Parallel word embedding based on Distributed Tensorflow

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

1. Idea and motivation

Firstly, we're going to explore the network bandwidth requirement for word embedding training system based on the traditional PS (parameters server) architecture. The following estimation is based on the skip-gram training algorithms.

d: vector dimension, 100 ~ 500;

w: context window size, 5 ~ 10;

n: the number of random negative examples per context word, 5 ~ 10;

For the storage of word vectors and weights, we use single precision floating point - that is 4 bytes per parameter. Then, we have the network traffic estimation (to get and put from and to the parameter server) for one training word.

$r(w, n, d) = (2 + w * n) * d * 4$, 2 means the input and output word.

For example, when w=10, n = 10, d=500, which are recommended in [2], each training word requires r(10, 10, 500) = 200,000 bytes transferred with each get and put. In [3], the authors gave a practical number of the bandwidth requirement - for a dataset with 50 billion words and one week training latency, the system required at least 1300Gbits/sec bandwidth. This would bring big burden for local network.

To solve the network burden problem, authors in [3] proposed a new system architecture, like Figure 1 Scalable word2vec on Hadoop[3]. This is a model parallelism method, and has the following features.

- Column-wise partitioning across word vectors and training weights;
- The computation done in the PS servers not in worker side;
- Network traffic is word index, not the word vectors.

The loss function for skip-gram training algorithm is as follow [2].

$$\log \sigma({v'_{w_O}}^{\top} v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^{\top} v_{w_I}) \right]$$

So, the training procedure is as follow.

Forward propagation:

- Worker sends word indices and seeds to every PS shard;
- Every PS shard does dot production, then sends back to the worker which sends the indices;
- Worker gather the part dot result, does sigmoid function, then sends

back to all PS shard;

Backward gradient propagation:

- The PS shard received the adjusting factor from worker, then calculates the gradient and adjust the word vectors and weights.

The proposed word embedding training system in [3] is based on the following tools.

- Programing Language: Java and Scala.
- Hadoop YARN: worker management.
- Slider: launch worker process in YARN.
- Spark: vocabulary generation(map-reduce).
- Hadoop Distributed file System (HDFS): dataset storage.
- Netty client-server library: RPC layer, worker and PS shard communication.

From the above, the traffic load for the proposed system is $r(w, n, d) = (2 + w * n) * 4$, reduced by a factor d (hundred level), and the improvement is significant.

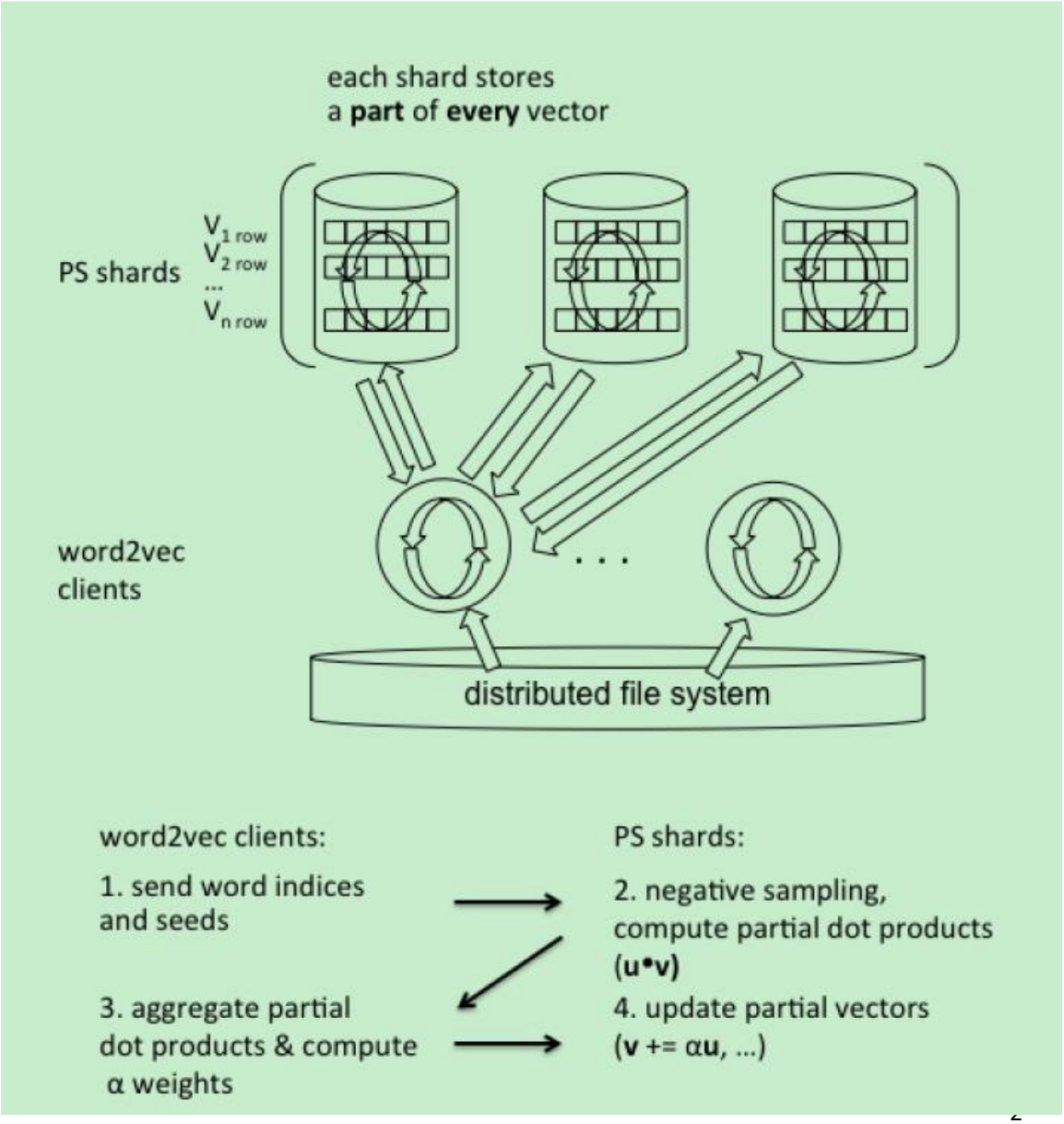

Figure 1 Scalable word2vec on Hadoop[3]

2. Distributed Tensorflow

Tensorflow has an elegant mechanism for distributed computing. The RPC service in Tensorflow makes the distributed computing friendly to users. The users just want to partition the computing graph and designate to the specified device, then RPC service in Tensorflow would deal with all the communications between devices or computing nodes [4].

Based on the distributed Tensorflow, we implemented the idea in [3], and developed our own word embedding training system. Our training system implements both skip-gram and cbow model, and supports large scale dataset.

3. Proposed System architecture

Our system is as Figure 2 architecture of our word embedding training system. The "Slice n" has the same function as "PS shard" in Figure 1 Scalable word2vec on Hadoop[3]. In our system, the "Slice n" is the sub computing graph; it stores the column-wise word vectors and weights, and does the dot production for forward propagation and adjustment for backward propagation.

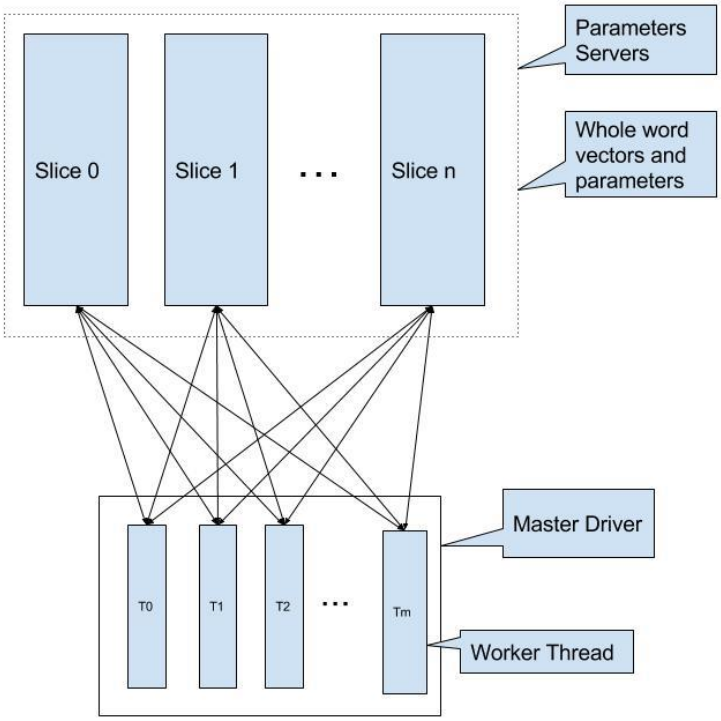

Figure 2 architecture of our word embedding training system

4. Implementation

Our system supports the following features.

- CBOW and Skip-gram model;
- Dynamic context window;
- Negative Sampling to replace full softmax function;

- Subsampling of Frequent Words;

Firstly, we should prepare the vocabulary table (one line one word—word: frequency).

Our system contains two part: server (Slice) and Master Driver. The server is simple (refer to the “how to run” section).

For the “Master Driver”, the procedure is as follow.

- Load vocabulary
- Setup sub computing graph and sub Tensorflow session
- Setup worker thread
- Setup checker thread to do evaluation of current word vectors
- Wait until all workers are done, and assemble the word vectors

For worker thread, every worker is assigned a work directory, and the worker works as follow.

- Scan work directory to get the whole training files list
- Tokenize one training file, map to index according to the vocabulary table
- Generate training word pair and subsampling index, and send them to all Slice sub graphs
- Gather partial dot production result from all Slices, do sigmoid, and send back to all Slice sub graphs

The input parameters are as follow.

```
usage: Large_Scale_Word2Vector.py [-h] [--ps_hosts PS_HOSTS]
                                  [--job_name JOB_NAME]
                                  [--task_index TASK_INDEX]
                                  [--win_size WIN_SIZE] [--model MODEL]
                                  [--neg_samples NEG_SAMPLES] [--vsize VSIZE]
                                  [--iteration ITERATION] [--alpha ALPHA]
                                  [--sample SAMPLE] [--vocab VOCAB]
                                  [--vfreqT VFREQT] [--output OUTPUT]
                                  [--work_folder WORK_FOLDER]
                                  [--question_file QUESTION_FILE]
                                  [--embeding_quality EMBEDING_QUALITY]
                                  [--debug DEBUG]

optional arguments:
  -h, --help                    show this help message and exit
  --ps_hosts PS_HOSTS      Comma-separated list of hostname:port pairs
  --job_name JOB_NAME      One of 'ps', 'master'
  --task_index TASK_INDEX      Index of task within the job
  --win_size WIN_SIZE      the size of training window
  --model MODEL                 'skip-gram'(default) or 'cbow'
  --neg_samples NEG_SAMPLES the negtive sample count for negtive sampling
  --vsize VSIZE                 the dimension size of word vector, default is 100
  --iteration ITERATION        the iteration count, default is 5
```

```
  --alpha ALPHA                the start learning rate
  --sample SAMPLE              word sample rate, to recude frequent words
  --vocab VOCAB                the path to the vocabulary table
  --vfreqT VFREQT              the word frequency threshold to generate vocabulary
  --output OUTPUT              the output word vector file name, default is
                               'wordVector.dat'
  --work_folder WORK_FOLDER
                               Comma-separated list of work folder paths, which
store the text file
  --question_file QUESTION_FILE
                               questions, words file for checker
  --embeding_quality EMBEDING_QUALITY
                               average cos similality in question_file
  --debug DEBUG                debug level, default is 0
```

How to run?

Firstly, setup the Slice servers.

```
python Large_Scale_Word2Vector.py --ps_hosts=localhost:2222,localhost:2223 --job_name=ps --task_index=0
```

Note:

The “ps_hosts” parameter specifies the number of Slice servers, and you should login that machine and run the above command. For example, the above command has two Slice servers (localhost:2222 and localhost:2223, of course you could designate different machine using “ip_address:port” format), so you should run the command as follow.

```
python Large_Scale_Word2Vector.py --ps_hosts=localhost:2222,localhost:2223 --job_name=ps --task_index=0
python Large_Scale_Word2Vector.py --ps_hosts=localhost:2222,localhost:2223 --job_name=ps --task_index=1
```

Secondly, setup Master driver.

```
python Large_Scale_Word2Vector.py --ps_hosts=localhost:2222,localhost:2223 --job_name=master --work_folder=.
```

Note:

The “ps_hosts” should be the same as setup Slice servers, else the Master driver could not find the Slice servers.

The ”work_folder” is comma separated parameter, and the Master driver would create one worker thread for every work directory.

5. Result and evaluation

We used movie review dataset from [5]. The vocabulary reaches 60656 with word frequency large than 5, and the tokenized words reach 26,031,113.

The training parameters are as follow.

Start learning rate: 0.025

Negative samples: 10
High frequency sampling rate: 0.001
Word vector dimension: 100
Context window size: 5
What's more, we have 4 worker thread. For convenience, we only use one machine (AWS computing optimized VM, 4-kenel, 16G memory), with 2 Slice Server. Every day, it could finish 10 iterations.

The test questions are from [6], and the result is in Table 1 question-answer result – the final row is percentage, others are correct answers, while the numbers in first row (-15, -45) are the iteration count.

	Skg-15	Skg-45	Cbow-15	Cbow-45
Family(462)	160	169	46	120
Currency(40)	0	0	0	0
City-in-state(1500)	74	52	0	6
Capital-common-countries(420)	180	98	2	25
Capital-world(815)	249	162	1	15
Gram1-adjective-to-adverb(992)	14	69	0	6
Gram2-opposite(756)	4	4	0	2
Gram3-comparative(1332)	50	59	1	17
Gram4-superlative(1056)	27	33	2	13
Gram5-present-participle(930)	145	214	11	88
Gram6-nationality-adjective(1371)	901	924	38	217
Gram7-past-tense(1482)	127	263	30	103
Gram8-plural(1190)	370	370	24	143
Gram9-plural-verbs(812)	119	131	7	42
total	0.18	0.19	0.01	0.06

Table 1 question-answer result

The test questions from [6] is for common dataset, so the test result is not well in the movie review dataset. While the following table (Table 2 Synonyms), we pick up some words and try to find the similar words based on the trained word vectors. From the table, the skip-gram model works well than cbow model, and captures more semantics than cbow.

	Skg-15	Skg-45	Cbow-15	Cbow-45
good	'really', 'better', 'but', 'though', '.'	'really', 'better', 'well', 'but', 'very'	'one', 'but', 'is', 'really', ','	'better', 'much', 'done', 'well', 'was'
bad	'terrible', 'awful', 'ok', 'worst', 'least'	'terrible', 'awful', 'worst', 'worse', 'seriously'	'...', 'really', 'just', 'could', 'like'	'awful', 'worse', 'worst', 'terrible', 'seriously'

excellent	'great', 'superb', 'terrific', 'performances', 'outstanding'	'superb', 'outstanding', 'great', 'terrific', 'fine'	'wonderful', 'performances', 'cast', 'always', 'brilliant'	'brilliant', 'superb', 'outstanding', 'fantastic', 'performances'
man	'his', 'he', 'who', 'a', 'as'	'he', 'his', 'who', 'himself', 'him'	'he', 'who', 'him', 'men', 'girl'	'himself', 'his', 'who', 'him', 'sees'
woman	'her', 'she', 'herself','husband', 'mother'	'her', 'she', 'herself', 'girl', 'husband'	'she', 'who', 'mother', 'he', 'wife'	'himself', 'his', 'who', 'him', 'sees'
teacher	'school', 'teachers', 'students', 'student', 'teenager'	'school', 'students', 'teachers', 'teaching', 'student'	'brother', 'eventually', 'parents', 'loves', 'meet'	'parents', 'mother', 'student', 'loves', 'daughters'
apple	'break', 'broken', 'broke', 'along', 'pie'	'bottle', 'drinking', 'wine', 'drink', 'drunk'	'writer/director', 'bust', 'keeps', 'incompetent', 'alive'	'discontinuity', 'floor', 'sits', 'table', 'band'
red	'wearing', 'blue', 'around', 'hair', 'a'	'blue', 'wearing', 'hat', 'wear', 'black'	'guy', 'walk', 'gets', 'front', 'enter'	'blue', 'wearing', 'shadow', 'cigarette', 'torch'

Table 2 Synonyms

6. Future work

Some improvements are added to the to-do list.

- Auto deployment tool to simplify to the system setup process;
- Combine with Spark and HDFS to support larger dataset and enhance data parallelism.

References

[1] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
[2] 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).
[3] 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.
[4] https://www.tensorflow.org/deploy/distributed
[5] 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.
[6] http://download.tensorflow.org/data/questions-words.txt