ST446
DISTRIBUTED COMPUTING

FINAL PROJECT

Distributed Word2Vec and Dimensionality Reduction

Author
Aleksander Brynjulf Hübert

May 2020

Contents

Abstract

This paper will explore dimensionality reduction using Singular Value Decomopostion (SVD) of word vectors created using the `pyspark` implementation of `word2vec`. I first explored `word2vec` and limiatatiosn of its distributed implementation. I then explain dimensionality reduction using SVD. I then perfom a numerical experiment using word vectors created from DBLP using `word2vec` and compare their perfomance in the `QVEC` word vector test developed at Carnegie Mellon against vectors that have reduced dimensionality. I find that the vectors with reduced dimensionality still perform relatively well in teh `QVEC` scoring, but the limitation of my data and the `QVEC` makes it difficult to compare very low dimensional vectors.

1 Introduction

With the growth in text data new strides have been made in the realm of Natural Language Processing (NLP). The core of NLP is to computationally process language produced by humans, whether this is text data or speech. Much of NLP relies on the use of word vectors to represent language. Due to the complexity of human speech these vectors are an attempt to encode a vast amount of infomation about words into a simpified space. Though it is impossible currently for NLP to completely encapuslate language it has provided very useful in sentiment analysis, speech to text conversion, as well as text classification and continues to provide new uses.

This paper will explore the continuous bag of word model created using `word2vec`. I will then explore the use of Singular Value Decomopostion (SVD) to reduce the dimensionality of the vectors and compare the performace on a distributed RDD. I first attempted to train the model using the wikipedia dump dataset as a corpus but this was not feasible so I created smaller word vectors for testing purposes using DBLP titles. This allowed me to still compare vectors in relation to one another.

2 The Language Model

There are many ways to create word vectors, using Latent Semantic Analysis we can create word vectors through document frequencies and more recently Facebook's `fasttext` uses deep neural networks to creat word vectors using large corpuses of text. I will focus on the open sourced Google project `word2vec` implementation in pyspark. Though pretrained word vectors are available I wanted to focus on the usage of RDDs and `spark` and explore the creation of word vectors from text data.

2.1 word2vec

In NLP word vectors have shown to be a good method of processing text data. Google researchers developed their own method of creating word vectors, `word2vec` in 2013. This method hopes to encode data about language into large vectors. These vectors are trained based on closeness of words to one another, where words in similar contexts should be more similar than words that show up in completely different contexts.

Though there has been improvements since then the basic idea of the model relies on shallow neural network to create a continuous bag of words model. The model is based on a continuous skip-gram which with the objective of creating word vectors that are good at predicting nearby words [2].

The following formulation describes the skip-gram

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c} log(p(w_{t+j}|w_t))$$

The goal is to optimize the average log-probabilities over the training words represented by w_t where t represents the index of each word in the training sequence [2]. In the formulation the value of c represents the window in which we are accounting for. A larger c value means a larger window we are optimizing over which can lead to better accuracy but larger training times.

Though there are different formulations for optimizing $p(w_{t+j}|w_t)$, the basic fomulation is the *Hierarchical Softmax*. The formulation is as follows

$$p(w_{t+j}|w_t) = \frac{exp(u_{w_i}^T v_{w_j})}{\sum_{l=1}^{V} exp(u_l^T v_{w_j})}$$

where u and v are the vector representations of vocabulary words w_i and w_j and V is the size of the vocabulary. As seen in the formulation there are a few dot products required represented by $u_{w_i}^T v_{w_j}$, requiring many multiplies between vectors.

This is an efficient way to approximate *softmax* requirering $O(log_2(\mathrm{V}))$ compared to basic softmax which is evaltuated in $O(\mathrm{V})$ [2]. This is the optimization formulation that is used in the `pyspark` implementation. There are drawbacks in terms of overall acurracy, the speed-up allows processing of much larger vocabularies.

2.2 Distributed word2vec

There is a widely used implementation of `word2vec` in `pyspark.MLlib`. This implementation is closely connected to the original implementation and uses the *Hierarchical Softmax* optimization function without an option to change it. Using a distributed system has many distinct advantages in speed and storage but there are drawbacks associated with using a particular algorithm on such a system. Not every algorithm is able to fully take advantage of a distributed system.

Using `word2vec` when otimizing the *Hierarchical Softmax* function we are performing a series of dot products between word vectors $(u_{w_i}^T v_{w_j})$. Becaues of this formulation each vector must be stored on all spark executors [3]. If they are stored in a distributed fashion then a network call must be made if there are words that appear close to one another with vectors stored on different machines. With smaller vocaublary sizes this is not as large an issue, but when vocabularies become large either vectors must be stored on the same machine to complete the dot products or the network traffic in the distributed system becomes a bottleneck in calculations [3].

2.3 Singular Value Decomopostion

The singular value decomposition (SVD) is a factorization that given a matrix A we can factorize $A = USV^T$ where U is left singular vectors and V is the right singular vectors. When calculating the SVD we can choose the amount of leading singular values to use k. The SVD factorization offers many advangages such as fast computation of repeated multiplication but its complexity means it is rarely used to find solutions of linear systems. With a computational time of $O(n^3)$ FLOPS with a large constant that means its usage is typically tied to finding singular values and applications of them [1].

A usage of SVD that is common in distributed computing and machine learning is dimensionality reduction. The reduction works by multiplying US. Given a matrix A which is $n \times m$ then we factorize with parameter k. This give us matricies U which is $n \times k$, S which is $k \times k$. This new matrix $B = US$ is related to the principal components and therefore is often used in dimensionality reductions. This factorization though slightly expensive offers the ability to reduce the size of the vectors significantly depending on the leading singular values chosen allowing smaller memory usage and less computations when using the vectors for future calculations.

2.4 Dimensionality of Word Vectors

Exploring dimensionality of word vectors is important for several reasons. Memory concerns in storage of the vectors is important for systems that are not able to dedicate the memory to storing these vectors and a tradeoff of slightly worse perfomance can be acceptable for a reduction in storage size. Smaller vectors also allow easier calculations and easier transfer. With the availablity of pretrained vectors the problem of dimensionality reduction does not lie solely in training but also post processing. A word embedding matrix of 2.5 million tokens can take up 6Gb of memory on a system [4]. Thus transfer, storage, and usage are all considerations especially when running on systems that have many constraints but would benefit with quality word vectors.

3 Methodology

3.1 Data Selection

Using the Wikipedia dump I downloaded the compressed file containing the Wikipedia backup of approximately 5 million articles. This file was 14Gb and usign the `wikiextractor` I was able to uncompress these files and separate the text from the media leaving 5Gb of text files that I could format into an RDD. The code for creating the RDD can be seen in appendix C. With such a large amount of text data I was unable to use this for my experimentation. Though I was able to get the data into the RDD and correctly tokenized for the creation of word vectors the sheer volume of data along with credit and time limitations did not allow me to fully explore this data.

Due to the setbacks when using the Wikipedia data I decided to use the DBLP titles to create word vectors. This smaller dataset was both familiar since I had spent a lot of time familiarizing myself with the Wikipedia dump, and it was small enough to allow for faster experimentation. This data consists of authors, journal, titles, dates of computer science publications. I created an RDD of the titles tokenized which I could then use as my sentences when feeding them into the `word2vec.fit()` function. The final RDD included the titles text split by word into a list. I kept stopwords though I did remove numerics and punctuation from the titles.

3.2 Evaluating Word Vectors

Evaluating word vectors can be a difficult task. There is no intrinsic way to rate them and typically they are judge using their ability to perform NLP tasks such as sentiment analysis [5]. To evaluate my model I decided to use `QVEC` developed at Carnegie Mellon University. This method compares the componentwise correlations with manually constructed word vectors and sucess in this test has been shown to correlate well with performance in downstream NLP tasks [5]. The test works by aligning the dimensions of the word vectors against the lingistic standard. Then the correlation is measured against the aligned dimensions. This method allows for smaller dimensions of vectors and smaller vocaublaries to be tested. The steps to using the code can be seen in appendix D.

3.3 Experiment

First I created 100 dimensionaly word vectors using `word2vec`. I chose 100 as that is the default choice in the implementation and could act as a baseline for comparisons. I then used singular value decomposition to reduce the vectors to 75, 50, 25, 10 dimensions. This was done using pyspark's linear algebra library then writing the word vector files that could then be tested directly using `qvec`. I compared the `qvec` semantic scores and semantic scores using the created word vector files.

4 Results

The results for the numerical experiment proved to be interesting and showed some of the limitations of my study. In the Carnegie Mellon paper the `fasttext`, a different continuous bag of word model, with 300 dimensional word vectors scored 40.3 in the sentiment task [5]. This value represents word vectors that performed very well at sentiment based tasks and shows that a score of between 20 to 40 represents normal scores. The following table shows the comparison of the `QVEC` scores:

QVEC Scores of Word Vectors

Model	Semantic Score	Syntactic Score
Word2Vec	0.242659	0.195610
75 dimensions	0.215773	0.176279
50 dimensions	0.181817	0.150707
25 dimensions	0.201144	0.153671
10 dimensions	0.262562	0.191581

We see that the word2vec model with 100 dimensional vectors does worse than the vectors reduced to 10 dimensions. This I see as a limitation of the testing method, with such few dimensions it is easier to find higher correlated dimensions among the hand coded data and therefore smaller dimensional vectors may appear better through chance rather than their usefulness in NLP tasks.

The word vectors I created only scored ≈ 0.2427 in the semantic task and ≈ 0.1763 in the syntactic task. This I believe is an artifact of the source material. The way in which computer science papers and titles are written do not match well with human language naturally and therefore it would make sense for the syntactic score to be lower. The semantic score tells us that the vectors are slightly more useful for semantic tasks, such as sentiment analysis, but as the data was created from DBLP text the low score most likely is from the limited vocabulary of the data and the vectors only being 100 dimensions as apposed to 300 in the model used to test `QVEC`.

This analysis also shows that dimensionality reduction does reduce the score which correlates highly with perfomance in sentiment based tasks. It also shows that they word vectors still retain important information as the scores do not dramatically drop when the dimensionality is reduced. Other methods such as PCA have been explored as possible ways to reduce dimensionality and a hybrid method could improve performance.

5 Conclusion

In this paper I explored the distributed `word2vec` model, dimensionality redution using SVD in spark, and their performance using the `QVEC` perfomance metric. I originally attempted to use Wikipedia as the source of my text but the training of the `word2vec` model proved to not be feasible given the restrictions I was working with I therefore trained my models using DBLP text. This gave me decent results but also showed the shortcomings of using `QVEC` scores.

This project showed that dimensionality reduction of word vectors can create vectors that are still usable in NLP tasks. It also explored the difficulties in using a distributed system with large sums of data when dealing with text and word vector data. These results line up with previous work done on dimensionality reduction of word vector spaces, though SVD is less sucessful than other methods proposed to accomplish this task [4]. With larger vocabularies a certain level of dimensionality reduction can still allow for rich word vectors as well as saved memory usage.

Word vectors on distributed systems have their own special considerations. Using a distributed system can lead to speedups when processing data, but certain algorithms do not take full advantage of distributed architecture such as `word2vec` [3]. Therefore when presented with pretrained models we would like to be able to save memory perfoming dimensionality reduction post vector creation. I have shown this is possible but further investingations are required to determine the best suitible method and the effectiveness on large vocabularies.

References

[1] Ian Gladwell, James G. Nagy, and Warren E. Ferguson Jr. *Introduction to Scientific Computing using MATLAB*. Lulu Publishing, 2011.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *arXiv:1310.4546 [cs, stat]*, October 2013. arXiv: 1310.4546.

[3] Erik Ordentlich, Lee Yang, Andy Feng, Peter Cnudde, Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, and Gavin Owens. Network-Efficient Distributed Word2vec Training System for Large Vocabularies. *arXiv:1606.08495 [cs]*, June 2016. arXiv: 1606.08495.

[4] Vikas Raunak. Simple and Effective Dimensionality Reduction for Word Embeddings. *arXiv:1708.03629 [cs]*, November 2017. arXiv: 1708.03629.

[5] Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Guillaume Lample, and Chris Dyer. Evaluation of Word Vector Representations by Subspace Alignment. In *Proc. of EMNLP*, 2015.

A Github and Code Documentation

WikiExtractor:
`https://github.com/attardi/wikiextractor`
QVEC:
`https://github.com/ytsvetko/qvec.git`
Spark Linear Algebra:
`https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html #pyspark.mllib.linalg`
Word2Vec:
`https://code.google.com/archive/p/word2vec/`
Spark Word2Vec:
`https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html #pyspark.mllib.feature.Word2Vec`

B Setting Up Cloud Server

In my terminal

```
$ gsutil mb gs://{BUCKET_NAME}/

$ gcloud beta dataproc clusters create {CLUSTER_NAME} --project {PROJECT_NAME} \
    --bucket {BUCKET_NAME} --region europe-west1\
    --image-version=preview \
    --optional-components=ANACONDA,JUPYTER \
    --enable-component-gateway \
    --initialization-actions \
    gs://goog-dataproc-initialization-actions-europe-west1/python/pip-install.sh \
    --metadata 'PIP_PACKAGES=sklearn nltk pandas numpy datalab'
```

In the dataproc VM to ensure NLTK is set up properly

```
$python
>>> import nltk
>>> nltk.download('all')
>>> exit()

$sudo cp -r nltk_data/ /home/
```

C Wikipedia Data Extraction

First run the following commands in the shell.

```
$ git clone https://github.com/attardi/wikiextractor.git

$ wget https://dumps.wikimedia.org/enwiki/latest/
enwiki-latest-pages-articles.xml.bz2

$ mv enwiki-latest-pages-articles.xml.bz2 data/

$ python wikiextractor/WikiExtractor.py -o data/wiki/
    --no-templates --processes 8 data/
    enwiki-latest-pages-articles.xml.bz2
```

This creates a folder structure containing the wikipedia dump. Then importing the `wikidata.py` file, we can run `tokenize_data()` with the following file path `"gs://{BUCKET_NAME}/data/*/*"` Giving an RDD that contains the tokenized text of each document in a row.

D Using QVEC

First we download the `QVEC` package using git.

```
$ git clone https://github.com/ytsvetko/qvec.git
```

Then I copied the vectors from the `gs://{BUCKET_NAME}/` and ran the tests using the following commands:

```
$ gsutil cp gs://{BUCKET_NAME}/vectors/*.txt .

$ qvec/qvec_cca.py --in_vectors  ${VECTOR_FILE_NAME} --in_oracle
qvec/oracles/semcor_noun_verb.supersenses.en

$ qvec/qvec_cca.py --in_vectors  ${VECTOR_FILE_NAME} --in_oracle
qvec/oracles/ptb.pos_tags
```

The results were then output in the command line.

E List of Code Files

`wikidata.py`
`projectCode.ipynb`