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

1 Written: Understanding word2vec (23 points)

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

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

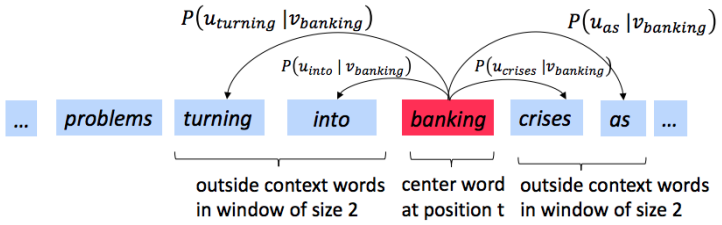

Figure 1: The word2vec skip-gram prediction model with window size 2

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

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

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

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

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

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

(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

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

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

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

Your answer should be one line.

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

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

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

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

Please compute the derivative of $\boldsymbol{\sigma}(x)$ with respect to $\boldsymbol{x}$, where $\boldsymbol{x}$ is a vector.

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

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

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

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

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

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

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

Write down three partial derivatives:

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

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

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

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

2 Coding: Implementing word2vec (20 points)

In this part you will implement the word2vec model and train your own word vectors with stochastic gradient descent (SGD). Before you begin, first run the following commands within the assignment directory in order to create the appropriate conda virtual environment. This guarantees that you have all the necessary packages to complete the assignment.

```
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
```

(a) (12 points) First, implement the `sigmoid` function in `word2vec.py` to apply the sigmoid function to an input vector. In the same file, fill in the implementation for the softmax and negative sampling loss and gradient functions. Then, fill in the implementation of the loss and gradient functions for the skip-gram model. When you are done, test your implementation by running `python word2vec.py`.

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

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

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

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

3 Submission Instructions

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

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

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

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

handwritten solution

a) $-\sum_w y_w \log(\hat{y}_w) = -\sum_{w \neq o} y_w \log(\hat{y}_w) - y_o \log(\hat{y}_o) = -\log \hat{y}_o$

b) $J(\vec{v}_c, \vec{o}, U) = -\log \frac{\exp(\vec{u}_o^T \vec{v}_c)}{\sum_w \exp(\vec{u}_w^T \vec{v}_c)}$, $\frac{\partial J}{\partial \vec{v}_c} = -\vec{u}_o + \sum_w \hat{y}_w u_w = -\vec{y}^T U + \hat{\vec{y}}^T U$

c) $\frac{\partial J}{\partial \vec{u}_w} = -\vec{v}_c \mathbb{1}_{\{w=o\}} + \frac{\sum \vec{v}_c \exp(\vec{u}_o^T \vec{v}_c)}{\sum \exp(\vec{u}_w^T \vec{v}_c)} = -\vec{v}_c \mathbb{1}_{(w=o)} + \sum \hat{y}_w \vec{v}_c = -\vec{v}_c \mathbb{1}_{\{w=o\}} + \hat{\vec{y}} \cdot \vec{v}_c$

d) $\sigma(\vec{x}) = (\sigma(x_1), \sigma(x_2), \cdots, \sigma(x_n))$, $\sigma'(x) = -\frac{1}{(1+e^{-x})^2}(-e^{-x}) = \frac{1}{1+e^{-x}} \frac{1}{1+e^{-x}} = \sigma(x)(1-\sigma(x))$

$\therefore \frac{\partial \sigma(\vec{x})}{\partial \vec{x}} = \text{diag}\{\sigma(x_1)(1-\sigma(x_1)), \cdots \sigma(x_n)(1-\sigma(x_n))\}$

e) $J_{\text{neg-sample}}(\vec{v}_c, \vec{o}, U) = -\log(\sigma(\vec{u}_o^T \vec{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\vec{u}_k^T \vec{v}_c))$, $\frac{\partial J}{\partial \vec{v}_c} = -\frac{\sigma(\vec{u}_o^T \vec{v}_c)(1-\sigma(\vec{u}_o^T \vec{v}_c))}{\sigma(\vec{u}_o^T \vec{v}_c)} \vec{u}_o - \sum_k \frac{\sigma(-\vec{u}_k^T \vec{v}_c)(1-\sigma(-\vec{u}_k^T \vec{v}_c))}{\sigma(-\vec{u}_k^T \vec{v}_c)} \vec{u}_k =$

$-(1-\sigma(\vec{u}_o^T \vec{v}_c))\vec{u}_o + \sum_{k=1}^{K} (1-\sigma(-\vec{u}_k^T \vec{v}_c))\vec{u}_k$

$\frac{\partial J}{\partial \vec{u}_o} = -(1-\sigma(\vec{u}_o^T \vec{v}_c))\vec{v}_c$, Note. $\vec{u}_k \neq \vec{u}_o$

$\frac{\partial J}{\partial \vec{u}_k} = \sum_{k=1} (1-\sigma(-\vec{u}_k^T \vec{v}_c))\vec{v}_c$

f) $\frac{\partial J}{\partial U} = \sum_{m=1}^{2w} \frac{\partial J(\vec{v}_c, u_{\text{word}}, U)}{\partial U}$

$\frac{\partial J}{\partial \vec{v}_c} = \sum \frac{\partial J(\vec{v}_c)}{\partial \vec{v}_c}$

$\frac{\partial J}{\partial \vec{v}_m} = 0$, $m \neq c$