Assignments

M3224.000300 Natural Language Processing with Neural Networks

Fall 2023

Due on Tuesday Oct. 10, 2023 by 9:30 am (before class)

(Assignment 2)

Graduate School of Data Science
Seoul National University

Instructor: Dr.Jay-yoon Lee
(lee.jayyoon@snu.ac.kr)
Head TA: Jonghyun Song
(hyeongoon11@snu.ac.kr)

Sungwoo PARK

Honor Pledge for Graded Assignments

"I, Sungwoo PARK, affirm that I have not given or received any unauthorized help on this assignment, and that this work is my own."

0 Instructions (2)

- Skeleton codes for problem 1 and 2 are at the directory `/skipgram` and `/sentimentAnalysis` each.
- Run the `bash collect_submission.sh` script to produce your 2023_abcde_coding.zip file. Please make sure to modify collect_sumbssion.sh file before running this command. (**abcde** stands for your student id)
- Modify this tex file into `a2_2023-abcde_written.pdf` with your written solutions
- Upload both `2023_abcde_coding.zip` and `2023_abcde_written.pdf` to etl website. **(2pts)**

1 Understanding Skip-Gram (48)

1.1 Softmax Loss Function

Word representation by **Word2vec** is theoretically supported by *distributional hypothesis*[1] In the example below, a 'center' word *banking* is surrounded by 'outside' words *turning, into, crises*, and *as* when the context window length is 2.

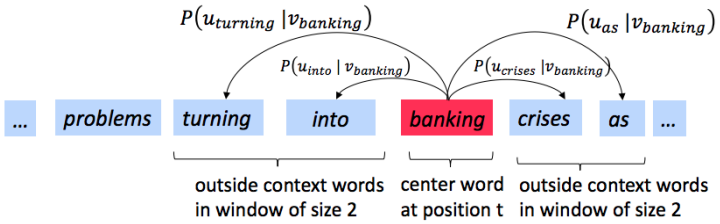

Figure 1: The skip-gram prediction model with window size 2

The objective of training Skip-gram is to learn the conditioned probability distribution of outside word O given center word C, $P(O = o|C = c)$. (i.e. the probability of the word o is an 'outside' word for word c) The probability can be obtained by taking the softmax function over the inner product of word vectors as below:

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

[1]The hypothesis that words that occur in similar contexts have similar meanings.

where u_o is the 'outside vector' representing outside word o, and v_c is the 'center vector' representing center word c. Be aware that outside and center vector representations of the same word are defined differently.

For the whole vocabulary, we can store outside vector u_w and center vector v_w in two matrices, U and V by columns. That is, the columns of U are all the outside vectors u_w and the columns of V are all the center vectors v_w for every $w \in Vocabulary$

The loss for a single pair of words c and o is defined as:

$$\begin{aligned} J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) &= -\log P(O = o|C = c) \\ &= -\log \frac{\exp(u_o^\top v_c)}{\sum_{w \in \text{Vocab}} \exp(u_w^\top v_c)} \\ &= -u_o^T v_c + \log \sum_{w \in vocab} \exp(u_w^T v_c) \end{aligned} \tag{2}$$

To learn the correct center and outside vector by gradient descent, we need to compute the partial derivative of the outside and center word vectors. Partial derivative of $J_{\text{naive-softmax}}$ with respect to v_c can be obtained as following:

$$\begin{aligned} \frac{\partial J}{\partial v_c} &= -u_o^T + \frac{\sum_{i \in vocab} u_i^T \exp(u_i^T v_c)}{\sum_{w \in vocab} \exp(u_w^T v_c)} \\ &= -u_o^T + \sum_{i \in vocab} P(O = i|C = c) u_i^T \\ &= -u_o^T + \sum_{w \in vocab} \hat{y}_w u_w^T \\ &= -\boldsymbol{y}^T \boldsymbol{U}^T + \hat{\boldsymbol{y}}^T \boldsymbol{U}^T \end{aligned} \tag{3}$$

where $\boldsymbol{y}$ is the one-hot vector which has 1 at the index of true outside word o ($y_o = 1$ and 0 for all other $i \neq o$, $y_i = 0$), and 0 everywhere else while $\hat{\boldsymbol{y}}$ stands for $P(O|C = c)$, a prediction made by Skip-gram model in equation (1).

(a) When is the gradient (3) is zero? **(2pts)** **Answer:** We can see that the equation 3 becomes 0 if and only if $\boldsymbol{y}^\top \boldsymbol{U}^\top$ is equal to $\hat{\boldsymbol{y}}^\top U^\top$. By the definition, $\boldsymbol{y}^\top$ is an indicator of the true outside word o and U is a matrix where each column is the vector representation of an outside word. This implies that $\boldsymbol{y}^\top \boldsymbol{U}^\top$ is the transpose of the vector representation of o, i.e. $u_o^\top$.

Meanwhile, $\hat{\boldsymbol{y}}^\top \boldsymbol{U}^\top$ is a weighted sum of all the outside word vectors based on the model's predicted probabilities, as in the second line of the equation (3). Given the fact that this weighted sum(or the linear combination of outside word vectors) to be equal to the true vector representation, it is implied that the vector of predicted probabilities is equal to the true indicator vector, i.e. the true one-hot vector.

Therefore, we can say that the gradient is zero if the model perfectly predicts the true outside word. However, we need special conditions, such as all u_i are linearly independent, to tell the converse is true as well.

(b) If we update v_c with this gradient, toward which vector is v_c getting pulled to? **(4pts)**

- **HINT**: The loss for a single pair of words c and o can be thought of as the cross-entropy between the true distribution y and the predicted distribution $\hat{y}$ for a particular center word c and outside word o.

$$\begin{aligned} \boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) &= -\log P(O = o|C = c) \\ &= -\sum_{w \in vocab} y_w \log(\hat{y}_w) \\ &= -\log \hat{y}_o. \end{aligned} \tag{4}$$

Answer: Like every other loss function, the cross entropy also measures the discrepancy between the true and the predicted output, and the parameters are updated in direction that decreases such discrepancy. This implies that the cross-entropy leads the predicted output to the true target.

Extending this notion and using the gradient given in (3), we can denote the parameter update by

$$v_c^{\text{new}} \leftarrow v_c^{\text{old}} - \alpha(-\boldsymbol{y}^\top \boldsymbol{U}^\top + \hat{\boldsymbol{y}}^\top \boldsymbol{U}^\top)$$

where $\alpha > 0$ is the learning rate, and the gradient indicates the discrepancy between the weighted sum of all outside word vectors based on the model's predicted probabilities, $\hat{\boldsymbol{y}}^\top \boldsymbol{U}^\top$ and the true outside vector $\boldsymbol{y}^\top \boldsymbol{U}^\top$, which is equivalent to $u_o^\top$.

Thus, like in the cross-entropy loss, the v_c is trained so that it is getting pulled towards the true outside vector representation, i.e. $\boldsymbol{y}^\top \boldsymbol{U}^\top$.

(c) Derive the partial derivatives of $J_{\text{naive-softmax}}$ with respect to the outside word vector matrix, U.

- **Note 1**: To get full credit, please write your answer in the form of $v_c, \mathbf{y}$, and $\hat{\mathbf{y}}$ and write the whole computation process. That is, the correct answer does not refer to other vectors but $v_c, \mathbf{y}$, and $\hat{\mathbf{y}}$.
- **Note 2**: Be careful about the dimensions of arrays and matrices. The final answer should have the same shape as U. (i.e. (vector length) $\times$ (vocabulary size))
- **HINT**: Start from outside word vector u_w, which is the column vector of U. Then, dividing into two cases may help: when $w = o$ and $w \neq o$ **(8pts)**

Answer: The given loss function $J(\boldsymbol{v}_c, o, \boldsymbol{U})$ can be re-arranged as below:

$$J(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\boldsymbol{u}_o^\top \boldsymbol{v}_c + \log\left(\exp\left(\boldsymbol{u}_1^\top \boldsymbol{v}_c\right) + \cdots + \exp\left(\boldsymbol{u}_{|V|}^\top \boldsymbol{v}_c\right)\right),$$

where $|V|$ denotes the number of words in the vocabulary.

First, consider the partial derivative of $\boldsymbol{u}_j^\top \boldsymbol{v}_c$ with respect to $\boldsymbol{u}_j$, a column vector of $\boldsymbol{U}$. Since $\boldsymbol{u}_j^\top \boldsymbol{v}_c = \sum_{i=1}^m u_{ij} v_{ic}$, where m denotes the embedding dimension, the partial derivative is equal to

$$\frac{\partial}{\partial \boldsymbol{u}_j} \boldsymbol{u}_j^\top \boldsymbol{v}_c = \left(\frac{\partial \sum_{i=1}^m u_{ij} v_{ic}}{\partial u_{1j}}, \ldots, \frac{\partial \sum_{i=1}^m u_{ij} v_{ic}}{\partial u_{mj}}\right)^\top = (v_{1c}, \ldots, v_{mc})^\top = \boldsymbol{v}_c.$$

Next step, consider the partial derivative of J with respect to $\boldsymbol{u}_j$, where $\boldsymbol{u}_j \neq \boldsymbol{u}_o$. By the chain rule,

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{u}_j} \log\left(\underbrace{\exp\left(\boldsymbol{u}_1^\top \boldsymbol{v}_c\right) + \cdots + \exp\left(\boldsymbol{u}_{|V|}^\top \boldsymbol{v}_c\right)}_{*}\right) &= \frac{\partial \log(*)}{\partial *} \cdot \frac{\partial *}{\partial \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right)} \cdot \frac{\partial \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right)}{\partial \boldsymbol{u}_j^\top \boldsymbol{v}_c} \cdot \frac{\partial \boldsymbol{u}_j^\top \boldsymbol{v}_c}{\partial \boldsymbol{u}_j} \\
&= \frac{\exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right) \cdot \boldsymbol{v}_c}{\sum_{i=1}^{|V|} \exp\left(\boldsymbol{u}_i^\top \boldsymbol{v}_c\right)} \\
&= \hat{y}_j \cdot \boldsymbol{v}_c.
\end{aligned}$$

If $\boldsymbol{u}_j = \boldsymbol{u}_o$, the partial derivative of the first term, $-\boldsymbol{u}_o^\top \boldsymbol{v}_c$, is added, so that

$$\frac{\partial J}{\partial \boldsymbol{u}_o} = -\boldsymbol{v}_c + \hat{y}_o \cdot \boldsymbol{v}_c = (\hat{y}_o - 1) \cdot \boldsymbol{v}_c.$$

Since $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{|V|})$, the partial derivative of J with respect to $\boldsymbol{U}$ is equal to $\left(\frac{\partial J}{\partial \boldsymbol{u}_1}, \ldots, \frac{\partial J}{\partial \boldsymbol{u}_{|V|}}\right)$, and using the intermediate results we have derived above,

$$\begin{aligned}
&\left(\hat{y}_1 \cdot \boldsymbol{v}_c, \ldots, (\hat{y}_o - 1) \cdot \boldsymbol{v}_c, \ldots, \hat{y}_{|V|} \cdot \boldsymbol{v}_c\right) \\
&= \left(\hat{y}_1, \ldots, \hat{y}_o, \ldots, \hat{y}_{|V|}\right) \cdot \boldsymbol{v}_c - (0, \ldots, 1_o, \ldots, 0) \cdot \boldsymbol{v}_c \\
&= \left((\hat{y}_1 - y_1) \cdot \boldsymbol{v}_c, \ldots, (\hat{y}_o - y_o) \cdot \boldsymbol{v}_c, \ldots, (\hat{y}_{|V|} - y_{|V|}) \cdot \boldsymbol{v}_c\right) \\
&= (\hat{\boldsymbol{y}} - \boldsymbol{y}) \circ \boldsymbol{v}_c,
\end{aligned}$$

where we define $\circ$ as an operator such that multiplies each element of $(\hat{\boldsymbol{y}} - \boldsymbol{y})$ to $\boldsymbol{v}_c$.

1.2 Negative Sampling Loss

Now we shall consider the Negative Sampling loss, which is an alternative to the Naive Softmax loss. Assume that K negative samples (words) are randomly 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 a center word c and an outside word o, the negative sampling loss function is given by:

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

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

(a) Compare the computational complexity of negative sampling with that of the softmax loss function presented in section 1.1. State the reason why negative sampling is computationally more efficient. **(4pts)**

Answer: From the point of computational complexity, the most influential term of the softmax loss is $\log \sum_{w \in \text{vocab}} \exp\big(\boldsymbol{u}_w^\top \boldsymbol{v}_c\big)$. For every word in the vocabulary, we need to compute a dot product between the outside word vector and the center word vector, and then take its exponent. Thus, for the entire vocabulary of size $|V|$, we have $O(|V|)$ dot product operations.

On the other hand, for the negative sampling loss, we have a dot product operation for the true outside word o and K dot product operations for the K negative samples. Thus, in total, we have $O(K+1)$ dot product operations.

Given that K is typically much smaller than $|V|$, the computational complexity of the negative sampling loss is less than that of the naive softmax.

(b) Compute the partial derivatives of $\boldsymbol{J}_{\text{neg-sample}}$ with respect to $\boldsymbol{v}_c$, $\boldsymbol{u}_o$, and 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 $s \in [1, K]$. (within five lines) **(8pts)** **Answer:** Negative sampling avoids the computationally expensive task of summing over all words in the vocabulary (as seen in the denominator of the softmax). Instead, it approximates the loss by considering only a small sample of negative words. By only computing dot products and other operations for one true outside word and K negative samples, negative sampling greatly reduces the computational cost, making it more efficient, especially when the vocabulary size $|V|$ is large.

1.3 Coding

In this part, you will implement the Skip-Gram model and train 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. You will be asked to implement the math functions above using the Numpy package at the root directory.

```
conda env create -f env.yml
conda activate a1
```

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

```
conda deactivate
```

For each of the methods you need to implement, we included approximately how many lines of code our solution has in the code comments. These numbers are included to guide you. You don't have to stick to them, you can write shorter or longer code as you wish. If you think your implementation is significantly longer than ours, it is a signal that there are some `numpy` methods you could utilize to make your code both shorter and faster. `for` loops in Python take a long time to complete when used over large arrays, so we expect you to utilize `numpy` methods. The sanity checking function is provided to check if your code works well.
To run the code command below, please move your directory to `/skipgram`.

(a) We will start by implementing a sampling method in `utils/treebank.py`.

(i) Implement the `sampleTokenIdx` method which first determines sampling frequency per token by reweighting the token frequency and samples a token according to the frequency. **(4pts)**

$$\text{index} = \arg\max(X), X = (X_1, \cdots, X_k)$$

$$X \sim Multinomial_k(1; p_1, p_2, ..., p_k)) \text{ where } p_i = \frac{(\text{Count}_{w_i})^{0.75}}{\Sigma_{j=1}^{k}(\text{Count}_{w_j})^{0.75}}$$

For more explanation on multinomial distribution, please refer to this reference.

(b) Now we will implement 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 naiveSoftmaxLossAndGradient method by running `python word2vec.py naiveSoftmaxLossAndGradient`. (Copy and paste it to your terminal)

(i) Implement the softmax loss and gradient in the `naiveSoftmaxLossAndGradient` method. **(6pts)**

(ii) Implement the negative sampling loss and gradient in the `negSamplingLossAndGradient` method. **(6pts)**

(c) When you are done, test your entire implementation by running `python word2vec.py`. Now we are going to load some real data and train word vectors with everything you just implemented! We are going to use The Recognizing Textual Entailment (RTE) dataset to train word vectors. (You can check out the raw datasets in the directory `./skipgram/utils/datasets/RTE`. 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 computing power of your machine **(an efficient implementation takes around three to four hours)**. Please start your homework as soon as possible!)

After 40,000 iterations, the script will finish and visualization for your word vectors will appear. It will also be saved as `word_vectors.png` in your project directory. **Include the plot below (6pts)**
Answer:

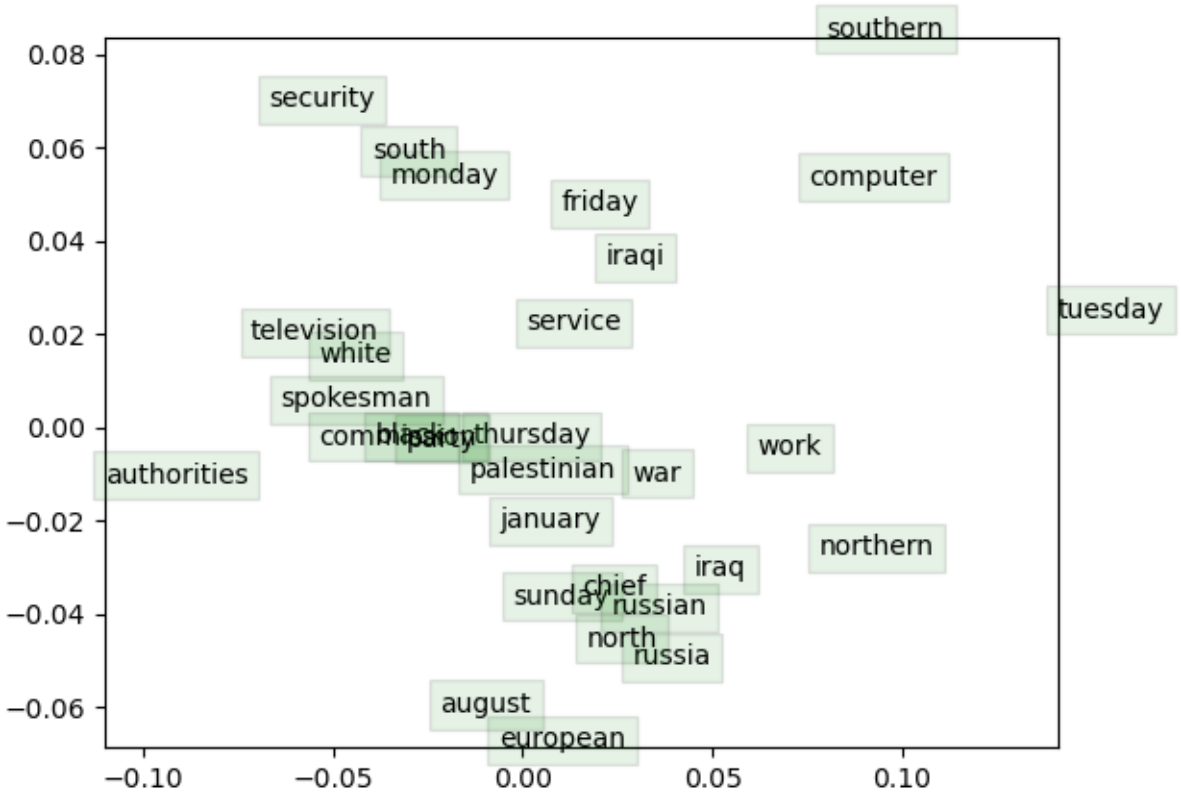

Figure 2: Visualization of Word Vectors

2 Sentiment Analysis using Neural Networks (50)

In class, we learned sequence representation and how language models are developed for different tasks. In this assignment, we will implement two neural network models for sentiment analysis task using IMDB dataset. Sentiment analysis in Natural Language Processing (NLP) is a task that involves classifying sentences or text into different categories based on the sentiment expressed. It aims to determine whether the sentiment of the text is positive, or negative. This analysis helps in understanding the overall opinion or emotion conveyed by the text.
For this question, please update the given jupyter notebook file in /sentimentAnalysis and submit it along with your answer to this latex file. Please note that this assignment is built and tested under Google Colaboratory. If you work on a local machine, you need to handle version issue on your own.

2.1 Multilayer Perceptron (MLP) (30 pts)

In this question, we are going to implement a simple Multilayer Perceptron (MLP) model to classify IMDB dataset. MLP is the classical type of neural network, and they are comprised of one or more layers of neurons. Data is fed to the input layer, there may be one or more hidden layers providing levels of abstraction, and predictions will be made on the output layer. Please refer to the jupyter notebook for detailed description.

2.2 Convolutional Neural Network (CNN) (20 pts)

Next, we will perform sentimental analysis on the same dataset with Convolutional Neural Network (CNN). In a CNN, text is organised into a matrix, with each row representing a word embedding. The CNN's convolutional layer scans the text like it would an image, breaks it down into features, and judges whether each feature matches the relevant label or not. Implement a CNN model following the instruction in jupyter notebook.