Sentiment Analysis with Neural Network Approaches

Wu Qingyue*
1155089156
1155089156@link.cuhk.edu.hk

Zhang Chengxing
1155089143
cxzhang@link.cuhk.edu.hk

Zhang Ning
1155084445
1155084445@link.cuhk.edu.hk

Zhang Yafei
1155095388
1155095388@link.cuhk.edu.hk

ABSTRACT

Online product reviews which deliver consumers' attitudes and favors with products they have bought are extraordinary valuable for both potential consumers to make informed purchase decisions and merchants to improve their products and services quality. One of the fundamental questions that arises in online review is to characterize, measure and infer the sentiment that each review conveys. In this project, we study sentiment analysis in Amazon reviews with the application of several deep learning or deep-learning-inspired approaches, such as word embedding methods and Recurrent Neural Networks (RNN). Experiments in our dataset show that RNN performs better compared with simple word embedding methods, but when taking local word order into consideration, word embedding methods can outperform RNN slightly in terms of accuracy. Overall, further researches and explorations are urgently needed both in RNN and word embedding approaches in domain of sentiment analysis.

Keywords

Sentiment analysis; Natural language processing, Word embedding; Recurrent neural networks

1. INTRODUCTION

In recent years, with' the rapid and unprecedented development of natural language processing, especially approaches in deep learning, and the increase in the availability of relevant data, many works have begun to emerge with the goal of understanding sentiment of human language. Previous studies have shown that product reviews have non-negligible influential impact to online shopping as consumers prefer to read online reviews when making purchase decisions [4, 5]. Online reviews mainly contain two crucial parts: *comments* and *ratings*, which are serving similar purpose. Comments are more about details of the product information and consumers' attitudes while ratings are the overall evaluation that consumers assign on products they have bought. As the complexity of dealing with online reviews has increased (e.g., spam reviews), it has become increasingly important to understand and infer how consumers organize their comments and ultimately result in reasonable overall ratings. For each review comment, there has a corresponding rating affiliated to it. With so many reviews, there must exist some underlying patterns behind comments and ratings, in other words, relationships can be built between review comments and ratings. For instance, expressions in comments like *'bad purchase'* or *'not worth it'* usually result in low ratings, while expressions like *'perfect product'* or *'really appreciate it'* often lead to high ratings. Therefore, sentiment analysis through review comments is a feasible and doable way. However, there are several challenges in studying sentiment analysis of reviews. Generally, meaning of words are diverse, even for a same word there can be different interpretations, and the meaning of phrase (i.e., combination of words) are thus difficult to detect. Furthermore, consumers'concerns about products are quite different and they usually weigh differently in meaning of words to deliver their attitudes. As shown in Figure 1, comments in upper and middle panel both contain only one word *'good'*, but the overall ratings they give are quite different, 3 stars and 5 stars, respectively. In addition, the length of reviews is even more variable, spanning from one word to tens or hundreds of words, which further enhances the difficulty of sentiment analysis based on online reviews. As shown in Figure 1, comments in bottom panel are quite longer than the others, in fact, the variability of review length is even severe in real datasets.

Recent years have witnessed a rapid development of Natural Language Processing (NLP) [2, 3, 11, 14, 16, 21], accompanied by the great success deep learning has achieved. Consequently, various deep learning or deep-learning-inspired methods have been proposed and applied in NLP, and achieved tremendous results. Among these approaches, word embeddings have attracted great attention in the past few years, where words or phrases from the vocabulary are mapped into vectors of low dimensional real vectors according to their co-occurrence relations. Word embeddings, although work in an unsupervised manner, have ben exceptionally successful in many NLP tasks, such as machine translation, syntactic parsing, sentiment analysis, and semantic change researches [7, 14, 19, 20].

Here in our study, we mainly focus on methods of distributed representation of words or sentences, such as *word2vec* [16], *doc2vec* [11], as well as recurrent neural networks [6, 8], and apply them in Amazon review sentiment analysis tasks.

*Authors listed in alphabetic order.

ENGG5108 Big Data Analytics, Semester A 2016-17, The Chinese University of Hong Kong, Shatin, NT, Hong Kong SAR, The People's Republic of China

ACM ISBN 123-4567-24-567/08/06.

DOI: 10.475/123_4

Figure 1: Examples of reviews on Amazon.com

Review comments in our data are of variable lengths, and ratings are categorically labeled from 1 star to 5 stars. For word embedding methods, we firstly achieve the distributed representation of review comments, and then conduct classification tasks with the guide of overall ratings that each comment affiliated. For Recurrent Neural Networks (RNN), we emphasize on Long Short-Term Memory (LSTM) architecture, and conduct sentiment analysis directly with each review comment as input.

2. RELATED WORK

Research on sentiment analysis has a long tradition and involves a variety of approaches, and methods they employed can be mainly divided as *count models* and *predict models* [1]. Traditional machine learning methods, including Naïve Bayes, maximum entropy, support vector machines and dictionary-based approach, are widely used in sentiment analysis before neural network is well accepted and most of them can be regarded as count models [9, 13, 15, 17, 18]. These count models are efficient in learning, but cannot capture comprehensive and latent relationships between words and phrases, which may lead to fail of understanding sentences in latent semantic domains.

For instance, Naïve Bayes method works in a straightforward way: For sentences with known labels or classes, we can achieve bag-of-words representation accordingly and feed them to a Naïve Bayes classifier to perform text classification. In addition, researches have realized that some words are commonly used to express opinions including good or bad, excellent or broken, therefore these specific words can be constructed as opinion word list and provide another way to learn sentiment efficiently. Dictionary-based and corpus-based approaches are two of the major approaches to compile the opinion word list. In dictionary-based approach, assume we have a small set of opinion words with known orientations, then by searching in well-known corpora for their synonyms and antonyms and adding the new words, the dictionary become bigger and bigger until no new words to be found [13]. Corpus-based approach is based on that opinion words can be context specific. Due to the idea that similar opinion words frequently appear together near each other in a body of text, two words appearing together frequently in the same context may have same polarity. So we can determine the polarity of an unknown word by finding the co-occurrence patterns using statistical approach [9, 13]. Traditional count models only care more about co-occurrence patterns of words but not their context. In other words, traditional natural language processing methods are competent to do tasks on manifest comprehension of language, but when it comes to latent level understanding they may fall to get the idea.

Recent years, especially in the past 5 years, word embedding methods have been widely used in sentiment analysis, owing to their impressive overall performance in NLP [3, 11, 14, 19]. Compared with count models, word embedding methods work in a way that semantically close words tend to have similar contextual distributions. In other words, word embedding methods take contextual information into consideration beyond co-occurrence patterns of words. Specifically, Maas et al. [14] have presented a mixed unsupervised and supervised model to learn word vectors with the goal of capturing both semantic term-document information and abundant sentiment content. The proposed model can make use of not only continuous and multi-dimensional sentiment information but also non-sentiment annotations as well, which is considered to be a breakthrough in sentiment analysis.

Socher et al. [19] bring in a Sentiment Treebank, which includes fine grained sentiment labels for more than 200,000 phrases in the parse trees of 11,855 sentences, i.e., words like good or great are assigned with a high sentiment label (e.g., 3 or 4), while words like bad or awfully are assigned with a low sentiment label (e.g., 0). When two words are combined together to form a binary tree, a new score is given according to their respective sentiment labels. This method works very well on short phrases, but when it comes to long phrases or sentences, the classification accuracy drops quickly. In addition, documents in datasets they have adopted are quite different in terms of lengths: every sample in Socher et al.'s dataset is a single sentence while every sample in Maas et al.'s dataset comprises several sentences. However, in reality, length of documents is variable, spanning from one word to several sentences, which brings in new challenges to sentiment analysis. To remedy this, several frameworks have been proposed. Inspired by vector representations of words using neural networks [2, 16, 21], Le et al. [11] construct paragraph-level vectors where paragraph vectors are asked to contribute to a prediction task about the next word given plenty of contexts sampled from the paragraph. Therefore, each paragraph can be represented by a unique real numerical vector. In the near past years, recurrent neural networks (RNN), especially for Long Short-Term Memory (LSTM), have shown their unreasonable effectiveness in sequence processing, which provide new inspiration and candidate approaches for sentiment analysis [6, 8]. LSTM, as a special kind of RNN, is explicitly designed to tackle the long-term dependency problem, i.e., it can remember information for a long period and thus has the ability to connect previous information to the present situation.

As much as datasets that have been adopted by previous studies are either polarizable (i.e., positive and negative only) or fine-grained (i.e., with pre-obtained sentiment

labels) [11, 14, 19]. Here in our study we will adopt a new dataset which comprises about 400, 000 online reviews, where each review is labeled with a star rating ranging from 1 to 5 but without fine-gained sentiment labels. What's more, length of each review comment is variable, in other words, we do not select reviews manually and try to make every review stays exactly as it was. Our work is inspired by the recent works on word embeddings and LSTM architecture, and we will explore these methods concentrating on sentiment classification tasks in the new dataset while employing neural network training strategies.

3. METHODS

Understanding the meaning of words or sentences is one of the core issues in natural language processing studies. Words are treated as discrete atomic symbols in traditional natural language processing systems, which makes it hard to unearth latent semantic relationships between words. For instance, in one-hot representation, *moon* and *star* are represented by distinct indices,

$$sun \qquad [0,0,0,0,0,0,0,0,1,0,0,0,0,...]$$

$$star \qquad [0,0,0,0,0,0,0,1,0,0,0,0,0,...]$$

which makes it hard to infer their direct relationships. Therefore, these encodings cannot provide enough useful information to the system regarding the relationships that may exist between the individual symbols. Traditional topic model methods, such as LSA (Latent Semantic Analysis) or LDA (Latent Dirichlet Allocation), are usually count-vector-based, and they care more about co-occurrence patterns of words but not their context. What's more, these traditional methods rely heavily on dimensionality reduction techniques, which may require more resources to handle on larger data. However, context-predicting models or neural language models stress more on contexts of words and thus can figure out semantic or syntactic relations deeper. Just as highlighted by Baroni's paper [1], *don't count, predict!*, context-predicting methods generally outperform than count models.

3.1 Distributed representation of words and sentences.

Among these context-predicting models, *word2vec* is an efficient embedding method which can provide state-of-the-art results on a lot of natural language tasks [12]. Furthermore, comparing *word2vec* with other neural-network-inspired word embedding models, *word2vec* is more robust and scales nicely. In other words, although *word2vec* might not be the best approach for every task, it does not significantly underperform in a lot of scenarios [12].

The *word2vec* model and application implemented by Tomas Mikolov and his colleagues [16] have attracted a great amount of attention since their release. *Word2vec* works in a way that is similar to deep learning approaches, but is computationally more efficient. It attempts to discover semantic relationships among words through word embeddings, a framework for vector representations of words. The vector representations of words learned by *word2vec* models have been shown to be efficient for learning high-quality vector representations of words from large amounts of unstructured text data, and proven to be useful in various NLP tasks.

Continuous bag-of-words model (CBOW) and Skip-gram model are two main techniques used in *word2vec* to build

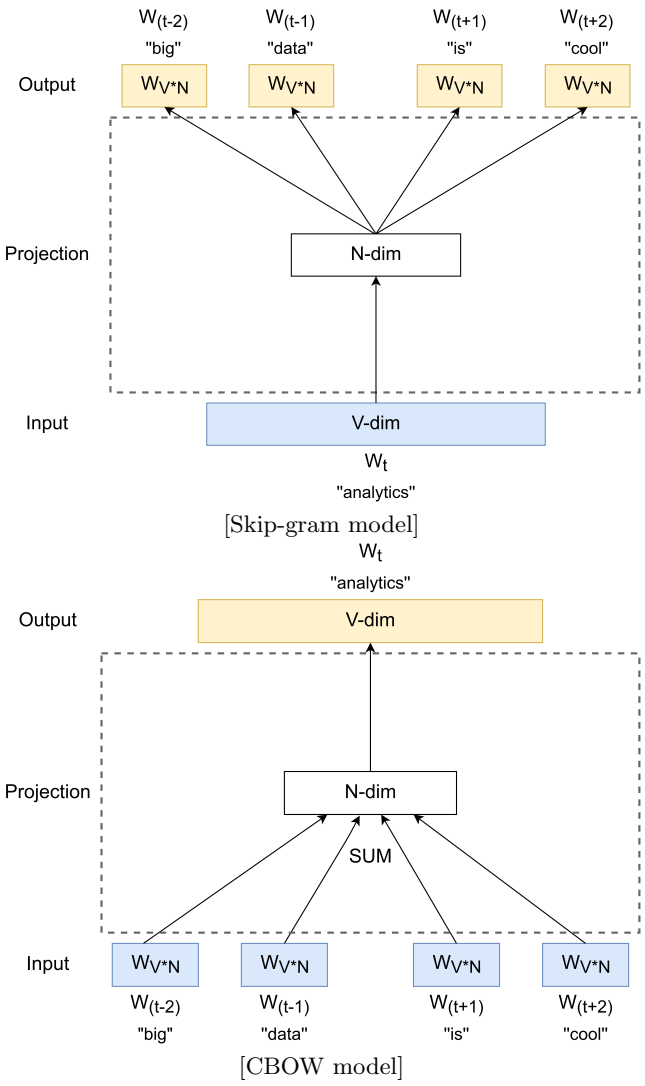

Figure 2: Main schemes of *word2vec*

a neural network that maps words to real-number vectors, with the expectation that words with more similar meanings will be mapped to more similar vectors.

CBOW: Assuming word inputs to the model could be w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}, and the output will be w_i, where the subscripts from $i-2$ to $i+2$ indicate the index of words in order. Hence we can consider the task as "*predicting the word given its context*". The main scheme of CBOW is shown in the bottom panel of Figure 2. In this scenario, for a given sentence, *'big data analytics is cool'*, surrounding words of word *'analytics'* are asked to predict *'analytics'*.

Skip-gram: While in this scenario, words input to the model is w_i, and the output could be w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}. So the task here is "*predicting the context given a word* ". Similarly, as shown in upper panel of Figure 2, *'analytics'* is asked to predict its neighbors. In addition, the window size of context is not limited to its immediate context, and training instances can be created by skipping a constant number of words corresponding to their contexts.

More generally, given word w and its contexts c, we can

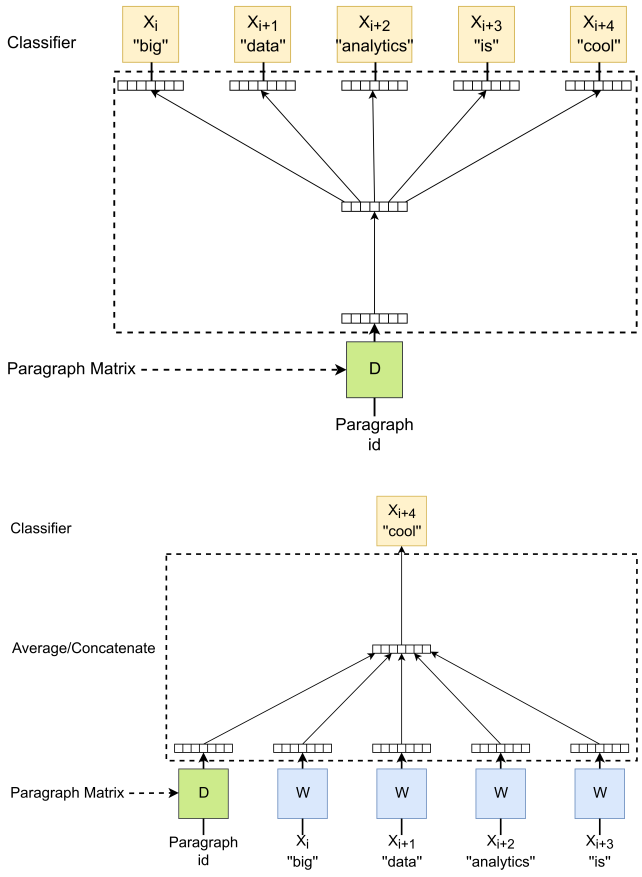

Figure 3: Framework for learning paragraph vector

consider conditional probabilities $p(c|w)$ based on a set of Ω which denotes all word and context pairs derived from the text. Then the objective of the Skip-gram model is to set the parameters θ of $p(c|w;\theta)$ so as to maximize the probability:

$$\arg\max_{\theta} \prod_{(w,c)\in\Omega} p(c|w;\theta) \tag{1}$$

Following a neural-network approach and softmax function, we can obtain:

$$\arg\max_{\theta} \sum_{(w,c)\in\Omega} \log p(c|w;\theta) = \sum_{(w,c)\in\Omega} (\log e^{v_c \cdot v_w} - \log \sum_{c'} e^{v_{c'} \cdot v_w}) \tag{2}$$

where v_c and v_w are vector representations for c and w respectively. Therefore, finding the best parameters θ, which aim to maximize objective function (2), will result in good embedding of words.

However, it's still computationally expensive to compute objective (2), therefore *word2vec* model also employs some other tricks, such negative sampling, to make the computation more tractable and efficient in real scenarios. Here we will not address about these methods any more due to space limitations (see Ref. [16] for more detail).

Inspired by *word2vec* architecture, Le et al. [11] designed a similar framework to construct paragraph-level vectors. In their architecture, each paragraph is mapped to a unique vector and contributes to a prediction task about the next word given plenty of contexts sampled from the paragraph. The context is fixed-length and randomly selected from a sliding window from the paragraph. One thing should keep in mind is that the paragraph vector is mutual across all contexts generated from the same paragraph but not across paragraphs, while the word vector is universal across all paragraphs. Figure 3 illustrates the main idea of paragraph vector learning, where paragraph vector of *'big data analytics is cool'* is asked to contribute to the prediction tasks.

Recently, Mikolov et al. proposed a new word embedding approach based on *word2vec* framework, which is not only much faster than most of deep learning classifiers but also maintaining competitive accuracy [3, 10]. In their model, named *fastText*, a bag of n-grams is added as additional feature to capture some partial information about local word sequence and thus each word can be represented as a bag of character n-grams instead of individual words. In our project, we will also adopt this approach and compared it with other methods.

3.2 Recurrent Neural Networks.

Recurrent Neural Networks (RNN) is one of the neural network structures we tried to improve the performance of the training process. The connections between RNN units form a directed cycle, and allow handling the correlation between the input information, providing data consistency. In traditional neural network models, the layers are started from input layers, then hidden layers and finally output layers [6, 8]. Neurons in a layer are unconnected. So, these models cannot do well in the scenes depending on the sequence of input signals, like most of the NLP problems. RNN will maintain the memory about previous information and apply it to the concurrent calculation which means that there are connections among neurons in hidden layers. In RNN, the last step information will affect this step's calculation. Consider a scene when we use an sigmoid function as the activation function to decide what to be updated and tanh to decide the value of the updated information:

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \tag{3}$$

$$h_t = o_t \cdot tanh(h_{t-1}) \tag{4}$$

First we accept the information from the input layer as the tradition neural network, And then we accept information from the last step. It decides what to be remained to the next step, and update the information carried each steps. A loop allows information to be passed from one step of the network to the next. For back propagation, we can just consider the loss of this step and next step, then combine them to find the residuals to get the derivatives. The LSTM (long short-term memory) architecture is one of the most famous RNN architectures. Hidden layers contain LSTM blocks to prevent information loss during a long time period of input, so that we can deal with the Long-Term Dependencies problems (Often occurs when the input is related to some information long time ago). A typical LSTM block structure is shown in Figure 4.

Cell states keep the information during the input period, with only some minor linear interactions. Information to just flow along layers unchanged, it gives us the ability that can keep track of the information with longterm dependencies. Gates have the ability to add or remove information to

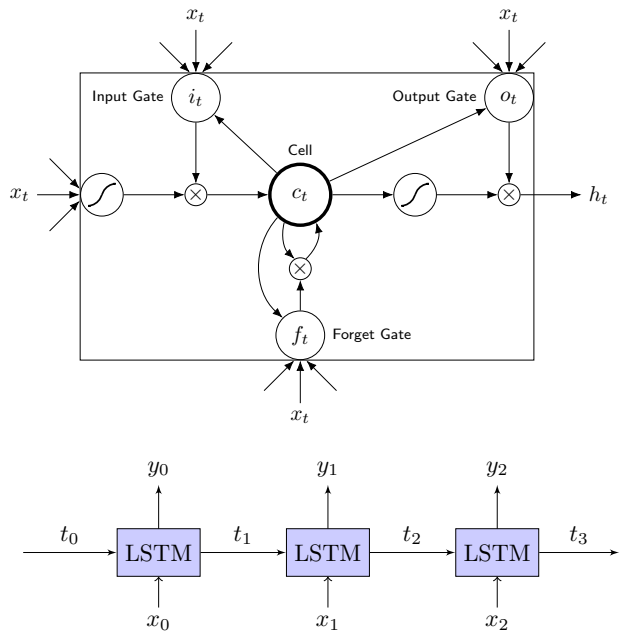

Figure 4: Digram of LSTM block structure

the cell states.There are three gates in the structure : Input Gate, Forget Gate, Output Gate. They can decide what information to be updated to the cell states and what to be remove when it comes to the next step. A sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. Zero represents nothing remained, and one represents we will keep everything. A tanh layer outputs numbers between minus one and one, describing the value of the input data that goes through the gate. So, First for the Forget Gate, it decide how much information (between 0 and 1) will be accepted and forgotten to the cell state. Typically, the older key words information tend to be forgotten when a newer key word arrives and then be updated to the new one , and if not, remain the key word information in the cell state. The information kept in the cell state is like this:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{5}$$

Then, for the Input Gate, we decide what new information we're going to store in the cell state.First, use a sigmoid layer to decide what to be updated and a tanh layer to produce values will be added to the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{6}$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{7}$$

And then we combine them to update our cell state. First, we remove the information we want to forget by multiplying previous cell states value and the information we want to keep. And then, we add the information of what we choose by multiplying input gate sigmoid value and candidate value, which means the part of information that can go through the gate.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{8}$$

Finally the Output Gate decides what we output for the next step's input. A sigmoid layer decides what part of the cell state we're going to output. Then, put the cell state through tanh and multiply it by the output of the sigmoid layer, to get what we choose to pass through the time line.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{9}$$

$$h_t = o_t * tanh(C_t) \tag{10}$$

We use a TensorFlow package to build the LSTM model. The model contains serval types of layers:

Embedding Layer: First, we construct a dictionary of 10000 words, and than break down the input reviews into an array that represented by the index of words in the dictionary. By doing this, we get the signals to convert to tensors used in the Embedding layer. In the Embedding Layer, we transfer the input into a vector that represent the review data.

LSTM layer: Contruct the LSTM, setting a dropout probability to apply the forget gate parameter and prevent overfitting. The activation function in this layer is tanh, and the inner activation layer is sigmoid.

Regression layer: In this layer, we use an Adam Optimizer for the optimize the weights and bias. It specify a TensorFlow gradient descent optimizer that will minimize the provided loss function(in our model , the cross entropy).

Softmax layer: It is a fully connected layer. Use a softmax function to map the input into the probability result of ratting 1-5.

4. EXPERIMENTAL RESULTS

4.1 Dataset

In this project, we adopt the Amazon product review dataset[1] as our source data. The raw data we used consists of 1.68 million records in category of *Electronics* only. The original data covers many aspects of reviews, such as ID of the reviewer, ID of the product, helpfulness rating of the review, text of the review and rating of the product that the reviewer gives. Here in our study, we mainly focus on data about review text and review rating. A sample of review likes this: {*review text: "I bought this for my husband who plays the piano. He is having a wonderful time playing these old hymns. The music is at times hard to read because we think the book was published for singing from more than playing from. Great purchase though!"; overall rating: 5.0*}.

However, due to the imbalance of the date set, we randomly select 80,000 (number of records in each rating) * 5 (number of unique ratings) non-empty reviews from raw data (i.e., 80 000 review data for ratings with 1 star, 80 000 review data for ratings with 2 stars, and so forth). Taken together, we have 400 000 randomly selected reviews in total with variable review length. The distribution of review length is shown in Figure 5. Owing to few reviews have unusual long texts, so we we transfer the review length to *log* form to display the distribution. Afterwards, we will process

[1]http://jmcauley.ucsd.edu/data/amazon.

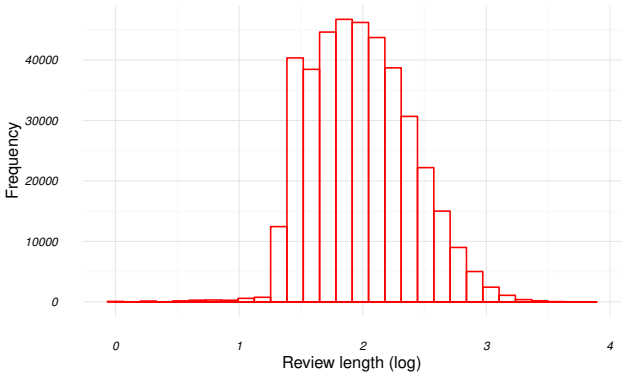

Figure 5: Distribution of review length.

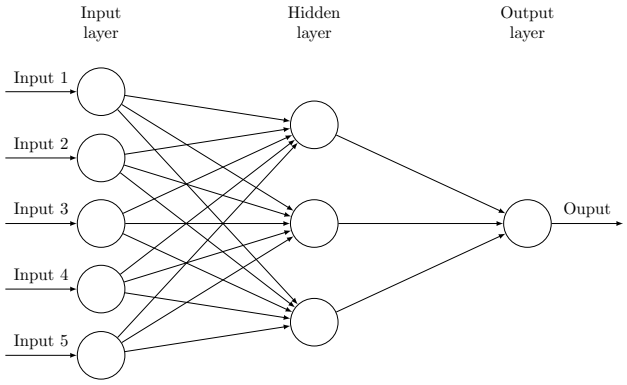

Figure 6: A typical structure of neural networks.

data accordingly, train them in the proposed model and then predict the overall rating based on review text only. To get the train and validation data separately, we randomly split the data into train/validation sets with a ratio of 8/2, i.e., 320 000 reviews are split into train set while 80 000 reviews are split into validation set.

4.2 Distributed representation approach

As we have announced above, distributed representation or word embedding method have the explicit advantage of capturing latent semantic relations, so firstly we use word or sentence features extracted from *word2vec* and *doc2vec* and then apply them to sentiment classification tasks through a simple neural network. A typical structure of neural networks we adopted here is shown in Figure 6.

4.2.1 *Word vector averaging*

Firstly, with the aid of *word2vec*, a given word is represented by a vector with a reasonable dimension (tens to hundreds, according to the specification assigned to it). In our case, we adopt 320 000 review texts and train them to get vectors of words with a dimension of 300. For example, given a word '*sun*', it can be represented by a 300 dimensional vector, say $v(\text{`}sun\text{'}) = [v_1, v_2, \ldots, v_{300}]$. Then, for a given sentence we can get the corresponding vector easily by averaging the word vectors in the sentence.

Note that, to get the average vector of sentence, we neglect stopwords (derived from NLTK[2].) as many of them don't convey rich information and may even bring in irrelevant influences. We should also notice that for words come form validation set, there may no corresponding existing word vectors because they are out of the vocabulary of train set, so in this scenario we neglect new words as well when computing sentence or paragraph-level vectors. After we get vectors corresponding to each review, we feed them to neural networks with one hidden layer to do classification tasks with the aid of *Tensorflow*, and the primary result of 5-way classification is shown in Table 1. Beyond that, we also conduct some additional experiment between different combination of classes, part of results are show in Appendix.

Table 1: Results of sentiment classification tasks.

Method	Accuracy for 5-way classification
Word vector averaging	47.3%
Paragraph vector	30.8 %
fastText	55.7 %
LSTM	54.3 %

4.2.2 *Paragraph vector*

Paragraph vector or *doc2vec* works in a way similar with *word2vec*, but the superior is that we can get the paragraph-level vector directly, which will liberate labors greatly. In the same way, we conduct sentiment classification tasks by training review vectors through a typical neural network with one layer. Result is reported in Table 1, obviously, we don't get a higher accuracy as expected. The reason may be that paragraph vector learned in Le et al. [11] contains two types of vectors: one learned by the standard paragraph vector with distributed memory (PV-DM) and the other learned by the paragraph vector with distributed bag-of-words (PV-DBOW), but in our study we only take PV-DM into consideration which may lose many rich information. In addition, they have made use of unlabeled data to train the model ,which may help a lot to distinguish boundaries between different classes in classification task, while we don't leverage any other data to train our model.

4.2.3 *FastText*

FastText works in a way similar with *word2vec* while taking local word order into consideration [3, 10]. Result in Table 1 shows that it can really achieve good performance compared with the other two word embedding methods. Moreover, as *fastText* is implemented in C++, as such it works fast indeed just as its name implies: in our task, only 1 minute is needed for the main function to get an accuracy of about 55% in five-way classification task.

4.3 Recurrent neural network approach

In our project, we mainly conduct experiment on LSTM (long short-term memory) architecture. Firstly, words are processed and mapped into a unique and distinct id, however, troublesome problem occurs consequently, as there are many typos in the review data resulting in a high dimensional word space. To remedy this, we only chose top 10 000 words by their term frequency in the data. Note that, 10 000

[2]http://www.nltk.org

words in english corpus are enough to help us capture the word information while training more efficient as it reduces possible word space.

Classification result in Table 1 shows that LSTM achieves significant improvement compared with word vector averaging or paragraph vector approaches but performs slightly poorly when compared with *FastText*. Generally, Recurrent Neural Networks are difficult to train, especially for people without professional knowledge of RNN, this may be the main reason in this scenario. Furthermore, due to limitation we just train RNN for several Iterative steps, which may be not enough to get a striking result.

5. DISCUSSION AND CONCLUSIONS

In our project, we investigate sentiment analysis through several approaches, including word vector averaging, paragraph vector, *fasetText* and LSTM. Different from previous studies, where datasets used are fine-gained or only contain two classes (i.e., positive and negative), dataset adopted in our study is randomly selected from raw review data and of variable text length. Experiments show that RNN/LSTM performs better than simple word embedding methods like *word2vec*, *doc2vec*, but when taking local word order into consideration, just as *fasetText* states, word embedding method can achieve competitive results compared with RNN/LSTM.

In addition, since PCA could decompose matrix into certain length while keeping the major information of the matrix, we also conduct a matrix factorization which is just the exact representation of quantified words, we put the vector of words as columns sequentially. And then we did the PCA decomposition with 4 components left, however we don't achieve expected results. This direction still needs to explore.

Taken together, through this project we recognize the amazing power of word embedding and neural networks. We still need to learn more in Big Data Analytics in this era of Big Data.

6. REFERENCES

[1] M. Baroni, G. Dinu, and G. Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL (1)*, pages 238–247, 2014.

[2] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *journal of machine learning research*, 3(Feb):1137–1155, 2003.

[3] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[4] J. A. Chevalier and D. Mayzlin. The effect of word of mouth on sales: Online book reviews. *Journal of marketing research*, 43(3):345–354, 2006.

[5] W. Duan, B. Gu, and A. B. Whinston. The dynamics of online word-of-mouth and product salesâĂŤan empirical investigation of the movie industry. *Journal of retailing*, 84(2):233–242, 2008.

[6] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[7] W. L. Hamilton, J. Leskovec, and D. Jurafsky. Diachronic word embeddings reveal statistical laws of semantic change. *arXiv preprint arXiv:1605.09096*, 2016.

[8] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

[10] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[11] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. In *ICML*, volume 14, pages 1188–1196, 2014.

[12] O. Levy, Y. Goldberg, and I. Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[13] B. Liu. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167, 2012.

[14] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

[15] W. Medhat, A. Hassan, and H. Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093–1113, 2014.

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[17] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

[18] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[19] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

[20] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[21] J. Turian, L. Ratinov, and Y. Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for

APPENDIX

Table 2: Sentiment classification results of word vector averaging.

Classification (different classes separated by ',')	Accuracy
123, 45	80.0%
12, 345	79.9%
12, 3, 45	68.2%
1, 2	67.6%
1, 2, 3	55.9%
1, 2, 3, 4, 5	47.3%
1, 5	90.0%
3, 4, 5	57.3%
4, 5	69.1%

Table 3: Sentiment classification results of paragraph vector.

Classification (different classes separated by ',')	Accuracy
123, 45	64.6%
12, 345	68.5%
1, 2	61.0%
1, 2, 3	45.8%
1, 2, 3, 4, 5	30.8%
1, 5	72.4%
3, 4, 5	40.0%
4, 5	52.9%