Learning word embeddings

The main question is how to represent the word to machine learning algorithms. Such algorithms understand numbers only; they see a picture as pixels which in turn are a matrix of numbers. In that sense, we need to find a way to represent the word numerically. This numerical representation for the word is called the ***word embeddings***. In simple machine learning algorithms, we can use TF-IDF that transforms text to feature vectors that can be used as input to an estimator.

Humans can guess the meaning of a word from the context it used in. Firth, J. R has a famous quotation: “You shall know a word by the company it keeps”. ***Embedding matrix*** is an attempt to model such relationships. As an example, you may find a numerical representation for the words “queen”, “king”, “man” and “woman” such that these representations can show that the relations between the two words “king” and “queen” is the same relation between the two words “man” and “woman”.

What happens is that you train a network over a large corpus. You select a target word in the corpus then you randomly sample from the neighborhood of that target word other words (context word) to train the network to predict that target word. But first, how you will represent a word in the first place. This is where a ***one-hot vector representation*** comes to solve this problem. Actually, you will have a vector of zeros that has a length that is equal to the number of unique words, say 50K words. To represent the word “the”, you will have a vector of length 50K that has a value of “1” at position say 3000 and zeros anywhere else. Each word/term in the corpus has a unique position in this vector.

When the network is fully trained, as you give it any “target word”, it gives you a probability distribution over all the context words (at output layer) which represents how likely each of these context words may show up within a certain window of that target word. If the system successfully can do this task, then the weights of the hidden layer have encoding some meaningful information about the target word based on its context. The weights of the hidden layer are the ***word embeddings***. This vector is a ***dense vector*** which means it does not have zeros. An ***embedding matrix*** consists of a list of words and their corresponding embedding vectors. One can see an embedding matrix as a digital dictionary of the words that appear in the corpus.

Word2vec and ***GloVe*** are two widely-known approaches to embedding words. What we have discussed above is a ***neural language model,*** such a model is very expensive in terms of computational complexity. However, ***Word2vec*** omits the hidden layer in the neural network to reduce the cost of computation. Word2vec comes into two flavors, actually what we have discussed in the last paragraph has the same goal of the ***skip-gram*** flavor; predict the contextual words that may surround a target word. The continuous bag of words or ***CBOW*** works the other way around; the network tries to predict the target word given the context words surrounding that target word.

GloVe tries to take advantage of words co-occurrence statistics. The main argument is that the meaning relations between words are related to the ratio of their co-occurrence probabilities with a certain word. As an example, the word “ice” is more related to “solid”, not “steam”, so t $P(solid|ice)/P(solid|steam)$ will be relatively high, as $P(solid|ice)$ is expected to be more than one and $P(solid|steam)$ is less than one. In that sense, GloVe takes the co-occurrence counts of words as an input to the network instead of the entire words in the corpus as in word2vec. Some studies applied a dimensionality reduction to the word

embeddings. The evaluation suggests that the ***reduced word embeddings*** perform better or similar than the original ones.

GloVe and Wordvec as we have discussed above are neural-based models that try to predict the surrounding words. On the other hand, the ***co-occurrence matrices*** are count-based models that count the co-occurrences of words and produce a term-term matrix. GloVe is also based on co-occurrence matrices; one can see that GloVe is a merge between the distributional semantic models and word2vec. The Word embeddings are trained using a huge corpus, maybe millions of words, But this heavily training part is done once. After that, the pre-trained word embeddings could be used to seed another model that works on a smaller corpus. Table 1 shows a comparison between different lexical meanings representation.

One of the ***drawbacks*** of the traditional word embeddings is that they cannot differentiate between different senses for the same words. For the two phrases “trees bark” and “dogs bark”; the word “bark” will have the same vector representation. As we have discussed before, to understand a word, you correlate it with the previous words; you do not forget about the previous words. Traditional feedforward neural networks cannot remember the previous prediction. The problem is known as *long-term dependencies* where you need to look to the previous information to perform the current prediction and the gap between the current and previous information is very large. Long Short Term Memory networks (***LSTM***s) are gated type of recurrent neural networks (RNN) that can deal with long-term dependencies. These networks have special memory gates that enable them to memorize and forget the previous states of the network (add and remove information to the current state).

Contextual word embeddings are introduced to overcome the drawback of having word representations that cannot differentiate between different word senses. ELMo employs a deep bidirectional language model (BLiM) to incorporate context into word representation. BLiM is a two LSTM, one is forward LSTM which predicts the next word based on the previous words while the second LSTM predicts in the reverse direction, i.e. predicts the previous word based on the words that are successive to it. Elmo uses the representation produced by each of the BLiM and the character encoding to produce different word representations that are based on context.

The ***probabilistic language*** model accounts for the context as it considers the word order and using n-gram models. As the number of unique words increases, the number of sequences increase. To overcome this problem, neural networks are trained to predict the probability distribution over the corpus words given some contexts.

Table 1 - Word embedding Comparison based on (Khattak 2019)

Model	Freq. based	Prediction based	Prediction	Sub-word info	Order of dim.
One-hot Encoding					Vocabolary_Size
Co-Occurrence Matrix	✓				$Vocabolary\_Size^2$
CBOW		✓	Target word based on context		N
Skip-gram		✓	Context based on the focus word		N
GloVe	✓				N
ELMo		✓	Left and right words	✓	N

Questions:

1. Can we instead of looking for surrounding words, connect words that have a relation in the dependency tree?
2. Do additional text processing/POS taggers or morphological analyzers improve the performance?
3. Are there any existing frameworks that integrate both DS and FS?
4. Can we use available resources for one language to another language that does not have enough resources or not standardized? (Classical Arabic and Dialect Arabic)

Bibliography

Khattak, Faiza Khan, et al. "A survey of word embeddings for clinical text." Journal of Biomedical Informatics: X (2019): 100057.