S20CS7.401 Introduction to NLP **(Date: January 18, 2020)**

Tutorial 1: Introduction to Word Representation

Instructor: Manish Shrivastava *TA:* Alok Debnath

Abstract

A computationally efficient manner of the representation of words in order to capture lexical meaning and the relation between words is a major topic of research in NLP, and for good reason. Efficacious representations capture syntactic, semantic and latent information about the use of words which is intuitive to human users of language, but there is no real mechanism to represent. Nonetheless, various mechanisms for representing lexical semantics in a distributional manner have come into being, one of the most common ones being known as `word2vec`. In this tutorial, we take a look at what this model represents, how to create such a model of word representation and introduce the readers to some of the notions of word embeddings research that are currently being persued.

1 Word Representations: Where the Story Begins

There are two basic assumptions that are considered in natural language processing:

- **Words are the basic unit of meaning.** As students of linguistics, we are prone to looking at a notion of meaning that is morphological, or based on constructions. As mentioned in class, processing of subword units is challenging due to the state of morphanalyzers. Similarly, multiword expressions and other constructions that consist of multiple words but are a single semantic unit, are ubiquitous in language. However, for the sake of ease in analysis, limited understanding of how to model larger or smaller units appropriately, and the relatively simpler notion of lexical semantics, we consider that words are the basic unit of meaning.
- **'A word is known by the company it keeps'** - J.R. Firth (1957). Firth's distributional hypothesis is the notion that we use to assign words their meaning, i.e. to suggest that a word is given meaning by the words around it. So if two words are often used in the same context, it is likely that they mean the same thing. The distributional hypothesis is one of the foundation stones of modern lexical semantics and the word representations that we see today are based on this underlying hypothesis.

With these assumptions in mind, we start the story of word representation from a very simple and intuitive concept, the document-term matrix. Let's say that we would like to compare the meaning of two large documents. One of the most intuitive ways of doing so is to list out all the words that exist in the document and then take a comparison of the common words between the two documents. This model is known as the document-term matrix model, and example of which can be seen below. Let us consider two documents:

- Document 1: John likes Mary
- Document 2: John loves Mary

The document-term matrix can be seen as:

	John	likes	loves	Mary
Doc 1	1	1	0	1
Doc 2	1	0	1	1

Figure 1: A Simple Document Term Matrix for two documents

Clearly, this is a very inefficient method of finding similarity between documents. The size of the matrix is $V \times D$ where $V : \{v_i \in V \text{ if } \exists v_i \in d_i \forall d_i \in D\}$ and D is the number of documents. For a large number of documents and for a large number of words, this leads to a large and mostly sparse matrix.

An evolved version of this concept of a binary document-term matrix, is a bag-of-words representation. As the name suggests, a bag-of-words model is a model that counts the number of words in a sequence (document,

sentence, corpus) in the form of a `(term, frequency)` tuple. The nature of similarity between the sequences can be constructed based on the same model as in the document-term matrix, except for weighted by the frequency of the terms. While the above mentioned models have shortcomings which are quite apparant, they play an important role in the representation of meaning in documents. These models suffer from the same shortcomings.

While the comparison of raw words is one of the estimations of the meaning of a large document or corpus, we realize these representations are quite memory intensive and sparse. In terms of the representation of meaning, it adds little information. In order to deal with these shortcomings, new models of of representing the meaning of documents were introduced. In these models, documents were represented as a vector. Each vector consisted of non-zero values, which was a function of the frequency with which terms occur in a document (read `tf-idf`. The dimensionality of the vector space is the number of distinct words in the corpus. As documents were represented as vectors, some of the interesting applications of similarity were now based on cosine similarity between the vectors rather than the number or weighted frequencies of the individual words, which provided a more continuous, albeit uninterpretable, notion of similarity.

Vector space models provide numerous benefits, which includes the ability to reduce the dimensionality of the vector space by using singular value decomposition, which led to the development of latent semantic analysis as a method of representing meaning of words in documents. LSA depends on the construction of a document-term matrix, which is then reduced in dimensionality by singular value decomposition, i.e. it provides a low-rank approximation of a of the document-term matrix. While this model has multiple advantages, note that it can not be generalized or scaled, as the representations are specific to the document on which it is trained. It is also computationally extremely expensive.

2 Introducing `word2vec`

With the introduction of the above models for document meaning, the question of word meaning arises as the degenerate case of a document. In this section, we discuss the meaning of words embedded in a vector space as introduced to us by Miklov and his team at Google in 2013, with two models, the continuous-bag-of-words and the continuous-skip-gram models of word embeddings, known collectively as `word2vec`.

Students of NLP are invariably bound to run into `word2vec` as the first known representation of word meaning that could efficiently estimate the meaning of words from large corpora with little loss in generality, and therefore became the foundation-stone of word embeddings research in the previous decade. It has been touted as one of the most influential works in the field due to the rate of evolution of the subject after its introduction, as for the first time, there was a scaleable and efficient representation of words which could be used for syntactic and semantic downstream tasks.

Internally, `word2vec` creates two representations of words, an 'input' and an 'output' representation. We shall now describe two models of learning word representations based on a loose notion of distributionality.

2.1 CBOW: Continuous Bag-of-Words Model

The continuous bag-of-words model aims to predict a output word based on n input words in a window. This is a task similar to language modeling. Therefore, based on the input representation ($\vec{w}_i$) of $\frac{n}{2}$ previous and $\frac{n}{2}$ next words, we predict a focus word. $\texttt{corpus}[i]$ represents the ith word of the corpus, $\vec{v}_o[w]$ is the output representation of word w and $\vec{v}_i[w]$ is the input representation.

Therefore, we can predict the output representation in a window of n input representation is computed as follows:

$$\vec{v}_o[\texttt{corpus}[(t)]] \simeq \vec{v}'_o[\texttt{corpus}[(t)]] = \sigma\left(\frac{\sum_1^{\frac{n}{2}} \vec{v}_i[\texttt{corpus}[(t-i)]] + \sum_1^{\frac{n}{2}} \vec{v}_i[\texttt{corpus}[(t+i)]]}{n}\right) \tag{2.1}$$

Here we provide a small code-snippet that illustrates the implementation of the CBOW model.

```
// Initialization
layer1_size = 100;
real *neu1 = (real *)calloc(layer1_size, sizeof(real));
real *neu1e = (real *)calloc(layer1_size, sizeof(real));
...

// CBOW training loop
cw = 0;
```

```
    for (a = b; a < window * 2 + 1 - b; a++)
    if (a != window) {
        c = sentence_position - window + a;
        if (c < 0) continue;
        if (c >= sentence_length) continue;
        last_word = sen[c];
        if (last_word == -1) continue;
        for (c = 0; c < layer1_size; c++)
            neu1[c] += syn0[c + last_word * layer1_size];
        cw++;
    }
    if (cw) {
        for (c = 0; c < layer1_size; c++) neu1[expc] /= cw;
    ...
    // HS or Neg Sampling
    ...

    // Update input vector
    for (c = 0; c < layer1_size; c++)
        syn0[c + last_word * layer1_size] += neu1e[c];
```

During training, the idea, we try to maximize $(v_o(t) \cdot v'_o(t))$, so as to predict the correct output representation based on the correct input representation. Therefore, the correct input representation is then used as the representation of the word which is used in downstream tasks. This model is called a continuous bag-of-words model, as the order of the vectors does not matter (as we are only taking an average), but unlike bag of words, the nature of the value associated to a word belongs to a continuous set rather than a discrete set of whole numbers (i.e. frequency).

2.2 Skip-Gram Model

Skip-gram aims to achieve the opposite of CBOW, in that, given an input word representation, it attempts to predict the surrounding context words associated with that input word. More formally, give a word w, its input vector representation $\vec{v}_i(w)$ and a context window of n, the model tries to maximize the total log-probability of the predicted $\frac{n}{2}$ previous and $\frac{n}{2}$ next words; i.e.:

$$p = \text{softmax}\left(\vec{v}_o[\texttt{corpus}[(t+j)]] \cdot \vec{v}_i[\texttt{corpus}[(t)]]\right) \qquad \text{where} \quad -c \leq j \leq c, j \neq 0 \tag{2.2}$$

```
    // Initialization
    layer1_size = 100;
    real *neu1 = (real *)calloc(layer1_size, sizeof(real));
    real *neu1e = (real *)calloc(layer1_size, sizeof(real));
    ...

    // Skip-Gram Training Loop
    for (a = b; a < window * 2 + 1 - b; a++)
        if (a != window) {
            c = sentence_position - window + a;
            if (c < 0) continue;
            if (c >= sentence_length) continue;
            last_word = sen[c];
            if (last_word == -1) continue;
            l1 = last_word * layer1_size;
            for (c = 0; c < layer1_size; c++) neu1e[c] = 0;
    ...

    // HS or Neg Sampling
    ...

    // Update input vector
    for (c = 0; c < layer1_size; c++) syn0[c + l1] += neu1e[c];
```

3 Training Mechanism

In this section, we discuss the mechanism in which these models are trained in depth. There are two main training mechanisms introduced, known as hierarchical softmax and negative sampling. The main difference

between the two is the treatment of the vectors within or outside the designated window of prediction. Both CBOW and skip-gram have both training mechanisms.

3.1 Hierarchical Softmax

Essentially, a hierarchical softmax is a computationally efficient approximation of the softmax over the entire probability distribution. In CBOW, the hierarchical softmax representation is used as the estimation function in order to provide the necessary non-linearity to the estimation of the output representation. In Skip-Gram, the hierarchical softmax is used to construct a binary tree with the words as leaves and each node being the relative probabilities of the child nodes, so the weights learned will be on the nodes traveresed rather than the individual probabilities themselves, which is far more efficient.

The code snippets for both are presented here.

```
    // Propagate hidden -> output (CBOW)
    for (c = 0; c < layer1_size; c++)
        f += neu1[c] * syn1[c + l2];
    ...

    // Propagate hidden -> output (Skip-Gram)
    for (c = 0; c < layer1_size; c++)
        f += syn0[c + l1] * syn1[c + l2];
    ...

    // Propagate errors output -> hidden
    for (c = 0; c < layer1_size; c++)
        neu1e[c] += g * syn1[c + l2];
    ...
    // Learn weights hidden -> output (CBOW)
    for (c = 0; c < layer1_size; c++)
        syn1[c + l2] += g * neu1[c];
    ...

    // Learn weights hidden -> output (Skip-Gram)
    for (c = 0; c < layer1_size; c++)
        syn1[c + l2] += g * syn0[c + l1];
```

3.2 Negative Sampling

One intuition behind hierarchical softmax is that it tries to increase the probability of different words with the same meaning being similar by pushing increasing their similarity. Negative sampling, in that sense, is the mechanism used to reduce the similarity of a word representation with other random words in the corpus iteratively, except for words that occur in the same context. Negative sampling chooses k negative samples and aims to get an efficient representation of words by distinguishing the output representation from a distribution of other randomly sampled words outside a context window.

We can see the code snippet for negative sampling below.

```
    // CBOW
    for (c = 0; c < layer1_size; c++)
        f += neu1[c] * syn1neg[c + l2]
    ...
    for (c = 0; c < layer1_size; c++)
        neu1e[c] += g * syn1neg[c + l2];
    for (c = 0; c < layer1_size; c++)
        syn1neg[c + l2] += g * neu1[c];
    ...

    // Skip-Gram
    for (c = 0; c < layer1_size; c++)
        f += syn0[c + l1] * syn1neg[c + l2];
    ...
    for (c = 0; c < layer1_size; c++)
        neu1e[c] += g * syn1neg[c + l2];
    for (c = 0; c < layer1_size; c++)
        syn1neg[c + l2] += g * syn0[c + l1];
```

3.3 "Hyperparameters"

A large portion of the performance of word embeddings such as `word2vec` depends on how well its hyperparameters are tuned. Hyperparameters may be thought of as variables that can not be learned by the network so have to be tweaked externally. Some of the hyperparameters `word2vec` include the number of iterations of training, the dimensionality of the word representation, the number of negative samples chosen, the size of the context window size chosen for prediction, the size and type of the input corpus and so on.

The reason these change the learning is because, roughly speaking, changing these modifies the amount of information learned during training, which can yield vastly different results even if the algorithm is implemented exactly as the authors intended it to be implemented. A large part of machine learning research depends on choosing and using the right hyperparameters for the task at hand, which is not always very intuitive.

4 Evaluation of Word Embeddings

Now that we have our word embeddings, or word representations, how do we say that they accurately represent the meaning of the words as we have expected of it? This is where the notion of evaluating word embeddings comes in. Word embedding evaluation is one of the most rapidly evolving fields of research in NLP, as the evaluation of these representations becomes a benchmark for future representations to be trained as well. One of the features of both of Mikolov's 2013 papers was the contruction of linguistically rooted evaluation mechanisms, lexical similarity and analogical reasoning.

Lexical similarity of a word $w \in W$ with all other words in the corpus $w' \in W; w' \neq w$ can now be easily estimated by maximizing the cosine similarity between the two vectors $\vec{v}_i[w]$ and $\vec{v}'_i[w']$. The authors choose to normalize the word vectors before taking the cosine similarity, presumably to reduce the noise due to the length of the vectors. Analogical reasoning, made famous by analogies such as `man : king :: woman : queen(?)` are also considered notions of lexical semantics, as the test can demonstrate how well the representation understands the relation between two words in the context of applying that relation to a third word. This is computed by using $\vec{v}_i[\texttt{king}] - \vec{v}_i[\texttt{man}] + \vec{v}_i[\texttt{woman}] \simeq \vec{v}_i[\texttt{queen}]$.

In practice, there are also a number of extrinsic evaluation mechanisms associated with each vector representation, i.e. their use in downstream tasks on text classification, language modeling and other NLP Applications. It was found that `word2vec` was the first efficient estimation of the representation of words in a large corpus which could therefore be used well for other tasks. A lot of machine learning based research in NLP was based on the use of these embeddings in order to maximize the performance of their networks and methods.

5 References

- Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).
- Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.
- HackerNews post and comments on word2vec post by Siddharth Bhat: https://news.ycombinator.com/item?id=20089515 (post is also linked)
- Word2Vec code for reference: https://github.com/tmikolov/word2vec