Scalable Word Embeddings

Zijian Zhang
Master of Science
Informatik
Leibniz Universität Hannover
zhangzijian0523@gmail.com

Avishek Anand
Post Doctor
L3S Research Center
KBS Leibniz Universität Hannover
anand@l3s.de

Abstract

In this paper, we propose a method of distributed word embedding with the help of gensim. At first we calculating dedicated sub-models using parts of corpora, then we combine them using simply sorting over each embedding dimension or Low Rank Alignment(LRA). The separation and combination may do little harm to evaluation result, but the speed acceleration of our new distributing way of training model may provide a new trade-off argumentation between training speed and performance.

1 Introduction

In NLP applications, the first problem to solve is to find a propose representation (or token) of words (Schütze, 2008). One can use a random generated integer token, one-hot encoding (Turian et al., 2010) or Huffman code (El Daher and Connor, 2006), with respect to data compression, deal with it, as long as the representation could be únderstoodb́y machines. However, representations such as simple integer don't take the semantic information into consideration (Le and Mikolov, 2014). While method such as one-hot encoding will also cause *curse of dimensionality* if being directly used in an NLP applications (Bengio et al., 2003). Word representations with respect of semantic and within low dimension are therefore need to be developed.

As human beings, we can understand the meaning of one word through looking for the corresponded entry in a dictionary and read the description. In order to understand the description we have to find a description of the description, i.e. meta-description, meta-meta-description and so on all the way run into the awkward stymie of self-reference. It's alright for a human brain but a catastrophe for modern computers who are based on formal arithmetic system. This self-reference can't be complete and conflict-free according to Gödel's incompleteness theorems (Gödel, 1931).

Thanks for the contribution by Zellig Harris, a basic hypothesis on distributional semantics was introduced that linguistic items with similar distributions have similar meanings. Meaning, or semantic similarity between two linguistic expression depends strongly on the circumstance or the context they appear (Harris, 1954). Fortunately, computers are champions in forming distributional properties by counting and regression.

This word representation dates back to 1986 due to Rumelhart, Hinton and Williams (Williams and Hinton, 1986). Word embedding uses co-occurrence of words and a softmax functional with help of stochastic gradient descent(SGD) to train a model, where each word is represented by, saying émbedded iná vector in a high-dimensional space. In which model metric of distance (e.g. L2-Norm) between a pair of synonyms, or other pairs of words representing similar concept is smaller than other pairs. Also those representation vectors of words have also ṕarallelismín word pairs, such that vec("Beijing")-vec("China")+vec("Germany") is close to vec("Berlin") (Le and Mikolov, 2014), which means vector from "China" to "Beijing" is somehow has the same direction and norm as the vector from "Germany" to "Berlin". Popular applications of word embedding are for example machine translation (Cho et al., 2014), image annotation (Weston et al., 2011) and so on.

Distributed word representation can be built through Point-wise Mutual Information (PMI) (Church and Hanks, 1990). Moreover, it was later on researched that introduction of Positive PMI (PPMI) (Bullinaria and Levy, 2007) and Shifted

PMI (SPPMI, first presented in (Goldberg and Levy, 2014)) improves performance of models, considering the the fact that introduction of hyper-parameter k will shift the to optimize $PMI(w, c)$, where k is the number of negative samples in Skip-gram with Negative Sampling and `w` is the central word and `c` is the native sample or context word (Levy and Goldberg, 2014). Methods such as PPMI and SVD are usually referred as *"count based"* methods (Levy et al., 2015), which focus on counting the co-occurrence of word and it's context as well as performing linear transformation of the co-occurrence matrix using techniques like lower-rank representation with help of SVD. To dig deeper with respect of co-occurrence matrix, GloVe (Pennington et al., 2014) introduces a loss function

$$J = \sum_{i,j=1}^{V} f(X_{i}j)(w_i^T \tilde{w_j} + b_i \tilde{b_j} - log X_{i}j)^2, \quad (1)$$

where $f(X_{i}j)$ is

$$f(x) = \begin{cases} (\frac{x}{x_{max}})^{\alpha} : \text{if } x < x_{max} \\ 1 : \text{otherwise} \end{cases}$$

Alternatively, vector-based models can be generated with *predictive* methods, which generally outperform the count-based methods (Levy et al., 2015). The most notable of which is Skip-gram with Negative Sampling (SGNS) with neural network referred in (Mikolov et al., 2013), which uses a neural network with only one hidden-layer to generate embeddings. Which implicitly factorizes a shifted PMI matrix (Levy and Goldberg, 2014). In this paper, the base-line model is generalized using framework `gensim`[1] with SGNS and stochastic gradient descent (SGD) to minimize a loss function that defined on both co-occurrence of words in a same window (positive co-occurrence of central word and context) and negative co-occurrence between central word of a window and negative sample words.

2 Brief introduction on word embedding with Skip-gram model

As illustrated in (Mikolov et al., 2013), the goal of training process of a distributed representation model is to find a participate parameter combination for a 1-(hidden-)layer neural network. In this section we will go through how a distributed representation model of words as vectors is trained.

2.1 Notions and Conventions

V: Set of all vocabularies. Before training the model using corpus, we need to identify all of the legal words, i.e. words do have meaning and are frequent. Normally a threshold τ_v is introduced, so that all the words occur less than τ_v times within whole corpus are pruned. For the sake of a systematic comparison, in our experiment we use vocabulary extracted from `GoogleNews-vectors-negative300.bin`[2], whose count is approximately same as (Levy et al., 2015). Following are the notions used within this paper.

v: number of vocabulary, i.e. $|V|$,

$X_{1\times v}$: input of the neural network, one-hot encoding of a word,

dim: dimension of embedded space,

$L1_{v\times dim}$: embedding layer, multiply a one-hot vector with it produces the embedding of the input word,

neg: number of random negtive sampling,

$L2_{dim\times v}$: hidden layer. Because of the using of negtive-sampling while calculating the embedding in our experiment, it usually occures only partially, denoted as $L2P_{dim\times(neg+1)}$, where the "+1" indicates the target word in the output of network.

With Skip-gram method in (Mikolov et al., 2013) using a simplified variant of Noise Contrastive Estimation (NCE) (Gutmann and Hyvärinen, 2012), the activation function in output layer is a logit function: $y = \frac{1}{1+\exp^{(-syn0\times W^T)}}$. The final loss function is according to (Mikolov et al., 2013) denoted as

$\log \sigma(v'_{wO})$In each round of training, we at first find the context word and target word in the same window, then after forward propagation with respect to network structure, the activation function calculates predicted labels, which can be treated as 'co-occurrence' between context word and target word, as well as 'co-occurrence' between negative sampled words and target word. Then the error are defined as difference between labels (context word = 1, negtive sampled word = 0) and predicted result, which is used to multiply with learning factor

[1] http://radimrehurek.com/gensim/

[2] found in https://code.google.com/archive/p/word2vec/

α in the backward propagation. These iterate until the result convergent.

3 System-level Optimization Methods

Training a well-performed natural language model requires huge size of corpora. However, this process is highly time consuming. For example, the standard experiment mentioned in this paper spends approximately 4 days. An optimizing method or several methods are thus needed to be performed. Hyper-parameters such as number of negative sampling, size of window, embedding dimension as well as minimum threshold of count of vocabulary all around corpora may do effect to training time. Also according to the paper (Levy et al., 2015), these hyper-parameters also affect performance of a trained model. Therefore in order to shorten the training time while preserving performance, our strategy is either to find a trade-off for those parameters, or keep those hyper-parameters in the most performing way then parallelize the training process. Several methods are later on to be introduced and used to compare training time as well as performance with baseline model.

There are already several ways of parallel computation models of words embedding. For example the work of Erik Ordentlich et. al (Ordentlich et al., 2016) figures out the bottleneck of building a distributed training system is the network overhead. The transferring of both output and input vectors of words to the word2vec client, as well as gradient to PS Shards, provoke huge network throughput. (Ordentlich et al., 2016)'s solution to that is divide each of embedding vectors ”vertically (divide a d-dimension vector into k d/k dimension vectors)” into several components, each of which is maintained by one of the PS Shards. Every time after each word2vec client select a central word with its context as well as negative samplings (e.g. in Skip-gram), instead of require embedding vectors from PS Shard, they use remote procedure call (RPC) to tell every PS Shard to calculate dot product and gradient locally, avoiding transferring of intermediate result and reduce the training time.

Another paper from Ji, Shao et. al. (Ji et al., 2016) provides a optimized scheme of leveraging BLAS level 3 function to accelerate forward propagation. Considering a word2vec training process using Hogwild (Recht et al., 2011) log-free strategy. Every time after a central word, its context word and negative samplings was selected, a dot product between embedding of context word and input weight of target(central) word and negative samples have to be calculated. in legacy Hogwild, negative samples and target words could be mutually different among different contexts. Under this circumstance the scalar product is only between a matrix and a vector, which means it can only leverage level 2 BLAS to accelerate. The paper propose that we could shard all the context words together in every single batch, using same target(central) word and negative samples. The scalar product is the between two matrices and BLAS level 3 functions could dedicate.

Contrastively, the work from Jeroen B.P. Vuurens et.al. (Vuurens et al., 2016) researches in the level of hardware and proposes an efficient way of using high-speed cache of CPU. They find the level 3 BLAS presented by Ji et.al. implicitly lowers the number of times writing and reading shared memory, which consequently provokes the conflict and increment of queuing time because of concurrent access of the same memory. They found if an effective cache strategy is exploited, hierarchical softmax can benefit due to its tree-type structure and thus frequent access of the root.

4 Dedicate sub-models training and combination

Unlike the systematic way mentioned in the section above, we present in this paper essentially such method that at first train dedicate sub-models separately, each of which consume part of corpus file, instead of the whole file. These processes can be run at different processors or even different mappers in a MapReduce cluster (Dean and Ghemawat, 2008). Then these sub-models are combined with several different strategies. Due to $iter$ times passing-through of whole corpus in the traditional training process, it's trivial that the training time of a distributed representation model depends substantially on size of corpus. All of word vectors form an ambient Euclidean space, the purpose of distributed word representation is construction of a low-dimensional representation of one of its subspace (Mahadevan and Chandar, 2015). Several ”coarse” sub-models can be trained by using information from different parts of corpus. Which can be irrelevant with each other or be sampled so that the distribution of words are preserved. Then these sub-models can be combined using different strategies to reduce the "coarseness" of each single one of them. There are various of strategies can be conducted while dividing corpus, aligning of manifolds in each models and combining them together. In the following subsections we will go through them one by one.

4.1 Dividing corpus

We can divide the corpora using two different ways, the following part of this sub-section will describe them briefly.

4.1.1 Interweaving divide

We can divide the corpus in such a interweaving way, that at first we denote sentences in all of documents in corpus with integers and if we divide the whole corpus into p pieces, where $p \in Z_+$ and the t-th piece becomes all sentences that denoted as $np + (t-1)$, where $n \in Z$ and $t < p$. For example if we have the sentences numbered as 0, 1, 2, 3, 4, 5, ... in the 0th piece becomes 0, 2, 4, ... while the 1st becomes 1, 3, 5, ... Division using this way enables each piece contains different sentences, while spread each documentation into different pieces by granularity of sentence. In this way, each model trained from a sub-corpus describes a partial information of original corpus. And because a document in the original corpus contains most of the time more than one sentences, also under the hypothesis that one word in the same document means throughout the same, only perturbation instead of huge deviation between embedding vectors of sub-models can be introduced.

4.1.2 Division using random sampling

If maintenance of word distribution is required for the sake of preserving the semantic properties of words, a sampling method can be leveraged. Like described before, at first all of sentences contained in corpus are denoted by integers. Then for each one of p sub-corpus, we sample $\frac{|D|}{p}$ sentences, where $|D|$ is the total number of sentences in corpus and like mentioned before, p is the number of sub-corpora.

While dividing the corpus into p pieces, each sentence in the original corpus is sampled with probability $\frac{1}{p}$, so that important statistical properties of words and their context can vary. Denote the count of the central word w_i within the original corpus as $\#(w_i)$, while $\#(w_i^t)$ represents count of this central word in the t-th sub-corpus. Considering the most common situation that each sentence may contain w_i once or absolutely not. The $\#(w_i)$ then equals to the count of sentences, where w_i appears. If the sentences in a single sub-corpus are sampled from original corpus in probability of $\frac{1}{p}$, $\#(w_i^t)$ is now $\#(w_i)$. Analogously for $\#(c_j)$ and $\#(c_j^t)$ of context words c_i, as well as number of co-occurrence $\#(w_i, c_j)$, $\#(w_i^t, c_j^t)$ of w_i and c_j.

4.2 Order while combining models

When the combination processes, extra error can be introduced because of calculation accuracy of modern computer, or some combination methods, like PCA, causes deviation per se. As combination goes by, these errors or deviations can accumulate. Therefore it's reasonable to take combination order into consideration, if we want to recover the global model in a most lossless way. In our experiment, three types of combination order are taken into the consideration. They are to introduce in the following part of this section.

4.2.1 Successive combination

The successive combination means that, if we want to merge 5 models with number [1, 2, 3, 4, 5] together, model 1 and 2 are at first combined. Then the result of this combination is used to combine with model 3... so on and so forth. This combination order should run the fastest because it introduces no additional data structures or calculation, also it treated model sequence as a stream and actually regardless of numeric order of them. However the perturbation introduced while combining two models keep increasing, because from the second combination on, one of two combinators is always the result of previous combination.

4.2.2 Binary order

Like the example before, the combination of models sequence [1, 2, 3, 4, 5] starts with poping out two left most model 1 and 2 and combining them together 1 and 2, the result is named as model 1_2 and append into end of models sequence. This pop→combine→append procedure is repeated again and again until there is only one single model left in the sequence (in this case, the final result should be named as 3_4_5_1_2). This strategy uses more RAM as it entailing all the combined models currently in sequence maintained, or dumped in hard disk, which is in turn time consuming.

4.2.3 Combine two with minimum Procrustes error

Schönemann's work (Schönemann, 1966) solved so-called Orthogonal Procrustes Problem perfectly. This problem is defined as alignment of two matrix A and B with transformation using a orthogonal matrix C so that the trace of the rest error matrix E multiply with its transpose is minimum. The matrix T is then

$$\min_T tr(E^\intercal \cdot E) = B - A \cdot T. \tag{2}$$

We define here the $tr(E^\intercal \cdot E)$ as the Procrustes Error. In the following sections of this paper this orthogonal transformation will be introduced as one kind of alignment methods between two models. In aspect of combination order, we can combine each time such pair of matrices, that the Procrustes Error is minimum among all pairs of models.

4.3 Alignment between models

After training sub-model using each sub-corpus dedicatedly, we can proceed to the combination phase of the whole work flow. One simple approach of combination is to just add different embedded vectors together, hoping the local information presented by a single model can also be maintained using single vector addition without any alignment. But inspecting only two word vector pairs from model 0 and model 1, say $(\vec{a}_0, \vec{b}_0)$ and $(\vec{a}_1, \vec{b}_1)$, without losing the generality. The inner product of each vector pair illustrates the similarity of word a and b in each model respectively. Simply adding vectors from model 0 with correspondent vectors in model 1 separately doesn't guarantee that information expressed by two dedicated models using inner product still preserves. We expect that in the merged model, inner product of $(\vec{a}_m, \vec{b}_m)$ should be a function that depends on only $\langle \vec{a}_0, \vec{b}_0 \rangle$ and $\langle \vec{a}_1, \vec{b}_1 \rangle$. However after using the simple vector addition, the new inner product is

$$\begin{aligned} &\langle (\vec{a}_0 + \vec{a}_1), (\vec{b}_0 + \vec{b}_1) \rangle \\ =&\langle \vec{a}_0, \vec{b}_0 \rangle + \langle \vec{a}_1, \vec{b}_1 \rangle + \langle \vec{a}_0, \vec{b}_1 \rangle + \langle \vec{a}_1, \vec{b}_0 \rangle, \end{aligned} \tag{3}$$

where the last two part in the right side depends on the between-model-distortion because of the non-alignment of models. Thus some alignment should be performed.

4.3.1 Alignment through Orthogonal Linear Transformation

Defined and solved in (Schönemann, 1966), the Orthogonal Procrustes Problem focuses on solve such problem: Given matrix A and matrix B, find a orthogonal transformation matrix T so that the squared mean error (SME) between transformed A using T and B is minimized. Mathematically speaking, define

$$T = \min_T tr[E^\intercal \cdot E], \tag{4}$$

where

$$E = B - A \cdot T, \tag{5}$$

under the constraint that

$$T \cdot T^\intercal = T^\intercal T = I. \tag{6}$$

This transformation minimize the inter-model distortion under the constraint that only orthogonal transformations are allowed. Thus reduce the last two items in the (3) in a way.

4.3.2 Low rank alignment

The work from Boucher et al. (Boucher et al., 2015) provides another approach of aligning different manifold together. Consider X and Y are two manifold to be aligned. They are at first decomposed using SVD such that $X = U_x S_x V_x^\intercal$ and $Y = U_y S_y V_y^\intercal$. With out losing the generality, S_x and V_x are partitioned into $V_x = [V_{x1} V_{x2}]$ and $S_x = [S_{x1} S_{x2}]$ according to

$$I_1 = \{i : s_i > 1 \forall s_i \in S\} \tag{7}$$

and

$$I_2 = \{i : s_i \leq 1 \forall s_i \in S\}. \tag{8}$$

This decomposition is used for preparation of low-rank-representation X and Y using Low rank embedding (LRE). The LRE problem is defined as that, given a data set X, finding a proper transformation matrix R, in order to minimize the loss function,

$$\min_R \frac{1}{2} \|X - XR\|_F^2 + \lambda \|R\|_*, \tag{9}$$

where $\lambda > 0, \|X\|_F = \sqrt{\sum_i \sum_j |x_{i,j}|^2}$ is called Frobenius norm, while $\|X\|_* = \sum_i \sigma_i(X)$ is the spectral norm and where by σ_i are singular values. (Candès and Tao, 2010) proved that the formula (9) is a convex relaxation of rank minimization problem and it's result R is the so-called reconstruction coefficients, which describe the intra-manifold relationship between points inside a single manifold. The (9) is showed in (Favaro et al., 2011) can be solved in closed form. This is where we use the decomposition of X and Y. For example, the optimal closed-form solution of reconstruction of coefficients for X is

$$R^{(X)} = V_x 1 (I - S_x 1^{-1}) V_x 1^{\intercal}. \tag{10}$$

Once the $R^{(X)}$ and $R^{(Y)}$ are calculated, they can be blocked as

$$R = \begin{bmatrix} R^{(X)} & 0 \\ 0 & R^{(Y)} \end{bmatrix}$$

and

$$C = \begin{bmatrix} 0 & C^{(X,Y)} \\ C^{(Y,X)} & 0 \end{bmatrix},$$

where $C^{(X,Y)}$ is inter-manifolds correspondence, defined as

$$C_{i,j}^{(X,Y)} = \begin{cases} 1 : X_i \text{ is in correspondence with } Y_i \\ 0 : \text{otherwise} \end{cases}$$

.

Defining $F \in \mathbb{R}^{(2N \times d)}$ as

$$F = \left[F^{(X)} \; F^{(Y)} \right],$$

where N is the number of points in each manifold and d is the dimension of both manifolds, $F^{(X)}$ and $F^{(Y)}$ are the aligned manifolds. The alignment precision of F can be described as loss function

$$\mathcal{Z}(F) = (1-\mu)\|F-RF\|_F^2 + \mu \sum_{i,j=1}^{N} \|F_i - F_j\|^2 C_{i,j}, \tag{11}$$

where $\mu \in [0, 1]$ is a hyper parameter that controls the inter-manifold correspondence or intra-manifold correspondence matters significantly. With help of the Lagrange multipliers method, equation (11) can be solved by finding the d smallest non-zero eigenvectors of the matrix

$$(1 - \mu)M + 2\mu L, \tag{12}$$

where

$$M = \begin{bmatrix} (I - R^{(x)})^2 & 0 \\ 0 & (i - R^{(Y)})^2 \end{bmatrix},$$

and

$$L = \begin{bmatrix} D^X & -C^{(X,Y)} \\ (-C^{(X,Y)})^{\intercal} & D^Y \end{bmatrix},$$

where by

$$D = \begin{bmatrix} D^X & 0 \\ 0 & D^Y \end{bmatrix}$$

is a diagonal matrix.

4.4 Normalized or Unnormalized?

According to work of Levy et al. (Levy et al., 2015), normalization of each vector enables that every time when we calculate inner product of two normalized vectors, we are actually calculate the cosine similarity between them. Considering float number of dimensions composing original (unnormalized) vectors in each sub-model can scale differently, a normalization that every vector is kept to length 1 also makes it possible that when an orthogonal transformation-vector addition of two models is employed, the result will not trend to the model, whose vectors has higher L2-norm.

4.5 Combination of sub-models

After alignment (or without it) and normalization (or without it), it's finally the time of combination. In our experiments, two kinds of combination are employed, thus direct vector addition and projection using PCA. In following sections they are to be briefly depicted how and why they might or might not work in our experiments.

4.5.1 Using vector addition

Add vectors together is one of the first intuition when talking about combination two bundle of vectors. The direct addition of two vectors from two sub-models can be seen as calculating middle point of two vectors when put them into a same space. However as depicted in (3), simple addition may introduce additional distortion because of nonalignment between models. Thus for comparison, we ran naked vector addition as well as vector addition with alignment using orthogonal transformation and low rank alignment.

4.5.2 Using PCA and projection

If vectors from two different models are just stacked together without any additional computation, all information from both models can be preserved. However because of the fact that we are using different part of same corpus, identical word does have same semantic within the same corpus. Also because semantic is under our hypothesis implied by the distribution of itself and it's context, only thing then has to be concerned about is the distribution of words with their context in different corpus fragments. These distributions can be similar with each other while some distribution-preserving sampling techniques of sentences are employed when slicing the original corpus. We denote the final result of SGNS model using entire original corpus as matrix W_i and C_i, which are vector expression of all central words and context words in the vocabulary, are according to the (Levy and Goldberg, 2014), there is relationship between these two matrices and PMI of each pair of w_i and c_j, such as

$$\begin{aligned} M_{ij}^{SGNS} &= W_i \cdot C_j = \vec{w_i} \cdot \vec{c_j} \\ &= PMI(w_i, c_j) - \log k \\ &= \log\left(\frac{\#(w_i, c_j) \cdot |D|}{\#(w_i) \cdot \#(c_j)}\right) - \log k. \end{aligned} \tag{13}$$

When the original corpus is divided using sampling method into d pieces, within each piece this relation ship according to 4.1.2 turns to be like

$$M_{ij}^{frag\_comb} = \log\left(\frac{\frac{\#(w_i, c_j)}{d} \cdot \frac{|D|}{d}}{\frac{\#(w_i)}{d} \cdot \frac{\#(c_j)}{d}}\right) - \log k, \tag{14}$$

where k as number of negative samples keeps the same. Based on the fact that each fragment of corpus focuses only on partial information, each generated sub-model introduces additional variance to $\vec{w}$ s and $\vec{c}$ s, while their expectations are preserved. Denoting the concatenated embedding matrix for central words as

$$W_{con} = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix},$$

where W_1 and W_2 are the embedded vectors from two models, the most principal component of the concatenated matrix $\vec{p_c^1 on}$ should be also the concatenation of two most principal components from both matrix, denoted as

$$\begin{bmatrix} a_{11}\vec{p_1^1} \\ a_{12}\vec{p_2^1} \end{bmatrix},$$

where $a_{11}, a_{12} \in \mathbb{R}$ are scaling factors, making the length (L_2 norm) of this base vector stays 1. So that in one hand according to the logic of PCA the sum of projections of all data vectors alone this direction is maximized, in the other hand transformation matrix is normalized and orthogonal, which enable a transformation of base merely through matrix multiplication. Analogously, $\vec{p_{con}^2}$ should also be like

$$\begin{bmatrix} a_{11}\vec{p_1^2} \\ a_{22}\vec{p_2^2} \end{bmatrix},$$

so on and so forth. A set of in d dimension embedded vector has maximum d principal component, with respect that there can be maximum d orthogonal basis that can span the subspace those vectors embedded in. Therefore only first d principal components of W_{con} are informative, meanwhile can form a orthogonal normalized matrix with shape of $(2d, d)$. This orthogonal matrix projects the concatenated vectors into vectors with "standard length", which makes them ready for the next round of combination.

5 Experiment results

"For this paper, we finished all experiments following the advice in (Levy et al., 2015), thus hyper-parameters are chosen as `win=10`, `neg=5`, `dim=500` and `iter=5`. English wiki dump (enwiki-latest-pages-articles on March 2nd, 2016[3]) is used as corpus. Parts of experiment results, with configuration in which performance the best, are demonstrated in table 1. Results for all combination of experiment setting is shown in appendix A. In those tables the line "baseline" represents the performance of model who uses whole corpus with out any division or combination. The lines named as "avg skip,sampling" represent the average performance over all fragments produced by two types of division methods. Each column in both tables stands for an evaluation dataset or time consumed for combination or (average) training. This benchmark is also kept the same with (Levy et al., 2015). All of the experiments ran on 18 cores of a server with dual-way Intel(R) Xeon(R)

[3]from https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2

CPU E5645 @ 2.40GHz with 126G of RAM. The size of entire corpus is approximately 14G. The combination and training times, however, because of public occupation of the server within institute, are for reference only. What is not mentioned in both table is that the average division time for corpus. For `skip` method this time is 2.487 sec, for `sampling` method it is 41.878 sec."

From the result we can draw several conclusions.

- `sampling` method performs under nearly all circumstances better than `skip` method,
- `PCA`, without any `LRA` or normalization, cooperating especially with `seq` order, performs the best,
- around most of data set in all three type of data set (categorization, similarity and analogy), `LRA` improves result of direct vector addition,
- `PCA` and orthogonal transformation should never occur together with `LRA`.

The reason for the first conclusion is like inferred in section 4.1, `sampling` method preserves distribution over all words and thus produces model fragments more 'smoothly'. This consequentially represents less disturbance among model fragments and thence better performance when combining several model fragments together. For the second conclusion, from the table we can see that under both types of division method the configuration `seq-PCA` without `LRA` or normalization seizures top-3 among all configurations on 10 out of 17 evaluation datasets. This also proves the hypothesis that compared with direct vector addition and vector addition after orthogonal transformation, PCA is the most optimal method while decreasing data-dimensions when most of dimensions provide no more information but redundancy. The `seq` (consecutive order) however, doesn't help much when the combination process runs in a e.g. Map-Reduce cluster. Therefore binary order can be a secondary choice when large scale parallel computation is needed. As to the third conclusion, if a `PCA` is too time and space consuming, considering the calculation demand of PCA or SVD, directly vector addition between two models can be a secondary substitution. A `LRA` might help to improve the performance of vector addition because of the alignment, but `LRA` itself is too time consuming and thus the profit doesn't cover the loss in the aspect of time. `LRA` provide an alignment approach with help of low-rank-representation, which according to (Boucher et al., 2015) guarantees only local linearity. Furthermore, according to the formulas in `LRA` algorithm, it treats disturbance between data as noise and try to diminish it, which firstly do harm to information to both model fragments, which is already be done by PCA and orthogonal transformation[4] and secondly make our central thought of 'enrich the detail through combination of several model fragments' meaningless.

6 Conclusion and future work

In this paper we provide several configurations to combine models trained using different corpus (fragments) together, and find with the help of `PCA` using consecutive order performs the best. But in the future when this combination process should run on a e.g. Map-Reduce cluster in order to decrease the calculation time one step further, binary order can be then chosen with only slice performance loss. Certainly for optimizing construction process of word embeddings there ere still a lot of other possible approaches may worth trying. For example when we concatenate all the vectors from sub-models together, we get actually a word embedding with dimension dk. An auto-encoder can then be used efficiently for dimension reduction (Hinton and Salakhutdinov, 2006). Furthermore, method like asynchronous VRSGD (Keuper and Pfreundt), (Keuper and Pfreundt, 2015) provides the possibility of optimizing SGD in a parallel way. This may help if we want to reform the word embedding fundamentally.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Janvin, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3(Feb):1137–1155. https://doi.org/10.1162/153244303322533223.

Thomas Boucher, CJ Carey, Sridhar Mahadevan, and Melinda Darby Dyar. 2015. Aligning mixed manifolds. In *AAAI*. pages 2511–2517.

John A Bullinaria and Joseph P Levy. 2007. Extracting semantic representations from word co-occurrence

[4] due to calculation accuracy of modern electronic computer

Configuration	AP	ESSLI_1a	ESSLI_2c	MEN	SimLex999	WS353	WS353R	Google	MSR	time (sec)
skip-seq-PCA	0.607	0.795	0.556	0.741	0.345	0.613	0.526	0.714	0.442	12
sampling-seq-PCA	0.642	0.795	0.556	0.741	0.345	0.614	0.527	0.713	0.441	267
sampling-seq-PCA-lra	0.453	0.659	0.6	0.051	0.036	-0.052	-0.005	0	0	24429
sampling-seq-PCA-lra-normed	0.48	0.636	0.622	0.038	-0.038	0.012	-0.055	0.001	0.001	32168
sampling-bin-PCA	0.622	0.818	0.6	0.741	0.344	0.612	0.525	0.712	0.44	150
sampling-MPE-PCA	0.619	0.795	0.556	0.741	0.344	0.613	0.525	0.713	0.441	351
skip-bin-vadd	0.595	0.795	0.533	0.713	0.302	0.561	0.475	0.654	0.378	2
skip-bin-vadd-lra	0.54	0.659	0.644	0.702	0.391	0.619	0.535	0.68	0.425	11064

Table 1: Part of experiment results

statistics: A computational study. *Behavior research methods* 39(3):510–526.

Emmanuel J Candès and Terence Tao. 2010. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory* 56(5):2053–2080.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* .

Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational linguistics* 16(1):22–29.

Jeffrey Dean and Sanjay Ghemawat. 2008. Mapreduce: simplified data processing on large clusters. *Communications of the ACM* 51(1):107–113.

Antoine El Daher and James Connor. 2006. Compression Through Language Modeling .

Paolo Favaro, René Vidal, and Avinash Ravichandran. 2011. A closed form solution to robust subspace estimation and clustering. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, pages 1801–1807.

Kurt Gödel. 1931. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik* 38(1):173–198.

Yoav Goldberg and Omer Levy. 2014. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722* (2):1–5. https://doi.org/10.1162/jmlr.2003.3.4-5.951.

Michael U Gutmann and Aapo Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research* 13(Feb):307–361.

Zellig S Harris. 1954. Distributional structure. *Word* 10(2-3):146–162.

Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* 313(5786):504–507.

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

Janis Keuper and Franz-Josef Pfreundt. ???? Asynchronous parallel stochastic gradient descent .

Janis Keuper and Franz-Josef Pfreundt. 2015. Asynchronous parallel stochastic gradient descent: a numeric core for scalable distributed machine learning algorithms. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. ACM, page 1.

Quoc V Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *ICML*. volume 14, pages 1188–1196.

Omer Levy and Yoav Goldberg. 2014. Neural Word Embedding as Implicit Matrix Factorization. *Advances in Neural Information Processing Systems (NIPS)* pages 2177–2185. https://doi.org/10.1162/153244303322533223.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics* 3:211–225.

Sridhar Mahadevan and Sarath A. P. Chandar. 2015. Reasoning about Linguistic Regularities in Word Embeddings using Matrix Manifolds. *Arxiv* pages 1–9. http://arxiv.org/abs/1507.07636.

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

Erik Ordentlich, Lee Yang, Andy Feng, Peter Cnudde, Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, and Gavin Owens. 2016. Network-efficient distributed word2vec training system for large vocabularies. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, pages 1139–1148.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in*

Natural Language Processing pages 1532–1543. https://doi.org/10.3115/v1/D14-1162.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*. pages 693–701.

Peter H Schönemann. 1966. A generalized solution of the orthogonal procrustes problem. *Psychometrika* 31(1):1–10.

Hinrich Schütze. 2008. Introduction to information retrieval. In *Proceedings of the international communication of association for computing machinery conference*. pages 22–27.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, pages 384–394.

Jeroen B. P. Vuurens, Carsten Eickhoff, and Arjen P. de Vries. 2016. Efficient Parallel Learning of Word2Vec. *ICML 2016 Machine Learning workshop* 48. http://arxiv.org/abs/1606.07822.

Jason Weston, Samy Bengio, and Nicolas Usunier. 2011. Wsabie: Scaling up to large vocabulary image annotation .

DRGHR Williams and GE Hinton. 1986. Learning representations by back-propagating errors. *Nature* 323:533–536.

A Results of Experiments

Configuration	AP	BLESS	Battig	ESSLI_1a	ESSLI_2b	ESSLI_2c	MEN	MTurk	RG65	RW	SimLex999	WS353	WS353R	WS353S	Google	MSR	SemEval2012_2	Combination Time (sec)
sampling-bin-lint	.550	.660	.362	.705	.625	.578	.650	.583	.656	.258	.284	.557	.480	.697	.610	.331	.133	193
sampling-bin-lint-lra	.388	.505	.208	.659	.625	.511	.556	.478	.601	.197	.318	.545	.485	.602	.544	.291	.120	14472
sampling-bin-lint-lra-normed	.388	.505	.208	.659	.625	.489	.556	.478	.601	.197	.318	.545	.485	.602	.544	.291	.120	15985
sampling-bin-lint-normed	.550	.660	.362	.705	.625	.578	.650	.583	.656	.258	.284	.557	.480	.697	.610	.331	.133	226
sampling-bin-PCA	.622	.785	.436	.818	.750	.600	.741	.680	.787	.291	.344	.612	.525	.756	.712	.440	.185	150
sampling-bin-PCA-lra	.172	.230	.094	.432	.550	.356	.020	.013	-.172	-.013	-.060	-.047	-.084	.028	.000	.000	.000	16020
sampling-bin-PCA-lra-normed	.172	.230	.094	.386	.550	.356	.020	.013	-.172	-.013	-.060	-.047	-.084	.028	.000	.000	.000	16652
sampling-bin-PCA-normed	.622	.785	.424	.795	.750	.533	.741	.680	.787	.291	.344	.612	.525	.756	.712	.440	.185	226
sampling-bin-vadd	.532	.780	.412	.795	.725	.533	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	246
sampling-bin-vadd-lra	.535	.700	.319	.659	.625	.556	.702	.506	.779	.268	.391	.619	.535	.740	.680	.425	.166	16213
sampling-bin-vadd-lra-normed	.535	.700	.319	.705	.625	.600	.702	.506	.779	.268	.391	.619	.535	.740	.680	.425	.166	15671
sampling-bin-vadd-normed	.562	.780	.410	.795	.725	.533	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	291
sampling-MPE-lint	.560	.800	.399	.750	.800	.556	.657	.630	.676	.238	.266	.483	.384	.621	.610	.304	.142	335
sampling-MPE-lint-lra	.562	.735	.381	.682	.675	.556	.636	.563	.621	.223	.263	.427	.313	.569	.618	.328	.156	265
sampling-MPE-lint-lra-normed	.562	.735	.381	.682	.725	.533	.636	.563	.621	.223	.263	.427	.313	.569	.618	.328	.156	236
sampling-MPE-lint-normed	.542	.800	.399	.750	.800	.600	.657	.630	.676	.238	.266	.483	.384	.621	.610	.304	.142	282
sampling-MPE-PCA	.619	.770	.426	.795	.750	.556	.741	.679	.792	.291	.344	.613	.525	.755	.713	.441	.184	351
sampling-MPE-PCA-lra	.619	.770	.426	.795	.750	.556	.741	.679	.792	.291	.344	.613	.525	.755	.713	.441	.184	435
sampling-MPE-PCA-lra-normed	.619	.805	.426	.795	.750	.556	.741	.679	.792	.291	.344	.613	.525	.755	.713	.441	.184	415
sampling-MPE-PCA-normed	.619	.770	.426	.795	.750	.556	.741	.679	.792	.291	.344	.613	.525	.755	.713	.441	.184	419
sampling-MPE-vadd	.562	.745	.416	.750	.725	.556	.715	.660	.728	.253	.309	.587	.493	.723	.651	.385	.161	232
sampling-MPE-vadd-lra	.575	.745	.412	.750	.725	.556	.715	.660	.728	.253	.309	.587	.493	.723	.651	.385	.161	236
sampling-MPE-vadd-lra-normed	.587	.745	.412	.750	.725	.556	.715	.660	.728	.253	.309	.587	.493	.723	.651	.385	.161	266
sampling-MPE-vadd-normed	.595	.745	.412	.750	.725	.556	.715	.660	.728	.253	.309	.587	.493	.723	.651	.385	.161	219
sampling-seq-lint	.483	.655	.395	.705	.650	.578	.672	.615	.566	.216	.299	.542	.457	.669	.623	.326	.131	147
sampling-seq-lint-lra	.398	.570	.223	.523	.675	.622	.584	.454	.746	.202	.328	.537	.438	.647	.594	.340	.151	14755
sampling-seq-lint-lra-normed	.398	.570	.223	.523	.675	.556	.584	.454	.746	.202	.328	.537	.438	.647	.594	.340	.151	15160
sampling-seq-lint-normed	.475	.650	.395	.705	.650	.489	.672	.615	.566	.216	.299	.542	.457	.669	.623	.326	.131	141
sampling-seq-PCA	.642	.835	.431	.795	.750	.556	.741	.680	.790	.291	.345	.614	.527	.755	.713	.441	.186	267
sampling-seq-PCA-lra	.453	.625	.282	.659	.500	.600	.051	-.036	-.024	.018	.036	-.052	-.005	-.120	.000	.000	-.011	24429
sampling-seq-PCA-lra-normed	.480	.620	.287	.636	.600	.622	.038	-.058	.222	-.042	-.038	.012	-.055	.037	.001	.001	-.002	32168
sampling-seq-PCA-normed	.612	.800	.431	.795	.750	.556	.741	.680	.790	.291	.345	.614	.527	.755	.713	.441	.186	302
sampling-seq-vadd	.562	.780	.426	.795	.725	.533	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	163
sampling-seq-vadd-lra	.560	.745	.327	.750	.625	.600	.685	.512	.785	.264	.362	.606	.516	.723	.661	.400	.158	15755
sampling-seq-vadd-lra-normed	.560	.745	.327	.750	.625	.689	.685	.512	.785	.264	.362	.606	.516	.723	.661	.400	.158	15241
sampling-seq-vadd-normed	.567	.780	.417	.795	.750	.622	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	201

Configuration	AP	BLESS	Battig	ESSLI_1a	ESSLI_2b	ESSLI_2c	MEN	MTurk	RG65	RW	SimLex999	WS353	WS353R	WS353S	Google	MSR	SemEval2012_2	(Average) Training Time (sec)
avg skip	.517	.685	.345	.706	.681	.553	.597	.519	.626	.220	.276	.487	.408	.603	.567	.329	.139	12483
avg sampling	.521	.689	.349	.712	.682	.551	.598	.521	.622	.221	.275	.486	.407	.601	.566	.329	.138	11484
baseline	.595	.820	.434	.795	.700	.578	.736	.694	.757	.299	.341	.611	.514	.754	.661	.440	.181	129014

Configuration	AP	BLESS	Battig	ESSLI_1a	ESSLI_2b	ESSLI_2c	MEN	MTurk	RG65	RW	SimLex999	WS353	WS353R	WS353S	Google	MSR	SemEval2012_2	time (sec)
skip-bin-lint	.465	.670	.367	.659	.575	.489	.614	.509	.631	.222	.249	.501	.443	.613	.596	.307	.109	17
skip-bin-lint-lra	.393	.540	.236	.523	.525	.578	.581	.456	.664	.198	.295	.503	.406	.609	.560	.297	.104	10966
skip-bin-lint-lra-normed	.393	.540	.236	.523	.525	.578	.581	.456	.664	.198	.295	.503	.406	.609	.560	.297	.104	11975
skip-bin-lint-normed	.465	.670	.367	.659	.575	.467	.614	.509	.631	.222	.249	.501	.443	.613	.596	.307	.109	12
skip-bin-PCA	.614	.800	.426	.795	.750	.533	.741	.680	.790	.291	.344	.613	.527	.757	.713	.441	.185	13
skip-bin-PCA-lra	.167	.220	.091	.432	.525	.400	-.030	.078	-.127	.032	.007	.074	.015	.049	.000	.000	-.006	11273
skip-bin-PCA-lra-normed	.167	.220	.091	.432	.550	.400	-.030	.078	-.127	.032	.007	.074	.015	.049	.000	.000	-.006	10860
skip-bin-PCA-normed	.604	.800	.428	.795	.750	.533	.741	.680	.790	.291	.344	.613	.527	.757	.713	.441	.185	11
skip-bin-vadd	.595	.780	.417	.795	.725	.533	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	2
skip-bin-vadd-lra	.540	.700	.319	.659	.625	.644	.702	.506	.779	.268	.391	.619	.535	.740	.680	.425	.166	11064
skip-bin-vadd-lra-normed	.535	.700	.319	.659	.625	.600	.702	.506	.779	.268	.391	.619	.535	.740	.680	.425	.166	11196
skip-bin-vadd-normed	.580	.780	.407	.795	.725	.533	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	2
skip-MPE-lint	.542	.800	.399	.750	.800	.600	.657	.630	.676	.238	.266	.483	.384	.621	.610	.304	.142	216
skip-MPE-lint-lra	.542	.800	.399	.750	.800	.556	.657	.630	.676	.238	.266	.483	.384	.621	.610	.304	.142	245
skip-MPE-lint-lra-normed	.542	.800	.399	.773	.800	.600	.657	.630	.676	.238	.266	.483	.384	.621	.610	.304	.142	265
skip-MPE-lint-normed	.542	.800	.399	.750	.800	.622	.657	.630	.676	.238	.266	.483	.384	.621	.610	.304	.142	242
skip-MPE-PCA	.602	.795	.437	.795	.750	.556	.741	.678	.791	.291	.346	.613	.525	.757	.714	.441	.184	95
skip-MPE-PCA-lra	.515	.715	.313	.705	.550	.644	.684	.509	.742	.243	.371	.620	.539	.719	.637	.370	.143	11125
skip-MPE-PCA-lra-normed	.515	.715	.313	.705	.575	.600	.684	.509	.742	.243	.371	.620	.539	.719	.637	.370	.143	11005
skip-MPE-PCA-normed	.624	.795	.437	.795	.750	.556	.741	.678	.791	.291	.346	.613	.525	.757	.714	.441	.184	95
skip-MPE-vadd	.577	.780	.407	.795	.725	.533	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	218
skip-MPE-vadd-lra	.582	.780	.418	.795	.725	.533	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	220
skip-MPE-vadd-lra-normed	.532	.780	.422	.795	.750	.533	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	218
skip-MPE-vadd-normed	.550	.780	.409	.795	.725	.533	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	244
skip-seq-lint	.483	.650	.395	.705	.650	.489	.672	.615	.566	.216	.299	.542	.457	.669	.623	.326	.131	125
skip-seq-lint-lra	.398	.570	.223	.523	.675	.556	.584	.454	.746	.202	.328	.537	.438	.647	.594	.340	.151	11917
skip-seq-lint-lra-normed	.398	.570	.223	.523	.675	.556	.584	.454	.746	.202	.328	.537	.438	.647	.594	.340	.151	16736
skip-seq-lint-normed	.475	.650	.395	.705	.650	.556	.672	.615	.566	.216	.299	.542	.457	.669	.623	.326	.131	172
skip-seq-PCA	.607	.815	.432	.795	.750	.556	.741	.681	.792	.291	.345	.613	.526	.755	.714	.442	.186	12
skip-seq-PCA-lra	.545	.700	.310	.682	.600	.556	.664	.514	.761	.236	.343	.598	.513	.708	.631	.373	.149	10273
skip-seq-PCA-lra-normed	.545	.700	.310	.614	.575	.644	.664	.514	.761	.236	.343	.598	.513	.708	.631	.373	.149	10382
skip-seq-PCA-normed	.627	.815	.432	.795	.750	.556	.741	.681	.792	.291	.345	.613	.526	.755	.714	.442	.186	12
skip-seq-vadd	.572	.780	.411	.795	.750	.533	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	127
skip-seq-vadd-lra	.560	.745	.327	.750	.625	.644	.685	.512	.785	.264	.362	.606	.516	.723	.661	.400	.158	11248
skip-seq-vadd-lra-normed	.560	.745	.327	.750	.625	.644	.685	.512	.785	.264	.362	.606	.516	.723	.661	.400	.158	11297
skip-seq-vadd-normed	.582	.780	.407	.795	.725	.533	.713	.661	.734	.256	.302	.561	.475	.700	.654	.378	.162	138

Configuration	AP	BLESS	Battig	ESSLI_1a	ESSLI_2b	ESSLI_2c	MEN	MTurk	RG65	RW	SimLex999	WS353	WS353R	WS353S	Google	MSR	SemEval2012_2	(Average) Training Time (sec)
avg skip	.517	.685	.345	.706	.681	.553	.597	.519	.626	.220	.276	.487	.408	.603	.567	.329	.139	12483
avg sampling	.521	.689	.349	.712	.682	.551	.598	.521	.622	.221	.275	.486	.407	.601	.566	.329	.138	11484
baseline	.595	.820	.434	.795	.700	.578	.736	.694	.757	.299	.341	.611	.514	.754	.661	.440	.181	129014