Parallel Word Embedding

Zhiping Fu, Zhiping_Fu@student.uml.edu
Weiwei Li, Weiwei_Li@student.uml.edu
Yufeng Yuan, Yufeng_Yuan@student.uml.edu

Introduction

Deep Learning, now is a very popular research topic in AI. Its original prototype came from Multilayer Perceptron Model (MPM) introduced in the 1950s. Because of the high demand of computing resource, the MPM didn't develop well in the following half of century until 2006. In [1], Hinton et al. invented a new training method (pre-train the parameters of each layer using Restricted Boltzmann Machines (RBM)), which could largely shorten the convergence time and get a good result comparing to the past training method. At the same time, they gave a new name, called Deep Learning. From then on, Deep Learning enters into Spring.

NLP, a big branch of AI, has obtained lots of amazing results under Deep Learning. Usually, the first step to apply Deep Learning to NLP tasks is word embedding, which is to covert each word to a vector. Compare to the traditional Bag-Of-Word model, which ignores the word sequence, word embedding has plenty of advantages: lower dimensions (hundred level vs. thousand or 10 thousand level of Bag-Of-Word), context related (keep word sequence), easy to compare similarity and so on. So, an efficient Word Embedding model is very critical to NLP tasks, and We're going to do something about Word Embedding.

Motivation

Now, we're in the information age of explosion, and the data generated everyday now is larger than the total data count of the last two thousand years. As a result, to train a Word Embedding model from such big data with good precision and within a reasonable time becomes more and more difficult. All of these call for a well-designed Deep Learning network and a high concurrency system. Our goal is to explore the data parallelism and model parallelism of the current Word Embedding algorithms, and try to develop an efficient system to train the Word Embedding model.

Related Work

Word or document vectorization could trace back to Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA). But both of them are computationally expensive when dealing with large corpus. Additionally, distributed representations of words based on neural networks show better performance, and they are the most popular Word Embedding model.

The prototype of neural network based Word Embedding was first introduced by Bengio et al. in 2003 [2], called Feedforward Neural Net Language Model (NNLM). As in Figure 1, the model tries to learn the current word probability based on the former context, and it turned the unsupervised problem to supervised problem.

Then, in 2013, the new Log-linear Models were proposed by Mikolov et al. [3]. They were Continuous Bag-of-Words Model (CBWM) and Continuous Skip-gram Model (Skip-gram), as in

Figure 2. Overall, they were the same as NNLM, but with the hidden layer removed. Moreover, the most famous Word Embedding tool now, called Word2vec, are developed from these Log-linear Models.

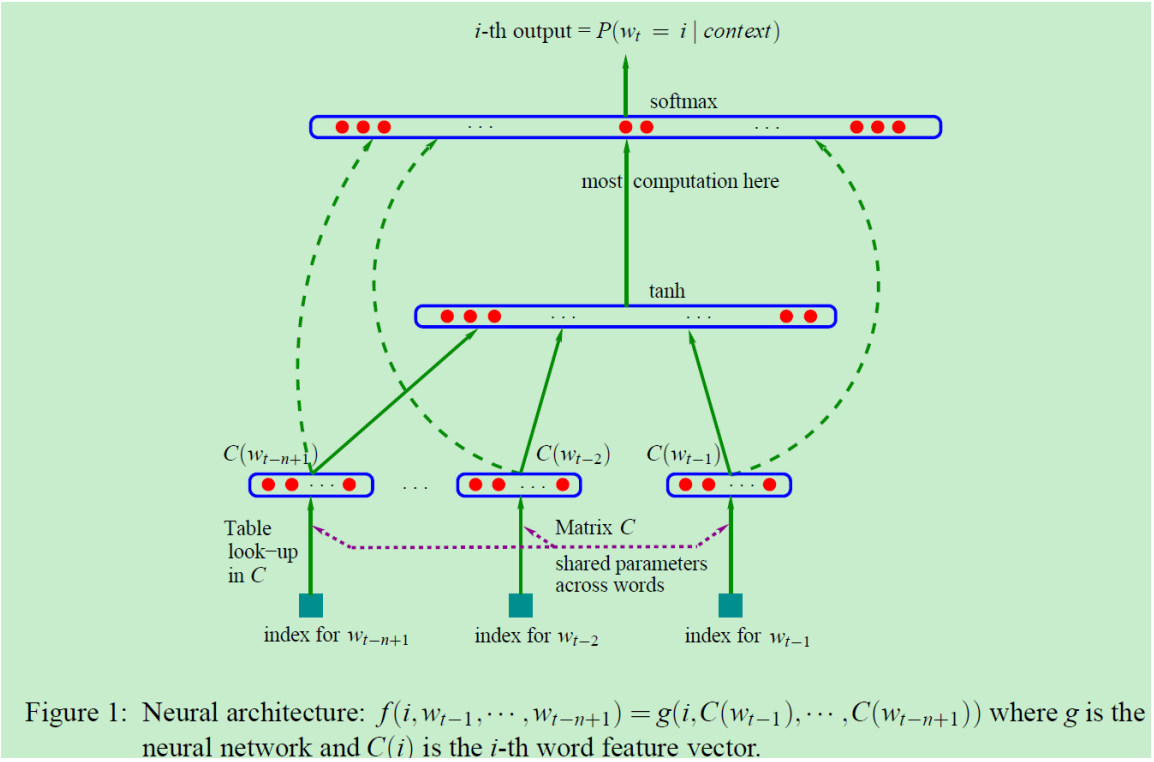

Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

Figure 1 Feedforward Neural Net Language Model (NNLM) [2]

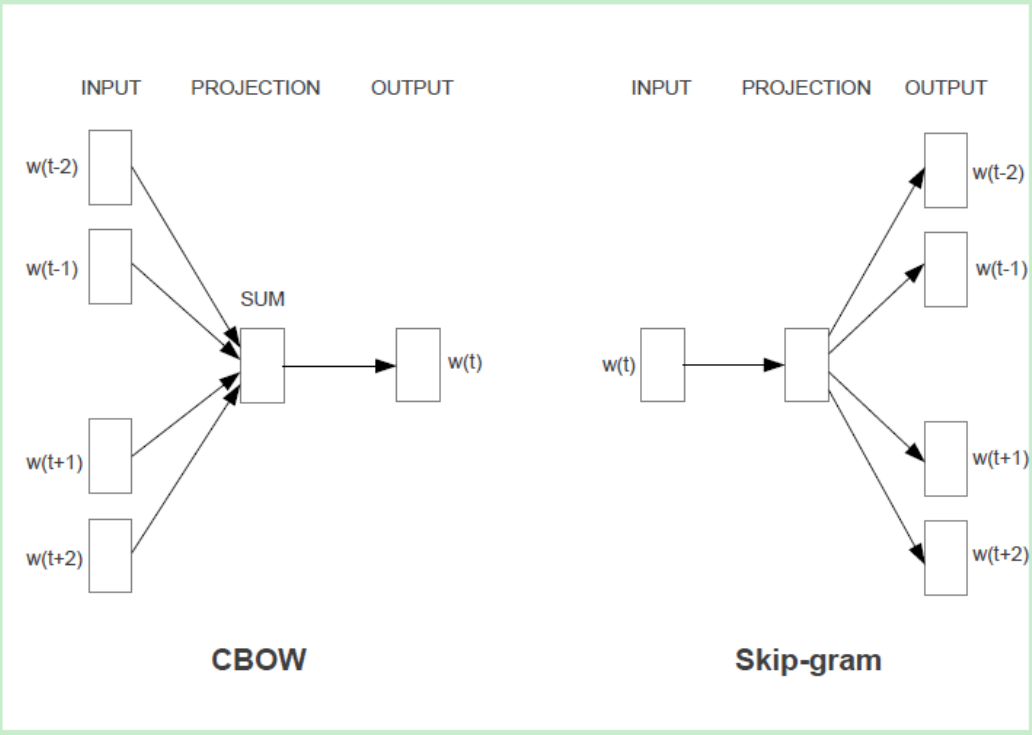

Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Figure 2 Log-linear Model [3]

Usually, the output layer of the neural network based Word Embedding Models contains the whole vocabulary table, which is ten thousand level. As a result, this leads to the low efficiency of training Word Embedding Models. In [4], Mikolov et al. summed up three methods to conquer this problem.

a. Hierarchical Softmax. The idea is encoding the whole vocabulary with tree, then the size of output layer could reduce to logarithm.
b. Negative Sampling. This is also called Noise Contrastive Estimation (NCE). It introduces some noisy words (less than 20, denoted as k) into the output layer. So, the size of output layer is 1 (target word) + k (noisy words), which dramatically reduces the size of output layer. Basic idea under NCE is that the classifier could easily distinguish the target from the noise additive sample.
c. Subsampling of Frequent Words. Just as it says, it depresses the frequent words, like "a", "the", to reduce computing.

To train a large scale neural network, Parameter Server is a popular architecture [5]. However, for Word Embedding training network (CBOW or Skip-gram), the communication load is terribly serious, according to [6]. So, Ordentlich at el. introduced a new structure to train the word vectors, which includes data and model parallelism. Their proposed algorithms give us a great inspiration to implement our own parallel Word Embedding system.

Proposed Research

We are going to develop a high parallel Word Embedding training system, which would include CBOW and Skip-gram models. Moreover, our training system would explore both the data parallelism and model parallelism, and our system includes the following features.

a. Suitable for big corpus, TB level.
b. High parallelism, include data and model level parallelism.
c. Reasonable training time.

Our proposed system would be using Java and Python programming language, and build on Hadoop, Spark and Google Tensorflow.

The training data set would be a movie review corpus [7], which contains 100-000 movie reviews, and the size of vocabulary table reaches 89527.

Evaluation

The word vectors contain interesting information, like Figure 3. The vector offset $(V_{kings} - V_{king})$ is parallel to $(V_{queens} - V_{queen})$, which means that the cos similarity between these two offsets is near one (almost the same). Base on this characteristic, Mikolov at el. came up with a vector offset method to test the quality of word vectors [8].

The test analogy question is like a:b vs. c:d, then we want to calculate the cos similarity:
$\cos(V_b - V_a + V_c, V_d)$, which should be near to one.

The test set contains 19558 questions [9], which includes 14 categories, such as capital-common-countries, capital-world, currency, city-in-state, family, gram1-adjective-to-adverb, gram2-opposite, gram3-comparative, gram4-superlative, gram5-present-participle, gram6-nationality-adjective, gram7-past-tense, gram8-plural and gram9-plural-verbs.

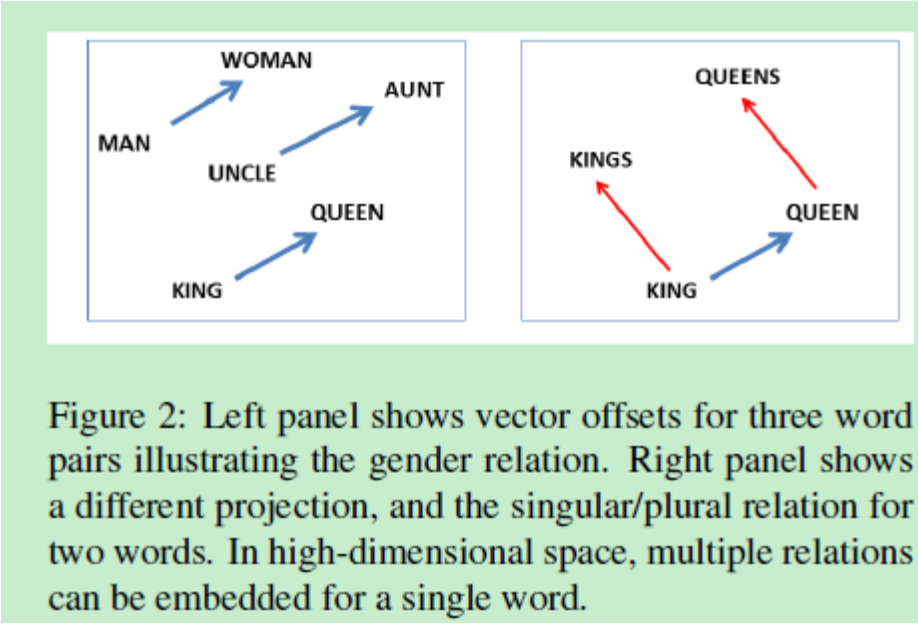

Figure 2: Left panel shows vector offsets for three word pairs illustrating the gender relation. Right panel shows a different projection, and the singular/plural relation for two words. In high-dimensional space, multiple relations can be embedded for a single word.

Figure 3 Word vectors semantic offset [8]

Timeline

The schedule is as following.

date	content	note
6th Feb ~ 12th Feb	Word Embedding literature review	
13th Feb ~ 19th Feb	Word Embedding literature review	Deep into the algorithms
20th Feb ~ 26th Feb	Hadoop and Spark	
27th Feb ~ 5th Mar	Tensorflow and initial System architecture	Confirm the system architecture
6th Mar ~ 12th Mar	Setup developing environment	
13th Mar ~ 19th Mar		Spring Break
20th Mar ~ 26th Mar	Detail implementation	
27th Mar ~ 2nd Apr	Debug and tuning	
3rd Apr ~ 9th Apr	Document	
10th Apr ~ 16th Apr	Prepare for presentation	

References

[1] Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, *18*(7), 1527-1554.

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

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

[4] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).

[5] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., ... & Ng, A. Y. (2012). Large scale distributed deep networks. In *Advances in neural information processing systems* (pp. 1223-1231).

[6] Ordentlich, E., Yang, L., Feng, A., Cnudde, P., Grbovic, M., Djuric, N., ... & Owens, G. (2016, October). Network-Efficient Distributed Word2vec Training System for Large Vocabularies. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (pp. 1139-1148). ACM.

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

[8] Mikolov, T., Yih, W. T., & Zweig, G. (2013, June). Linguistic Regularities in Continuous Space Word Representations. In *Hlt-naacl* (Vol. 13, pp. 746-751).

[9] http://download.tensorflow.org/data/questions-words.txt