Deep Learning in Natural Language Processing
Assignment 2

The Hong Kong University of Science and Technology, Spring 2021

Deadline: TUE 23:59, 9 March 2021

1 Introduction

The goal of this assignment is to implement the Continuous Bag of Words (CBOW) model in PyTorch and use it on a sentiment classification task. You need to: 1) Implement and train a CBOW model using the full user reviews of cars and hotels corpus *reviews_data.txt*. Please download the dataset here; 2) Use the learned word representations to retrain models for the sentiment classification task in assignment 1.

2 Training Part (6pt)

In this section, you will need to create your own PyTorch data loader to load the data, including dataset preprocessing and cleaning. Then you can start implement the CBOW algorithms. Finally, implement a training loop to try different hyper-parameters.

2.1 Data Loader

The first step is to load and pre-process the data. You need to override and extend two pytorch classes:

- **Dataset**: this class allows to create a custom dataset.

```
from torch.utils.data import Dataset
class MyData(Dataset):
    def __init__(self, filename="twitter_sentiment.csv.gz"):
        # TODO
    def __getitem__(self, index):
        # TODO
    def __len__(self):
        # TODO
```

- **DataLoader**: this creates an iterator over the Dataset object.

```
from torch.utils.data import DataLoader
bsz = 32
dataset = MyData()
train_loader = DataLoader(dataset=dataset,
                          batch_size=bsz,
                          shuffle=True)
```

In the Dataset class you can include your data cleaning that you have implemented in assignment 1. Then, you need to extract the context and the target from the sentences in the corpus. The train_loader object is an iterator which returns the following couple:

- contexts words: a tensor of dimension $[bsz, context\_size]$. ***Note 1)*** the bsz dimension is added by the `DataLoader` object, you do not need to create the batch manually. ***Note 2)*** *context_size*

dimension are numbers which represent the position in the embedding matrix. ***Note 3)*** The context size, number of word around the target word, is a parameter, but you may fix it to 2.

- target word: a tensor of dimension $[bsz]$ represents the target words.

2.2 Continuous Bag of Words (CBOW)

The second step is to implement the neural network class.

```
import torch
import torch.nn as nn
class CBOW(torch.nn.Module):

    def __init__(self, vocab_size, embedding_dim):
        super(CBOW, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        # TODO

    def forward(self, inputs):
        # TODO
        return out

    def get_word_embedding(self, word):
        # TODO
        return word_vector
```

In the `nn.Embedding(vocab_size, embedding_dim)`, `vocab_size` is the dimension of the vocabulary and `embedding_dim` is the embeddings dimension. The `get_word_embedding` takes a word and convert it to a vector. You need also to define the loss function and the optimizer (e.g. `nn.torch.optim.` etc.).

2.3 Training CBOW

Finally you need to implement the training loop. You need to use the `DataLoader` created before and should print the loss value at each iteration. Since we do not have a validation set to tune our hyper-parameters we try to have the loss as small as possible. Hence try at least 5 different combination of hyper-parameters (e.g. different learning rate value, or batch size or hidden size), and report the loss values in a table.

3 Evaluation Part (2pt)

Differently from the previous assignment, we do not have a proper test set for our embeddings. Normally, the learned embeddings matrix (i.e. `self.embeddings`) is used in a different classification task. Therefore, we are going to perform a qualitative and a quantitative evaluation:

Word similarity Implement the `get_similarity` function, which gets two word (i.e. as simple string) as input and the trained model, and it return the cosine similarity of the two respective vectors. The cosine similarity formula is the following:

$$\text{similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{1}$$

where you can implement $\|\|$ by using torch.norm(). This allows us to check whether similar words are close each other in the learned space. Hence, select 5 pairs of words, and report their cosine similarity. Try to select meaningful pairs to show possibly interesting property.

Sentiment Classifier To verify if the word embedding matrix is successfully learned, we will plug the word vectors as input feature for the assignment 1 task. Hence, instead of the BOW representation, we are going to use the learned word embedding representation. First, the embedding matrix is saved by using:

```
word_emb_mat = model.embeddings.weight.numpy()
np.savetxt('w_emb_mat.txt', word_emb_mat)
```

and then it can be loaded into the assignment 1 script using:

```
word_emb_mat = np.loadtxt('w_emb_mat.txt')
```

Note that, each row correspond to one word in the vocabulary, so you need to align the word of assignment 1 with the current embedding matrix. Now train your logistic regression classifier using this new input and report the accuracy.

4 Questions (2pt)

1. Implement the Dataset and the DataLoader class in `loader.py`. In this file you may also include all the data utilities.
2. Implement CBOW model, including the loss and the optimizer. Use a separate file named `CBOW.py`.
3. Implement the training function, and report the loss value (at the end of the training) of 5 different hyper-parameter configuration. Use a separate file named `main.py`.
4. Implement the two evaluation:
   - Word Similarity: implement cosine similarity and 5 couple of words score.
   - Sentiment Classifier: use the learned embedding matrix for assignment 1 and submit the prediction results of `twitter-sentiment-testset.csv` file again.
5. Does the learned representation improve the result from assignment 1? If yes, why? if no, can you suggest a strategy to improve the accuracy?

5 Bonus (1pt)

- Instead of training our own word embeddings, there are many other pre-trained embeddings available, such as Word2Vec, Glove, etc. Please try to download and use them in your experiment. Report the difference in validation set.
- Use the learned embedding matrix, but at this time train a Neural Network instead of logistic regression. Report the results in the validation set using two settings: 1) fix the word embeddings while training, and 2) keep updating the word embeddings. Compare this results with the logistic regression, and explain how keep updating the word embeddings influence the model performance.

6 Submission

Turn this in by **TUE 23:59, 9 March 2021** by emailing your solution and code to hkust.hltc.courses@gmail.com, with subject line "DLNLP-HW2". Remember to state your name, student ID in the email. The zip should only include the following materials:

- Short report of the assignment in pdf format (maximum 3 pages).
- A folder which contains: `loader.py`, `CBOW.py`, `main.py`, `sample-submission.csv`, and all the other file you have used in your experiments. **Please do not send back the dataset**.

7 Enjoy your HW2!