Homework 1

DUE DATE: OCT/01/2024 23:59

Word2Vec (5 Points)[1]

1 Written: Understanding Word2Vec

In the framework of word2vec, consider a 'center' word c surrounded by a set of 'context (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':

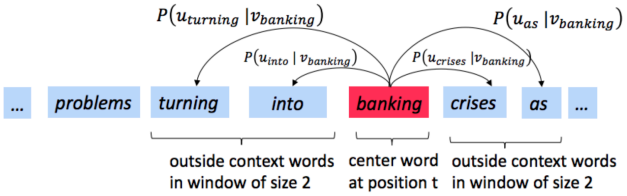

Figure 1: The word2vec skip-gram prediction model with window size 2

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:

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

For each word, we learn vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, where $\boldsymbol{u}_o$ is the 'outside' vector representing the outside word o, and $\boldsymbol{v}_c$ is the 'center' vector representing center word c. We store these parameters in two matrices, U and V. The columns of U are all the 'outside' vectors $\boldsymbol{u}_w$; the columns of V are all of the 'center' vectors $\boldsymbol{v}_w$. Both U and V contain a vector for every $w \in$ Vocabulary.[2]

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

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

[1] This homework is adapted from Stanford CS224N.

[2] 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 U and the 'outside' word vector for the word indexed by k. $\boldsymbol{v}_k$ is both the k^{th} column of 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.

We can view this loss as the cross-entropy[3] 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. The predicted distribution $\hat{\boldsymbol{y}}$ is the probability distribution $P(O|C = c)$ given by our model in equation (1).

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

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

(b) Compute the partial derivative of $J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, U)$ with respect to $\boldsymbol{v}_c$. Please write your answer in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and U. Show your work (the whole procedure) to receive full credit.

(c) Compute the partial derivatives of $J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, U)$ 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 part, 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. Show your work (the whole procedure) to receive full credit.

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

(e) Now let us consider the Negative Sampling loss, which is a more efficient 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 $\mathbf{u}_{w_1}, \mathbf{u}_{w_2}, \ldots, \mathbf{u}_{w_K}$. 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:

$$J_{\text{neg-sample}}(\mathbf{v}_c, o, U) = -\log(\sigma(\mathbf{u}_o^\top \mathbf{v}_c)) - \sum_{s=1}^{K} \log(\sigma(-\mathbf{u}_{w_s}^\top \mathbf{v}_c)) \tag{4}$$

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

for a sample $w_1, \ldots, w_K$, where $\sigma(\cdot)$ is the sigmoid function.[4]

(i) Please repeat parts (b) and (c), computing the partial derivatives of $J_{\text{neg-sample}}$ with respect to $\mathbf{v}_c$, with respect to $\mathbf{u}_o$, and with respect to the sth negative sample $\mathbf{u}_{w_s}$. Please write your answers in terms of the vectors $\mathbf{v}_c$, $\mathbf{u}_o$, and $\mathbf{u}_{w_s}$, where $s \in \{1, \ldots, K\}$.

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

(f) 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:

$$J_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots, w_{t+m}, U) = \sum_{-m \le j \le m, j \neq 0} J(\boldsymbol{v}_c, w_{t+j}, U). \tag{5}$$

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

Write down three partial derivatives:

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

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

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

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

Once you're done: Given that you computed the derivatives of $J(\boldsymbol{v}_c, w_{t+j}, U)$ with respect to all the model parameters U and V, you have now computed the derivatives of the full loss function $J_{skip-gram}$ with respect to all parameters. You're ready to implement word2vec!

[Instruction: For this section, write down your answers (manually or using word-processors) and submit the file named written.pdf via LMS.]

2 Practice: Word2Vec on the Gensim Library

Before diving into the actual implementation of Word2Vec, you will first experience its' effectiveness using "Gensim", an open-source library of Word2Vec. Before you begin, first run

[4] The sigmoid function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$.

the following commands within the assignment directory in order to create the appropriate conda virtual environment.[5] This helps you get prepared with most of the necessary packages to complete the assignment.

Windows users may wish to install the Linux Windows Subsystem.[6]

```
conda env create -f env.yml
conda activate HW1
```

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

```
conda deactivate
```

(a) For this practice, we will be using GloVe word vectors converted into word2vec format. Refer to the provided `gensim.ipynb` file. The first three cells does all the necessary processing required to use the word2vec model for you, so all you need to do is just execute them in order.

(b) The cells in section (b) offers functions that lets you see the relationship and similarities between words. Follow the instructions written in the markdown and try them out!

(c) Section (c) includes a function that takes a list of words as input and visualizes the vector space of those words. Run the cell with the predefined list of words. And also, try and come up with at least 20 words to fill the list `your_words` before executing the last cell. (Running the cells will give you two images `gensim.png` and `yourwords.png`.)

3 Coding: Implementing Word2Vec

In this part, you will implement the word2vec model and train your own word vectors with stochastic gradient descent (SGD). 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.

Note: If you are using Windows and have trouble running the .sh scripts used in this part, we recommend manually running commands in the scripts.

[5] Assume that you've installed Anaconda, a package manager for Python, on your system. For more info, visit https://www.anaconda.com.

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

(a) 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 skipgram method by running `python word2vec.py skipgram`.

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

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

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

Test your entire implementation by running `python word2vec.py`.

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

(c) 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. 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. Plan accordingly! (In the worst case scenario this can even take up to 5 hours!)*

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. (Try to compare this visualization with the gensim visualization `gensim.png`.)

[Instruction: compress all of your code (`*.py`, `*.ipynb`) and generated plots (`*.png`) with the name coding.zip, and upload the zipped file on LMS.]