Author: Benjamin Smidt
Created: October 3rd, 2022
Last Updated: January 17th, 2023

CS 224N A2: Word Vectors

Note to the reader. This is my work for assignment 2 of Stanford's course CS 224N: Natural Language Processing with Deep Learning. You can find the Winter 2021 lectures on YouTube here. This document is meant to be a reference, explanation, and resource for assignment 2. If there's a typo or a error, please email me at benjamin.smidt@utexas.edu so I can fix it. Finally, here is a link to my GitHub repo.

Contents

1 Written Understanding **2**
- 1.1 Problem A . . . 2
- 1.2 Problem B . . . 3
- 1.3 Problem C . . . 4
- 1.4 Problem D . . . 6
- 1.5 Problem E . . . 6
- 1.6 Problem F . . . 7
- 1.7 Problem G . . . 7
  - 1.7.1 (i) . . . 8
  - 1.7.2 (ii) . . . 10
  - 1.7.3 (iii) . . . 10
- 1.8 Problem H . . . 11
- 1.9 Problem I . . . 12

2 Programming **12**
- 2.1 Sigmoid . . . 12
- 2.2 Naive Softmax Loss and Gradient . . . 13
- 2.3 Negative Sampling Loss and Gradient . . . 13
- 2.4 Skipgram . . . 14
- 2.5 Stochastic Gradient Descent . . . 14

3 References **15**

1 Written Understanding

1.1 Problem A

Instructions

Prove that the naive-softmax loss 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).

Solution

Remember, we're using the skip-gram model (see lecture 1 notes, very helpful for definitions and general understanding). To start with, our naive-softmax loss is defined as

$$\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log P(O = o|C = c) \tag{1}$$

where

$$P(O = o \mid C = c) = \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{2}$$

Let's define our variables. Recall that $\boldsymbol{v}_c$ is the vector embedding for our given center word ($\boldsymbol{v}_c = Vx$ where $V \in \mathbb{R}^{n \times |Vocab|}$ and x is the one hot vector for the center word where $x \in \mathbb{R}^{|Vocab|}$). $\hat{\boldsymbol{y}}$ is our score vector and $\hat{\boldsymbol{y}_w}$ is the scalar at index w (denoting a given word) within $\hat{\boldsymbol{y}}$.

$$\hat{\boldsymbol{y}} = \frac{\exp(\boldsymbol{U}\boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \quad \text{and} \quad \hat{\boldsymbol{y}_w} = \frac{\exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{3}$$

Note that $\hat{\boldsymbol{y}} \in \mathbb{R}^{|Vocab|}$ (it is a vector of length equal to the vocabulary size). Additionally, it's numerator is of the same dimension as $\hat{\boldsymbol{y}}$ but its denominator is a scalar. You can interpret the expression to mean that each index of vector $\exp(\boldsymbol{U}\boldsymbol{v}_c)$ is divided by $\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)$ to yield $\hat{\boldsymbol{y}}$ (in fact this is exactly how you might think of programming something like this). Thus, each index w in the vector $\hat{\boldsymbol{y}}$ (scalar $\hat{\boldsymbol{y}_w}$) can be interpeted as the probability that the corresponding word (using the the index w and its one hot vector) is a context (or "outside") word given the center word (by eqn. 2)

$\boldsymbol{y}$ is defined as the one hot vector of the true outside word such that index o in $\boldsymbol{y}$ is 1 and all other indices are 0 for this particular problem. Then

$$-\sum_{w \in \text{Vocab}} \boldsymbol{y}_w \log(\hat{\boldsymbol{y}}_w) = -\boldsymbol{y}_1 \log(\hat{\boldsymbol{y}}_1) + ... + -\boldsymbol{y}_o \log(\hat{\boldsymbol{y}}_o) + ... + -\boldsymbol{y}_w \log(\hat{\boldsymbol{y}}_w) \tag{4}$$

$$-\sum_{w \in \text{Vocab}} \boldsymbol{y}_w \log(\hat{\boldsymbol{y}}_w) = -(0) \log(\hat{\boldsymbol{y}}_1) + ... + -(1) \log(\hat{\boldsymbol{y}}_o) + ... + -(0) \log(\hat{\boldsymbol{y}}_w) \tag{5}$$

$$-\sum_{w \in \text{Vocab}} \boldsymbol{y}_w \log(\hat{\boldsymbol{y}}_w) = -\log(\hat{\boldsymbol{y}}_o) \tag{6}$$

Remember that $\boldsymbol{y}_o$ represents the word at index o, who's one hot vector is a 1 at index o, indicating that it is a context word given $\boldsymbol{v}_c$. That is,

$$\hat{\boldsymbol{y}}_o = P(O = o \mid C = c) = \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{7}$$

Hence, by substitution

$$-\sum_{w\in\text{Vocab}} \boldsymbol{y}_w \log(\hat{\boldsymbol{y}}_w) = -\log(\frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w\in\text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}) \tag{8}$$

$$-\sum_{w\in\text{Vocab}} \boldsymbol{y}_w \log(\hat{\boldsymbol{y}}_w) = -\log P(O=o|C=c) \tag{9}$$

$$-\sum_{w\in\text{Vocab}} \boldsymbol{y}_w \log(\hat{\boldsymbol{y}}_w) = \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) \tag{10}$$

I know the instructions said one line but I was going for clarity here. Obviously, I could not define the variables (leave it to you to figure out what they mean) and just write some one liner that connects the dots. However, I wanted to make this as clear to understand as possible. Hopefully this leaves no room for ambiguity.

1.2 Problem B

Instructions

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 $\boldsymbol{U}$. Additionally, answer the following two questions with one sentence each: (1) When is the gradient zero? (2) Why does subtracting this gradient, in the general case when it is nonzero, make $\boldsymbol{v}_c$ a more desirable vector (namely, a vector closer to outside word vectors in its window)?

Solution

We start with our definition of $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$.

$$\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -log\frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w\in\text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{11}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = \frac{\partial}{\partial \boldsymbol{v}_c} - log\frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w\in\text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{12}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = \frac{\partial}{\partial \boldsymbol{v}_c} - \boldsymbol{u}_o^\top \boldsymbol{v}_c + \frac{\partial}{\partial \boldsymbol{v}_c}\, log \sum_{w\in\text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c) \tag{13}$$

Everything up to this point should be easy to follow if you remember algebra (although see the first lecture where he goes through these exact steps in detail if you are confused). Note that in the coming step, I do a change of variables to ensure I know what I'm taking my partial derivative with respect to.

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = -\boldsymbol{u}_o^\top + \frac{1}{\sum_{w\in\text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \frac{\partial}{\partial \boldsymbol{v}_c} \sum_{j\in\text{Vocab}} \exp(\boldsymbol{u}_j^\top \boldsymbol{v}_c) \tag{14}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = -\boldsymbol{u}_o^\top + \frac{\sum_{j\in\text{Vocab}} \boldsymbol{u}_j^\top \exp(\boldsymbol{u}_j^\top \boldsymbol{v}_c)}{\sum_{w\in\text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{15}$$

Recall that

$$\hat{\boldsymbol{y}}_w = \frac{\exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{16}$$

By substitution, (I change back the j to w since there's only one sum now)

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = -\boldsymbol{u}_o^\top + \sum_{j \in \text{Vocab}} \boldsymbol{u}_w^\top \hat{\boldsymbol{y}}_w \tag{17}$$

Note that the expression to the right of the addition (above) is each index w of matrix $U \in \mathbb{R}^{|Vocab| \times n}$ (vector $\boldsymbol{u}_w$) being weighted by each index w of vector $\hat{\boldsymbol{y}}$ (scalar $\hat{\boldsymbol{y}}_w$). Rewriting this using matrices (remember $\boldsymbol{y}$ is one hot where $\boldsymbol{y}_o = 1$), we find

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = -\boldsymbol{U}^\top \boldsymbol{y} + \boldsymbol{U}^\top \hat{\boldsymbol{y}}_{\boldsymbol{w}} \tag{18}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = \boldsymbol{U}^\top (\hat{\boldsymbol{y}} - \boldsymbol{y}) \tag{19}$$

You can verify the shape of this vector is the same shape as $\boldsymbol{v}_c \in \mathbb{R}^n$.

1. The gradient is zero when $\hat{\boldsymbol{y}} = \boldsymbol{y}$. Obviously if our predicted and correct vectors are equivalent then our accuracy is perfect and there's no update that could improve the loss.
2. Because we're doing gradient descent (as opposed to ascent), the update adds some portion of $\boldsymbol{U}^\top \boldsymbol{y}$ and subtracts some portion of $\boldsymbol{U}^\top \hat{\boldsymbol{y}}$. This makes intuitive sense because adding the correct vector $\boldsymbol{u}_o^\top$ makes it more like that vector (which is what we want) and subtracting by $\boldsymbol{U}^\top \hat{\boldsymbol{y}}$, the weighted average of our incorrect vectors that are producing the loss, makes it less like those vectors (again, what we want).

1.3 Problem C

Instructions

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 $\boldsymbol{y}_1$, $\boldsymbol{y}_2$, ...). Note that $\boldsymbol{u}_w$ is a vector while $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots$ are scalars.

Solution

Case 1: $\boldsymbol{u}_w = \boldsymbol{u}_o$

We start with the case that $\boldsymbol{u}_w = \boldsymbol{u}_o$.

$$\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -log \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{20}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = \frac{\partial}{\partial \boldsymbol{u}_o} - log \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{21}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = \frac{\partial}{\partial \boldsymbol{u}_o} - \boldsymbol{u}_o^\top \boldsymbol{v}_c + \frac{\partial}{\partial \boldsymbol{u}_o} log \sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c) \tag{22}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = -\boldsymbol{v}_c + \frac{1}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \frac{\partial}{\partial \boldsymbol{u}_o} \sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c) \tag{23}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = -\boldsymbol{v}_c + \frac{\boldsymbol{v}_c \exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{24}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = -\boldsymbol{v}_c + \boldsymbol{v}_c(\hat{\boldsymbol{y}}_{\boldsymbol{o}}) \tag{25}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = \boldsymbol{v}_c(\hat{\boldsymbol{y}}_{\boldsymbol{o}} - 1) \tag{26}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = \boldsymbol{v}_c(\hat{\boldsymbol{y}}^\top \boldsymbol{y} - 1) \tag{27}$$

Revisit eqn. 3 if going from step 24 to step 25 isn't clicking. We choose to represent $\hat{\boldsymbol{y}}_{\boldsymbol{o}}$ as $\hat{\boldsymbol{y}}^\top$ from the direction's constraints of how to represent the equation. I wasn't sure if eqn. 26 was good enough or it was meant to be more generalized.

Case 2: $\boldsymbol{u}_w \neq \boldsymbol{u}_o$

Now we move onto the case that $\boldsymbol{u}_w \neq \boldsymbol{u}_o$. That is, the gradient with respect to any vector $\boldsymbol{u}_w$ that isn't $\boldsymbol{u}_o$. I'll use the notation $\boldsymbol{u}_j$ to indicate the particular $\boldsymbol{u}_w$ we want to find the gradient with respect to and (hopefully) prevent any confusion.

$$\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -log \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{28}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_j} = \frac{\partial}{\partial \boldsymbol{u}_j} - log \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{29}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_j} = \frac{\partial}{\partial \boldsymbol{u}_j} - \boldsymbol{u}_o^\top \boldsymbol{v}_c + \frac{\partial}{\partial \boldsymbol{u}_j} \, log \sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c) \tag{30}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_j} = \frac{1}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \frac{\partial}{\partial \boldsymbol{u}_j} \sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c) \tag{31}$$

$$= \frac{1}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \frac{\partial}{\partial \boldsymbol{u}_j} (\exp(\boldsymbol{u}_1^\top \boldsymbol{v}_c) + ... + \exp(\boldsymbol{u}_j^\top \boldsymbol{v}_c) + ... + \exp(\boldsymbol{u}_{|Vocab|}^\top \boldsymbol{v}_c)) \tag{32}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_j} = \frac{\boldsymbol{v}_c \exp(\boldsymbol{u}_j^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \tag{33}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_j} = \boldsymbol{v}_c \; \hat{\boldsymbol{y}}_{\boldsymbol{j}} \tag{34}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_j} = \boldsymbol{v}_c(\hat{\boldsymbol{y}}_{\boldsymbol{j}} - \boldsymbol{y}_j) \tag{35}$$

Note that $\hat{\boldsymbol{y}}_{\boldsymbol{j}}$ is a scalar and $\boldsymbol{y}_j$ is 0 since the word j is defined as not being an outside word.

1.4 Problem D

Instructions
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 $\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_1}$, $\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_2}$, $\cdots$, $\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c,o,\boldsymbol{U})}{\partial \boldsymbol{u}_{|\text{Vocab}|}}$. The solution should be one or two lines long.

Solution
We already know

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_j} = \boldsymbol{v}_c(\hat{\boldsymbol{y}}_{\boldsymbol{j}} - \boldsymbol{y}_j) \text{ and } \frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = \boldsymbol{v}_c(\hat{\boldsymbol{y}}_{\boldsymbol{0}} - 1) \tag{36}$$

Since $\boldsymbol{y}$ is a one hot vector, then With that we can write

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{U}} = \frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_1}, ..., \frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_V} \tag{37}$$

where V is the index of the last word vector in $\boldsymbol{U}$. It follows then that

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{U}} = \boldsymbol{v}_c\,(\hat{\boldsymbol{y}}_{\boldsymbol{1}} - \boldsymbol{y}_1), \boldsymbol{v}_c\,(\hat{\boldsymbol{y}}_{\boldsymbol{2}} - \boldsymbol{y}_2), ..., \boldsymbol{v}_c\,(\hat{\boldsymbol{y}}_{\boldsymbol{o}} - 1), ..., \boldsymbol{v}_c\,(\hat{\boldsymbol{y}}_{|\boldsymbol{V}|} - \boldsymbol{y}_{|V|}) \tag{38}$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{U}} = \boldsymbol{v}_c(\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \tag{39}$$

Our current equation suggests $\boldsymbol{U} \in \mathbb{R}^{n \times |Vocab|}$. However, $\boldsymbol{U} \in \mathbb{R}^{|Vocab| \times n}$, the transpose of what we currently have. This comes from the fact that $\boldsymbol{u}_w$ is thought of as a column vector when alone (or at least that's how I interpret it) but it is represented as a row vector in $\boldsymbol{U}$. So, we fix it with the following to achieve the correct shape.

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{U}} = (\hat{\boldsymbol{y}} - \boldsymbol{y})\boldsymbol{v}_c^\top \tag{40}$$

1.5 Problem E

Instructions
The ReLU (Rectified Linear Unit) activation function is given by the Equation:

$$f(x) = \max(0, x) \tag{41}$$

Please compute the derivative of $f(x)$ with respect to x, where x is a scalar. You may ignore the case that the derivative is not defined at 0.

Solution

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x > 0 \end{cases} \tag{42}$$

$$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{43}$$

1.6 Problem F

Instructions

The sigmoid function is given by:

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{44}$$

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 $\sigma(x)$.

Solution

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \quad \textit{Multiply by } \frac{e^x}{e^x} \tag{45}$$

$$\sigma'(x) = \frac{e^x(e^x+1) - e^x e^x}{(e^x+1)^2} \quad \textit{Product Rule} \tag{46}$$

$$\sigma'(x) = \frac{e^{2x} + e^x - e^{2x}}{(e^x+1)^2} \tag{47}$$

$$\sigma'(x) = \frac{e^x}{(e^x+1)^2} \tag{48}$$

$$\sigma'(x) = \frac{e^x}{e^x+1}\ \frac{1}{e^x+1} \tag{49}$$

$$\sigma'(x) = \frac{1}{1+e^{-x}}\ \frac{1}{e^x+1} \quad \textit{Multiplied left by } \frac{\frac{1}{e^x}}{\frac{1}{e^x}} \tag{50}$$

$$\sigma'(x) = \sigma(x)\ \frac{1}{e^x+1} \tag{51}$$

$$\sigma'(x) = \sigma(x)\ \frac{e^x+1-e^x}{e^x+1} \tag{52}$$

$$\sigma'(x) = \sigma(x)\ (\frac{e^x+1}{e^x+1} - \frac{e^x}{e^x+1}) \tag{53}$$

$$\sigma'(x) = \sigma(x)\ (1 - \frac{1}{1+e^{-x}}) \tag{54}$$

$$\sigma'(x) = \sigma(x)\ (1 - \sigma(x)) \tag{55}$$

1.7 Problem G

Instructions

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 $\boldsymbol{u}_{w_1}, \boldsymbol{u}_{w_2}, \ldots, \boldsymbol{u}_{w_K}$. For this question, assume that the K negative samples are distinct. In

other words, $i \neq j$ implies $w_i \neq w_j$ for $i, j \in \{1, \ldots, K\}$. 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:

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{s=1}^{K} \log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)) \tag{56}$$

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

1.7.1 (i)

Instructions

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 $s \in [1, K]$. **Note:** you should be able to use your solution to part (f) to help compute the necessary gradients here.

Part B Recap:

(b) Compute the partial derivative of $\boldsymbol{J}_{naive\text{-}softmax}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to $\boldsymbol{v}_c$. Please write your answer in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{U}$. Additionally, answer the following two questions with one sentence each: (1) When is the gradient zero? (2) Why does subtracting this gradient, in the general case when it is nonzero, make $\boldsymbol{v}_c$ a more desirable vector (namely, a vector closer to outside word vectors in its window)?

Solution

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{s=1}^{K} \log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)) \tag{57}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = \frac{\partial}{\partial \boldsymbol{v}_c} - \log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \frac{\partial}{\partial \boldsymbol{v}_c} \sum_{s=1}^{K} \log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)) \tag{58}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \frac{\partial}{\partial \boldsymbol{v}_c} \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - \sum_{s=1}^{K} \frac{\partial}{\partial \boldsymbol{v}_c} \log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)) \tag{59}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \frac{\partial}{\partial \boldsymbol{v}_c} \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - \sum_{s=1}^{K} \frac{1}{(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c))} \frac{\partial}{\partial \boldsymbol{v}_c} (\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)) \tag{60}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = -\frac{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} [1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)] \boldsymbol{u}_o - \sum_{s=1}^{K} \frac{\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)}{(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c))} [1 - \sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)](-\boldsymbol{u}_{w_s}) \tag{61}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = [\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1] \boldsymbol{u}_o + \sum_{s=1}^{K} [1 - \sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)] \boldsymbol{u}_{w_s} \tag{62}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = [\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1] \boldsymbol{u}_o + \sum_{s=1}^{K} [1 - \frac{1}{1 + \exp(-(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c))}] \boldsymbol{u}_{w_s} \tag{63}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = [\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1]\boldsymbol{u}_o + \sum_{s=1}^{K}[\frac{1 + \exp(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)}{1 + \exp(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)} - \frac{1}{1 + \exp(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)}]\boldsymbol{u}_{w_s} \quad (64)$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = [\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1]\boldsymbol{u}_o + \sum_{s=1}^{K}[\frac{\exp(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)}{1 + \exp(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)}]\boldsymbol{u}_{w_s} \quad (65)$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = [\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1]\boldsymbol{u}_o + \sum_{s=1}^{K}[\frac{1}{1 + \exp(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)}]\boldsymbol{u}_{w_s} \quad (66)$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = [\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1]\boldsymbol{u}_o + \sum_{s=1}^{K}\sigma(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)\boldsymbol{u}_{w_s} \quad (67)$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = -[1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)]\boldsymbol{u}_o + \sum_{s=1}^{K}\sigma(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)\boldsymbol{u}_{w_s} \quad (68)$$

You can see that the gradient is zero when $[1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)]\boldsymbol{u}_o = \sum_{s=1}^{K}\sigma(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)\boldsymbol{u}_{w_s}$. For our first term, if $\boldsymbol{u}_o^\top \boldsymbol{v}_c$ is a large negative number (dissimilar) then $1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)$ evaluates to be close to 1. Thus, in our gradient descent update, we'll be adding $\approx \boldsymbol{u}_o$ to $\boldsymbol{v}_c$ (recall we're doing gradient descent so $-(-[1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)]\boldsymbol{u}_o) = [1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)]\boldsymbol{u}_o \approx \boldsymbol{u}_o$). Intuitively this makes sense because adding $\boldsymbol{u}_o$ to $\boldsymbol{v}_c$ will make $\boldsymbol{v}_c$ more like $\boldsymbol{u}_o$, which is what we want.

If $\boldsymbol{u}_o^\top \boldsymbol{v}_c$ is a large positive number (similar) then $1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)$ evaluates to a small number and our gradient update for the first term adds approximately 0. Again, this is intuitive because if the vectors are similar already then we don't need to change v_c much.

For our second term, a similar line of reasoning can be invoked to see that we're subtracting out $\approx \boldsymbol{u}_{w_s}$ from $\boldsymbol{v}_c$ if $\sigma(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)$ is large positive (vectors are similar) and doing nothing if $\sigma(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)$ is large negative (vectors dissimilar). Again, this makes sense because we want $\boldsymbol{v}_c$ to be less like the vectors that we negative sample and don't need to change $\boldsymbol{v}_c$ if that's already the case.

(c) Compute the partial derivatives of $\boldsymbol{J}_{naive\text{-}softmax}(\boldsymbol{v}_c, o, \boldsymbol{U})$ 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 $\boldsymbol{y}_1$, $\boldsymbol{y}_2$, ...). Note that $\boldsymbol{u}_w$ is a vector while $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots$ are scalars.

Case 1: $\frac{\partial}{\partial \boldsymbol{u}_o}$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = \frac{\partial}{\partial \boldsymbol{u}_o} - \log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \frac{\partial}{\partial \boldsymbol{u}_o}\sum_{s=1}^{K}\log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)) \quad (69)$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = -\frac{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}[1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)]\boldsymbol{v}_c \quad (70)$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = -[1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)]\boldsymbol{v}_c \tag{71}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_o} = [\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1]\boldsymbol{v}_c \tag{72}$$

Case 2: $\frac{\partial}{\partial \boldsymbol{u}_{w_s}}$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_{w_s}} = \frac{\partial}{\partial \boldsymbol{u}_{w_s}} - \log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \frac{\partial}{\partial \boldsymbol{u}_{w_s}} \sum_{s=1}^{K} \log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)) \tag{73}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_{w_s}} = -\frac{\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)}{\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)}[1 - \sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)](-\boldsymbol{v}_c) \tag{74}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_{w_s}} = [1 - \sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)]\boldsymbol{v}_c \tag{75}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_{w_s}} = \sigma(\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)\boldsymbol{v}_c \tag{76}$$

See part (b) of this problem for that last jump, it's the exact same as before.

1.7.2 (ii)

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

Solution
Since we compute the sigmoid function so much we could reuse the following:

$$\sigma(\boldsymbol{U}_o \boldsymbol{v}_c)$$

1.7.3 (iii)

Instructions
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.

Solution
This loss function is much more efficient because, provided we have enough negative samples, the negative sampling we use will approximate the gradient update we would've gotten for

the entire vocabulary but with literally a fraction of the vocabulary used. Instead of iterating through the entire vocabulary we can just iterate though negative samples which can be orders of magnitude smaller in size.

1.8 Problem H

Instructions

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:

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{s=1}^{K} \log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)) \tag{77}$$

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 $\boldsymbol{u}_{w_s}$. Please write your answers in terms of the vectors $\boldsymbol{v}_c$ and $\boldsymbol{u}_{w_s}$, where $s \in [1, K]$. 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.

Solution

I'm going to use j to iterate over the sum in this equation instead of s to make the notation more clear.

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

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

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

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

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

1.9 Problem I

Instructions

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:

$$\boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}) = \sum_{\substack{-m \le j \le m \\ j \neq 0}} \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}) \tag{78}$$

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.

Solution

At first I thought these solutions were supposed to be substituted and simplified but I'm pretty sure this is literally all the question is asking.

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U})}{\partial \boldsymbol{U}} = \sum_{\substack{-m \le j \le m \\ j \neq 0}} \frac{\partial \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})}{\partial \boldsymbol{U}} \tag{79}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = \sum_{\substack{-m \le j \le m \\ j \neq 0}} \frac{\partial \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})}{\partial \boldsymbol{v}_c} \tag{80}$$

$$\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U})}{\partial \boldsymbol{v}_w} = 0 \tag{81}$$

This last one should make sense because $\boldsymbol{v}_w$ isn't part of the loss function at all.

2 Programming

If you don't know how to use NumPy (or Python more generally) I'd suggest going through this tutorial to learn the basics

2.1 Sigmoid

This one is a gimme if you know how to use NumPy at all. Literally just write down the sigmoid function using *np.exp()* and you're done.

2.2 Naive Softmax Loss and Gradient

Here is where all the math we've done really comes in handy. Literally 90% of this assignment is understanding and deriving the equations while the other 10% is just writing down our mathematical results for the computer to calculate. Now, let's get to programming. As we saw in Problem A, our loss function is

$$\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)})$$

Let's rearrange this a bit and make it more numerically stable. These exponentials can easily blow up to numbers much larger than we can store in a computer. We first add a constant $\boldsymbol{C}$ to both the numerator and the denominator, which you can see doesn't affect the equation at all.

$$\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\frac{\boldsymbol{C}\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \boldsymbol{C}\exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)})$$

Then we move the constant inside the exponential.

$$\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c + log\boldsymbol{C})}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c + log\boldsymbol{C})})$$

We can then set $log\boldsymbol{C} = -max(\boldsymbol{u}_w^\top \boldsymbol{v}_c)$ so that the maximum value of our exponential will be 1. This is prevents the exponential from blowing up the loss such that it makes our code numerically unstable. For more info see these notes from CS231N.

In my implementation, I first found the softmax probabilities ($\hat{\boldsymbol{y}}$) for every vector in $\boldsymbol{U}$ since we'll need it later for the gradient. Speaking of which, here are our gradients computed in problems B, C, and D.

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{v}_c} = \boldsymbol{U}^\top(\hat{\boldsymbol{y}} - \boldsymbol{y})$$

$$\frac{\partial \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{U}} = (\hat{\boldsymbol{y}} - \boldsymbol{y})\boldsymbol{v}_c^\top$$

I chose not to use a one-hot vector $\boldsymbol{y}$ in my code because it's quite simple to just subtract 1 from the value $\hat{\boldsymbol{y}}_o$ in our $\hat{\boldsymbol{y}}$ vector. A one-hot vector would just be a waste of space and excess computation. Hopefully my code is easy to follow. It's labeled exactly the same as the math so if you know how to use NumPy it should be easy to follow along.

2.3 Negative Sampling Loss and Gradient

Onto the more complicated negative sampling loss and gradient. Our loss from Problem G is

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

Our first step is to get our negative samples. This is implemented for us already (yay!? I'm always curious as to how things work but I'll leave my curiosity for more important things). Then, we get our matrix $\boldsymbol{U}_{o,\{w_1,\ldots,w_K\}} = \left[\boldsymbol{u}_o, -\boldsymbol{u}_{w_1}, \ldots, -\boldsymbol{u}_{w_K}\right]$. We matrix multiply $\boldsymbol{U}_o \boldsymbol{v}_c$,

pass the result through our sigmoid, take the negative log of every term, and sum everything to find our loss. And boom, loss calculated.

For the gradient with respect to $\boldsymbol{v}_c$ we have

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

Note that this isn't the final form I placed the gradient in but this is actually the easiest one to work with. We can rearrange this a bit to find

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

We can easily reuse $\boldsymbol{U}_o$ and the sigmoid we calculated before using this simple line of code

```
gradCenterVec = (sigmoid_vec - 1)  @ Uo
```

I'll leave it to you to figure out why this works. Onto our final gradient, the gradient with respect to $\boldsymbol{u}_o$ and $\boldsymbol{u}_{w_s}$. From problem G part (ii) we know

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

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

We use the following lines of code to find $\frac{\partial \boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{U}_o}$

```
sigmoidVec[0] -= 1
sigmoidVec[1:] = 1 - sigmoidVec[1:]
gradUo = np.outer(sigmoidVec, centerWordVec)
```

Finally, we iterate through $\boldsymbol{U}_o$ and add the vector to its proper place in our gradient matrix. I racked my brain for a long time to try and do this without using a for loop but I could not find one. Please let me know if you did, I would thoroughly enjoy knowing.

2.4 Skipgram

This function is pretty simple. All you need to do is iterate over all the words, add all the losses, add all the gradients, and spit back the output. Nothing much going on here to be honest. Although I will note that in the gradient checker, if you switch the order in which you check the two different functions (negative sampling vs. softmax), you actually get a different loss. I'm not totally sure what that's about but it was something worth mentioning.

2.5 Stochastic Gradient Descent

The update rule is pretty simple here. We just use the following code per usual

```
loss, gradient = f(x)
x -= step * gradient
```

Anyways, that's all for this assignment! My implementation ran in 30 minutes on Macbook with M1 chip for reference.

3 References

1. Python Tutorial
2. Softmax