Ngram2vec: Learning Improved Word Representations from Ngram Co-occurrence Statistics

Zhe Zhao[1,2]
helloworld@ruc.edu.cn

Tao Liu[1,2]
tliu@ruc.edu.cn

Shen Li[3,4]
shen@mail.bnu.edu.cn

Bofang Li[1,2]
libofang@ruc.edu.cn

Xiaoyong Du[1,2]
duyong@ruc.edu.cn

[1] School of Information, Renmin University of China
[2] Key Laboratory of Data Engineering and Knowledge Engineering, MOE
[3] Institute of Chinese Information Processing, Beijing Normal University
[4] UltraPower-BNU Joint Laboratory for Artificial Intelligence, Beijing Normal University

Abstract

The existing word representation methods mostly limit their information source to word co-occurrence statistics. In this paper, we introduce ngrams into four representation methods: SGNS, GloVe, PPMI matrix, and its SVD factorization. Comprehensive experiments are conducted on word analogy and similarity tasks. The results show that improved word representations are learned from ngram co-occurrence statistics. We also demonstrate that the trained ngram representations are useful in many aspects such as finding antonyms and collocations. Besides, a novel approach of building co-occurrence matrix is proposed to alleviate the hardware burdens brought by ngrams.

1 Introduction

Recently, deep learning approaches have achieved state-of-the-art results on a range of NLP tasks. One of the most fundamental work in this field is word embedding, where low-dimensional word representations are learned from unlabeled corpora through neural models. The trained word embeddings reflect semantic and syntactic information of words. They are not only useful in revealing lexical semantics, but also used as inputs of various downstream tasks for better performance (Kim, 2014; Collobert et al., 2011; Pennington et al., 2014).

Most of the word embedding models are trained upon <*word, context*> pairs in the local window. Among them, word2vec gains its popularity by its amazing effectiveness and efficiency (Mikolov et al., 2013b,a). It achieves state-of-the-art results on a range of linguistic tasks with only a fraction of time compared with previous techniques. A challenger of word2vec is GloVe (Pennington et al., 2014). Instead of training on <*word, context*> pairs, GloVe directly utilizes word co-occurrence matrix. They claim that the change brings the improvement over word2vec on both accuracy and speed. Levy and Goldberg (2014b) further reveal that the attractive properties observed in word embeddings are not restricted to neural models such as word2vec and GloVe. They use traditional count-based method (PPMI matrix with hyper-parameter tuning) to represent words, and achieve comparable results with the above neural embedding models.

The above models limit their information source to word co-occurrence statistics (Levy et al., 2015). To learn improved word representations, we extend the information source from co-occurrence of ***'word-word'*** type to co-occurrence of ***'ngram-ngram'*** type. The idea of using ngrams is well supported by language modeling, one of the oldest problems studied in statistical NLP. In language models, co-occurrence of words and ngrams is used to predict the next word (Kneser and Ney, 1995; Katz, 1987). Actually, the idea of word embedding models roots in language models. They are closely related but are used for different purposes. Word embedding models aim at learning useful word representations instead of word prediction. Since ngram is a vital part in language modeling, we are inspired to integrate ngram statistical information into the recent word representation methods for better performance.

The idea of using ngrams is intuitive. However, there is still rare work using ngrams in recent representation methods. In this paper, we introduce ngrams into SGNS, GloVe, PPMI, and its SVD factorization. To evaluate the ngram-based models, comprehensive experiments are conducted on word analogy and similarity tasks. Experimental results demonstrate that the improved word representations are learned from ngram co-occurrence statistics. Besides that, we qualitatively evaluate the trained ngram representations. We show that they are able to reflect ngrams' meanings and syntactic patterns (e.g. 'be + past participle' pattern). The high-quality ngram representations are useful in many ways. For example, ngrams in negative form (e.g. 'not interesting') can be used for finding antonyms (e.g. 'boring').

Finally, a novel method is proposed to build ngram co-occurrence matrix. Our method reduces the disk I/O as much as possible, largely alleviating the costs brought by ngrams. We unify different representation methods in a pipeline. The source code is organized as ***ngram2vec*** toolkit and released at `https://github.com/zhezhaoa/ngram2vec`.

2 Related Work

SGNS, GloVe, PPMI, and its SVD factorization are used as baselines. The information used by them does not go beyond word co-occurrence statistics. However, their approaches to using the information are different. We review these methods in the following 3 sections. In section 2.4, we revisit the use of ngrams in the deep learning context.

2.1 SGNS

Skip-gram with negative sampling (SGNS) is a model in word2vec toolkit (Mikolov et al., 2013b,a). Its training procedure follows the majority of neural embedding models (Bengio et al., 2003): (1) *Scan the corpus and use <word, context> pairs in the local window as training samples.* (2) *Train the models to make words useful for predicting contexts (or in reverse).* The details of SGNS is discussed in Section 3.1. Compared to previous neural embedding models, SGNS speeds up the training process, reducing the training time from days or weeks to hours. Also, the trained embeddings possess attractive properties. They are able to reflect relations between two words accurately, which is evaluated by a fancy task called word analogy.

Due to the above advantages, many models are proposed on the basis of SGNS. For example, Faruqui et al. (2015) introduce knowledge in lexical resources into the models in word2vec. Zhao et al. (2016) extend the contexts from the local window to the entire documents. Li et al. (2015) use supervised information to guide the training. Dependency parse-tree is used for defining context in (Levy and Goldberg, 2014a). LSTM is used for modeling context in (Melamud et al., 2016) Sub-word information is considered in (Sun et al., 2016; Soricut and Och, 2015).

2.2 GloVe

Different from typical neural embedding models which are trained on <*word, context*> pairs, GloVe learns word representation on the basis of co-occurrence matrix (Pennington et al., 2014). GloVe breaks traditional 'words predict contexts' paradigm. Its objective is to reconstruct non-zero values in the matrix. The direct use of matrix is reported to bring improved results and higher speed. However, there is still dispute about the advantages of GloVe over word2vec (Levy et al., 2015; Schnabel et al., 2015). GloVe and other embedding models are essentially based on word co-occurrence statistics of the corpus. The <*word, context*> pairs and co-occurrence matrix can be converted to each other. Suzuki and Nagata (2015) try to unify GloVe and SGNS in one framework.

2.3 PPMI & SVD

When we are satisfied with the huge promotions achieved by embedding models on linguistic tasks, a natural question is raised: where the superiorities come from. One conjecture is that it's due to the neural networks. However, Levy and Goldberg (2014c) reveal that SGNS is just factoring PMI matrix implicitly. Also, Levy and Goldberg (2014b) show that positive PMI (PPMI) matrix still rivals the newly proposed embedding models on a range of linguistic tasks. Properties like word analogy are not restricted to neural models. To obtain dense word representations from PPMI matrix, we factorize PPMI matrix with SVD, a classic dimensionality reduction method for learning low-dimensional vectors from sparse matrix (Deerwester et al., 1990).

2.4 Ngram in Deep Learning

In the deep learning literature, ngram has shown to be useful in generating text representations. Recently, convolutional neural networks (CNNs) are reported to perform well on a range of NLP tasks (Blunsom et al., 2014; Hu et al., 2014; Severyn and Moschitti, 2015). CNNs are essentially using n-gram information to represent texts. They use 1-D convolutional layers to extract ngram features and the distinct features are selected by max-pooling layers. In (Li et al., 2016), ngram embedding is introduced into Paragraph Vector model, where text embedding is trained to be useful to predict n-grams in the text. In the word embedding literature, a related work is done by Melamud et al. (2014), where word embedding models are used as baselines. They propose to use ngram language models to model the context, showing the effectiveness of ngrams on similarity tasks. Another work that is related to ngram is from Mikolov et al. (2013b), where phrases are embedded into vectors. It should be noted that phrases are different from ngrams. Phrases have clear semantics and the number of phrases is much less than the number of ngrams. Using phrase embedding has little impact on word embedding's quality.

3 Model

In this section, we introduce ngrams into SGNS, GloVe, PPMI, and SVD. Section 3.1 reviews the SGNS. Section 3.2 and 3.3 show the details of introducing ngrams into SGNS. In section 3.4, we show the way of using ngrams in GloVe, PPMI, and SVD, and propose a novel way of building n-gram co-occurrence matrix.

3.1 Word Predicts Word: the Revisit of SGNS

First we establish some notations. The raw input is a corpus $T = \{w_1, w_2, ..., w_{|T|}\}$. Let W and C denote word and context vocabularies. θ is the parameters to be optimized. SGNS's parameters involve two parts: word embedding matrix and context embedding matrix. With embedding $\vec{w} \in R^d$, the total number of parameters is $(|W|+|C|)*d$.

The SGNS's objective is to maximize the conditional probabilities of contexts given center words:

$$\sum_{t=1}^{|T|} \left[\sum_{c \in C(w_t)} log\, p(c|w_t; \theta) \right] \quad (1)$$

where $C(w_t) = \{w_i, t - win \leq i \leq t + win \, and \, i \neq t\}$ and *win* denotes the window size. As illustrated in figure 1, the center word 'written' predicts its surrounding words 'Potter', 'is', 'by', and

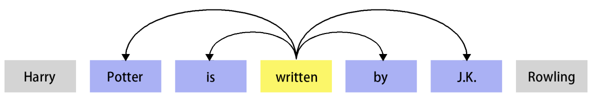

Figure 1: Illustration of 'word predicts word'.

'J.K.'. In this paper, negative sampling (Mikolov et al., 2013b) is used to approximate the conditional probability:

$$p(c|w) = \sigma(\vec{w}^T \vec{c}) \prod_{j=1}^{k} E_{c_j \sim P_n(C)} \sigma(-\vec{w}^T \vec{c_j}) \quad (2)$$

where σ is sigmoid function. k samples (from c_1 to c_k) are drawn from context distribution raised to the power of n.

3.2 Word Predicts Ngram

In this section, we introduce ngrams into context vocabulary. We treat each ngram as a normal word and give it a unique embedding. During the training, the center word should not only predict its surrounding words, but also predict its surrounding n-grams. As shown in figure 2, center word 'written' predicts the bigrams in the local window such as 'by J.K.'. The objective of 'word predicts ngram' is similar with the original SGNS. The only difference is the definition of the *C(w)*. In ngram case, *C(w)* is formally defined as follows:

$$C(w_t) = \bigcup_{n=1}^{N} \{w_{i:i+n} | w_{i:i+n} \; is \; not \; w_t \; AND \\ t - win \leq i \leq t + win - n + 1\} \quad (3)$$

where $w_{i:i+n}$ denotes the ngram $w_i w_{i+1} ... w_{i+n-1}$ and N is the order of context ngram. Two points need to be noticed from the above definition. The first is how to determine the distance between center word and context ngram. In this paper, we use the distance between the word and the ngram's far-end word. As show in figure 2, the distance between 'written' and 'Harry Potter' is 3. As a result, 'Harry Potter' is not included in the center word's context. This distance definition ensures that the ngram models don't use the information beyond the pre-specified window, which guarantees fair comparisons with baselines. Another point is whether the overlap of word and n-gram is allowed or not. In the overlap situation, ngrams are used as context even they contain the center word. As the example in figure 2 shows,

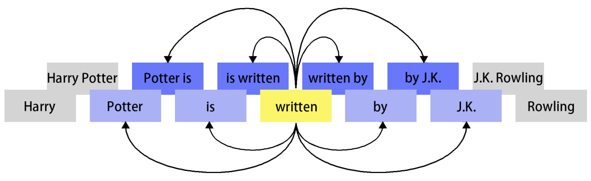

Figure 2: Illustration of 'word predicts ngram'.

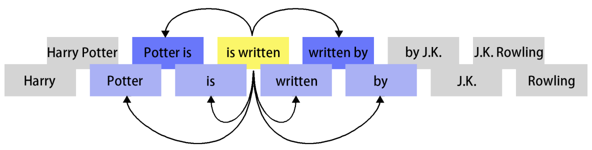

Figure 3: Illustration of 'ngram predicts ngram'.

ngram 'is written' and 'written by' are predicted by the center word 'written'. In the non-overlap case, these ngrams are excluded. The properties of word embeddings are different when overlap is allowed or not, which will be discussed in experiments section.

3.3 Ngram Predicts Ngram

We further extend the model to introduce ngrams into center word vocabulary. During the training, center ngrams (including words) predict their surrounding ngrams. As shown in figure 3, center bigram 'is written' predicts its surrounding words and bigrams. The objective of 'ngram predicts ngram' is as follows:

$$\sum_{t=1}^{|T|} \sum_{n_w=1}^{N_w} \left[\sum_{c \in C(w_{t:t+n_w})} log\, p(c|w_{t:t+n_w};\theta) \right] \quad (4)$$

where N_w is the order of center ngram. The definition of $C(w_{t:t+n_w})$ is as follows:

$$\bigcup_{n_c=1}^{N_c} \{w_{i:i+n_c} | w_{i:i+n_c} \ is\ not\ w_{t:t+n_w}\ AND \\ t - win + n_w - 1 \leq i \leq t + win - n_c + 1\} \quad (5)$$

where N_c is the order of context ngram. To this end, the word embeddings are not only affected by the ngrams in the context, but also indirectly affected by co-occurrence statistics of 'ngram-ngram' type in the corpus.

SGNS is proven to be equivalent with factorizing pointwise mutual information (PMI) matrix (Levy and Goldberg, 2014c). Following their work, we can easily show that models in section 3.2 and 3.3 are implicitly factoring PMI matrix of 'word-ngram' and 'ngram-ngram' type. In the next section, we will discuss the content of introducing ngrams into positive PMI (PPMI) matrix.

3.4 Co-occurrence Matrix Construction

Introducing ngrams into GloVe, PPMI, and SVD is straightforward: the only change is to replace word co-occurrence matrices with ngram ones. In the above three sections, we have discussed the way of taking out *<word(ngram), word(ngram)>* pairs from a corpus. Afterwards, we build the co-occurrence matrix upon these pairs. The rest steps are identical with the original baseline models.

Win	Type	#Pairs
2	uni_uni	0.36B
	uni_bi	1.14B
	uni_tri	1.40B
	bi_bi	2.78B
	bi_tri	3.65B
5	uni_uni	0.91B
	uni_bi	2.79B
	uni_tri	3.81B
	bi_bi	7.97B

Table 1: The number of pairs at different settings. The type column lists the order of ngrams considered in center word/context vocabularies. For example, uni_bi denotes that center word vocabulary contains unigrams (words) and context vocabulary contains both unigrams and bigrams. The setting of other hyper-parameters is discussed in Section 4.2.

However, building the co-occurrence matrix is not an easy task as it apparently looks like. The introduction of ngrams brings huge burdens on the hardware. The matrix construction cost is closely related to the number of pairs (**#Pairs**). Table 1 shows the statistics of pairs extracted from corpus wiki2010 [1]. We can observe that **#Pairs** is huge when ngrams are considered.

To speed up the process of building ngram co-occurrence matrix, we take advantages of 'mixture' strategy (Pennington et al., 2014) and 'stripes' strategy (Dyer et al., 2008; Lin, 2008). The two strategies optimize the process in different aspects. Computational cost is reduced significantly when they are used together.

[1] http://nlp.stanford.edu/data/WestburyLab.wikicorp.201004.txt.bz2

When words (or ngrams) are sorted in descending order by frequency, the co-occurrence matrix's top-left corner is dense while the rest part is sparse. Based on this observation, the 'mixture' of two data structures are used for storing matrix. Elements in the top-left corner are stored in a 2D array, which stays in memory. The rest of the elements are stored in the form of <*ngram, H*>, where *H*<*context, count*> is an associative array recording the number of times the *ngram* and *context* co-occurs ('stripes' strategy). Compared with storing <ngram, context> pairs explicitly, the 'stripes' strategy provides more opportunities to aggregate pairs outside of the top-left corner.

Algorithm 1 shows the way of using the 'mixture' and 'stripes' strategies together. In the first stage, pairs are stored in different data structures according to *topLeft* function. Intermediate results are written to temporary files when memory is full. In the second stage, we merge these sorted temporary files to generate co-occurrence matrix. The *getSmallest* function takes out the pair <*ngram, H*> with the smallest *key* from temporary files. In practice, algorithm 1 is efficient. Instead of using computer clusters (Lin, 2008), we can build the matrix of 'bi_bi' type even in a laptop. It only requires 12GB to store temporary files (win=2, subsampling=0, memory size=4GB), which is much smaller than the implementations in (Pennington et al., 2014; Levy et al., 2015) . More detailed analysis about these strategies can be found in the ***ngram2vec*** toolkit.

4 Experiments

4.1 Datasets

The tasks used in this paper is the same with the work of Levy et al. (2015), including six similarity and two analogy datasets. In similarity task, a scalar (e.g. a score from 0 to 10) is used to measure the relation between the two words. For example, in a similarity dataset, the 'train, car' pair is given the score of 6.31. A problem of similarity task is that scalar only reflects the strength of the relation, while the type of relation is totally ignored (Schnabel et al., 2015).

Due to the deficiency of similarity task, analogy task is widely used as benchmark recently for evaluation of word embedding models. To answer analogy questions, relations between the two words are reflected by a vector, which is usually obtained by the difference between word

Algorithm 1: An algorithm for building n-gram co-occurrence matrix

```
Input  : Pairs $P$, Sorted vocabulary $V$
Output: Sorted and aggregated pairs
The 2D array $A[\,][\,]$;
The dictionary $D < ngram, H >$;
The temporary files array $tfs[\,]$; $fid$=1;
for pair $p < n, c >$ in $P$ do
    if $topLeft(n, c) == 1$ then
        $A[getId(n)][getId(c)]$ += 1;
    else
        $D\{n\}\{c\}$ += 1;
        if Memory is full or $P$ is empty then
            Sort $D$ by key (ngram);
            Write $D$ to $tfs[fid]$;
            $fid$ += 1;
        end
    end
end
Write $A$ to $tfs[0]$ in the form of $< ngram, H >$;
$old = getSmallest(tfs)$ ;
while !(All files in $tfs$ are empty) do
    $new = getSmallest(tfs)$ ;
    if $old.ngram == new.ngram$ then
        $old$ =
        $< old.ngram, merge(old.H, new.H) >$;
    else
        Write $old$ to disk;
        $old = new$
    end
end
```

embeddings. Different from a scalar, the vector provides more accurate descriptions of relations. For example, capital-country relation is encoded in *vec(Athens)-vec(Greece)*, *vec(Tokyo)-vec(Japan)* and so on. More concretely, the questions in the analogy task are in the form of 'a is to b as c is to d'. 'd' is an unknown word in the test phase. To correctly answer the questions, the models should embed the two relations, *vec(a)-vec(b)* and *vec(c)-vec(d)*, into similar positions in the space. Following the work of Levy and Goldberg (2014b), both additive (add) and multiplicative (mul) functions are used for finding word 'd'. The latter one is more suitable for sparse representation in practice.

4.2 Pipeline and Hyper-parameter Setting

We implement SGNS, GloVe, PPMI, and SVD in a pipeline, allowing the reuse of code and intermediate results. Figure 4 illustrates the overview of the pipeline. Firstly, <*word(ngram), word(ngram)*> pairs are extracted from the corpus as the input of SGNS. Afterwards, we build the co-occurrence matrix upon the pairs. GloVe and PPMI learn word representations on the basis of co-occurrence

Win	Type		Google Tot. / Sem. / Syn.		MSR	
			Add	Mul	Add	Mul
2	uni_uni		.579 / .543 / .608	.597 / .561 / .627	.513	.533
	overlap	uni_bi	.587 / .651 / .533	.626 / .681 / .580	.473	.508
		uni_tri	.505 / .615 / .414	.553 / .657 / .466	.358	.396
		bi_bi	**.664** / **.739** / .602	**.680** / **.739** / .631	.547	.575
		bi_tri	.572 / .695 / .470	.601 / .713 / .508	.416	.447
	non-overlap	uni_bi	.610 / .558 / .653	.633 / .581 / .676	.568	.595
		bi_bi	.644 / .607 / **.674**	.659 / .613 / **.696**	**.590**	**.616**
5	uni_uni		.653 / .669 / .639	.668 / .678 / .660	.511	.535
	overlap	uni_bi	.696 / .745 / .655	.714 / .752 / .683	.518	.542
		uni_tri	.679 / .738 / .630	.699 / .750 / .657	.542	.549
		bi_bi	.704 / **.764** / .654	.718 / **.764** / .681	.537	.560
	non-overlap	uni_bi	.696 / .722 / .675	.716 / .731 / .703	.549	.579
		uni_tri	.687 / .711 / .668	.705 / .717 / .696	.542	.574
		bi_bi	**.712** / .745 / **.684**	**.725** / .742 / **.710**	**.569**	**.607**

Table 2: Performance of (ngram) SGNS on analogy datasets.

Win	Type	Sim.	Rel.	Bruni	Radinsky	Luong	Hill
2	uni_uni	.745	.586	.713	.635	.387	.419
	uni_bi	.739	**.600**	.698	.627	.395	**.429**
	uni_tri	.700	.535	.658	.591	.380	.415
	bi_bi	**.757**	.574	**.724**	**.644**	**.408**	.407
	bi_tri	.724	.564	.669	.605	.403	.412
5	uni_uni	.789	.648	.756	.652	.407	.401
	uni_bi	.794	.681	.752	.653	.437	.431
	uni_tri	.783	.673	.743	.652	.432	**.436**
	bi_bi	**.816**	**.703**	**.760**	**.671**	**.446**	.421

Table 3: Performance of (ngram) SGNS on similarity datasets.

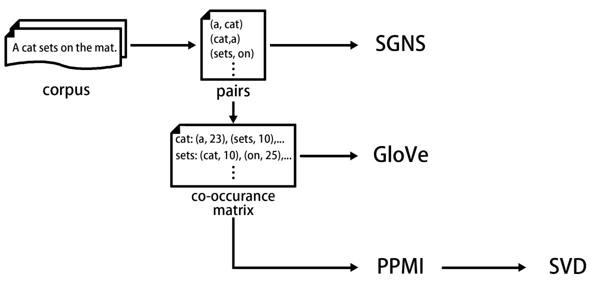

Figure 4: The pipeline.

matrix. SVD factorizes the PPMI matrix to obtain low-dimensional representation.

Most hyper-parameters come from 'corpus to pairs' part and four representation models. 'corpus to pairs' part determines the source of information for the subsequent models and its hyper-parameter setting is as follows: low-frequency words (n-grams) are removed with a threshold of 10. High-frequency words (ngrams) are removed with sub-sampling at the degree of 1e-5 [2]. Window size is set to 2 and 5. Clean strategy (Levy et al., 2015) is used to ensure no information beyond pre-specified window is included. Overlap setting is used in default. For hyper-parameters of four representation models, we use the embeddings of 300 dimensions in dense representations. SGNS is trained by 3 iterations. The rest strictly follow the baseline models [3]. We consider unigrams (words), bigrams, and trigrams in this work. The implementation of higher-order models and their results will be released with ***ngram2vec*** toolkit.

4.3 Ngrams on SGNS

SGNS is a popular word embedding model. Even compared with its challengers such as GloVe, S-GNS is reported to have more robust performance with faster training speed (Levy et al., 2015). Table 2 lists the results on analogy datasets. We can observe that the introduction of bigrams provides significant improvements at different hyper-parameter settings. The SGNS of 'bi_bi' type provides the highest results. It is very effective on capturing semantic information (Google semantic). Around 10 percent improvements are wit-

[2]Sub-sampling is not used in GloVe, which follows its original setting.

[3]http://bitbucket.org/omerlevy/hyperwords for SGNS, PPMI and SVD; http://nlp.stanford.edu/projects/glove/ for GloVe.

Win	Type	Google		MSR		Sim.	Rel.	Bruni	Radinsky	Luong	Hill
		Add	Mul	Add	Mul						
2	uni_uni	.403 / .441 / .372	.614 / .725 / .522	.235	.419	.709	.593	.705	.603	.293	.385
	uni_bi	.403 / .550 / .281	.733 / .854 / .632	.241	.572	.731	.581	.721	.627	.341	.394
5	uni_uni	.423 / .505 / .355	.580 / .740 / .447	.198	.339	.721	.619	.712	.619	.252	.341
	uni_bi	.453 / .590 / .338	.730 / .841 / .637	.281	.579	.707	.547	.696	.619	.296	.378

Table 4: Performance of (ngram) PPMI on analogy and similarity datasets.

Win	Type	Google		MSR		Sim.	Rel.	Bruni	Radinsky	Luong	Hill
		Add	Mul	Add	Mul						
2	uni_uni	.535 / .599 / .482	.540 / .610 / .481	.444	.445	.681	.529	.698	.608	.381	.351
	uni_bi	.543 / .601 / .493	.549 / .612 / .496	.464	.472	.686	.545	.695	.631	.389	.352
5	uni_uni	.625 / .689 / .572	.626 / .696 / .568	.476	.490	.747	.600	.735	.657	.389	.347
	uni_bi	.631 / .699 / .575	.633 / .703 / .574	.477	.504	.752	.610	.737	.631	.395	.342

Table 5: Performance of (ngram) GloVe on analogy and similarity datasets.

Win	Type	Google		MSR		Sim.	Rel.	Bruni	Radinsky	Luong	Hill
		Add	Mul	Add	Mul						
2	uni_uni	.419 / .388 / .446	.439 / .394 / .477	.321	.353	.714	.593	.712	.625	.410	.344
	uni_bi	.387 / .322 / .440	.410 / .327 / .479	.372	.402	.739	.546	.688	.636	.427	.347
5	uni_uni	.433 / .426 / .439	.460 / .463 / .458	.290	.321	.752	.633	.731	.623	.411	.326
	uni_bi	.410 / .340 / .468	.446 / .365 / .513	.374	.416	.751	.559	.698	.639	.426	.363

Table 6: Performance of (ngram) SVD on analogy and similarity datasets.

nessed on semantic questions compared with uni_uni baseline. For syntactic questions (Google syntactic and MSR datasets), around 5 percent improvements are obtained on average.

The effect of overlap is large on analogy datasets. Semantic questions prefer the overlap setting. Around 10 and 3 percent improvements are witnessed compared with non-overlap setting at the window size of 2 and 5. While in syntactic case, non-overlap setting performs better by a margin of around 5 percent.

The introduction of trigrams deteriorates the models' performance on analogy datasets (especially at the window size of 2). It is probably because that trigram is sparse on wiki2010, a relatively small corpus with 1 billion tokens. We conjecture that high order ngrams are more suitable for large corpora and will report the results in our future work. It should be noticed that trigram is not included in vocabulary in non-overlap case at the window size of 2. The shortest distance between a word and a trigram is 3, which exceeds the window size.

Table 3 illustrates the SGNS's performance on similarity task. The conclusion is similar with the case in analogy datasets. The use of bigrams is effective while the introduction of trigrams deteriorates the performance in most cases. In general, the bigrams bring significant improvements over SGNS on a range of linguistic tasks. It is generally known that ngram is a vital part in traditional language modeling problem. Results in table 2 and 3 confirm the effectiveness of ngrams again on SGNS, a more advanced word embedding model.

4.4 Ngrams on PPMI, GloVe, SVD

In this section, we only report the results of models of 'uni_uni' and 'uni_bi' types. Using higher order co-occurrence statistics brings immense costs (especially at the window size of 5). Levy and Goldberg (2014b) demonstrate that traditional count-based models can still achieve competitive results on many linguistic tasks, challenging the dominance of neural embedding models. Table 4 lists the results of PPMI matrix on analogy and similarity datasets. PPMI prefers Multiplicative (Mul) evalution. To this end, we focus on analyzing the results on Mul columns. When bigrams are used, significant improvements are witnessed on analogy task. On Google dataset, bigrams bring over 10 percent increase on the total accuracies. At the window size of 2, the accuracy in semantic questions even reaches 0.854, which is the state-of-the-art result to the best of our knowledge. On MSR dataset, around 20 percent improvements are achieved. The use of bigrams does not always bring improvements on similarity datasets. PPMI matrix of 'uni_bi' type improves the results on 5 datasets at the window size of 2. At the window size of 5, using bigrams only improves the results on 2 datasets.

Table 5 and 6 list GloVe and SVD's results. For GloVe, consistent (but minor) improvements are achieved on analogy task with the introduction of bigrams. On similarity datasets, improvements are witnessed on most cases. For SVD, bigrams sometimes lead to worse results in both analogy and similarity tasks. In general, significant improvements are not witnessed on GloVe and SVD. Our preliminary conjecture is that the default hyper-parameter setting should be blamed. We strictly follow the hyper-parameters used in baseline models, making no adjustments to cater to the introduction of ngrams. Besides that, some common techniques such as dynamic window, decreasing weighting function, dirty sub-sampling are discarded. The relationships between ngrams and various hyper-parameters require further exploration. Though trivial, it may lead to much better results and give researchers better understanding of different representation methods. That will be the focus of our future work.

4.5 Qualitative Evaluations of Ngram Embedding

In this section, we analyze the properties of ngram embeddings trained by SGNS of 'bi_bi' type. Ideally, the trained ngram embeddings should reflect ngrams' semantic meanings. For example, *vec(wasn't able)* should be close to *vec(unable)*. *vec(is written)* should be close to *vec(write)* and *vec(book)*. Also, the trained ngram embeddings should preserve ngrams' syntactic patterns. For example, 'was written' is in the form of 'be + past participle' and the nearest neighbors should possess similar patterns, such as 'is written' and 'was transcribed'.

Table 7 lists the target ngrams and their top nearest neighbours. We divide the target ngrams into six groups according to their patterns. We can observe that the returned words and ngrams are very intuitive. As might be expected, synonyms of the target ngrams are returned in top positions (e.g. 'give off' and 'emit'; 'heavy rain' and 'downpours'). From the results of the first group, it can be observed that bigram in negative form 'not X' is useful for finding the antonym of word 'X'. Besides that, the trained ngram embeddings also preserve some common sense. For example, the returned result of 'highest mountain' is a list of mountain names (with a few exceptions such as 'unclimbed'). In terms of syntactic patterns, we can observe that in most cases, the returned ngrams are in the similar form with target ngrams. In general, the trained embeddings basically reflect semantic meanings and syntactic patterns of ngrams.

With high-quality ngram embeddings, we have the opportunity to do more interesting things in our future work. For example, we will construct a antonym dataset to evaluate ngram embeddings systematically. Besides that, we will find more scenarios for using ngram embeddings. In our view, ngram embeddings have potential to be used in many NLP tasks. For example, Johnson and Zhang (2015) use one-hot ngram representation as the input of CNN. Li et al. (2016) use ngram embeddings to represent texts. Intuitively, initializing these models with pre-trained ngram embeddings may further improve the accuracies.

5 Conclusion

We introduce ngrams into four representation methods. The experimental results demonstrate ngrams' effectiveness for learning improved word representations. In addition, we find that the trained ngram embeddings are able to reflect their semantic meanings and syntactic patterns. To alleviate the costs brought by ngrams, we propose a novel way of building co-occurrence matrix, enabling the ngram-based models to run on cheap hardware.

Acknowledgments

This work is supported by National Natural Science Foundation of China (Grant No. 61472428 and No. 71531012), the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University of China No. 14XNLQ06. This work is partially supported by ECNU-RUC-InfoSys Joint Data Science Lab and a gift from Tencent.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Phil Blunsom, Edward Grefenstette, and Nal Kalchbrenner. 2014. A convolutional neural network for modelling sentences. In *Proceedings of ACL 2014*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Pattern	Target	Word	Bigram
Negative Form	**wasn't able**	**unable(.745), couldn't(.723), didn't(.680)**	**was unable(.832), didn't manage(.799)**
	don't need	**don't(.773), dont(.751), needn't(.715)**	**dont need(.790), don't have(.785), dont want(.769)**
	not enough	**enough(.708), insufficient(.701), sufficient(.629)**	**not sufficient(.750), wasn't enough(.729)**
Adj. Modifier	**heavy rain**	**torrential(.844), downpours(.780), rain(.766)**	**torrential rain(.829), heavy rainfall(.799)**
	strong supporter	**supporter(.828), proponent(.733), admirer(.602)**	**staunch supporter(.870), vocal supporter(.810)**
	high quality	**high-quality(.867), quality(.744)**	**good quality(.813), top quality(.751)**
Passive Voice	**was written**	**written(.793), penned(.675), co-written(.629)**	**were written(.785), is written(.744), written by(.739)**
	was sent	**sent(.844), dispatched(.661), went(.630)**	**then sent(.779), later sent(.776), was dispatched(.774)**
	was pulled	**pulled(.730), yanked(.629), limped(.593)**	**were pulled(.706), pulled from(.691), was ripped(.682)**
Perfect Tense	**has achieved**	**achieved(.683), achieves(.680), achieving(.625)**	**has attained(.775), has enjoyed(.741), has gained(.733)**
	has impacted	**interconnectedness(.679), pervade(.676)**	**have impacted(.838), is affecting(.773), have shaped(.772)**
	has published	**authored(.722), publishes(.705), coauthored(.791)**	**has authored(.852), has edited(.795), has written(.791)**
Phrasal Verb	**give off**	**exude(.796), fluoresce(.789), emit(.754)**	**gave off(.837), giving off(.820), and emit(.816)**
	make up	**comprise(.726), constitute(.616), make(.541)**	**makes up(.705), making up(.702), comprise the(.672)**
	picked up	**picked(.870), snagged(.544), scooped(.538)**	**later picked(.712), and picked(.682), then picked(.681)**
Common Sense	**highest mountain**	**muztagh(.669), prokletije(.664), cadair(.658)**	**highest peak(.873), tallest mountain(.857)**
	avian influenza	**h5n1(.870), zoonotic(.812), adenovirus(.806)**	**avian flu(.885), the h5n1(.870), flu virus(.868)**
	computer vision	**human-computer(.789), holography(.767)**	**image processing(.850), object recognition(.818)**

Table 7: Target bigrams and their nearest neighbours associated with similarity scores.

Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. 1990. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407.

Christopher Dyer, Aaron Cordova, Alex Mont, and Jimmy Lin. 2008. Fast, easy, and cheap: Construction of statistical machine translation models with mapreduce. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 199–207.

Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard H. Hovy, and Noah A. Smith. 2015. Retrofitting word vectors to semantic lexicons. In *Proceedings of NAACL 2015*, pages 1606–1615.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Proceedings of NIPS 2014*, pages 2042–2050.

Rie Johnson and Tong Zhang. 2015. Effective use of word order for text categorization with convolutional neural networks. In *Proceedings of NAACL 2015*, pages 103–112.

Slava M. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. Acoustics, Speech, and Signal Processing*, 35(3):400–401.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of EMNLP 2014*, pages 1746–1751.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of ICASSP 1995*, pages 181–184.

Omer Levy and Yoav Goldberg. 2014a. Dependency-based word embeddings. In *ACL (2)*, pages 302–308.

Omer Levy and Yoav Goldberg. 2014b. Linguistic regularities in sparse and explicit word representations. In *Proceedings of CoNLL 2014*, pages 171–180.

Omer Levy and Yoav Goldberg. 2014c. Neural word embedding as implicit matrix factorization. In *Proceedings of NIPS 2014*, pages 2177–2185.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *TACL*, 3:211–225.

Bofang Li, Zhe Zhao, Tao Liu, Puwei Wang, and Xiaoyong Du. 2016. Weighted neural bag-of-n-grams model: New baselines for text classification. In *Proceedings of COLING 2016*, pages 1591–1600.

Yitan Li, Linli Xu, Fei Tian, Liang Jiang, Xiaowei Zhong, and Enhong Chen. 2015. Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In *Proceedings of IJCAI 2015*, pages 3650–3656.

Jimmy J. Lin. 2008. Scalable language processing algorithms for the masses: A case study in computing word co-occurrence matrices with mapreduce. In *Proceedings of EMNLP 2008*, pages 419–428.

Oren Melamud, Ido Dagan, Jacob Goldberger, Idan Szpektor, and Deniz Yuret. 2014. Probabilistic modeling of joint-context in distributional similarity. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning, CoNLL 2014, Baltimore, Maryland, USA, June 26-27, 2014*, pages 181–190.

Oren Melamud, Jacob Goldberger, and Ido Dagan. 2016. context2vec: Learning generic context embedding with bidirectional LSTM. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, pages 51–61.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS 2013*, pages 3111–3119.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of EMNLP 2014*, pages 1532–1543.

Tobias Schnabel, Igor Labutov, David M. Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings. In *Proceedings of EMNLP 2015*, pages 298–307.

Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of SIGIR 2015*, pages 373–382.

Radu Soricut and Franz Josef Och. 2015. Unsupervised morphology induction using word embeddings. In *Proceedings of NAACL 2015*, pages 1627–1637.

Fei Sun, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. 2016. Inside out: Two jointly predictive models for word representations and phrase representations. In *Proceedings of AAAI 2016*, pages 2821–2827.

Jun Suzuki and Masaaki Nagata. 2015. A unified learning framework of skip-grams and global vectors. In *Proceedings of ACL 2015, Volume 2: Short Papers*, pages 186–191.

Zhe Zhao, Tao Liu, Bofang Li, and Xiaoyong Du. 2016. Cluster-driven model for improved word and text embedding. In *Proceedings of ECAI 2016*, pages 99–106.