AI605: Deep Learning for NLP

Spring 2020

Homework #1

March 23, Due on **April 1(Wed), 11:59pm (in Classum)**

1. **Written (45 points)**

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' o. The goal of the skip-gram word2vec algorithm is to accurately learn the probability distribution $P(O \mid C)$. Given a particular word o and a particular word c, we want to calculate $P(O = o \mid C = c)$, which is the probability that word o is and 'outside' word for c, i.e., the probability that o falls within the contextual window of c. In word2vec, the conditional probability distribution is given by taking vector dot-products and applying the softmax function as

$$P(O = o \mid C = c) = \frac{\exp\left(\mathbf{u}_o^T \mathbf{v}_c\right)}{\sum_{w \in \text{Vocabulary}} \exp\left(\mathbf{u}_w^T \mathbf{v}_c\right)} \tag{1}$$

Here, $\mathbf{u}_o$ is the 'outside' vector representing the outside word o, and $\mathbf{v}_c$ is the 'center' vector representing the center word c. To contain these parameters, we have two matrices, U and V. The columns of U are all the 'outside' vectors $\mathbf{u}_w$. The columns of V are all of the 'center' vectors $\mathbf{v}_w$. Both U and V contain a vector for every $w \in$ Vocabulary . Recall from lectures that, for a single pair of words c and o, the loss is given by

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

Another way to view this loss is the cross-entropy between the true distribution y and the predicted distribution $\widehat{y}$. Here, both y and $\hat{y}$ are vectors with the 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 1 for the true outside word o, and 0 everywhere else. The predicted distribution $\hat{y}$ is the probability distribution $P(O \mid C = c)$ given by our model in Equation (1).

(a) (5 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

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

Your answer should be written in just one line.

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

(c) (5 points) From this derivation, we can obtain

$$\frac{\partial}{\partial \mathbf{v}_c} \log P(O = o \mid C = c) = \left(\mathbf{u}_o - \mathbb{E}_{p(w|c)} \mathbf{u}_w\right) \tag{4}$$

Give an intuitive explanation about the meaning of this gradient term when using it to update $\mathbf{v}_c$.

(d) (optional, 5 points) Suppose we use just a single vector for each word, i.e., $\mathbf{u}_i = \mathbf{v}_i$, for $i = 1, ..., V$, where V is the vocabulary size. Would it make the derivation of the gradient in (b) easier or more difficult? Justify your answer, possibly by giving some high-level insights.

(e) (5 points) Compute the partial derivatives of $\boldsymbol{J}_{\text{naive-softmax}}(\mathbf{v}_c, o, U)$ with respect to each of the 'outside' word vectors, $\mathbf{u}_w$'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 y, $\hat{y}$, and $\mathbf{v}_c$.

(f) (5 points) The sigmoid function is given by

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

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

(g) (5 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 $w_1, \ldots, w_K$ and their outside vectors as $\mathbf{u}_1, \ldots, \mathbf{u}_K$. 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:

$$\boldsymbol{J}_{\textbf{neg-sample}}(\mathbf{v}_c, o, U) = -\log\left(\sigma\left(\mathbf{u}_o^T\mathbf{v}_c\right)\right) - \sum_{k=1}^{K}\log\left(\sigma\left(-\mathbf{u}_k^T\mathbf{v}_c\right)\right) \tag{6}$$

for a sample $w_1, \ldots, w_k$ where $\sigma(\cdot)$ is the sigmoid function.

Please repeat parts (b) and (e), computing the partial derivatives of $\boldsymbol{J}_{\text{neg-sample}}$ with respect to $\mathbf{v}_c$, with respect to $\mathbf{u}_o$, and with respect to negative sample $\mathbf{u}_k$. Please write your answers in terms of the vectors $\mathbf{u}_o$, $\mathbf{v}_c$, and $\mathbf{u}_k$, 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 (f) to help to compute the necessary gradients here.

(h) (15 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 written as

$$\boldsymbol{J}_{\text{skip-gram}}(\mathbf{v}_c, w_{t-m}, \ldots, w_{t+m}, U) = \sum_{-m\le j\le m, j\ne 0} \boldsymbol{J}(\mathbf{v}_c, w_{t+j}, U) \tag{7}$$

Here, $\boldsymbol{J}(\mathbf{v}_c, w_{t+j}, U)$ represents an arbitrary loss term for the center word $c = w_t$ and outside word w_{t+j}. $\boldsymbol{J}(\mathbf{v}_c, w_{t+j}, U)$ could be $\boldsymbol{J}_{\text{naive-softmax}}(\mathbf{v}_c, w_{t+j}, U)$ or $\boldsymbol{J}_{\text{neg-sample}}(\mathbf{v}_c, w_{t+j}, U)$, depending on your implementation.

Write down three partial derivatives:

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

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

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

Write your answers in terms of $\partial\boldsymbol{J}(\mathbf{v}_c, w_{t+j}, U)/\partial U$ and $\partial\boldsymbol{J}(\mathbf{v}_c, w_{t+j}, U)/\partial \mathbf{v}_c$. This is very simple - each solution should be written in just one line.

2. Coding: Implementing word2vec (50 points)

In this part, you will pre-process sentences and train the word2vec model with your own word vectors with stochastic gradient descent. 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 assn1
```

Once you are done with the assignment, you can deactivate this environment by running:

```
conda deactivate
```

You can also install required dependencies by pip3 installation. Your python version should be upper than 3.6.

```
pip3 install -r requirements.txt
```

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. Satisfying this given code length will be challenging but try to achieve it. However, apart from your code lengths, your script should be efficient. If your code is fast enough, you will get bonus scores on the implementation part.

- (3 points) We will start by preprocessing sentences of Multi30k dataset. All sentences are stored in `sentences.txt` files and preprocessing functions are defined in `dataset.py` file. Since the dataset is divided into sentence level, you first separate them into word level. This preprocessing is called as tokenizing. Complete the function `tokenize` in `dataset.py`. Please refer to the comments in the file for detailed description. When you are done, test your implementation by running `python dataset.py`.

- (5 points) Since our implementation works with numbers, you should convert separated words into unique numbers. To achieve this, you need functions that maps words to their numbers and vice versa. Implement `vocab_builder` that creates these functions from the given sentences. Each vocabulary must have a unique number. In addition to mapper functions, you need to change words with too few frequencies to <UNK> tokens to prevent overfitting. Also, since the frequency of each word is required for negative sampling, this should also be implemented here.
- (2 points) In order to train skipgram model, you need a function that extracts a center word and its outside words from each sentence. Implement `skipgram` function that generates a center word and list of its outside words when the sentence and the location of the center word are given.

Now run `dataset.py` to check whether your database works well. It will show a sampled center word index, outside word indices and negative indices. When you are done with the database, let's go to `word2vec.py`.

- (10 points) Implement `naive_softmax_losses` that calculate the naïve softmax losses between a center word's embedding and an outside word's embedding. When using GPU, it is efficient to perform a large calculation at once, so batching is used generally. In addition, using a large batch size reduces the variance of samples in SGD, making training process more effective and accurate. To practice this, let's calculate batch-sized losses of skipgram at once. <PAD> tokens are appended for batching if the number of outside words is less than 2 * window_size. However, these arbitarily inserted <PAD> tokens have no meaning so should NOT be included in the loss calculation. When you are done, test your implementation by running `python word2vec.py`.
- (10 points) Implement `neg_sampling_loss` to calculate the negative sampling loss. As same with `naive_softmax_loss`, all inputs are batched.

When you are done with `word2vec.py`, now you are ready to train word2vec.

- (10 points) In the `run.py`, you can select naive softmax loss or negative sampling loss for training. After you run the `run.py`, the script will finish and a visualization for your word vectors will appear. It would be saved as `word_vectors.png` in your project directory. Include the plot in your homework write up. Briefy explain in at most three sentences what you see in the each plot. Also compare the two methods and choose a better one, and describe why you thought so.
- (10 points) If your code is fast enough, you will get extra performance points. We will check out with our own GPUs. You could get this bonus scores unless your code is too slow.

3. Submission Instructions

Run the `collect_submission.sh` script or manually zip the python files, png files, and word2vec.pth to produce your assign1.zip file. Then, upload following files to following Classum link:`https://tinyurl.com/kaistnlp2020`.

(a) `assign1_[student name]_[student number].zip`

(b) The report named '`report1_[student name]_[student number]`.pdf'. It should contain your solutions about written problems and the implementation report.