IFT6289-A-H26 Assignment 1: Skip-gram Word2Vec

- *Due Date:* ***February 20, 2026 (23:59 pm, EST timezone)***
- *Assignment 1 should be completed by individuals, which is worth* ***15%*** *of your grade.*
- *Maximum* ***2*** *late days. Late submissions would result in a lower grade.*
- *Please Submit your code(.zip) and report (PDF) on Gradescope.*

In this assignment, you will implement the skip-gram word2vec model. Then, you will train the model on the Penn Treebank dataset and analyze the results.

The skip-gram word2vec model

Let us wrap up the skip-gram word2vec algorithm, which tries to guess the neighboring words using the current center word. Specifically, we have an 'input' word w_i, and the neighboring words are called 'output' words, which lie in the contextual window, where the input word is at the center of the window.

For example, in Figure 1, when 'a' is considered as the 'input word', the 'output' words are 'not', 'make', 'machine', 'in' with a contextual window size of 2.

Figure 1: The word2vec prediction model with a contextual window size of 2

Conditional probability. In the skip-gram word2vec model, we want to optimize the conditional probability of each 'output' word w_o given the 'input' word w_i. The conditional probability distribution is given by equation 1.

$$p(w_o|w_i) = \frac{\exp\left(\boldsymbol{u}_o^\top \boldsymbol{v}_i\right)}{\sum_{w\in\text{Vocab}} \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_i\right)} \tag{1}$$

Where $\boldsymbol{u}_o$ and $\boldsymbol{v}_i$ are the representation vectors (or embedding vectors) for the 'output' word w_o and the 'input' word w_i, respectively. These embedding vectors come from two matrices: an Embedding matrix $\mathbf{V}$ and a Context matrix $\mathbf{U}$, where each column of the matrix is an embedding vector for a word. For the 'input' word, we look in the Embedding matrix $\mathbf{V}$. For the 'output' word, we look in the Context matrix $\mathbf{U}$.

Both $\mathbf{U}$ and $\mathbf{V}$ contain the vectors of all words in the vocabulary. A vocabulary is typically a mapping from "tokens" to consecutive integers, where a token in our scenario is a word, and the corresponding integer acts as the index to retrieve the token's embedding from the embedding matrix.

The naïve softmax loss. First, we compute the classical cross-entropy loss based on the conditional probability $p(w_o|w_i)$, that is, we predict the output token with a softmax on the complete vocabulary. This is the original skip-gram word2vec training objective.

More specifically, for an input word w_i and an output word w_o, the naive softmax loss function is shown in equation 2:

$$\mathbf{L}_{\text{naive}}(\boldsymbol{v}_i, w_o, \mathbf{U}) = -\log p(w_o|w_i) = -\sum_{w \in \text{Vocab}} y_w \log(\hat{y}_w) = -\log(\hat{y}_o) \tag{2}$$

Where $\boldsymbol{y}$ is the true distribution, and $\hat{\boldsymbol{y}}$ is the predicted distribution of the output word w_o with softmax. The true distribution $\boldsymbol{y}$ is a one-hot vector with a 1 for the true output word, and 0 for the other words. $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ have the same length, which equal to the number of words in the vocabulary.

The negative sampling loss. Compared to the naïve softmax loss, a more efficient training objective is the negative sampling word2vec objective proposed by Mikolov et al. [2013].

Assume there are K negative samples (tokens) from the vocabulary, $\{w_1, w_2, ..., w_K\}$, which satisfies $w_o \notin \{w_1, w_2, ..., w_K\}$. The negative sampling loss will be written as equation 3.

$$\mathbf{L}_{\text{neg\_sampling}}(\boldsymbol{v}_i, w_o, \mathbf{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_i)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_i)) \tag{3}$$

Where $\sigma(\cdot)$ is the sigmoid function, which is given by equation 4:

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

The skip-gram loss For the skip-gram loss, it adds the losses for each single token in the contextual window:

$$\mathbf{L}_{\text{skip\_gram}}(\boldsymbol{v}_i, w_{t-m}, \ldots, w_{t+m}, \mathbf{U}) = \sum_{-m \le j \le m, j \neq 0} \mathbf{L}(\boldsymbol{v}_c, w_{t+j}, \mathbf{U}) \tag{5}$$

Where $i = w_t$ is the input center word, and $\mathbf{L}(\cdot)$ can be any training objective such as the naïve softmax loss or the negative sampling loss.

Part 1: Derive the gradients for training

In the first part of the assignment, you need to derive the gradients for the naïve softmax loss and the negative sampling loss. **Write the steps and results in your report.**

1. (1pt) Derive the gradient of the naïve softmax loss w.r.t. the input word vector $\frac{\partial \mathbf{L}_{\text{naive}}(\boldsymbol{v}_i, w_o, \mathbf{U})}{\partial \boldsymbol{v}_i}$. Please show the result in the report in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$ and $\mathbf{U}$.

2. (1pt) Derive the gradient of the naïve softmax loss w.r.t. each outside word vectors $\frac{\partial \mathbf{L}_{\text{naive}}(\boldsymbol{v}_i, w_o, \mathbf{U})}{\partial \boldsymbol{u}_w}$. There are two cases: when $w = o$, which refers the true 'output' word vector, and when $w \neq o$ for the other word vectors. Please show the result in the report in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$ and $\boldsymbol{v}_i$.

3. (1pt) Derive the gradient of the negative sampling loss w.r.t. the input word vector $\frac{\partial \mathbf{L}_{\text{neg\_sampling}}(\boldsymbol{v}_i, w_o, \mathbf{U})}{\partial \boldsymbol{v}_i}$. Please show your result in the report in terms of $\boldsymbol{v}_i$, $\boldsymbol{u}_o$ and $\boldsymbol{u}_k$.

4. (1pt) Derive the gradient of the negative sampling loss w.r.t. each true output word vectors $\frac{\partial \mathbf{L}_{\text{neg\_sampling}}(\boldsymbol{v}_i, w_o, \mathbf{U})}{\partial \boldsymbol{u}_o}$. Please show your result in the report in terms of $\boldsymbol{v}_i$, $\boldsymbol{u}_o$ and $\boldsymbol{u}_k$.

5. (1pt) Derive the gradient of the negative sampling loss w.r.t. a negative word vector $\frac{\partial \mathbf{L}_{\text{neg\_sampling}}(\boldsymbol{v}_i, w_o, \mathbf{U})}{\partial \boldsymbol{u}_k}$, where $k \in [1, K]$. Please show your result in the report in terms of $\boldsymbol{v}_i$, $\boldsymbol{u}_o$ and $\boldsymbol{u}_k$.

Part 2: Implement skip-gram word2vec

In this part, you will implement the skip-gram word2vec based on the given code and the derivatives you derived. Writing explainable comments for your code is not only a good behavior for coding, but also will be easier for grading.

1. (0.5pt) Complete the `sigmoid()` function in `utils/utils.py`.
2. Complete `word2vec.py`.
   (a) (1.5pt) Implement the `naive_softmax_loss_and_gradient()` function , which returns three values:
      i. The naïve softmax loss $\mathbf{L}_{\text{naive}}(\boldsymbol{v}_i, w_o, \mathbf{U})$;
      ii. The gradient w.r.t. the input word vector $\frac{\partial \mathbf{L}_{\text{naive}}(\boldsymbol{v}_i, w_o, \mathbf{U})}{\partial \boldsymbol{v}_i}$;
      iii. The gradient w.r.t. all output word vectors $\frac{\partial \mathbf{L}_{\text{naive}}(\boldsymbol{v}_i, w_o, \mathbf{U})}{\partial \mathbf{U}}$.
   (b) (1.5pt) Implement the `neg_sampling_loss_and_gradient()` function, which returns three values:
      i. The negative sampling loss $\mathbf{L}_{\text{neg\_sampling}}(\boldsymbol{v}_i, w_o, \mathbf{U})$;
      ii. The gradient w.r.t. the input word vector $\frac{\partial \mathbf{L}_{\text{neg\_sampling}}(\boldsymbol{v}_i, w_o, \mathbf{U})}{\partial \boldsymbol{v}_i}$;
      iii. The gradient w.r.t. all output word vectors $\frac{\partial \mathbf{L}_{\text{neg\_sampling}}(\boldsymbol{v}_i, w_o, \mathbf{U})}{\partial \mathbf{U}}$.
   (c) (2pt) Implement the `skipgram()` function.
3. (0.5pt) Complete the `sgd()` function in `sgd.py` file.

Part 3: Analyzing your word vectors

(4pt) In this part, you will train your skip-gram word2vec model on the training set of the Penn Treebank dataset and analyze your results. **Write your analysis in your report.**

1. Train your skip-gram word2vec using SGD by running `run.py`. This would train the model for $35,000$ iterations, and will generate visualizations for some example words (specified in `utils/utils.py`) every $5,000$ iterations.[1] Now analyze the results and **include both the plots and your analysis in the report**:
   (a) What's your expectations of the characteristics of these word vectors? This can include but not limit to the clusters and the arithmetic characteristics.
   (b) Are the results in line with your expectation?
2. The Penn Treebank dataset uses several approaches to reduce the vocabulary size. Observe the Penn Treebank corpus in `data/penntreebank/sentences.txt` and **include your analysis in the report**:
   (a) Penn Treebank only keeps the most frequent 10K words and discards the remaining with an ⟨UNK⟩, which stands for unknown. What are the positive and negative consequences of this approach? Can you come up with ways that can reduce ⟨UNK⟩s while not introducing too much computational overhead? (*Hint: think about the definition of "tokens" instead of words!*)
   (b) Penn Treebank substitudes all numbers with an N. What are the positive and negative consequences of this approach? (Brainstorming: Can you come up with ways that can treat numbers in the corpus reasonably?)

Writing your report

1. The template of the assignment 1 report is the same as that for the reading assignment, which can be downloaded from StudiUM or slack.
2. Please indicate in your report if you used any open-source codes or materials.

[1] Training on Penn Treebank may take 0.5 – 1 hour, depending on your CPU hardware.

Submission Instructions

Submit your PDF report and code (.zip file) in on Gradescope. When submitting the code, please zip all files in your project, **instead of** zipping the project's root folder.

References

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.