Word Embedding

Yu-Chun Chen
yuchun_chen@berkeley.edu

Jiayi Wang
jiayi_wang@berkeley.edu

Yujie Xu
yux121@berkeley.edu

Text representation has always been a challenge in the field of machine learning, particularly due to the messy and unorganized nature of raw text. Past effort includes using previous knowledge of the language, which fails to scale massively. In order to transform text into fixed-length and numerical form that computer algorithms favor, and at the same time retaining the rich meanings included in the context, several word embedding techniques have been developed to meet the challenge.

1 Bag of Words

An introductory skill to text feature extraction is called Bag of Words. BoW is a simple and flexible method, making it a reasonable start for our journey into text representation. The technique focuses on counting occurrences of known vocabularies and, as a result, discards the structure and order of sentences (thus a "bag" or words). With BoW, a piece of text is transformed into a vector of integers, with each integer representing the number a certain vocabulary occurs in this text.

For instance, consider the following two sentences:

1. I love machine learning!
2. Machine learning is an interesting field.

Now we first construct our known vocabulary (also called tokens in some papers) from all the unique words found: { i, love, machine, learning, is, an, interesting, field }. Then for each sentence, we measure the occurrences of each word in the vocabulary. As a result, the two sentences become vectors: [1 1 1 1 0 0 0 0] and [0 0 1 1 1 1 1 1]. We can observe that the similarity of these sentences represents how similar are the occurrences of used words. Note that here we also discard punctuation and lower-case all the terms. In the real world, eliminating common stop-words such as “is” and “an” is also a common practice to reduce vector dimension that is relatively less helpful in representing meaningful information.

Due to the simpleness of the technique, BoW also have obvious drawbacks. The biggest problem lies in the fact that, as mentioned above, BoW cannot retain information of structure and order of words. In addition, size of vocabulary and length of vectors increase rapidly, while entries are mostly 0, with number of texts we want to represent. As a result, we want to learn more complex text representation techniques that can use dense vectors to capture meanings in grammar and context.

2 TF-IDF

TF-IDF stands for “Term Frequency Inverse Document Frequency.” It’s a numeric statistics to score the importance of a word to a document based on how frequently it appears across multiple documents in a collection.

Intuitively, if a word appears frequently in a document, it’s important to this document. We need to give this word a high score. But if a word appears in many documents, this means that it’s not a unique identifier for a document. We need to give this word a low score. For example, common

words like “the” and “for,” which appear in many documents, will be scaled down. Words that appear frequently in a single document will be scaled up [1].

As you can see, there are two parts of TF-IDF:

Term Frequency (tf) captures the frequency of each word in a document. There are many ways of calculating this frequency. The simplest one is a raw count of word occurrences in a document. There are other ways to adjust the frequency by the length of a document or by the frequency of the most frequent word in a document. We will use the ratio of number of times the word appears in a document compared to the total number of words in that document. The TF score of word t in document d is

$$tf_{t,d} = \frac{n_{t,d}}{\sum_k n_{k,d}} \tag{1}$$

Inverse Document Frequency (idf) measures how common or rare a word is in the entire document. It upweights less frequent terms and downweights more frequent terms. It is the ratio of total number of documents over the number of documents that contain a word.

However, the TF score is a proportion, so it is a value between 0 and 1. Let’s say if there are 100 million documents and the number of docs with the given word is 10, then IDF is 100M/10 = 10M, which is much larger than TF score. The value of TF-IDF is biased since only IDF dominates the score. Thus, in order to dampen the effect of large IDF numbers, we apply the logarithmic transformation. The IDF score of the word t is

$$idf(t) = \ln(\frac{N}{df_t}) \tag{2}$$

Combining equation (1) and equation (2), we come up with the TF-IDF score for the word t in the document d from the document set D is:

$$tfidf(t, d, D) = \frac{n_{t,d}}{\sum_k n_{k,d}} * \ln(\frac{N}{df_t}) \tag{3}$$

Note that same word in different documents might have different tf-idf score.

3 Co-occurrence Matrix

Co-occurrence matrix is another intuitive count-based strategy, but a special one because it captures the semantics among words by collecting statistics on word co-occurrences, which both BoW and TFIDF approaches fail. Essentially, we want to measure how frequently that two words could co-occur in, say, three consecutive words in a corpus. An advantage of co-occurrence matrix is that it can be used at anytime once computed.

In our co-occurrence matrix X, entry X_{ij} represents the number of times word i appears in the context of word j. We could define our own window size for how far we consider two words "co-occur." Then, we can apply PCA/SVD to the co-occurrence matrix to find lower-dimension word-representation vectors. Note that sentence embedding with word vectors is another long topic, thus we will focus on the basic strategy: averaging word vectors for a sentence/document.

4 Word2Vec

The principles behind frequency-based approaches are intuitive, and thus easier to implement. However, a common drawback among them is their inability to capture word orders and semantics. Although Co-occurrence matrix preserves some semantic relationships, it requires huge memory to both compute and store the matrix. As a result, predictive models, driven by measures of loss in semantic and contextual information, are developed to make use of context and provide vector representations of words. We hope that, by minimizing the loss of semantic and contextual information, these word vectors can capture semantics and orders of the words. For example, words "teacher" and "professor" should have vectors closer in the vector space than words "teacher" and "apple".

Word2Vec is an effective predictive model that creates words' distributed representations using shallow neural networks. The distributed representation describes the relationship between words. The context words have closer relationship with the center word. Therefore, the objective for Word2Vec model is to maximize the similarity of word representation of the context words to the center word. The Word2Vec model has two methods, Continuous Bag Of Words (CBOW) and Skip Gram.

Context window The context words for a center word are the words near the center word within a context window whose size is set by you. For example, with window size = 2, given a sentence "The quick brown fox jumps over the lazy dog." [**The** quick brown], [The **quick** brown fox], [The quick **brown** fox jumps], [quick brown **fox** jumps over] are the context window for the first four words respectively.

Input/One hot encoding One hot encoding is a way of representing categories into vector forms. After building the word to index mapping, a word can be represented by a vector with its corresponding index being 1 and 0 elsewhere. The neural network takes input in the form of one hot vector.

Softmax Classification is a function that turns a vector of V real values into a vector of the same length that sums to 1, and each value is between 0 and 1, so that they can be interpreted as probabilities. The network performs the Softmax Classication by applying softmax function (4) to the output layer. For Skip Gram, $W_{O,j}$ represents the probability for j^{th} word of being the context word for W_I.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}} \tag{4}$$

Output Denote u as the output layer of the network. Take Skip Gram as example, the probability for $W_{O,j}$ of being the context word of the center word W_I is given by

$$\frac{e^{u_j}}{\sum_{j'=1}^{V} e^{u_{j'}}} \tag{5}$$

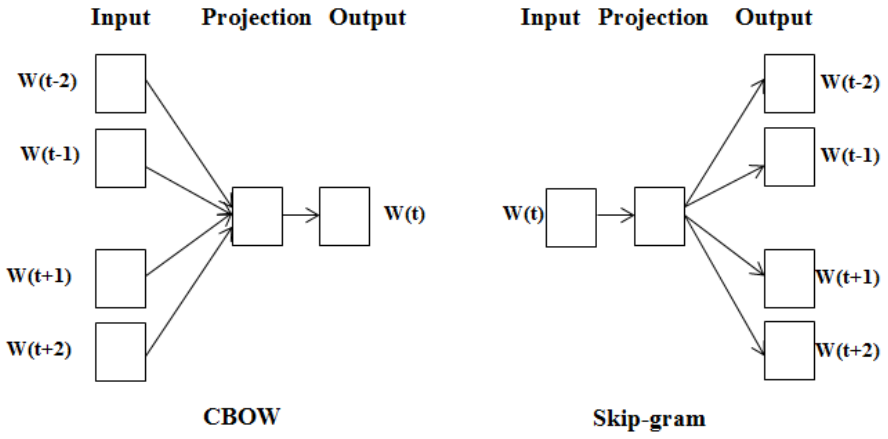

Figure 1: word2vec

4.1 CBOW

CBOW takes in the context of the word and predicts the word corresponding to the context. The CBOW model uses the average of the context words as input, and applies softmax layer to give the prediction of the center word.

Objective The objective for CBOW is to maximize the probability of the center word given context words. The loss function is given by

$$E = -\log p(W_O|W_{I,1}, ..., W_{I,C}) = -\log \frac{e^{u_{j^*}}}{\sum_{j'=1}^{V} e^{u_{j'}}} = -u_{j^*} + \log \sum_{j'=1}^{V} e^{u_{j'}} \tag{6}$$

Where W_O is the center word, $W_{I,1}$, ..., $W_{I,C}$ are C context words, and u_{j^*} is the node of the center word in the output layer.

4.2 Skip Gram

Skip Gram is the reverse of CBOW, and it takes in the center word and predicts the context.

Objective The objective for Skip Gram is to maximize the probability of context words given a center word. The likelihood equation is given by

$$P(W_{O,1}, W_{O,2}, ..., W_{O,C}|W_I) = \prod_{c=1}^{C} \frac{e^{u_{j_c^*}}}{\sum_{j'=1}^{V} e^{u_{j'}}} \tag{7}$$

Where $W_{O,1}$, ..., $W_{O,C}$ are C context words, W_I is the center word, and $u_{j_c^*}$ is the node of c^{th} context word in the output layer. The loss function for this problem is the negative logarithm of the maximum likelihood, and the derivation of loss function is left to the coding assignment.

4.3 Comparing CBOW and SG

Since in Skip Gram we are predicting the context words while in CBOW we are only trying to predict the center word, it takes much longer time to train a Skip Gram Word2Vec. Moreover, as CBOW uses all context words for center word prediction, it performs better on large training datasets. Skip Gram also works well on smaller sets of training data, and represents less frequent words better.

5 GloVe

GloVe is an unsupervised algorithm combining both the perspectives of count-based and predictive approaches to create word representation: it uses co-occurrence matrix as part of the objective to train and obtain word vectors.

First of all, different from standard co-occurrence matrix, the algorithm gives less weights to more distant words. Moreover, counts are normalized and log-smoothed in the co-occurrence matrix. Given the co-occurrence matrix, the objective of GloVe algorithm is to find word vectors such that the dot product of w_i and w_j (plus some bias terms) is approximately the ijth entry. This is done by first setting random weights for every word vectors, then using stochastic gradient descent to obtain final word vectors with the following loss function:

$$L = \sum_{i=1}^{V} \sum_{j=1}^{V} f(G_{ij})(w_i^T w_j + bias - G_{ij})^2 \tag{8}$$

In this equation, G is the normalized and log-smoothed transformation of co-occurrence matrix obtained, while $f(G_{ij})$ is a weighting function to prevent learning from only extremely common word combinations.

The reason these transformations of co-occurrence matrix and the above objective are desired is beyond the scope, but the intuition behind the dot product is that it captures the linear structure of word vectors, while normalizaton and log-transformation produce a measure for probabilities of co-occurrence from raw counts. They together enable the trained GloVe vectors to capture nuance of words quantitatively. For instance, $w_{king} - w_{man} + w_{woman} \approx w_{queen}$. Ideally, we expect better performance on classification tasks from GloVe since it incorporates global statistics given in the co-occurrence matrix while Word2Vec does not. However, they turn out to perform similarly overall in practical use.

The authors of GloVe provide several different sets of pre-trained word vectors that save a lot of work for engineers, which are also used in industry and researches. These word vectors were trained with corpus from different common sources and have various dimensions. Your focus will be learning how to use these pre-trained vectors in machine learning tasks.

References

[1] Roshan Kumar Gupta. *An Introduction to TF-IDF Using Python*. Medium, Analytics Vidhya. https://medium.com/analytics-vidhya/an-introduction-to-tf-idf-using-python-5f9d1a343f77.

[2] Xin Rong. *word2vec Parameter Learning Explained*. https://arxiv.org/abs/1411.2738

[3] Chris McCormick. *Word2Vec Tutorial - The Skip-Gram Model* http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

[4] Matthias Richter. *Comparing Word Embeddings*. Towards Data Science. https://towardsdatascience.com/comparing-word-embeddings-c2efd2455fe3

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. *GloVe: Global Vectors for Word Representation*. https://nlp.stanford.edu/pubs/glove.pdf