# Parallelizing Word2Vec in Shared and Distributed Memory

Team: April Cai, Kirti Bhandari

January 11, 2018

**Abstract**

Word2Vec is a group of models used to obtain low-dimensional vector representations of words. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space. Hence these representations are quite useful in various Natural Language Processing (NLP) applications like similarity detection, analogies etc.Previous multi-core implementations of word2vec are based on vector-vector operations that are memory-bandwidth intensive and do not efficiently use computational resources. In our reference paper, we scale up the word2vec performance nearly linearly by using minibatching, which allows us to express the problem using matrix multiply operations.

## 1 Introduction

Word2vec [8] is a group of related models used to produce vector representations of words in a continuous space. Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus. For word representations, similar words tend to be close to each other and words can have multiple degrees of similarity. The purpose and usefulness of Word2vec is to group the vectors of similar words together in vector space. Word2vec's applications extend beyond parsing sentences in the wild. It can be applied just as well to genes, code, likes, playlists, social media graphs and other verbal or symbolic series in which patterns may be discerned.

The remainder of this report is organized as follows. In section 2 we discuss the word2vec algorithm. In section 3, we highlight the limitations of original algorithm and the approaches [1] the improve the performance in multi-threaded and multi-core systems. Lastly, we summarize our findings and provide further directions to the work.

## 2 Word2Vec

### 2.1 Word Vectors

A Word vector is a vector of weights. In the 1-of-N(or 'one-hot') encoding, the encoding of a given word is simply the vector in which the corresponding element is set to one, and all other elements are zero. Using such encoding, it is only providing the equality testing between word vectors. A distributed representation of a word is used in Word2vec. Each word that with high dimension is represented by a distribution of weights across these elements. Instead of one-to-one mapping of between an element in the vector and a word, the representation of a word is spread across all of the elements in the vector, and each element in the vector contributes to the definition of many words.

As we can see from the Figure 1, words that are close in content, are closer in distance in vector space. Such a vector comes to represent in some abstract way the 'meaning' a word and also help to measure the similarity of words.

[1] Code: https://github.com/kirtisbhandari/cs240-pWord2Vec

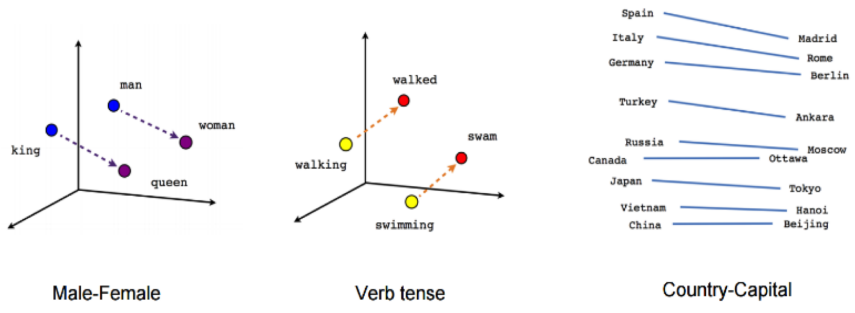


Figure 1: Analogies using word vectors

## 2.2 Why Word2Vec?

Suppose we have a query, *a:b; c:?* where *a, b, c* are the words from vocabulary and the answer to the query must be semantically related to *c* in the way *b* is related to *a*. For example, given the query *king:queen; man:?* We expect the model output *women*. As we can see, the vectors can provide answers to analogy questions.

## 2.3 The Word2Vec Model

There are two models we use in word2vec. One is the Continuous Bag-of-Words model (CBOW) and another one is the Skip-Gram model. Though CBOW and skip-gram are just inverted methods to each other, since CBOW can use many context words to predict the 1 target word, it can offer a good performance when the input data is not so large. However the skip-gram model is more fine grained so we are able to extract more information and essentially have more accurate embeddings when the data set is large.

### 2.3.1 Skip-Gram model

The idea behind skip-gram is to take in a word and predict the context words. Let's see how we would generate training data with the sentence:

***The quick brown fox jump over the lazy dog.***

We need to predict context words from a target word where the context words are the words to the left and to the right of the target word. We will choose a window size of 1. The training data for the sentence above will look like this:

***(quick, the) (quick, brown) (brown, quick) (brown, fox), . . . (lazy, dog)***

The idea is to be able to train our weights so that we predict context words from target words. So when we input 'quick', we should receive 'the'. We do this across the entire training set.

### 2.3.2 Continuous Bag-of-Words model (CBOW)

For the CBOW method, we predict target from context. From the example sentence above, the training data for the sentence will look like this:

***('the brown', 'quick') ('quick fox', 'brown') ('brown jump', 'fox), ...***

## 2.4 The Word2Vec Algorithm

Word2vec algorithm judges similarity between words and their surrounding words using the dot product of their word vectors. The goal is to minimize the distance between words and their

surrounding words (as obtained through the dot product).

The algorithm uses Stochastic Gradient Descent (SGD) [5] to solve the optimization problem. It randomly picks a pair of word and its surrounding word and computes the gradient of objective function w.r.t. this pair. It then updates the word vector representations of these words using the gradient values.

# 3 Efficient Word2Vec

Subsections below discuss the limitations of original word2vec implementation. Two approaches in which word2vec model can be parallelized to increase the efficiency of training are also highlighted. We also describe our experiments and resultant speed-up in each case.

### 3.0.1 Limitations of Word2Vec

Original word2vec implementation uses dot product of word vectors during Stochastic Gradient Descent (SGD) optimization phase. This is a level-1 BLAS operation [4] and cannot make use of computing power of multiple cores.

Also, Hogwild implementation [9] of SGD is limited by false sharing of caches and inter-thread communication.

## 3.1 Parallelized Word2Vec

pWord2Vec [6], a parallelized implementation of word2vec in distributed and shared memory systems tries to overcome the limitations of original word2vec and obtain a great speed-up.

The model uses a scheme based on minibatching and shared negative sampling as shown in Figure 2. Due to this, a level-1 BLAS operation for dot product can be converted into a level-3 BLAS operation for matrix-matrix multiplication.

This scheme is then parallelized over multiple batches of input while ignoring conflicting updates. This reduces the total number of updates to the shared model and limits the inter-thread communication.

We performed experiments by varying the number of threads from 2 to 72 using the above scheme. Table 1 shows the resultant speed in Millions of words per second. The experiments were conducted on a 1-billion words dataset. We have compared the results of these experiments with original word2vec implementation in Figure 3.

**Dataset:** 1 Billion words benchmark with a vocabulary of 1,115,011 words

**Model Parameters:** In the experiments on the one billion word benchmark, we follow the parameters (dim=300, negative samples=5, window=5, sample=1e-4, vocabulary of 1,115,011 words).

**Hardware:** We performed these experiments on dual-socket 24-core Intel Haswell E5-2680 v3 CPU.

**Observations:** As seen in Figure 3, this implementation leads to almost a 6X speed-up over original word2vec implementation. Scaling with number of threads leads to a linear increase in speed, however as the number of threads go above 24, the speed-up is lower than 6X. We suspect this is due to the fact that the machine has 24 cores, Our performance result is better than a reported 3.6X speed-up in the original paper.

## 3.2 Distributed Word2Vec

Efficiency of training word2vec model can be increased by using distributed training using parameter server framework [7]. There are various ways of training a model in distributed fashion. Either

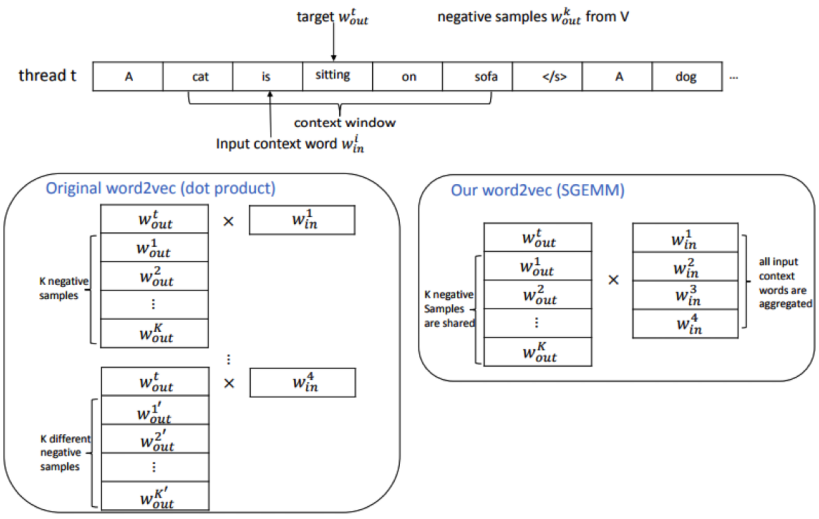


Figure 2: The parallelization schemes of the original word2vec(left) and our optimization(right).

the model or the data can be split into different workers. Since word2vec is a small model, the data was split into multiple workers and gradient updates were parallelized, while keeping the model parameters shares.and In a synchronized setting, multiple batches are processed at the same time. Once all the workers are done, the parameter updates are averaged and the only one update is performed. In an asynchronized setting, every worker updates the model parameters as soon as it is done. Figure 4 shows the intuition behind synchronous and asynchronous training.

**Dataset:** 17 Million words benchmark with a vocabulary of 71,291 words

**Model Parameters:** In the experiments on the one billion word benchmark, we follow the parameters (dim=300, negative samples=5, window=5, sample=1e-4, vocabulary of 71,291 words). The model was trained using 2 workers and 1 parameter server.

**Hardware:** We performed these experiments on dual-socket 24-core Intel Haswell E5-2680 v3 CPU.

**Observations:**

We experimented with asynchronous training with data parallelism using 2 workers and 1 parameter server. We recorded the time taken for training on 17 million words benchmark for 10 epochs with and without distributed training.

Distributed training with 2 workers and 1 parameter server led to a 2X increase in efficiency for training 500k steps ( 7 epochs). We also sampled few words and obtained 8 similar words using this trained word2vec models. Some of the obtained results are as follows in the decreasing order of similarity.

| |
|---|
| **Nearest to known**: regarded, defined, used, seen, such, described, served, needed, |
| **Nearest to into**: through, from, towards, within, thaler, across, almagest, build, |
| **Nearest to who**: she, still, which, he, never, often, defraud, intonation, |
| **Nearest to if**: when, though, where, before, although, then, while, after, |
| **Nearest to other**: various, including, numerous, many, dicrostonyx, subkey, reginae, individual, |

# 4 Conclusion

Word2vec model for generation of vector representations of words can be parallelized heavily through the use of techniques like mini-batching and negative sample sharing. Asynchronous

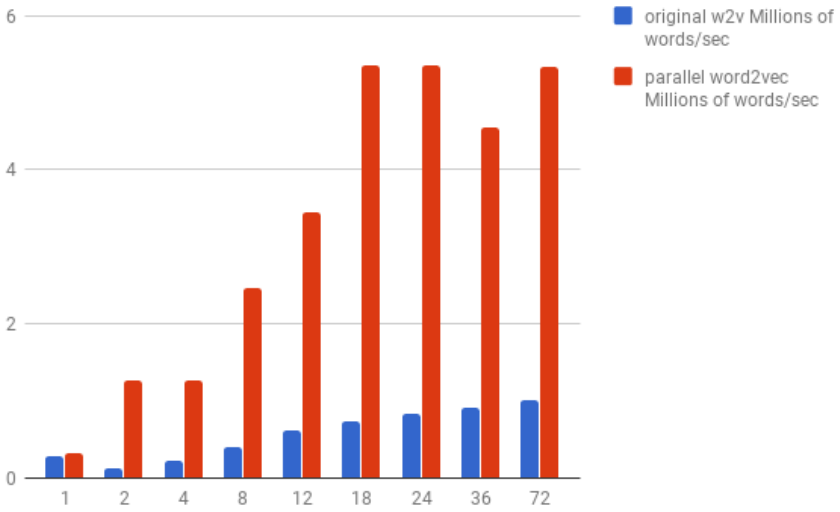


Figure 3: Scalability of original word2vec and parallelized word2vec using single Intel Haswell node and multiple threads

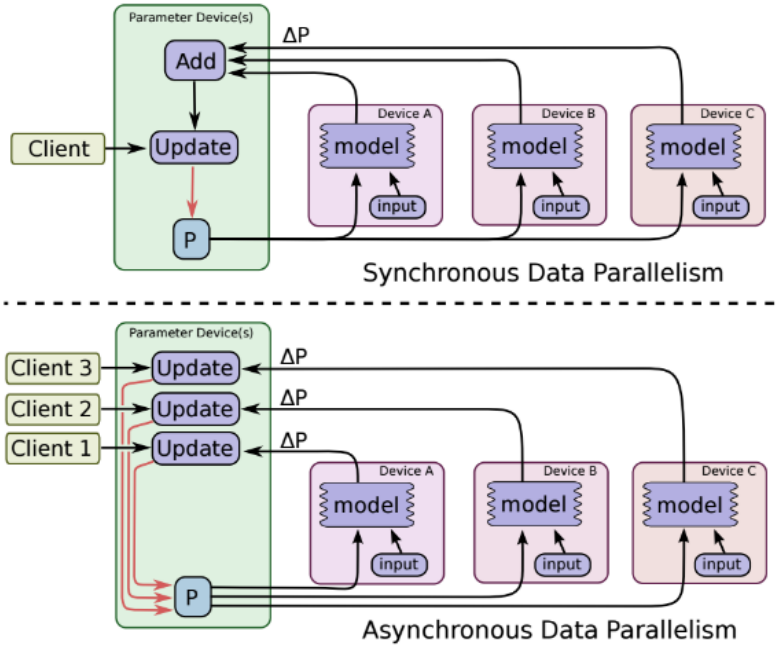


Figure 4: Synchronous and asynchronous training with data parallelism

| Threads | Speed (Mil Words/Sec) |
|---|---|
| 1 | 0.3301 |
| 2 | 1.2666 |
| 4 | 1.2708 |
| 8 | 2.4726 |
| 12 | 3.4414 |
| 16 | 5.3517 |
| 24 | 5.3610 |
| 36 | 4.5470 |
| 72 | 5.3405 |

Table 1: Performance of parallelized word2vec implementation with varying number of threads

model updates through parameter server model can also lead to an increase in efficiency. Two approaches discussed above can be combined to improve pWord2Vec implementation. Further studies to selectively synchronize the model may also lead to significant performance improvement.

# References


[1] https://github.com/caicloud/tensorflow-demo/tree/master/distributed.

[2] https://github.com/IntelLabs/pWord2Vec/.

[3] https://www.tensorflow.org/deploy/distributed#replicated_training.

[4] L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[5] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[6] Shihao Ji, Nadathur Satish, Sheng Li, and Pradeep Dubey. Parallelizing word2vec in shared and distributed memory. *arXiv preprint arXiv:1604.04661*, 2016.

[7] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 1, page 3, 2014.

[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[9] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

**Data for all the experiments can be obtained as**

```
cd parallelized_w2v/multithreaded_par_w2v/data
.\getText8.sh
.\getBillion.sh
```

**Running Google's original word2vec**

```
cd parallelized_w2v/original_w2v
sbatch run_single.sh
```

**Running parallelized word2vec**

```
cd parallelized_w2v/multithreaded_par_w2v
sbatch run_single.sh
```

**Running Google's tensorflow implementation of word2vec**

```
cd distributed_tensorflow_w2v/original_tf_w2v/
python word2vec.py --train_data=/tmp/text8 --eval_data=/tmp/questions-words.txt
```

**Running distributed tensorflow implementation of word2vec**

```
cd distributed_tensorflow_w2v/distributed-w2v/
kubectl exec -it <pod_id> -- python word2vector.py --worker_grpc_url=grpc://tf-worker0
```