Sanjana Agarwal, Krish Mahajan, and Dwipam Katariya

ASSIGNMENT 9

1) The word2vec model maps each word from the training data into a low-dimensional continuous vector-space. It constructs a vocabulary from the training data and then learns vector representation of words. Words that are semantically and syntactically similar tend to occur next to one another when the model maps the words. For the output, they're actually maximizing two similarity terms and minimizing a third dis-similarity.
Word2vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text. It comes in two flavors, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. Algorithmically, these models are similar, except that CBOW predicts target words (e.g. 'mat') from source context words ('the cat sits on the'), while the skip-gram does the inverse and predicts source context-words from the target words. Both these can be used with or without negative sampling and also with hierarchical softmax. CBOW is similar to the feedforward NNLM, where the non-linear hidden layer is removed and the projection layer is shared for all words. CBOW classifier is trained with four future and four history words at the input, where the training criterion is to correctly classify the current (middle) word. As unlike standard bag-of-words model, it uses continuous distributed representation of the context.
In the paper, none of the previously proposed architectures have been successfully trained on more than a few hundreds of millions of words, with a modest dimensionality of the word vectors between 50 - 100. Usually NLP systems and techniques treat words as atomic units - there is no notion of similarity between words. In the reference 1 the paper contains results to show how this paper is better than other earlier techniques.

References

[1] https://www.tensorflow.org/versions/r0.11/tutorials/word2vec/index.html
[2] http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
[3] https://arxiv.org/pdf/1301.3781.pdf **Efficient Estimation of Word Representations in Vector Space**
[4] https://www.microsoft.com/en-us/research/publication/linguistic-regularities-in-continuous-space-word-representations/ **Linguistic Regularities in Continuous Space Word Representations**

2) Problem Statement: Print the odd word amongst a group of words.

The code uploaded is code.py
I have considered using the vectors that have been pre-trained by Google. Several articles on the internet, used the Google-News binary files and I considered using the same.
https://github.com/mmihaltz/word2vec-GoogleNews-vectors

Approach:

1) The code requires the user to input a number of words with one word semantically different from the rest.
2) These words are then converted to a list and passed as an input to the method named 'odd.'
3) Using word2vec's inbuilt function, doesnt_match, we get the odd word from the list.
4) For the function to work, we need to train it on some existing data for which we use the google-news-vectors binary file. The model is trained with the load_word2vec_format method.

Efficiency:

1) According to the papers, word2vec is the most efficient tool since it deals with word vectors and has the best accuracy.
2) It can be used to find the dissimilarities and the odd items in various fields.
3) Very accurately, it displays the results of the odd words that the user inputs.

3) This was my first implementation with word2vec and it seems to be a very interesting too. It is very easy to use, and is incredibly fast. Google developers have taken the old word sense disambiguation method and simplified it drastically. They trade the model's complexity for efficiency, that is done by making the tool train on more data instead of making it more complex and train on less data.

It consists of CBOW and skip-gram and it is good how a user can pick which algorithm to work with and add other variants like negative sampling and hierarchal softmax. A user can pick according to what fits one's problem statement. Although similar. they are different in the way that the skip-gram model predicts the neighboring words given the current word. In contrast, the CBOW model predicts the current word *w,* given the neighboring words in the window. It was predicted that skip-gram gives better output than CBOW but that's true only when one trains over more data. This is because CBOW smooth's over a lot of the distributional statistics by averaging over all context words while skip-gram does not.

Although it is easy to use, one big disadvantage is its applications of use, for now there have been only a few places where you can use this like

- To complete incomplete knowledge base or fill in missing data
- To annotate images with phrases/words

Its applications are currently limited. It can be readily used in finding antonym/synonym, spelling correction and stemming. Despite its remarkability, it is not quite clear how this ability can be used in an application.

While I was working on this, I also thought of trying to see if I could train it for a neutral class. I could not find a way to train it to classify into 3 groups, however I may be wrong and there might be a way but it isn't easy as the other parts of the tools. Being able to classify things into just two groups I feel is a drawback and a solution must be found for it.

The papers talk about calculations on word vectors, about addition and subtraction of vectors, a question stands I suppose would be whether multiplication instead of addition and division instead of subtraction will give better results?

Another drawback I feel is that since each word has its vector, joint words like ‘Indiana University,’’ Indiana’ and ‘University’ would not be near each other although there is a similarity. For that one has to use phrase2vec

I would like to conclude by saying it is one of the greatest advances in technology. In spite of the numerous drawbacks stated, it is one of the fastest and most accurate tool till now.