Homework 5

4190.408 001, Artificial Intelligence

December 7, 2023

1 Submission

In this homework we will make our own word2vec embedding using CBOW model.

- Assignment due: 2023-12-18
- Individual Assignment
- Submission through ETL. Please read the instructions in this PDF and ipynb file. **Fill out the TODO in your ipynb file, and submit model.pt, vocab.pt and your ipynb file including outputs**. The name of the ipynb file should be in the format of "{student_ID}_{first name}_{last name}.ipynb", e.g. "2023-11111_Jisoo_Kim.ipynb". Please write your name in English.
- We recommend to use Google Colab.
- Collaboration to solve homework is permitted. Discussion is encouraged, but you must think about the problems on your own.
- If you collaborate with someone or use a book or website, you are expected to write up your solution independently. That is, close the book and all of your notes before starting to write up your solution.
- Make sure you cite your work/collaborators at the end of the homework.
- **Honor Code**: This document is exclusively for Fall 2023, 4190.408 students with Professor Hanbyul Joo at Seoul National University. Any student who references this document outside of that course during that semester (including any student who retakes the course in a different semester), or who shares this document with another student who is not in that course during that semester, or who in any way makes copies of this document (digital or physical) without consent of Professor Hanbyul Joo is guilty of cheating, and therefore subject to penalty according to the Seoul National University Honor Code.

2 Implementation

In this homework, we will train word embedding using Wikitext2 dataset with CBOW model.

2.1 CBOW Model Architecture

Word Embedding is a example of dense representation of vocab. We will train word embedding by implementing CBOW architecture that predicts the target word using the left and right words of a specific window size relative to the target word. This is kind of unsupervised learning, which trains the model to make embedding(look up table) in pipeline for predicting target word using context information around the target word, thereby obtaining a dense representation of the words associated with that context.

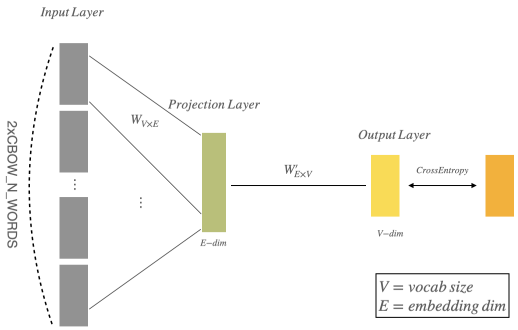

Fig 1. Architecture of CBOW Model

2.1.1 Make Vocab and collate Function

In order to use WikiText2 dataset to train word embedding, we should process data in several steps.

- Try printing some of the raw dataset(like `train_dataset['text'][11]`). We should tokenize sentences, delete not useful tokens like (. , \n ...) and lowercase each word. We also need to index the words.
- To tokenize sentences and delete not useful token, just use prepared tokenizer in TorchText. We recommend `basic_english` tokenizer for lowercasing words.
- To make vocabulary, just use `build_vocab_from_iterator`, You don't need to consider how to make vocabulary in this Homework.

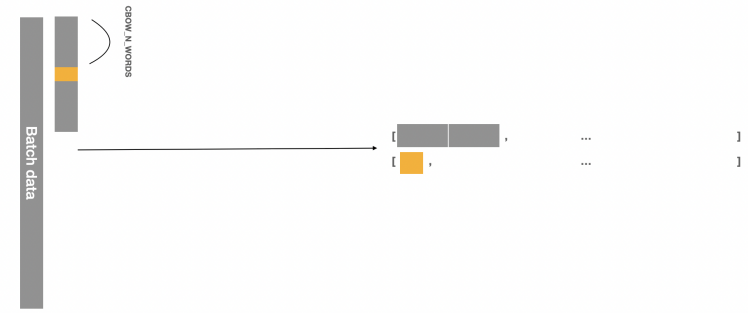

Fig 2. collate function

Collate function is required by Dataloader to process batch data into a training format (input - target output).

- collate function's input is each batch sentences.
- In collate function, you should make train/test dataset iterating over batch and extract target word and concatenate it together with the left and right window words.
- `text_pipeline` is lambda function which return indexes of vocabs after tokenizing input sentence.
- If you want to use another logic, you can delete `text_pipeline` in the function and change it to another.

2.1.2 Make CBOW Model

- Fill in two TODOs in CBOW model.
- Original explanation about Word2Vec used one hot vector to represent word in sparse matrix. But in this homework, it is not necessary to use one hot vector, as you could use python embedding function `nn.Embedding` to represent word in dense embedding.

```
class CBOW_Model(nn.Module):
 def __init__(self, vocab_size: int, EMBED_DIMENSION, EMBED_MAX_NORM)
   ...
 def forward(self, inputs_):
   ...
```

3 Training

As we finished building the model, we will implement optimizer and loss criterion for training. The example details are as follows:

- **Optimizer:** `optim.Adam`
- **Loss function**: `nn.CrossEntropyLoss()` as each word vocab index works like class
- **Learning rate:**: 0.025

If you want to change above optimizer, loss function, learning rate, you can do so.

For 4-1, 4-2, you have to fill in two functions in `Train_CBOW` class.

```
class Train_CBOW:
 def __init__( self, model, epochs, train_dataloader, ... ) :
   ...
 def train(self):
   ...
 def _train_epoch(self)
   ...
 def _validate_epoch(self)
   ...
   ...
```

- `_train_epoch` and `_validate_epoch` have to iterate over `self.train_dataloder`, `self.val_dataloder`.
- After getting each model's result of input data, compute loss and append to each target `self.loss`
- Backpropagate only for train part
- (Optional) You can use lr scheduler. There is an example in ipynb file.
- In function `train`, call `_train_epoch` and `_validate_epoch` each epoch and print training loss changes.
- Please set `epochs` to 30 or more. We don't expect specific performances or specific losses. However, we will visually check whether the train loss and validation loss decrease as the epoch passes.

4 Visualization/ Inference

As word embedding is trained, we will inference trained word embedding in 2 ways.

4.1 t-SNE

Before inferencing to get t-SNE and cos similarities, it is usual to normalize word2vec. It is because only direction of vector is needed and useful when computing similarities. Please look into this link .

- Plot word embedding vectors using t-SNE.
- t-SNE is used pretty often when visualizing distribution of high dimensional embedding by compressing to smaller dimension keeping the relative similarity of vectors as remained as possible.
- After get compressed position of vectors in 2D plot, color only for numeric values(others, just color black).
- Mind it the final saved ipynb file should have t-SNE output.

4.2 Finding Similar Words

Let's make function return top N similar words of input word.

- Fill in TODO to return output as dictionary datatype composed of Top N words having similar embedding and their similarity scores.
- Output should be sorted descending order of similarity score.
- The similarity score with itself should be 1, which should be the highest similarity score. In other words, there must not be a word with a higher similarity than the input word itself, or the similarity must not exceed 1(This does not mean that the result should include the input word. Rather, the result should not contain the input word).
- Check whether the input word exists in the vocab, and if not, throw an exception (you can simply handle it using a print statement).

5 Result Report

After finishing all TODOs, submit your output ipynb file with trained model and vocab file.
The results should include:

- vocab.pt and model.pt file in weights directory in local directory or google drive (when you train using colab)
- Filled TODOs
- train loss / val loss result of trainer.train()'s output
- t-SNE graph output
- The name of the ipynb file should be in the format of "{student_ID}_{first name}_{last name}.ipynb", e.g. "2023-11111_Jisoo_Kim.ipynb". Please write your name in English.