you expect to cluster but do not.

[9]https://techcommunity.microsoft.com/t5/windows-11/how-to-install-the-linux-windows-subsystem-in-windows-11/m-p/2701207

Answer:

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]".