Homework 5

4190.408 001, Artificial Intelligence

December 4, 2025

1 Submission

In this homework we will make our own Word2Vec embedding using CBOW model.

- Assignment due: 2025-12-18
- Individual Assignment
- Submission through ETL. Please read the instructions in this `.pdf` and `.ipynb` file, and fill in the TO DOs - [1,2,3,4,5]) in the `.ipynb` file and submit the `.zip` file including `model.pt`, `vocab.pt`, `.ipynb` file, and the `image.png`. The `.zip` file name should be in the format of "{student_ID}_{first name}_{last name}.zip", *e.g.*, "2025-11111_Jisoo_Kim.zip". The unzipped folder should have same name without the extension. Please write your name in English. Failure to follow the above format may result in a penalty to the final score.
- We recommend to use Google colab to run the file.
- Collaborations on solving the homework is allowed. Discussions are encouraged but you should think about the problems on your own.
- If you do 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 2025, 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 and we will train embedding by implementing CBOW architecture which predict target word using before and after words in specific window size. We can obtain word's dense representation connected with its context by training the model to predict the target word using context information around the target word. It is kind of unsupervised learning which makes word's embedding (look-up table).

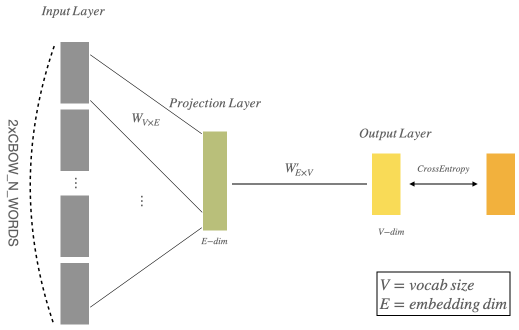

Fig 1. Architecture of CBOW Model

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.

- As you can check printing dataset like `train_dataset['text'][10]`, we should tokenize sentences, delete useless tokens like (. , \n ...) and lowercase each word and change word to integer.
- You may use special token `<unk>` or `<pad>`.
- To tokenize sentences and delete useless tokens, just use prepared tokenizer in torchtext. We recommend `basic_english` tokenizer as it lowercase words as well.
- To make vocabulary, just use `build_vocab_from_iterator`, You don't need to write how to make vocabulary in this homework.
- You can change vocab and model's parameters like `MIN_WORD_FREQUENCY` or `CBOW_N_WORDS`.

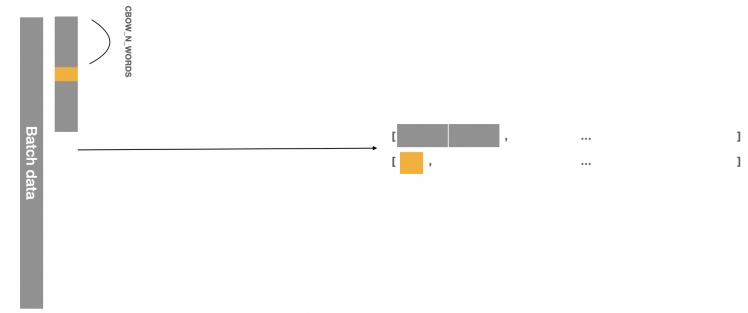

Fig 2. Collate Function

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

- The input of the collate function is a batch of sentences.
- Consider to use `MAX_SEQUENCE_LENGTH` to crop or skip data with many words.
- `CBOW_N_WORDS` is the number of words on each side (before and after) used for predicting the target word, which is an hyperparameter.
- In collate function, you should make train/test dataset iterating over batch data by extracting target word (output), and concatenating before and after window words together (input).
- `text_pipeline` is lambda function which return indices of vocabs after tokenizing input sentence.
- You can replace `text_pipeline` it with other logic if you want.

2.1.2 Make CBOW model

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

- Fill in two TO DOs in CBOW model.
- Original Word2Vec uses one-hot vector to represent word in sparse matrix. However, in this homework, we use python embedding function `nn.Embedding` to represent dense word embedding.
- Use `nn.Linear` to construct the CBOW model.
- Your input is the sequence of word embeddings, and your output is the predicted target word embedding. One way to handle the dimension reduction is using `mean`. You can also use other ways to match the dimension.

3 Training

After building the model, we will implement optimizer and loss criterion for training. The default settings are as follows. It is allowed to change the settings including optimizer, loss function, and learning rate.

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

For 4-1, 4-2, you need to complete two functions in the `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`.
- Use back propagation only for the training part.
- `LambdaLR` is the learning rate scheduler. It changes the learning rate in accordance with the epoch.
- In function `train`, call `_train_epoch` and `_validate_epoch` each epoch and print training loss changes.

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 direction of vector is only needed and useful when computing similarities. Please look into this link .

- Plot word embedding vectors using t-SNE.
- You can use `TSNE` from scikit-learn for implementation.
- t-SNE is used pretty often when visualizing distribution of high dimensional embedding by compressing embedding to smaller dimension keeping the relative similarity of vectors as remained as possible.
- After get compressed position of vectors in 2D plot, color numeric values (*e.g.*, "1", "2", "3") only with different color.

4.2 Getting Similar words

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

- Fill in TO DO to return output as dictionary datatype composed of top-N words having similar embedding and its similarity score.
- The top-N words must not include the input word.
- Output should be sorted descending order of similarity score.

5 Result Report

After finishing all TO DOs. Submit the `.zip` file including followings:

- `AI_HW5.ipynb` file with filled TO DOs, and log results of training / validation losses.
- Saved `vocab.pt` and `model.pt` files.
- Result image of t-SNE graph output named `image.png`.

The `.zip` and the unzipped folder name should be in the format of "{student_ID}_{first name}_{last name}.zip", and "{student_ID}_{first name}_{last name}". One example is "2025-11111_Jisoo_Kim.zip" for the `.zip` file and "2025-11111_Jisoo_Kim" for the unzipped folder. Please write your name in English.