DNLP Assignment #1 (57 Points)

Due on Monday Oct. 20, 2025 by **11:59pm**

This assignment consists of two parts. In part 1, you will be asked to solve exercise problems. In part 2, you will implement the word2vec models. Please submit your report of part 1 and part 2 as a single pdf file and name it 'assignment1_written.pdf'. For part 2, submit your implementations as a zip file named 'assignment1_code.zip'. Please bundle the 'assignment1_written.pdf' and 'assignment1_code.zip', and submit as one zip file with name 'student#_name.zip' (ex. 2025XXXX_gildonghong.zip).

Part 1

1 Understanding word2vec (23 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':

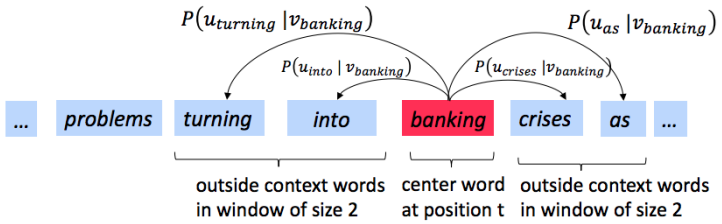

Figure 1: The word2vec skip-gram prediction model with window size 2

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:

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

[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.**

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).

(a) (3 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 $\boldsymbol{y}, \hat{\boldsymbol{y}}$ 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) (5 points) 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 $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, $\boldsymbol{U}$.

(c) (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.

[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.

(d) (3 points) The sigmoid function is given by Equation 4:

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

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

(e) (7 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 $\boldsymbol{u}_{w_1}, \boldsymbol{u}_{w_2}, \ldots, \boldsymbol{u}_{w_K}$.[4] 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:

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

for a sample $w_1, \ldots w_K$, where $\sigma(\cdot)$ is the sigmoid function.[5]

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

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

[4]Note: In the notation for part (e), we are using words, not word indices, as subscripts for the outside word vectors.

[5]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 Problems (16 points)

2.1 Multi-choice (1 point each)

For each of the following questions, choose the single correct answer and provide a brief **explanation**.

(1) What does the 'Sparsity Problem' mean, one of the main limitations of N-gram language models?

(a) The issue where the computation speed decreases exponentially as the model size increases.

(b) The issue of predicting a probability of 0 for an N-gram that did not appear in the training data (corpus).

(c) The issue where the model has too many parameters, leading to a shortage of storage space.

(d) The issue where the N-gram model fails to capture the long-term dependency of a sentence.

Explanation:

(2) What is the fundamental cause of the 'Vanishing Gradient' problem in a Recurrent Neural Network (RNN)?

(a) Because a single weight matrix is used repeatedly at every timestep.

(b) Because information is lost as the hidden state is continuously overwritten.

(c) Because gradients with values less than 1 are repeatedly multiplied during the backpropagation process.

(d) Because the gradient calculation becomes unstable as the number of model parameters becomes too large.

Explanation:

(3) What is the main reason for using a non-linear activation function in a neural network?

(a) To increase the learning speed of the model.

(b) To reduce the number of model parameters and prevent overfitting.

(c) To enable the neural network to represent functions more complex than a linear transformation, even when stacking many layers.

(d) To constrain the output values to a range between 0 and 1 for interpretation as probabilities.

Explanation:

(4) In a Seq2Seq model, what is the problem called where the encoder's final hidden state vector must compress all the information from the source sentence?

(a) Overfitting

(b) Sparsity

(c) Information bottleneck

(d) Exploding Gradient

Explanation:

(5) You are training a 4-gram language model. Given the following counts from a corpus, what is the probability of the word 'books' appearing after 'students opened their', i.e., $P(\text{books}|\text{students opened their})$?

- `count(students opened their)` = 1000

- `count(students opened their books)` = 250

(a) 0.40

(b) 4.00

(c) 0.25

(d) 0.10

Explanation:

2.2 True/False (1 point each)

For each of the following statements, indicate whether it is **true** or **false**.

(6) The CBOW (Continuous Bag of Words) model in word2vec is trained by using a center word to predict its surrounding context words.

(7) In Recurrent Neural Networks (RNNs), the Gradient Clipping technique is used to solve the Vanishing Gradient problem.

(8) An LSTM can effectively preserve long-term information by maintaining a separate cell state.

(9) Stacking multiple layers with a Linear Activation Function in a neural network can create non-linear decision boundaries.

(10) The BLEU score is a metric that directly measures the grammatical perfection of a machine translation's output.

2.3 Descriptive Question (2 points each)

(11) Unlike fixed-window models, what is the key architectural reason that Recurrent Neural Networks (RNNs) can process variable-length sequences?

(12) A standard Seq2seq model is often limited by an "information bottleneck" because it must rely on a single, fixed-size vector from the encoder. Explain how the Attention mechanism fundamentally overcomes this limitation to better handle long input sentences.

(13) Standard RNNs struggle to maintain information over many time steps due to the vanishing gradient problem. Describe how the introduction of the 'cell state' and gate mechanisms in an LSTM's architecture specifically addresses this challenge of learning long-term dependencies.

Part 2

3 Coding: 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.

```
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
```

(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`.

(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 report.** 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.

* Run the `collect_submission.sh` script to produce your `assignment1_code.zip` file.