CS 224n Assignment #2: word2vec (49 Points)

Due on Tuesday Jan. 23, 2024 by **4:30pm (before class)**
Please do not forget to tag questions on Gradescope

1 Understanding word2vec (31 points)

Recall that the key insight behind `word2vec` is that *'a word is known by the company it keeps'*. Concretely, consider a 'center' word c surrounded before and after by a context of a certain length. We term words in this contextual window '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':

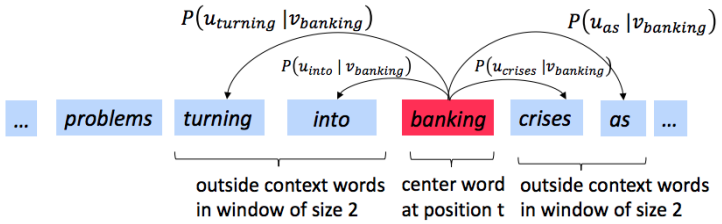

Figure 1: The word2vec skip-gram prediction model with window size 2

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:

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

For each word, we learn vectors u and v, where $\boldsymbol{u}_o$ is the 'outside' vector representing outside word o, and $\boldsymbol{v}_c$ is the 'center' vector representing center word c. We store these parameters in 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:

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

We can view this loss as the cross-entropy[2] 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.[3] The predicted distribution $\hat{\boldsymbol{y}}$ is the probability distribution $P(O|C = c)$ given by our model in equation (1).

Note: Throughout this homework, when computing derivatives, please use the method reviewed during the lecture (i.e. no Taylor Series Approximations).

[1] 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 $\boldsymbol{U}$ 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 word k and the index of word k.**

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

[3] Note that the true conditional probability distribution of context words for the entire training dataset would not be one-hot.

(a) (2 points) Prove that the naive-softmax loss (Equation 2) is the same as the cross-entropy loss between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$, i.e. (note that $\boldsymbol{y}$ (true distribution), $\hat{\boldsymbol{y}}$ (predicted distribution) are vectors and $\hat{\boldsymbol{y}}_o$ is a scalar):

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

Your answer should be one line. You may describe your answer in words.

Answer: $\boldsymbol{y}_w$ is zero for all $\boldsymbol{w}$ except for $\boldsymbol{o}$, and $\boldsymbol{y}_o$ is 1

(b) (6 points)

(i) 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 $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, $\boldsymbol{U}$, and show your work to receive full credit.

- **Note**: Your final answers for the partial derivative should follow the shape convention: the partial derivative of any function $f(x)$ with respect to x should have the **same shape** as x.[4]
- Please provide your answers for the partial derivative in vectorized form. For example, when we ask you to write your answers in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{U}$, you may not refer to specific elements of these terms in your final answer (such as $\boldsymbol{y}_1$, $\boldsymbol{y}_2$, ...).

(ii) When is the gradient you computed equal to zero?
Hint: You may wish to review and use some introductory linear algebra concepts.

(iii) The gradient you found is the difference between the two terms. Provide an interpretation of how each of these terms improves the word vector when this gradient is subtracted from the word vector v_c.

Answer:

(i)

$$\begin{aligned}
\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) &= -\log P(O = o \mid C = c) = -\sum_{w \in \text{Vocab}} \boldsymbol{y}_w \log(\hat{\boldsymbol{y}}_w) = -\log(\hat{\boldsymbol{y}}_o) \\
&= -\boldsymbol{u}_o^\top \boldsymbol{v}_c + \log\Big(\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)\Big)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{v}_c} &= -\boldsymbol{u}_o^\top + \frac{\sum_{w \in \text{Vocab}} \boldsymbol{u}_w^\top \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \\
&= -\boldsymbol{u}_o^\top + \sum_{w \in \text{vocab}} P(O = o \mid C = c) \boldsymbol{u}_w^\top \\
&= -\boldsymbol{u}_o^\top + \sum_{w \in \text{vocab}} \hat{\boldsymbol{y}}_w \boldsymbol{u}_w^\top \\
&= \sum_{w \in \text{vocab}} (\hat{\boldsymbol{y}}_w - \boldsymbol{y}_w) \boldsymbol{u}_w^\top \\
&= (\boldsymbol{U}(\hat{\boldsymbol{y}} - \boldsymbol{y}))^\top
\end{aligned}$$

[4] This allows us to efficiently minimize a function using gradient descent without worrying about reshaping or dimension mismatching. While following the shape convention, we're guaranteed that $\theta := \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$ is a well-defined update rule.

(ii) The gradient is zero when the predicted distribution equals the true distribution
(iii) When the gradient is subtracted from the word vector:

$$\mathbf{v}_c \leftarrow \mathbf{v}_c - \eta \frac{\partial J_{\text{naive-softmax}}}{\partial \mathbf{v}_c}$$
$$\mathbf{v}_c \leftarrow \mathbf{v}_c - \eta(U^\top \hat{y} - U^\top y)$$

This update has the following effects:
Corrective Adjustment: The term $\boldsymbol{Uy}$ ensures that $\mathbf{v}_c$ is adjusted to better reflect the true outside word. It acts as a corrective force.
Prediction Refinement: The term $\boldsymbol{U\hat{y}}$ represents the model's current prediction. By subtracting this term, we correct the model's current over- or under-estimations.
Gradient Descent: The overall effect is a gradient descent step that minimizes the cross-entropy loss, thereby refining the center word vector $\mathbf{v}_c$ to improve future predictions.

(c) (1 point) In many downstream applications using word embeddings, L2 normalized vectors (e.g. $\mathbf{u}/||\mathbf{u}||_2$ where $||\mathbf{u}||_2 = \sqrt{\sum_i u_i^2}$) are used instead of their raw forms (e.g. $\mathbf{u}$). Let's consider a hypothetical downstream task of binary classification of phrases as being positive or negative, where you decide the sign based on the sum of individual embeddings of the words. When would L2 normalization take away useful information for the downstream task? When would it not?

Hint: Consider the case where $\mathbf{u}_x = \alpha \mathbf{u}_y$ for some words $x \neq y$ and some scalar α. Give examples of words x and y which satisfy the given relation and why would they be affected/not affected due to the normalization?

Answer: L2 normalization, which involves dividing each vector by its Euclidean norm, is often used in downstream applications involving word embeddings. In a hypothetical downstream task like binary classification of phrases as positive or negative based on the sum of individual word embeddings, L2 normalization could potentially affect the task's performance.

When L2 normalization might take away useful information:

1. Magnitude-based distinctions: If the magnitude of word embeddings carries important information for the downstream task, L2 normalization might remove this information. For instance, if the intensity or importance of a word in a phrase is correlated with the magnitude of its embedding, normalization could blur or remove these distinctions.

2. Semantic contrasts: In cases where words with different meanings have embeddings with different magnitudes, L2 normalization might hinder the model's ability to distinguish between them. For example, if "good" and "great" have different magnitudes in their raw embeddings, normalizing them could make them more similar in magnitude, potentially affecting the model's ability to differentiate between them.

When L2 normalization might not affect useful information:

1. Semantic similarity: In tasks where semantic relationships between words are more important than their individual magnitudes, L2 normalization might not significantly impact performance. For example, in tasks like semantic similarity or word analogy, where the relative distances between embeddings matter more than their absolute magnitudes, normalization may not cause much loss of information.

2. Relative contributions: If the downstream task is robust to variations in the magnitude of individual word embeddings and focuses more on their relative contributions to the overall phrase representation, L2 normalization might not be detrimental. For instance, in sentiment analysis where the sentiment of a phrase

is determined by the combination of positive and negative words, the relative contributions of words may matter more than their individual magnitudes.

Example: Consider the words "big" and "huge" where the embedding of "huge" is a scaled version of the embedding of "big", i.e., $\boldsymbol{u}_{huge} = \alpha \boldsymbol{u}_{big}$ for some scalar α. L2 normalization would affect these embeddings because it would make the direction of "big" and "huge" identical, potentially making it harder for the downstream task to differentiate between them based on their magnitudes. However, if the downstream task is more concerned with the semantic similarity between "big" and "huge" rather than their individual magnitudes, normalization might not be detrimental.

(d) (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 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. Show your work to receive full credit.

Answer: The answer is on the handwritten file.

(e) (1 point) 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 the column vectors $\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_1}$, $\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_2}$, $\cdots$, $\frac{\partial \boldsymbol{J}(\boldsymbol{v}_c, o, \boldsymbol{U})}{\partial \boldsymbol{u}_{|\text{Vocab}|}}$. No derivations are necessary, just an answer in the form of a matrix.

Answer: The answer is on the handwritten file.

(f) (2 points) The Leaky ReLU (Leaky Rectified Linear Unit) activation function is given by Equation 4 and Figure 2:

$$f(x) = \max(\alpha x, x) \tag{4}$$

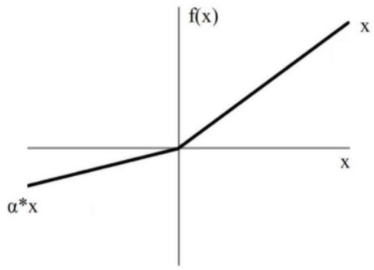

Figure 2: Leaky ReLU

Where x is a scalar and $0 < \alpha < 1$, please compute the derivative of $f(x)$ with respect to x. You may ignore the case where the derivative is not defined at 0.[5]

Answer: The answer is on the handwritten file.

[5]If you're interested in how to handle the derivative at this point, you can read more about the notion of subderivatives.

(g) (3 points) The sigmoid function is given by Equation 5:

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{5}$$

Please compute the derivative of $\sigma(x)$ with respect to x, where x is a scalar. Please write your answer in terms of $\sigma(x)$. Show your work to receive full credit.

Answer: The answer is on the handwritten file.

(h) (6 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 $\boldsymbol{u}_{w_1}, \boldsymbol{u}_{w_2}, \ldots, \boldsymbol{u}_{w_K}$.[6] 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:

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{s=1}^{K} \log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)) \tag{6}$$

for a sample $w_1, \ldots w_K$, where $\sigma(\cdot)$ is the sigmoid function.[7]

(i) Please repeat parts (b) and (d), 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 $s \in [1, K]$. Show your work to receive full credit. **Note:** you should be able to use your solution to part (g) to help compute the necessary gradients here.

(ii) In the lecture, we learned that an efficient implementation of backpropagation leverages the reuse of previously computed partial derivatives. Which quantity could you reuse (we store the common terms in a single matrix/vector) amongst the three partial derivatives calculated above 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.[8] Additional terms and functions (other than $\boldsymbol{U}_{o,\{w_1,\ldots,w_K\}}$ and $\mathbf{1}$) can be used in your solution.

(iii) 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.

[6] Note: In the notation for parts (g) and (h), we are using words, not word indices, as subscripts for the outside word vectors.
[7] 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.
[8] Note: NumPy will automatically broadcast 1 to a vector of 1's if the computation requires it, so you generally don't have to construct $\mathbf{1}$ on your own during implementation.

Answer:
(i) The answer is on the handwritten file.
(ii) The answer is on the handwritten file.
(iii) The negative sampling loss function is more efficient to compute than the naive-softmax loss because it only updates a small subset of parameters corresponding to the positive and negative samples, rather than updating all parameters for each training instance.

(i) (2 points) 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:

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{s=1}^{K} \log(\sigma(-\boldsymbol{u}_{w_s}^\top \boldsymbol{v}_c)) \tag{7}$$

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 $s \in [1, K]$. Show your work to receive full credit. 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, such as in Equation 8.

Answer: The answer is on the handwritten file.

(j) (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:

$$\boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}) = \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}) \tag{8}$$

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.

Answer: The answer is on the handwritten file.

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

2 Implementing word2vec (18 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. **Windows users** may wish to install the Linux Windows Subsystem[9]. Also note that you probably want to finish the previous math section before writing the code since you will be asked to implement the math functions in Python. You'll probably want to implement and test each part of this section in order, since the questions are cumulative.

```
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
```

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. We will be checking the efficiency of your code. You will be able to see the results of the autograder when you submit your code to `Gradescope`, we recommend submitting early and often.
Note: If you are using Windows and have trouble running the .sh scripts used in this part, we recommend trying Gow or manually running commands in the scripts.

(a) (12 points) 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 sigmoid method by running `python word2vec.py sigmoid`.

(i) Implement the `sigmoid` method, which takes in a vector and applies the sigmoid function to it.

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

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

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

When you are done, test your entire implementation by running `python word2vec.py`.

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

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

(c) (2 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 and the compute power of your machine ***(an efficient implementation takes one to two hours)****. 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.** In at most three sentences, briefly explain what you see in the plot. This may include, but is not limited to, observations on clusters and words that you expect to cluster but do not.

Answer: The plot is right below the answer space.

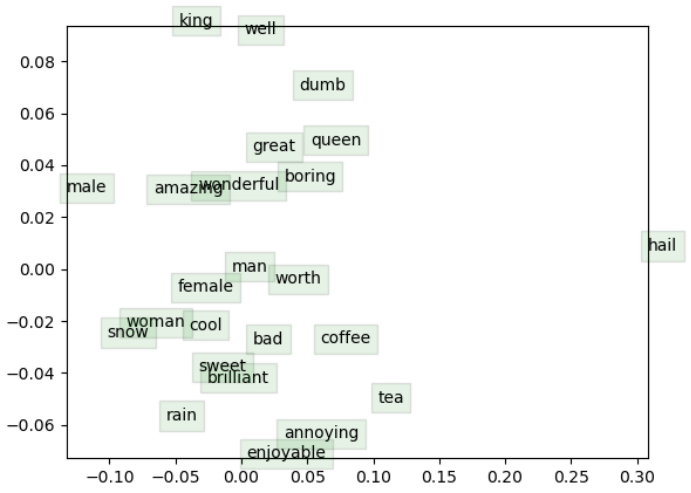

Figure 3: The plot of word vectors by word2vec skip-gram prediction model

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]".