Mini-project 1: Training a CBoW Model

In this mini-project, you'll learn to train and evaluate a Continuous Bag of Words (CBoW) model which was introduced by Mikolov et al (2013). At the end of the mini-project, you will have a set of embeddings for a certain set of words, known as the vocabulary, we have provided. You'll be submitting the embedding file along with the code and report.

CBoW Overview

At a high level, CBoW asks: Given the surrounding words around a word that is hidden, can we predict the hidden word? To do so, it defines what is essentially a two layer neural network that projects the context words to a low dimensional vector space, and then tries to use this embedding as a feature vector to predict the hidden word. The CBoW model consists of two components: 1) the projection layer, and 2) the classifier layer.

The projection layer projects a one-hot encoded vector for a word into the embedding space. That is,

$$h = x \cdot E$$

Here, E is a learnable projection matrix of size $|V| \times d$ where $|V|$ is the size of the vocabulary and d is the embedding dimensions. The vector x is a one-hot encoded vector of size $|V|$. Consequently, the hidden embedding h is a d dimensional vector.

The classifier layer is another matrix which maps from the embedding space to the vocabulary.

$$z = h \cdot W$$

where W is a second learnable matrix with size $d \times |V|$. Hence, the vector z is $|V|$ dimensional.

During training, we obtain the hidden embedding of a target word by summing over the hidden embeddings of its context words. CBoW defines the context of a word w_0 to be the n words before and after it. Given a target word w_0 in context [w_{-n}, …, w_{-2}, w_{-1}, w_0, w_1, w_2, ..., w_n], we compute hidden embedding using projection matrix for all words except w_0 giving us [h_{-n}, …, h_{-2}, h_{-1}, h_1, h_2, ..., h_n] . Summing over these hidden embeddings gives us h_0.

The embedding h_0 is then multiplied with matrix W to give us logits (or, raw scores) over the vocabulary. While the original paper used a different loss metric, for this project, we will explore using Cross Entropy Loss to compute the loss given the logits and the target word, which is w_0.

To obtain the final embedding once training is complete, we simply take the transpose of the W matrix to get the embeddings. That is, the embedding p for a one-hot encoded b is defined to be:

$$p = b\,W^T$$

Processing Steps

You will need to follow the steps mentioned below to implement CBoW:

1. Create a mapping between a word and an index for every word in the vocabulary.
2. Load the data from the files and map the words to their respective index.
3. Create contexts depending on your context window size and targets from the data created above. Suppose your context window is of size 2, you should consider two words before and two words after the target word as your context. Note that, for the first and the last two words you have to use appropriate padding.

   As an example, for a context window of size 2 and sentence - "I met her", we'll have the following instances:

Context	**Target word**
["[PAD]", "[PAD]", "met", "her"]	I
["[PAD]", "I", "her", "[PAD]"]	met
["I", "met", "[PAD]", "[PAD]"]	her

   To ease implementation, you can add the appropriate number of "[PAD]" tokens at the front and back of the data prepared in step 2.
4. Create appropriate datasets and data loaders. You are free to choose the batch size you want. We recommend a size of 64.
5. Create the model and implement the training loop. Use the Cross Entropy loss for loss computation. You can use your favorite optimizer.
6. Save your best model.
7. Load the best model and compute the embeddings for every word in the vocabulary and dump it into an embeddings file (See format below).
8. Perform the evaluations asked below.

Hyperparameters and Parameters

The only tunable hyper-parameter is the learning rate. Train your model using the following learning rates: {0,.01, 0.001, 0.0001}. Consider the epoch with the best loss on the dev set across all learning rates to compute your embeddings.

Other parameters:

- Embedding dimension: 100
- Batch Size: Free to choose (64 recommended)
- Maximum epochs: 10
- Context size: 5

Data Files

We have given you the following files in the compressed folder:

1. Data Files (located under `/data/`): These are the data files for training your CBoW model. This folder contains two sub-folders for the 'train', and 'dev splits'. Each sub-folder contains multiple `*.txt` files. Each `.txt` file contains line-separated words (i.e., each line has one word) which when put together creates a document.
2. Vocab file (located at vocab.txt): This is the vocab file which contains a fixed set of words which we would like to consider as our vocabulary. Our aim is to get embeddings for each of these words. At the end, you'll see two special words- i) "[UNK]" which will be used to replace any out-of-vocabulary(OOV) words that we encounter, and ii) "[PAD]" which will be used for padding inputs.
3. Test files (located under `/test_sets/`): Contains files for downstream tasks to test your embeddings. We will evaluate the embedding on two tasks:
    a. Word Similarity: Computes similarity between two words. For example, "tiger" and "cat" are similar, but "human" and "plane" is not similar. We will have ground truth values of these similarity. To evaluate our embeddings, we compute similarity of two words by calculating the cosine similarity between the embeddings of the two words. We compute the Pearson's and Spearman's correlation to evaluate how our embeddings perform w.r.t the ground truth.
    b. Analogy: The idea is to compute answers to questions like "If king is to queen, then man is to what?". Hence, the problem is that given an input triplet like [king,queen,man], complete the set. We compute the accuracy metric for this task

    You'll be given a test file each for the two tasks. In addition, we'll have a blind test set for each of the tasks that will be evaluated post-submission.

Helper Methods and Resources

We will be providing you with some utility and evaluation methods.

Utility functions are given under scripts/utils.py. Here is a brief description of the methods provided:

1. `get_word2ix()`: This method gives you a mapping between the words in the vocab file to unique indices. You need to provide the path to the vocab_file. It returns a dictionary where keys are the words in your vocabulary and their respective value is the unique index.
2. `get_files()`: Given a directory path, it returns a list of files with the directory. This will be helpful to enumerate all the files in the `/data/<split>/` directories.
3. `process_data()`: Given a list of files, it loads the data iteratively from each of the files. All words in a file are written to a list. Subsequently, we map all the words to that unique indices (as obtained from the get_word2ix method). If the words are not in the vocabulary, we map it to the index corresponding to the “[UNK]” token. We also pad the list for a file with the “[NULL]” token index appropriate for the context window chosen.

Some helpful PyTorch methods

1. torch.nn.Embedding takes a list of indices (like the ones returned by `process_data`) and projects it to their corresponding set of embeddings. Simply put, it maps an index to a embedding, In the CBoW architecture, this can be used as the projection layer.

Evaluation

The eval_embs.py takes in the embedding file generated (see below for format details), and evaluates the embeddings across the word similarity and analogy task. You can run the evaluation using the following command:

```
python eval_embs.py --emb_file <full_path_to_emb_file>
```

Format of Embedding File

Important: We need you to submit an embeddings.txt file. The first line of the file should contain space-separated values of vocabulary size and the embedding dimension. In every subsequent line, you’ll have the word and embeddings which should be separated by a space. Each value in the embedding should also be separated. Below is an example of an embedding file with vocabulary size of 5 and embedding dimension of 3:

```
5 3
the 0.10 0.21 -4.13
a 1.1 2.3 0.45
an 1.55 3.14 -2.0
[UNK] -1.1 -3.4 0.34
[PAD] -9.3 1.4 1.5
```

What to submit

1. **[40 points]** Submit the embedding file. We will evaluate the embeddings you submit on a set of analogy and word similarity examples that are not provided to you.
2. A report as described below

What to Report

1. **[5 points]** Report the best learning rate and the lowest development set loss.
2. Using your embeddings compute the answers to the following:
   (Use Cosine similarity (https://en.wikipedia.org/wiki/Cosine_similarity) to compute the similarity between two vectors.**)** Report the similarity score as well.
   a. **[5 points]** Which pair is closer in the word embedding space?
      i. [cat, tiger] or [plane, human]
      ii. [my, mine] or [happy, human]
      iii. [happy, cat] or [king, princess]
      iv. [ball, racket] or [good, ugly]
      v. [cat, racket] or [good, bad]
   b. **[5 points]** Using the most similar word in the vocabulary with the vector (w_a - w_b + w_c), complete the following analogies:
      i. king:queen, man: ?
      ii. king:queen, prince: ?
      iii. king:man, queen: ?
      iv. woman:man, princess: ?
      v. prince:princess, man: ?
3. Using your embeddings, generate three examples each:
   a. **[5 points]** Word Similarity test. You should produce two sets of word pairs where one pair is much more similar than the other pair. All words should belong to the vocabulary
   b. **[5 points]** Word Analogy test. Present an analogy set according to your embeddings where “a:b, then c:d”
4. **[20 points]** Evaluate your embedding on the test files included and report the results.
5. **[15 points]** Read the following short paper
   Joulin, Armand, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. "Bag of tricks for efficient text classification." *arXiv preprint arXiv:1607.01759* (2016).

   Explain (briefly, no more than 2-3 paragraphs) in your own words how the technique described in the paper differs from CBoW.

Extra Credit

1. **[20 points]** Plot a 2-D projection of the embeddings for words in the list below.

```
horse, cat, dog, I, he, she, it, her, his, our, we, in, on,
from, to, at, by, man, woman, boy, girl, king, queen, prince,
princess
```

You may use PCA (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html) or SVD (https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html) to compute the projection.

Explain your observations and also explain why using a 2 dimensional projection may be misleading.

References

- Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).
- Joulin, Armand, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. "Bag of tricks for efficient text classification." *arXiv preprint arXiv:1607.01759* (2016).
- Wikipedia: Cosine Similarity