Word2vec Implementation using Torch

Data-warehousing and Data mining (CSE445)
IIIT, Hyderabad

Chinmay Patel
201405627

Dharak Kharod
201405583

Pawankumar P.
201405637

Vinayak Bharti
201405522

ABSTRACT

In this report we provide the details of how we implemented the continuous bag of words (CBOW) model of Word2vec using the scientific computing framework Torch. Word2vec takes a text corpus as input and produces word vectors as output. Word2vec has two famous architectures one is CBOW and the other is the skip-gram model. For a text corpus, given a window size of n words around a word w, 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. For the scope of this project we have implemented the CBOW model and have tried to evaluate the quality of word vectors obtained by measuring the syntactic and semantic accuracy. The original models of Word2vec are insensitive to word order, we try to address the problem here.

Keywords

word2vec; word-embedding; CBOW; torch; lua

1. INTRODUCTION

Word2Vec provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. These representations can be subsequently used in many natural language processing applications and for further research. The word2vec tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words. The resulting word vector file can be used as features in many natural language processing and machine learning applications.

A simple way to investigate the learned representations is to find the closest words for a user-specified word. The distance tool serves that purpose. For example, if you enter 'france', distance will display the most similar words and their distances to 'france',

In many natural language processing tasks, words are often represented by their tf-idf scores. While these scores give us some idea of a word's relative importance in a document, they do not give us any insight into its semantic meaning. Word2Vec is the name given to a class of neural network models that, given an unlabelled training corpus, produce a vector for each word in the corpus that encodes its semantic information. These vectors are useful for two main reasons.

- We can measure the semantic similarity between two words are by calculating the cosine similarity between their corresponding word vectors.
- We can use these word vectors as features for various supervised NLP tasks such as document classification, named entity recognition, and sentiment analysis. The semantic information that is contained in these vectors make them powerful features for these tasks.

It was recently shown that the word vectors capture many linguistic regularities, for example vector operations vector('Paris') - vector('France') + vector('Italy') results in a vector that is very close to vector('Rome'), and vector('king') - vector('man') + vector('woman') is close to vector('queen')

There are two main learning algorithms in word2vec : continuous bag-of-words and continuous skip-gram. The switch -cbow allows the user to pick one of these learning algorithms. Both algorithms learn the representation of a word that is useful for prediction of other words in the sentence.

Several factors influence the quality of the word vectors:

- Amount and quality of the training data
- Dimension of the vectors
- Training algorithm

The quality of the vectors is crucial for any application. However, exploration of different hyper-parameter settings for complex tasks might be too time demanding. Thus, we designed simple test sets that can be used to quickly evaluate the word vector quality.

2. PROBLEM

Finding embedding for a word in given dimension space is a problem with many facets. The basic intuition for

WOODSTOCK '97 El Paso, Texas USA
© 2015 ACM. ISBN 123-4567-24-567/08/06...$15.00
DOI: 10.475/123_4

Word2vec comes from the distributional hypothesis, for it states that words in similar contexts have similar meanings. Intuitively, this means that words that share many contexts will be similar to each other (note also that contexts sharing many words will also be similar to each other). Thus we can make out a word from its context and vice versa. This is the concept of CBOW model described in section 3.1. The major aspects of the Word2vec problem are as below:

- Word vectorization requires a lot of text.
- Bad influence of rare words onto the vectors
- Maintaining the word order becomes computationally intensive.
- Training requires high-memory and high-computing power.

3. APPROACH

We tried two approaches of language models to assign probabilities to a sequence of tokens namely, Skip gram and CBOW(continuous bag of words). These can be explained in brief using an example. Let's take the following statement as an example:

"The cat jumped over the puddle."

Continuous bag of words(CBOW) approach treats the words ("The", "cat", "over", "the", "puddle") as context and from these words predicts or generates the center word "jumped".

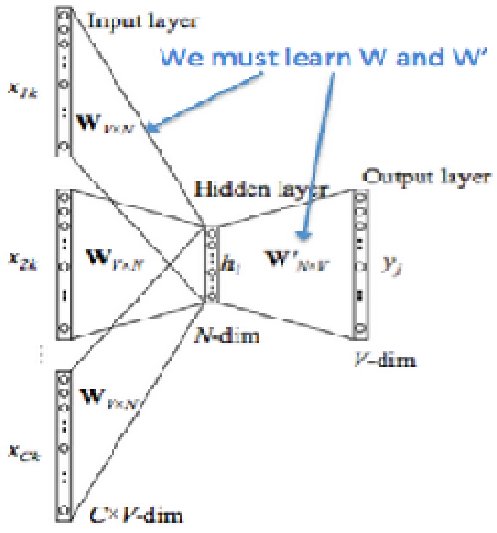

Figure 1: This image demonstrates how CBOW works and how we must learn the transfer matrices

Whereas in the skip-gram model we create a model such that given the center word "jumped", the model will be able to predict or generate the surrounding words "The", "cat", "over", "the", "puddle". Here call the word "jumped" the context.

We tried the skip gram model but in the end we only implemented the CBOW model, which will be covered in detail in the following section.

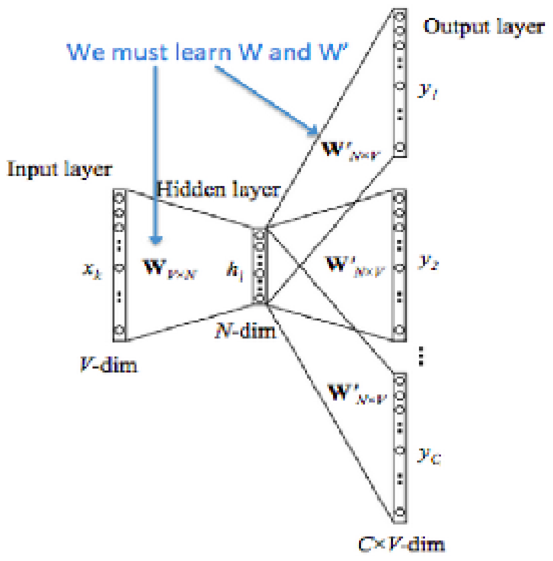

Figure 2: This image demonstrates how Skip gram works and how we must learn the transfer matrices

3.1 Continuous Bag-of-Words(CBOW)

We have followed following steps to implement CBOW model for word2vec:

Creation of vocabulary: Creation of vocabulary: The vocabulary size of given corpus is very high. We have added one parameter for discarding some of the input words: words appearing less than min frequency are not considered as either words or contexts. Importantly, these words are removed from the text before generating the contexts. This has the effect of increasing the effective window size for certain words.

Training: First we are reading file in batchsize because we cannot read whole file in main memory. Each one batch represents single window. For example, if window size is 7 and batchsize is 100, then we are taking total 700 words (100 overlapping windows) in main memory. Then we train each window one by one using deep neural network. Here we have implemented CBOW model, so we are predicting middle word based on neighbour words according to specified window size. After each iteration, network generates vector of predicting(middle) word. In our model we are not averaging the weights in middle layer of network to maintain syntactic structure of the words. We have use stochastic gradient function for calculating prediction error.

Output: Output file contains word and its vector separated by space. The dimensionality of vector depends upon given input parameter. The resulting word vector file can be used as features in many natural language processing and machine learning applications.

Model of our project:

– Create the neural net.
model = nn.Sequential();
– vecdim specifies size of resulting vector
wordvecs = nn.LookupTable(vocabsize, vecdim);
model:add(wordvecs);
– numwords represents the size of neighbour words
model:add(nn.Reshape(numwords*vecdim));
– hidden represents total nodes in hidden layer

```
model:add(nn.Linear(numwords*vecdim,nhidden));
– activation function
model:add(nn.Sigmoid());
model:add(nn.Linear(nhidden,vocabsize));
model:add( nn.LogSoftMax() );
```

4. RESULTS AND LEARNING

Word order was preserved to obtain more syntactic accuracy. CBOW model captured more semantics in the data as compared to the skip-gram model as the CBOW model takes into consideration the neighbour in both sides to predict the word while the skip-gram only considers the previous words for training. The algorithm proceeds by having a sliding window which trains our model on the basis of left and right window words for the center word prediction. We experimented on window sizes ranging from 3-11 obtaining optimal performances when we used a window size of 7. The learning rate which is a hyperparameter in the neural network applies a portion of adjustment to the old weights. Although, iterating through the dataset for multiple times increases the quality of the model we were limited by time and computational power which resulted in us testing out for 5 iterations at max. We tested with the following transfer functions: SoftMax, tanh, Sigmoid, LogSoftMax. The best result was obtained when using the Sigmoid function. The following were the results obtained:

Google/Our Model	**Window**	**Epochs**	**Semantic**	**Syntactic**
Our Model	7	5	1.01	0.02
Google	7	5	2.29	0.96

5. REFERENCES

1. http://cs224d.stanford.edu/lecturenotes/LectureNotes3.pdf
2. http://cs224d.stanford.edu/lecture notes/LectureNotes1.pdf.
3. https://code.google.com/p/word2vec/.
4. https://github.com/yoonkim/word2vec torch
5. https://github.com/torch/tutorials/tree/master/coursera neural nets/assignment2/solution_torch_nn
6. https://github.com/torch/nn/blob/master/doc/training.md#nn.traningneuralnet.dok
7. http://www.lua.org/manual/5.3/.
8. http://tylerneylon.com/a/learn-lua/.