XCS224N Assignment 2 Understanding and Implementing Word2Vec

Due Sunday, October 26 at 11:59pm PT.

Guidelines

1. If you have a question about this homework, we encourage you to post your question on our Slack channel, at http://xcs224n-scpd.slack.com/
2. Familiarize yourself with the collaboration and honor code policy before starting work.
3. For the coding problems, you must use the packages specified in the provided environment description. Since the autograder uses this environment, we will not be able to grade any submissions which import unexpected libraries.

Submission Instructions

Written Submission: Some extra credit questions in this assignment require a written response. For these questions, you should submit a PDF with your solutions online in the online student portal. As long as the PDF is legible and organized, the course staff has no preference between a handwritten and a typeset LaTeX submission. If you wish to typeset your submission and are new to LaTeX, you can get started with the following:

- Type responses only in `submission.tex`.
- Submit the compiled PDF, **not** `submission.tex`.
- Use the commented instructions within the `Makefile` and `README.md` to get started.

Coding Submission: Some questions in this assignment require a coding response. For these questions, you should submit **all files indicated in the question** to the online student portal. For further details, see Writing Code and Running the Autograder below.

Honor code

We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the problem set the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions. More information regarding the Stanford honor code can be foudn at https://communitystandards.stanford.edu/policies-and-guidance/honor-code.

Writing Code and Running the Autograder

All your code should be entered into the `src/submission/` directory. When editing files in `src/submission/`, please only make changes between the lines containing `### START_CODE_HERE ###` and `### END_CODE_HERE ###`. Do not make changes to files outside the `src/submission/` directory.

The unit tests in `src/grader.py` (the autograder) will be used to verify a correct submission. Run the autograder locally using the following terminal command within the `src/` subdirectory:

```
$ python grader.py
```

There are two types of unit tests used by the autograder:

- `basic`: These tests are provided to make sure that your inputs and outputs are on the right track, and that the hidden evaluation tests will be able to execute.

- `hidden`: These unit tests are the evaluated elements of the assignment, and run your code with more complex inputs and corner cases. Just because your code passed the basic local tests does not necessarily mean that they will pass all of the hidden tests. These evaluative hidden tests will be run when you submit your code to the Gradescope autograder via the online student portal, and will provide feedback on how many points you have earned.

For debugging purposes, you can run a single unit test locally. For example, you can run the test case `3a-0-basic` using the following terminal command within the `src/` subdirectory:

```
$ python grader.py 3a-0-basic
```

Before beginning this course, please walk through the [Anaconda Setup for XCS Courses] to familiarize yourself with the coding environment. Use the env defined in `src/environment.yml` to run your code. This is the same environment used by the online autograder.

Test Cases

The autograder is a thin wrapper over the python `unittest` framework. It can be run either locally (on your computer) or remotely (on SCPD servers). The following description demonstrates what test results will look like for both local and remote execution. For the sake of example, we will consider two generic tests: `1a-0-basic` and `1a-1-hidden`.

Local Execution - Hidden Tests

All hidden tests rely on files that are not provided to students. Therefore, the tests can only be run remotely. When a hidden test like `1a-1-hidden` is executed locally, it will produce the following result:

```
----- START 1a-1-hidden:  Test multiple instances of the same word in a sentence.
----- END 1a-1-hidden [took 0:00:00.011989 (max allowed 1 seconds), ???/3 points] (hidden test ungraded)
```

Local Execution - Basic Tests

When a basic test like `1a-0-basic` passes locally, the autograder will indicate success:

```
----- START 1a-0-basic:  Basic test case.
----- END 1a-0-basic [took 0:00:00.000062 (max allowed 1 seconds), 2/2 points]
```

When a basic test like `1a-0-basic` fails locally, the error is printed to the terminal, along with a stack trace indicating where the error occurred:

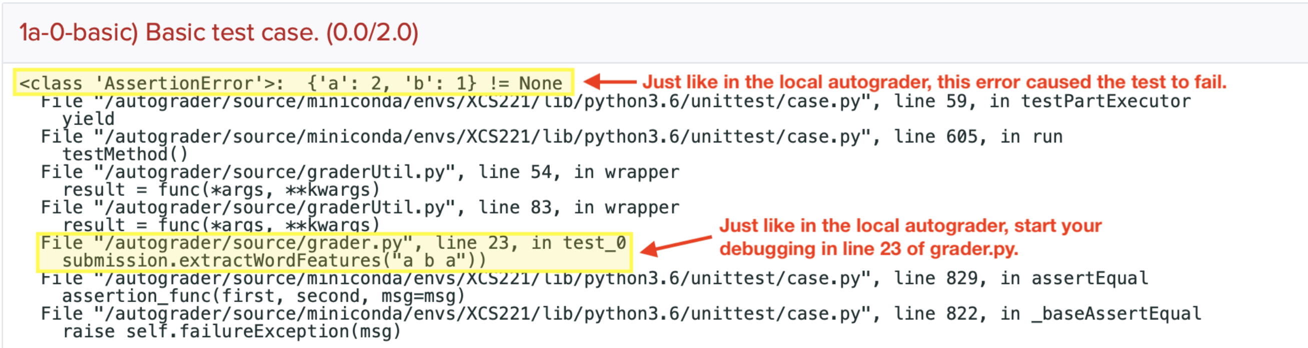
```
----- START 1a-0-basic:  Basic test case.
<class 'AssertionError'>
{'a': 2, 'b': 1} != None
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/grader.py", line 23, in test_0
    submission.extractWordFeatures("a b a"))
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
----- END 1a-0-basic [took 0:00:00.003809 (max allowed 1 seconds), 0/2 points]
```

This error caused the test to fail.

In this case, start your debugging in line 23 of grader.py.

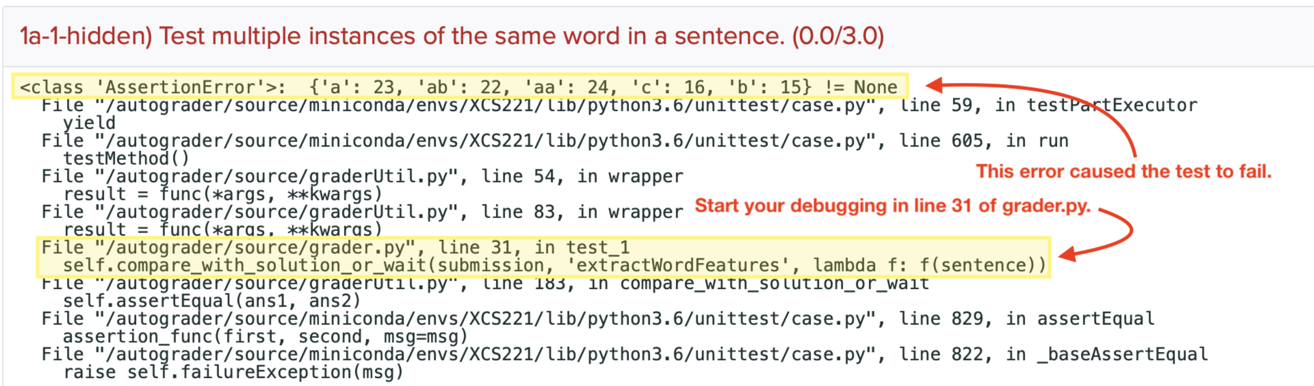
Remote Execution

Basic and hidden tests are treated the same by the remote autograder. Here are screenshots of failed basic and hidden tests. Notice that the same information (error and stack trace) is provided as the in local autograder, now for both basic and hidden tests.



1a-0-basic) Basic test case. (0.0/2.0)

```
<class 'AssertionError'>:  {'a': 2, 'b': 1} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 23, in test_0
    submission.extractWordFeatures("a b a"))
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```

1a-1-hidden) Test multiple instances of the same word in a sentence. (0.0/3.0)

```
<class 'AssertionError'>:  {'a': 23, 'ab': 22, 'aa': 24, 'c': 16, 'b': 15} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 31, in test_1
    self.compare_with_solution_or_wait(submission, 'extractWordFeatures', lambda f: f(sentence))
  File "/autograder/source/graderUtil.py", line 183, in compare_with_solution_or_wait
    self.assertEqual(ans1, ans2)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```


Finally, here is what it looks like when basic and hidden tests pass in the remote autograder.

1a-0-basic) Basic test case. (2.0/2.0)

1a-1-hidden) Test multiple instances of the same word in a sentence. (3.0/3.0)

1 Understanding word2vec (Refresher)

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.

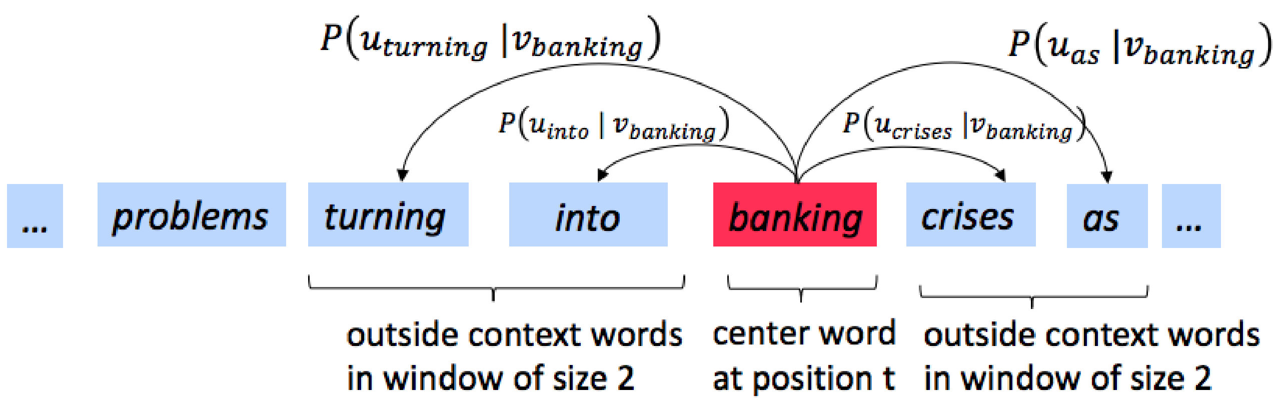


Figure 1: The word2vec skip-gram prediction model with window size 2

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

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

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). There are 2 optional questions below which you may attempt (*Note: Bonus points will be awarded for the optional assignments*)

[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 $-\sum_i p_i \log(q_i)$.

(a) **[2.50 points (Written, Extra Credit)] Extra Credit Challenge I**
The partial derivative of $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to $\boldsymbol{v}_c$ in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{U}$ is given below:

$$\frac{\partial J}{\partial \boldsymbol{v}_c} = \boldsymbol{U}(\hat{\boldsymbol{y}} - \boldsymbol{y}) \tag{3}$$

or equivalently,

$$\frac{\partial J}{\partial \boldsymbol{v}_c} = -\boldsymbol{u}_o + \sum_{w=1}^{V} \hat{y}_w \boldsymbol{u}_w \tag{4}$$

The naive-softmax loss given in Equation (2) is the same as the cross-entropy loss between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$; This is equivalent to:

$$-\sum_{w \in Vocab} y_w \log(\hat{y}_w) = -\log(\hat{y}_o). \tag{5}$$

Since $\boldsymbol{y}$ is a one-hot vector, all $\boldsymbol{y}_k = 0$ where $k \neq o$. $\boldsymbol{y}_o = 1$, so we are left with $-\log(\hat{\boldsymbol{y}}_o)$.

Write the steps to arrive at equation 3 or 4, the partial derivative of $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to $\boldsymbol{v}_c$, starting from equation 5. Please write your answer in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{U}$. The first few steps have been provided below (loss function $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$) and the rest of the proof may take 4 or 5 steps.

$$\begin{aligned}
J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) &= -\log(\hat{y}_o) && \text{(refer to equation 5)} \\
&= -\log\left(\frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}\right) \\
&= -\left(\log\big(\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\big) - \log\big(\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)\big)\right) \\
&= -\boldsymbol{u}_o^\top \boldsymbol{v}_c + \log\big(\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)\big)
\end{aligned}$$

(b) **[2.50 points (Written, Extra Credit)] Extra Credit Challenge II**
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 is given below:

$$\frac{\partial J}{\partial \boldsymbol{U}} = \boldsymbol{v}_c(\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \tag{6}$$

or equivalently:

$$\frac{\partial J}{\partial \boldsymbol{u}_w} = \begin{cases} (\hat{y}_w - 1)\boldsymbol{v}_c & \text{if } w = o \\ \hat{y}_w \boldsymbol{v}_c & \text{otherwise} \end{cases} \tag{7}$$

Write the steps required to arrive at the partial derivative 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 are two cases you need to consider: 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 $\boldsymbol{v}_c$. The proof may take 4 or 5 steps. The loss function $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ is:

$$J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\boldsymbol{u}_o^\top \boldsymbol{v}_c + \log\left(\sum_{w' \in \text{Vocab}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_c)\right)$$

2 Coding: Implementing word2vec

In this part you will implement the word2vec model and train your own word vectors with stochastic gradient descent (SGD). Make sure to take a look at the equations presented in sections 1a and 1b, especially **equations (3) for gradient (of loss function $J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$) with respect to $\boldsymbol{v}_c$ and (6) for gradient with respect to each of the 'outside' word vectors, $\boldsymbol{u}_w$'s** which we'll be using in our gradient calculation code.

Note: *A2 code can be found here :* *https://github.com/scpd-proed/XCS224N-A2*

(a) **[12 points (Coding)]** First, implement the `sigmoid()` function in `src-word2vec/submission.py` to apply the sigmoid function to an input vector. In the same file, fill in the implementation for the `naive_softmax_loss_and_gradient()` function. Then, fill in the implementation of the loss and gradient functions for the skip-gram model (`skipgram()`).
To complete these steps you will need to refer to Sections 1a and 1b, Equations (3) and (6).

Pseudo code for skip-gram

Algorithm 1 Skipgram

```
procedure SKIPGRAM(outside_words, loss_and_gradient_func, **kwargs)    ▷ The center word vector
    loss ← 0
    grad_center ← np.zeros(*args)
    grad_outside ← np.zeros(*args)
    for word in outside_words do                                      ▷ Iterate over outside words
        loss_current, gradc, grado ← loss_and_gradient_func(**kwargs)
        loss ← loss + loss_current                                    ▷ Loss is accumulated
        grad_center ← grad_center + gradc
        grad_outside ← grad_outside + grado
    end for
    return loss, grad_center, grad_outside                            ▷ Return loss and gradients
end procedure
```

(b) **[4 points (Coding)]** Complete the implementation for your SGD optimizer in the `sgd()` function in `src-word2vec/submission.py`.

(c) **[4 points (Coding)]** 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. 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` and the corresponding wordvectors as `sample_vectors_(soln).json` in your project directory.

You must upload the generated `sample_vectors_(soln).json` along with `src-word2vec/submission.py` to achieve full credit.

3 Quiz

This remainder of this homework is a series of multiple choice questions related to the word2vec algorithm. Please input your answers into the Gradescope Online Assessment `A2 Online Assessment`.

1. **[2 points (Online)]** Please input answer to question 1 of the Gradescope online assessment `A2 Online Assessment`.

   Choose all the equations that represent the differential of a sigmoid (When choosing the right option(s) consider the given input x to be a scalar instead of a vector).

   (a)
   $$\frac{\partial \sigma(x)}{\partial x} = \sigma(x) \cdot (1 - \sigma(x))$$

   (b)
   $$\frac{\partial \sigma(x)}{\partial x} = 1 - \sigma(x) \cdot (1 - \sigma(x))$$

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

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

2. **[1 point (Online)]** Please input answer to question 2 of the Gradescope online assessment `A2 Online Assessment` .

   What is the shape of the matrices U and V where (please refer to reading material from the handout):

   U is the outside vector matrix

   V is center vector matrix

   `num_tokens`: Number of unique words in the dataset

   `embed_size`: size of the word vector(word2vec size)

   (a) U : (`embed_size` $\times$ 1); V : (`embed_size` $\times$ `num_tokens`)
   (b) U : (`num_tokens` $\times$ `num_tokens`) ; V : (`embed_size` $\times$ 1)
   (c) U : (`embed_size` $\times$ `num_tokens`) ; V : (`embed_size` $\times$ `num_tokens`)
   (d) U : (`embed_size` $\times$ 1) ; V : (`embed_size` $\times$ 1)

3. **[2 points (Online)]** Please input answer to question 3 of the Gradescope online assessment `A2 Online Assessment`.

   What is the shape of the gradient $\frac{\partial J_{\text{naive-softmax}}}{\partial v_c}$ matrix/vector where:

   $$J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\boldsymbol{u}_o^\top \boldsymbol{v}_c + \log\left(\sum_{w' \in \text{Vocab}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_c)\right)$$

   $$\frac{\partial J}{\partial \boldsymbol{v}_c} = \boldsymbol{U}(\hat{\boldsymbol{y}} - \boldsymbol{y})$$

   U is the outside vector matrix

   $\hat{y}$ is the predicted output

y is the actual output

`num_tokens`: Number of unique words in the dataset

`embed_size`: size of the word vector(word2vec size)

(a) `num_tokens` $\times$ 1

(b) `embed_size` $\times$ 1

(c) `embed_size` $\times$ `num_tokens`

4. **[2 points (Online)]** Please input answer to question 4 of the Gradescope online assessment `A2 Online Assessment`.

What is the shape of the gradient $\frac{\partial J_{\text{naive-softmax}}}{\partial U}$ (will be referred to $\frac{\partial J}{\partial U}$ for simplicity) matrix/vector where:

$$J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\boldsymbol{u}_o^\top \boldsymbol{v}_c + \log\left(\sum_{w' \in \text{Vocab}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_c)\right)$$

$$\frac{\partial J}{\partial \boldsymbol{U}} = \boldsymbol{v}_c(\hat{\boldsymbol{y}} - \boldsymbol{y})^\top$$

V_c is the center vector

$\hat{y}$ is the predicted output

y is the actual output

`num_tokens`: Number of unique words in the dataset

`embed_size`: size of the word vector(word2vec size)

(a) `num_tokens` $\times$ 1

(b) `embed_size` $\times$ 1

(c) `embed_size` $\times$ `num_tokens`

5. **[1 point (Online)]** Please input answer to question 5 of the Gradescope online assessment `A2 Online Assessment` .

Which of the below equations represents a general form of SGD where $g(x)$ is the gradient of loss function and η is the learning rate:

(a)
$$x = x - \eta \cdot g(x)$$

(b)
$$x = x - \eta \cdot \frac{\partial g(x)}{\partial x}$$

(c)
$$x = x - \frac{\eta}{g(x)}$$

(d)
$$x = x - \eta \cdot \frac{\partial x}{\partial g(x)}$$

6. **[1 point (Online)]** Please input answer to question 6 of the Gradescope online assessment `A2 Online Assessment` .

Negative sampling was briefly introduced the Lecture 2 video Word2Vec: Model Variants. Here and here you can find more written information about negative sampling.

In skip-gram Word2Vec you have two ways of calculating the gradients. Namely, naive softmax and negative sampling. The goal of skip-gram Word2Vec is to accurately learn the representation of the conditional probability distribution(below is the equation):

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

Where:

$P(O = o|C = c)$ (i.e., the probability that o falls within the contextual window of c) represents the conditional probability

o is an 'outside' word for context word c.

u_o is the 'outside' vector representing outside word o.

v_c is the 'center' vector representing center word c.

Which of the following statements are true?

(a) In naive softmax, the gradient is dependent on summation across all classes. In the case of word2vec, this means summing across all the words in the vocabulary (as can be seen in the above equation). This tends to slow down the training process.

(b) In negative sampling, computation is cheaper. Instead of summing across all words (represented by uw in the above equation) we only sum across a few context/negative words(termed as negative samples).

(c) In negative sampling, the gradient is dependent on summation across all classes. In the case of word2vec, this means summing across all the words in the vocabulary(as can be seen in the above equation). This tends to slow down the training process.

4 Appendix (Extra Knowledge In Case You're Curious!)

Negative sampling is briefly introduced in Lecture 2: Word2Vec: Model Variants and an implementation is provided in the Assignment 2 coding assignment. For detailed notes on the math behind negative sampling, see below.

1. Negative Sampling loss 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:

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)) \tag{8}$$

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

Below we compute 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 $\boldsymbol{u}_k$.

$$\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{v}_c} &= -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \times \frac{\partial}{\partial \boldsymbol{v}_c}\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - \sum_{k=1}^{K} \frac{1}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} \times \frac{\partial}{\partial \boldsymbol{v}_c}\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c) && \text{(chain rule on } \log\text{)} \\
&= -\frac{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\boldsymbol{u}_o}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} - \sum_{k=1}^{K} -\frac{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)(1-\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\boldsymbol{u}_k}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} && \text{(chain rule on } \sigma\text{)} \\
&= -(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\boldsymbol{u}_o - \sum_{k=1}^{K} -(1-\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\boldsymbol{u}_k && \text{(cancel)} \\
&= (\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1)\boldsymbol{u}_o - \sum_{k=1}^{K} (\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c) - 1)\boldsymbol{u}_k && \text{(rearrange)}
\end{aligned}$$

Secondly:

$$\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{u}_o} &= \frac{\partial}{\partial \boldsymbol{u}_o}\Big(-\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\Big) \\
&= -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \times \frac{\partial}{\partial \boldsymbol{u}_o}\Big(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\Big) && \text{(chain rule on } \log\text{)} \\
&= -\frac{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\boldsymbol{v}_c}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} && \text{(chain rule on } \sigma\text{)} \\
&= -(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c))\boldsymbol{v}_c && \text{(cancel)} \\
&= (\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1)\boldsymbol{v}_c && \text{(rearrange)}
\end{aligned}$$

Thirdly, for all $k = 1, 2, \ldots, K$:

$$\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{u}_k} &= \frac{\partial}{\partial \boldsymbol{u}_k}\Big(-\log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\Big) \\
&= \frac{1}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} \times \frac{\partial}{\partial \boldsymbol{u}_k}\Big(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\Big) && \text{(chain rule on } \log\text{)} \\
&= \frac{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)(1-\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\boldsymbol{v}_c}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} && \text{(chain rule on } \sigma\text{)} \\
&= (1-\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))\boldsymbol{v}_c && \text{(cancel)}
\end{aligned}$$

The naive-softmax loss contains a summation over the entire vocabulary as part of computing the $P(O = o \mid C = c)$ term. Here, we don't do that calculation, approximating it with K samples (where K is much smaller than the vocabulary size).

[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.

2. 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:

$$\boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}) = \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}) \tag{9}$$

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 the implementation.

The proofs for the three partial derivatives are given below:

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

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

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

Note that the final derivatives of $\boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$ with respect to all the model parameters $\boldsymbol{U}$ and $\boldsymbol{V}$ are provided in the appendix part 1 derivation

Given a loss function J, we already know how to obtain the following derivatives

$$\frac{\partial J(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})}{\partial \boldsymbol{U}} \text{ and } \frac{\partial J(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})}{\partial \boldsymbol{v}_c}$$

Therefore, for skip-gram, the gradients for the loss of one context window can be expressed in terms of these:

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

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

$$\frac{\partial J_{\text{skip-gram}}(w_{t-m} \ldots w_{t+m})}{\partial \boldsymbol{v}_w} = \boldsymbol{0}, \text{when } w \neq c.$$

This handout includes space for every question that requires a written response. Please feel free to use it to handwrite your solutions (legibly, please). If you choose to typeset your solutions, the `README.md` for this assignment includes instructions to regenerate this handout with your typeset LaTeX solutions.

1.a

Starting from:

$$J_{\text{naive-softmax}}(v_c, o, U) = -u_o^T v_c + \log\left(\sum_{w \in \text{Vocab}} \exp(u_w^T v_c)\right)$$

I am taking the derivative with respect to v_c because that is the gradient we are looking for:

$$\frac{\partial J}{\partial v_c} = \frac{\partial}{\partial v_c}\left[-u_o^T v_c + \log\left(\sum_{w \in \text{Vocab}} \exp(u_w^T v_c)\right)\right]$$

For the first term:

$$\frac{\partial}{\partial v_c}(-u_o^T v_c) = -u_o$$

For the second term, using chain rule:

$$\frac{\partial}{\partial v_c}\log\left(\sum_{w \in \text{Vocab}} \exp(u_w^T v_c)\right) = \frac{1}{\sum_{w \in \text{Vocab}} \exp(u_w^T v_c)} \cdot \frac{\partial}{\partial v_c}\left(\sum_{w \in \text{Vocab}} \exp(u_w^T v_c)\right)$$

$$= \frac{1}{\sum_{w \in \text{Vocab}} \exp(u_w^T v_c)} \cdot \sum_{w \in \text{Vocab}} \exp(u_w^T v_c) \cdot u_w$$

$$= \sum_{w \in \text{Vocab}} \frac{\exp(u_w^T v_c)}{\sum_{w' \in \text{Vocab}} \exp(u_{w'}^T v_c)} \cdot u_w$$

The clue is to recognize that $\hat{y}_w = \frac{\exp(u_w^T v_c)}{\sum_{w' \in \text{Vocab}} \exp(u_{w'}^T v_c)}$ which I enter into the equation to get:

$$= \sum_{w \in \text{Vocab}} \hat{y}_w \cdot u_w = U\hat{y}$$

And therefore:

$$\frac{\partial J}{\partial v_c} = -u_o + U\hat{y} = U\hat{y} - Uy = U(\hat{y} - y)$$

where y is a one-hot vector with 1 at position o.

1.b

Starting from:

$$J_{\text{naive-softmax}}(v_c, o, U) = -u_o^T v_c + \log\left(\sum_{w' \in \text{Vocab}} \exp(u_{w'}^T v_c)\right)$$

The goal is to find $\frac{\partial J}{\partial u_w}$ for all $w \in$ Vocab.
Case 1: When $w = o$ (the true outside word)

$$\frac{\partial J}{\partial u_o} = \frac{\partial}{\partial u_o}\left[-u_o^T v_c + \log\left(\sum_{w' \in \text{Vocab}} \exp(u_{w'}^T v_c)\right)\right]$$

First term: $\frac{\partial}{\partial u_o}(-u_o^T v_c) = -v_c$
Second term: Using chain rule (similar to 1.a),

$$\frac{\partial}{\partial u_o} \log\left(\sum_{w' \in \text{Vocab}} \exp(u_{w'}^T v_c)\right) = \frac{1}{\sum_{w' \in \text{Vocab}} \exp(u_{w'}^T v_c)} \cdot \exp(u_o^T v_c) \cdot v_c$$

$$= \frac{\exp(u_o^T v_c)}{\sum_{w' \in \text{Vocab}} \exp(u_{w'}^T v_c)} \cdot v_c = \hat{y}_o \cdot v_c$$

Therefore: $\frac{\partial J}{\partial u_o} = -v_c + \hat{y}_o v_c = (\hat{y}_o - 1)v_c$
Case 2: When $w \neq o$

$$\frac{\partial J}{\partial u_w} = \frac{\partial}{\partial u_w}\left[\log\left(\sum_{w' \in \text{Vocab}} \exp(u_{w'}^T v_c)\right)\right]$$

$$= \frac{1}{\sum_{w' \in \text{Vocab}} \exp(u_{w'}^T v_c)} \cdot \exp(u_w^T v_c) \cdot v_c = \hat{y}_w \cdot v_c$$

In matrix form: $\frac{\partial J}{\partial U} = v_c(\hat{y} - y)^T$