Word Embedding

Machine Learning for Natural Language Processing, Lab Report II

October 30, 2022

Word Embedding is essential for natural language processing (NLP) since it maps alphabetical contexts into vector spaces, which allows performing further NLP tasks such as language classification, machine translation, etc. Continuous bag of words (CBOW) is an architecture that generates word embeddings by taking semantic similarities into account. In this report, the data will first be preprocessed, and CBOW will be used to generate the corresponding embeddings for each word. Finally, discussions about choosing the dimension of the embedding and the result analysis will be presented.

1 Word Embedding

Word embedding is the vector representation of natural language vocabularies(arvindpdmn, n.d.)(Brownlee, n.d.). It is essential for tasks like natural language processing (NLP), and its quality is one of the factors that effect the accuracy and performance of the models that is trained on the word embedding.

There are ways to generate word embeddings. In this report, continuous bag of words (CBOW) will be introduced. CBOW is intuitive, but has several short comings such as failing to identify combined words[1](Basnet, n.d.) and having trouble with homonyms[2].

1.1 Continuous Bag of Words (CBOW)

CBOW is the an intuitive way to generate word embeddings. CBOW trains the embedding weight matrix by predicting a target word with words around it(Prabhu, n.d.)(Khandelwal, n.d.)(Fangyug, n.d.). To implement the algorithm, each word is represented by an one-hot encoding vector, and is treated as input of a shallow network. In the single hidden layer, a matrix of an initialised embedding weight matrix is multiplied by one-hot encoding vector. The output of the hidden layer then sent to a non-linear activation function for generating a vector of probabilities of the target word should be.

Take a corpus of three words for example as shown in Equation 1, the first and third word is used to predict the second word (the target word), and thus the first and third position in the one-hot encoding is denoted as 1. The embedding weight matrix is initialised with random values with dimension of 3×dimension-of-embedding, which is a hyperparameter that can be chosen freely. Normally, the dimension of embedding is between 50 and 300. The output of the the hidden layer is of dimension 4, which is the dimension of the embedding. This will be the input of the activation function. The output of the activation function will be of dimension 3, the total number of input vocabularies. Each entries of the output is the probability of the target word.

$$\underbrace{\begin{bmatrix}1 & 0 & 1\end{bmatrix}}_{\text{one-hot encoding}} \times \underbrace{\begin{bmatrix}0.2 & 0.3 & 0.1 & 0.4\\ 0.6 & 0.5 & 0.1 & 0.2\\ 0.8 & 0.2 & 0.4 & 0.4\end{bmatrix}}_{\text{embedding weight matrix}} = \underbrace{\begin{bmatrix}1 & 0.5 & 0.5 & 0.8\end{bmatrix}}_{\text{hidden layer output}} \tag{1}$$

Practically, this can be done by `nn.Embedding` of the `PyTorch` module. It is worth noting that, instead of putting one-hot encoding into `PyTorch` embedding module, it takes the index of the words to be trained directly. `PyTorch` embedding module handles the one-hot encoding internally.

```
import torch.nn as nn
# INPUT WORD INDEX VECTOR
idxs = torch.tensor([word_idx_array])
# INITIALISE PYTORCH EMBEDDING
emb = nn.Embedding(vocab_size, embedding_dim)
# GENERATE HIDDEN LAYER OUTPUT
hidden = emb(idxs)
# INPUT ACTIVATION FUNCTION FOR PROB.
```

[1]For instance, *New York* should be considered as a single word. However, CBOW tends to treat it as two separate words and have different word embeddings for each of them.

[2]Words are called homonyms if they have the same pronunciation and spelling, but different meanings.

```
prob = softmax(hidden)
```

Listing 1: *Embedding by* `PyTorch`

This is the core structure of the CBOW. The complete architecture requires putting the core structure into a shallow neural network with appropriate batch size and epoch. The whole architecture is shown in the pseudocode below.

```
# ITERATE THROUGH EPOCH
for epoch in epoch_size do:
    batch_loss = 0
    # ITERATE THROUGH BATCH
    for batch in batch_size do:
        # WORD INDEX VECTOR
        idxs = [training_word_indexes]
        # PYTORCH EMBEDDING
        emb = nn.Embedding(_, _)
        # HIDDEN LAYER
        hidden = emb(idxs)
        # ACTIVATION FUNCTION FOR PROB.
        prob = softmax(hidden)
        # GET LOSS
        loss = NLLLoss(prob, target_idx)
        # UPDATE LOSS OF THIS BATCH
        batch_loss += loss
    # UPDATE LOSS TO EMBEDDING MATRIX
    loss.backward()
    update parameters in embedding
```

Listing 2: *Pseudocode*

1.2 CBOW Pros and Cons

Since CBOW rely on surrounded words to train the target word, it can capture the semantic similarities between words. For instance, since cat and dog are mostly in the same context, and thus, CBOW is more likely to give the a shorter distance in the embedding vector space than other words that are not used often in the same context.

However, CBOW processes word by word, and thus combined words such as *New York* will be treated as two words, which are *New* and *York*. Treating combined words into separate words often leads to false meaning and results in incorrect vector in the embedding vector space, making the relationship between words distorted. In addition, since each word has one embedding for CBOW, homonyms will end up with only one meaning even when used in different context. Contextualised embedding algorithms like BERT, ELMo will give different embeddings for a word when used under different context. For instance, the word *bank* in the following two sentence have different meanings.

- This is a river *bank*.
- I went to the *bank* this morning.

Unfortunately, CBOW gives them identical embeddings when they should have ended up with different embeddings.

2 Methods

To generate word embeddings, there are several steps to save memory and computational resources. Context preprocessing are first performed, and then CBOW architecture, a shallow neural network, is used to train the embedding weight matrix.

2.1 Datsets

In this task, two datasets are used for training with same models. One dataset is Trip Advisor hotel review, and the other one is the sci-fi stories.

Due to the computational resources limitation, the datasets are trained with 40% and 10% repectively. The purpose of this task is to demonstrate CBOW instead of pursuing an ulltimate accuracy.

2.2 Preprocessing

The goal of preprocessing is to remove words that are not interesting or helpful for tasks, including HTML, numbers, punctuations and stop words. This significantly reduces the memory and computation resources. In addition, it converts words to lower cases so that "Cat" and "cat" will not be treated differently only because one is at the beginning of the sentence.

These are the steps taken in the preprocessing:

- remove HTMLs
- remove numbers
- convert to lower case
- remove punctuations
- remove stop words

After data cleaning, the sentences are concatenated into a corpus for each dataset. The corpus are converted into the following format for training: (w stands for a word in the corpus)

$$([w[i-2], w[i-1], w[i+1], w[i+2]], w[i]) \quad (2)$$

The first list, $[w[i-2], w[i-1], w[i+1], w[i+2]]$, is served as the input, while the second element in the format, $w[i]$, is the target word.

2.3 CBOW

To train the embedding, CBOW architecture is used. The training sentences are first concatenated into a corpus, and chunks of words in the corpus are input as training data with the middle word as the target word, and its surrounding words as input words to the shallow network. For instance, for the corpus, *I am a good student*, "I", "am", and "good", "student" are treated as input, while "a" is the target word. The neural network uses the input and the target word to update the embedding weight matrix in the hidden layer.

As mentioned in section 1.1, `PyTorch` takes word indexes vector, converts it into one-hot encoding, and then multiply it with the initialised embedding weight matrix. Hence, each word is first given a unique index. Then the indexes of the two previous and future words of the target word are treated as the input of the neural network. The task is done by following the code in Listing 1. Due to the limitation of memory, each data is treated as one batch. This will cause the unstable gradient for back-propagation. However, this is a trade-off for the memory limitation. But with more epoch, the result will converge eventually theoretically. On the other hand, epoch for Trip Advisor dataset is set to 12 so that it doesn't take too long to train the model. Unfortunately, sci-fi story dataset has only 2 epoch trained due to computational limitation. Please refer this paragraph to the pseudocode below.

```
# ITERATE THROUGH EPOCH
for epoch in epoch_size=12 do:
    loss = 0
    # ITERATE THROUGH BATCH
    for each data do:
        # WORD INDEX VECTOR
        idxs = [training_word_indexes]
        emb = nn.Embedding(_, _)
        hidden_1 = emb(idxs)
        # ACTIVATION FUNCTION FOR PROB.
        prob = softmax(hidden_1)
        # GET LOSS
        loss = NLLLoss(prob, target_idx)
        # UPDATE LOSS TO EMBEDDING MATRIX
        loss.backward()
        update parameters in embedding
```

Listing 3: *Pseudocode*

The standard CBOW requires an input, one hidden layer and an activation function in the end. In this task, more layers are used to capture the non-linearity of the training data.

2.4 Train in Both Directions

In this task, two models of training in different directions are implemented. One model reads the corpus from left to right like the usual way, while the other model reads the corpus backwards from right to left. The result is expected to be the same since the input has the same weight for all input words, and thus, the result should not depend on the direction of the data.

2.5 Loss Function

The loss function used in the task is negative log-likelihood loss. It is defined as the following in `PyTorch`:

$$l_n = -w_{y_n} x_{n,y_n} \tag{3}$$

Thus, if the probability vector is $\begin{bmatrix}0.5 & 0.1 & 0.3 & 0.1\end{bmatrix}$, and the index of the target word is $\begin{bmatrix}0\end{bmatrix}$, then the loss will be -1$\times$0.5=-0.5 with default weight.

2.6 Choice of Embedding Dimension

Usually, the dimension of the embedding is between 50 and 300. However, the dimension is not the more the merrier due to the expensive computational and memory costs. A paper(Patel and Bhattacharyya, 2017) proves that,

> *The number of pairwise equidistant words enforce a lower bound on the number of dimensions that should be chosen for training word embeddings on the corpus.*

In other words, if the maximum number of pairwise equidistant words can be found, the dimension of the embedding can be determined accordingly.

With a simple example, the paper points out that the minimum dimension of space that can properly represent distances between four objects is three. If four objects that have same equidistances in a three dimensional space are put into a two dimensional space, and are trained for their embedding, the embeddings will not capture the information within the occurances properly. This leads to badly trained embeddings which do not include all necessary of the words.

According to the paper, the following steps are proposed:

1. Calculate the co-occurrence matrix.
2. Calculate the cosine-similarity matrix
3. Get the maximum clique.
4. Look up the equiangular lines table according to the maximum clique.(Barg and Yu, 2013)

These can be done by `sklearn` module. Take three sentences for instance,

- I love dog
- I am a student
- My dog is cute

The co-occurance matrix will be the one in figure 1. With this matrix, cosine similarities between each pairs can be calculated accordingly. The number of dimension can be looked up once the maximum clique is calculated by the cosine similarity matrix. In this example, the maximum clique is 12, and by looking up the equiangular lines table, the dimension of the embedding should be 6. Please refer to the following code for practical implementation.(Notes, n.d.)

```
from sklearn.metrics.pairwise import
    cosine_similarity
from sklearn.feature_extraction.text import
    CountVectorizer
from scipy import sparse
cv = CountVectorizer(ngram_range=(1,1))
# CO-OCCURRANCE MATRIX
X = cv.fit_transform(X)
Xc = (X.T * X)  # MATRIX MANIPULATION
Xc.setdiag(0)
# COSINE SIMILARITY MATRIX
similarities_sparse = cosine_similarity(Xc,
    dense_output=False)
```

	am	cute	dog	is	love	my	student
am	0	0	0	0	0	0	1
cute	0	0	1	1	0	1	0
dog	0	1	0	1	1	1	0
is	0	1	1	0	0	1	0
love	0	0	1	0	0	0	0
my	0	1	1	1	0	0	0
student	1	0	0	0	0	0	0

Figure 1: *Co-occurance matrix*

```
# GET MAXIMUM CLIQUE
similarity_count = dict(Counter(
    similarity_lst))
max_pair = max(similarity_count.values())
```

Listing 4: *Pseudocode*

Unfortunately, the implementation works for small datasets and not large datasets like the ones in the task. The RAM crashed with over roughly 150 rows.

2.7 Testing Performance

It is hard for word embedding to test for accuracy since there is no testing dataset in this task. All data are put into the model for calculating the embeddings. In this case, a way to obtain the performance of the model is to get the closest neighbour of a word in the embedding space and see if the model gives a similar word instead of a total non-related one.

For this task, the closest neighbour is calculated for some words. Words are chosen from the corpus, and then get the closest word in the embedding space. Three verbs, three adjectives, and three nouns are given to test the embeddings. Two words from each category will be selected from the most frequent 100 words in the corpus, while one word from each category will not.

3 Discussion

The performance of the models do not turn out good for both datasets. Both results in the dataset do not make sense. For both datasets and models, no related output words are obtained. The loss after 12 epochs is still pretty high, not to mention sci-fi dataset results with only 2 epoch trained. In addition, updating the embedding weight matrix for each data also causes the unstable gradient. Plus, the epoch is not enough for it to converge, leading to poor results.

In addition to the causes from the training process, due to memory and computational limitations, datasets are trained on subsets, which means a lot of words are not trained in various contexts, causing the performance to suffer.

Theoretically, reading from left to right or reading from right to left should not affect the result. Since all input words have equal weights, CBOW is not sequential. Although the result is not ideal, the sci-fi dataset result does show more or less one or two same closest neighbours for the same given word regardless of reading from left or from right.

Besides, with the same words, both datasets do not provide the same neighbours. This is because words are used under different contexts in different datasets, and thus the closest neighbours might not be the same.

3.1 Performance Analysis Result

In the table below, closest neighbours obtained from the model with a word provided from the corpus will be shown. Words in blue are the most frequent 100 words in the corpus, while words in red are the least frequent 100 words in the corpus.

For Trip Advisor dataset:

- Adjective (left to right)
  - great: acurately, brewery, ginza, weekendat, yumi
  - good: scrape, combination, value, restaurantitalianwe, vacate
  - dreadful: claredon, againanyway, daynightammidnightnoontime, exuded, pitched
- Adjective (right to left)
  - great: hotplate, personi, sented, siteexpedia, affordable
  - good: bakeware, phobic, lapses, sublime, mildred
  - dreadful: unfamiliar, beautifulwhat, familila, coordinator, breadrolls
- Verb (left to right)
  - stayed:lorologio, disappointedupon, amazingthe, byi, leaguewe
  - like: resortl, empressed, mcdonaldsthe, inappropriate, anight
  - talk: sister, legroom, varadero, exhibitioncongress, banged
- Verb (right to left)
  - stayed: imperceptible, hatsoff, steakhouseexcellent, definitley, welcomedmedical
  - like: condodwellers, kulturforum, unbeatable, burgundy, oom
  - talk: reinforse , stuckday, founty, rate, smile
- Noun (left to right)
  - hotel: david, nonrelaxing, sideafter, fugazi, cupbring

  - room: weredo, debated, inducing', experiencewithout, kayaking
  - tea: rhumba, flatwe, balestri, oppossed, chandelier
- Noun (right to left)
  - hotel: fantasticlovely, alhough, bugsfor, expectations, niceback
  - room: oasisbeach, areasquite, irregardless, nicola, aircraftoutside
  - tea: boutiqueish, sunbathingthe, cheerfulcould, roadall, sherryport

For Sci-fi story dataset:

- Adjective (left to right)
  - actual: pillow, scan, onist, candidacy, statement
  - new: awfullooking, vibrator, brightcolored, peers, lucidate
  - pending: retrieve, adantapr, walltalkie, rabbleman, depressing
- Adjective (right to left)
  - actual: lu, prevented, watch, scarlet, statement
  - new: cutlets, selfeffacement, esophagus, donkeys, optimum
  - pending: hangdog, lax, thirsty, something, hobbying
- Verb (left to right)
  - published: resumed, strategists, giliu, nightbirds, helo
  - put: eggshell, ethnological, glimmer, abstractly, uhhh
  - buy: buster, cloing, aperture, desperately, tareai
- Verb (right to left)
  - published: fearful, flunked, bienvenu, coaxed, electronic
  - put: eggshell, script, boychild, rayburn, peers
  - buy: detonator, pages, cloing, tareai, pranced
- Noun (left to right)
  - story: dumps, german, lizardhead, ranchtype, wideeyed
  - coffee: ddt, strives, refraction, mansion, withering
  - magazine: associative, laboratory, degenerating, butler, drudgery
- Noun (right to left)
  - story: chaos, joining, german, radioak, plantings
  - coffee: toughed, refraction, oiled, suikuitureand, storyline
  - magazine: obstetrics, degenerating, classified, butler, dialect

Two words that appear in both datasets are also output for comparison.

- tea (left to right)
  - Trip Advisor: humba, flatwe, balestri, oppossed, chandelier
  - Sci-fi Story: odder, passageway, slang, dancers, unstuck
- tea (right to left)
  - Trip Advisor: outiqueish, sunbathingthe, cheerful- could, roadall, sherryport
  - Sci-fi Story: sorcerers, whereof, dandy, whistlings, greyed
- great (left to right)
  - Trip Advisor: acurately, brewery, ginza, weekendat, yumi
  - Sci-fi Story: richest, decorated, mont, irrational, help
- great (right to left)
  - Trip Advisor: hotplate, personi, sented, siteexpedia, affordable
  - Sci-fi Story: irritation, mont salient, buca, decorated

Bibliography

arvindpdmn, devbot5S (n.d.). *Word Embedding*. URL: https://devopedia.org/word-embedding. (accessed: 27.10.2022).

Barg, Alexander and Wei-Hsuan Yu (2013). "New bounds for equiangular lines." In: *Discrete geometry and algebraic combinatorics* 625, pp. 111–121.

Basnet, Bikal (n.d.). *CBOW / Skip-gram Drawbacks : Sense Embedding*. URL: https://deepdatascience.wordpress.com/2017/06/18/cbow-skip-gram-drawbacks/#:~:text=Question\%20\%3A\%20Is\%20there\%20any\%20other,and\%20York\%20two\%20different\%20words.. (accessed: 27.10.2022).

Brownlee, Jason (n.d.). *What Are Word Embeddings for Text?* URL: https://machinelearningmastery.com/what-are-word-embeddings/#:~:text=A\%20word\%20embedding\%20is\%20a,challenging\%20natural\%20language\%20processing\%20problems.. (accessed: 27.10.2022).

Fangyug (n.d.). *NLP: Word Embedding Techniques for Text Analysis*. URL: https://medium.com/sfu-cspmp/nlp-word-embedding-techniques-for-text-analysis-ec4e91bb886f. (accessed: 27.10.2022).

Khandelwal, Renu (n.d.). *Word Embeddings for NLP*. URL: https://towardsdatascience.com/word-embeddings-for-nlp-5b72991e01d4. (accessed: 27.10.2022).

Notes, Rainy (n.d.). *Word Co-occurrence Matrix Implementation and Visualization*. URL: https://rainynotes.net/co-occurrence-matrix-visualization/. (accessed: 27.10.2022).

Patel, Kevin and Pushpak Bhattacharyya (Nov. 2017). "Towards Lower Bounds on Number of Dimensions for Word Embeddings". In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Taipei, Taiwan: Asian Federation of Natural Language Processing, pp. 31–36. URL: https://aclanthology.org/I17-2006.

Prabhu (n.d.). *Understanding NLP Word Embeddings — Text Vectorization*. URL: https://towardsdatascience.com / understanding - nlp - word - embeddings - text - vectorization - 1a23744f7223. (accessed: 27.10.2022).