Assignment 1

Deep Learning Natural Language Processing (DNLP) – CSED554

Introduction

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 as a pdf and name the file "Assignment1_written".

In part 2, submit your implementations as zip file "assignment1_code"
Please bundle the "assignment1_written.pdf" and "assignment1_code.zip", and submit as one zip file with name '*your student number_your name*' (ex. 2023XXXX_이채빈.zip or 2023XXXX_ChaebinLee.zip).

Part 1 (40 points)

Problem 1 (20 points)

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*.

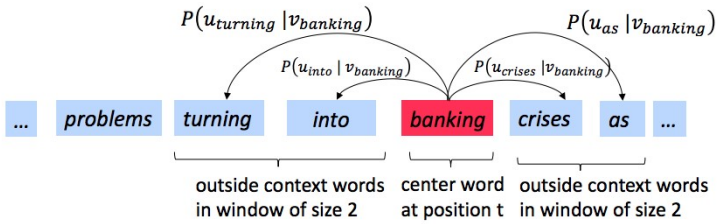

Figure 1: The word2vec skip-gram prediction model with window size 2

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

$$\boldsymbol{P(O = o|C = c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in Vocab} \exp(u_w^T v_c)}} \quad (1)$$

Here, u_o is the 'outside' vector representing outside word *o*, and v_c is the 'center' vector representing center word *c*. To contain these parameters, we have two matrices, *U* and *V* . The columns of *U* are all the 'outside' vectors

u_w. The columns of V are all of the 'center' vectors v_w. Both U and 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:

$$\boldsymbol{J_{\text{naïve-softmax}}(v_c, o, U) = -\log P(O = o|C = c)} \qquad (2)$$

Another way to view this loss is as the cross-entropy[2] between the true distribution y and the predicted distribution $y\hat{}$. Here, both y and $\hat{y}$ 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 y is a one-hot vector with a 1 for the true outside word o, and 0 everywhere else. The predicted distribution $\hat{y}$ is the probability distribution $P(O|C = c)$ given by our model in equation (1).

(a) (3 points) Show that the naive-softmax loss given in Equation (2) is the same as the cross-entropy loss between y and $\hat{y}$; i.e., show that

$$-\sum_{w \in \text{Vocab}} y_w \log(\hat{y}_w) = -\log(\hat{y}_o) \qquad (3)$$

Your answer should be one line.

(b) (5 points) Compute the partial derivative of $J_{\text{naive-softmax}}(v_c, o, U)$ with respect to v_c. Please write your answer in terms of y, $\hat{y}$, and U.

(c) (5 points) Compute the partial derivatives of $J_{\text{naive-softmax}}(v_c, o, U)$ with respect to each of the 'outside' word vectors, 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 you answer in terms of y, $y\hat{}$, and v_c.

---

[1] Assume that every word in our vocabulary is matched to an integer number k. u_k is both the k^{th} column of U and the 'outside' word vector for the word indexed by k. v_k is both the k^{th} column of 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 $-\sum_i p_i \log(q_i)$.

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

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

Please compute the derivative of $\sigma(x)$ with respect to x, where x is a vector.

(e) (4 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 $w_1, w_2, \dots, w_K$ and their outside vectors as $u_1, \dots, u_K$.
Note that $o \notin \{w_1, \dots, w_K\}$. For a center word c and an outside word o, the negative sampling loss function is given by:

$$J_{\text{neg-sample}}(v_c, o, U) = -\log\left(\sigma(u_o^T v_c)\right) - \sum_{k=1}^{K\log} \left(\sigma\left(-u_k^T v_c\right)\right) \qquad (5)$$

for a sample $w_1, w_2, \dots, w_K$, where $\sigma(\cdot)$ is the sigmoid function.

Please repeat parts (b) and (c), computing the partial derivatives of $J_{\text{neg-sample}}$ with respect to v_c, with respect to u_o, and with respect to a negative sample u_k. Please write your answers in terms of the vectors u_o, v_c and u_k, where$k \in [1, K]$. After you've done this, describe with one sentence why this loss function is much more efficient to compute than the naive-softmax loss. Note, you should be able to use your solution to part (d) to help compute the necessary gradients here.

Problem 2 (5 points)

For each of the following questions, check all the square you think are correct. No explanations are required.

(a) (1 points) Which of the following statements is true of language models?
- ☐ Neural window-based models share weights across the window.
- ☐ Neural window-based language models suffer from the sparsity problem.
- ☐ The number of parameters in an RNN language model grows with the number of time steps.
- ☐ Neural window-based models can be parallelized, but RNN language models cannot.

(b) (1 points) If you are training a neural network for classification and you observe that the training loss is significantly lower than the validation loss, what are some methods that can be used to resolve this issue? Please choose all applicable options.
- ☐ Use a network with fewer layers.
- ☐ Decrease dropout probability.
- ☐ Increase L2 regularization weight.
- ☐ Increase the size of each hidden layer.

(c) (1 points) If a classifier assigns the same probability to all possible classes, what would be the value of cross-entropy error for a single instance when there are 5 classes?
- ☐ $-\log(5)$
- ☐ $-0.2\log(2)$
- ☐ $-\log(0.2)$
- ☐ $-5\log(0.2)$

(d) (1 points) When the gradient is propagated backwards through the sigmoid or tanh non-linearities, it is impossible for it to switch signs.
- ☐ True
- ☐ False

(e) (1 points) We are using stochastic gradient descent on minibatches to train a neural network. Is the following statement true or false: If we divide the learning rate by the minibatch size while summing the cost across the minibatch, it would be equivalent to averaging the cost across the minibatch.
- ☐ True
- ☐ False

Problem 3 (9 points)

In lectures you were introduced to transition-based dependency parsing. Another approach to sentence parsing is graph-based dependency parsing. This method receives a sentence $[w_1, w_2, \dots, w_T]$ as input and converts it into a sequence of d-dimensional vectors $[h_1, h_2, \dots, h_T]$ using bidirectional LSTM. Then, it assigns a score $s(i, j)$ to each possible dependency $i \rightarrow j$, representing the connection between word w_i and word w_j in the dependency graph. The score is computed using $s(i, j) = h_i^T A h_j$ where A is $d \times d$ weight matrix. Additionally, there is a score for having an edge going from ROOT to a word w_j, given as $s(ROOT, j) = w^T h_j$ where w is a d-dimensional vector of weights. Lastly, the model assigns the head to each word j whose edge scores the highest among $\text{argmax}_{x \in [1,\dots,T]} s(i, j)$

(a) (3 points) Can this model generate a type of parse tree that is not possible with a transition-based dependency parser? If yes, what is the name of this type of tree?

(b) (3 points) Suppose the model calculates the score of each edge using a simple dot product: $s(i, j) = h_i^T h_j$. Can you explain why this method may not perform as effectively as the previously mentioned method?

(c) (3 points) What is a drawback of using graph-based dependency parsers when compared to transition-based dependency parsers?

Problem 4 (6 points)

RNNs are a flexible option! During our lesson, we were informed that this group of artificial neural networks has numerous crucial benefits and can be applied to various assignments. They are frequently utilized in many state-of-the-art architectures for natural language processing (NLP).

For each of the following tasks, state how you would run an RNN to do that task. In particular, specify how the RNN would be used at test time (not training time), and specify

1. how many outputs (i.e. number of times the soft-max $\hat{y}^{(t)}$ is called from your RNN. If the number of outputs is not fixed, state it as *arbitrary*)
2. what each $\hat{y}^{(t)}$ is a probability distribution over (e.g. distributed over all species of cats)
3. which inputs are fed at each time step to produce each output

The inputs are specified below

i. (3points) Named-Entity Recognition: For each word in a sentence, classify that word as either a person, organization, location, or none.

   Inputs: A sentence containing n words.

ii. (3 points) Language models: generating text from a chatbot that was trained to speak like you by predicting the next word in the sequence.

Input: A single start word or token that is fed into the first time step of the RNN.

Part 2 – Coding: Implementing word2vec (20 points)

You will create and train your own word vectors using stochastic gradient descent (SGD) by implementing the word2vec model in this section. 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) First, implement the `sigmoid` function in `word2vec.py`, which will enable you to apply it to an input vector. Within the same file, complete the implementation of the softmax and negative sampling loss and gradient functions. Then, fill in the implementation of the loss and gradient functions for the ski-gram model. Once you have completed these tasks, run `python word2vec.py` to test your code.

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

(c) (4 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 dataset 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 (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` in your project directory. Include the plot in your homework write up. Briefly explain in at most three sentences what you see in the plot.

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