Batch IS NOT Heavy: Learning Word Representations From All Samples

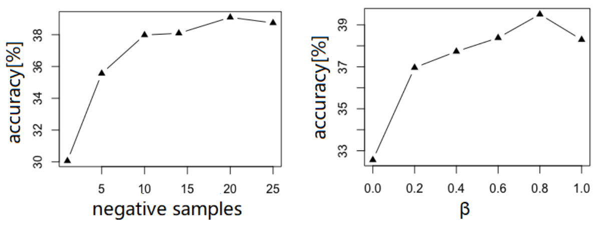
*Xin Xin[1], *Fajie Yuan[1], Xiangnan He[2], Joemon M.Jose[1]
[1]School of Computing Science, University of Glasgow, UK
[2]School of Computing, National University of Singapore
{x.xin.1,f.yuan.1}@research.gla.ac.uk, xiangnanhe@gmail.com

Abstract

Stochastic Gradient Descent (SGD) with negative sampling is the most prevalent approach to learn word representations. However, it is known that sampling methods are biased especially when the sampling distribution deviates from the true data distribution. Besides, SGD suffers from dramatic fluctuation due to the one-sample learning scheme. In this work, we propose AllVec that uses batch gradient learning to generate word representations from *all* training samples. Remarkably, the time complexity of AllVec remains at the same level as SGD, being determined by the number of positive samples rather than all samples. We evaluate AllVec on several benchmark tasks. Experiments show that AllVec outperforms sampling-based SGD methods with comparable efficiency, especially for small training corpora.

1 Introduction

Representing words using dense and real-valued vectors, aka word embeddings, has become the cornerstone for many natural language processing (NLP) tasks, such as document classification (Sebastiani, 2002), parsing (Huang et al., 2012), discourse relation recognition (Lei et al., 2017) and named entity recognition (Turian et al., 2010). Word embeddings can be learned by optimizing that words occurring in similar contexts have similar embeddings, i.e. the well-known distributional hypothesis (Harris, 1954). A representative method is skip-gram (SG) (Mikolov et al., 2013a,b), which realizes the hypothesis using a shallow neural network model. The other family of methods is count-based, such as GloVe (Pennington et al., 2014) and LexVec (Salle et al., 2016a,b), which exploit low-rank models such as matrix factorization (MF) to learn embeddings by reconstructing the word co-occurrence statistics.



(a) (b)

Figure 1: Impact of different settings of negative sampling on skip-gram for the word analogy task on Text8. Clearly, the accuracy depends largely on (a) the sampling size of negative words, and (b) the sampling distribution ($\beta = 0$ means the uniform distribution and $\beta = 1$ means the word frequency distribution).

By far, most state-of-the-art embedding methods rely on SGD and negative sampling for optimization. However, the performance of SGD is highly sensitive to the sampling distribution and the number of negative samples (Chen et al., 2018; Yuan et al., 2016), as shown in Figure 1. Essentially, sampling is biased, making it difficult to converge to the same loss with all examples, regardless of how many update steps have been taken. Moreover, SGD exhibits dramatic fluctuation and suffers from overshooting on local minimums (Ruder, 2016). These drawbacks of SGD can be attributed to its one-sample learning scheme, which updates parameters based on one training sample in each step.

To address the above-mentioned limitations of SGD, a natural solution is to perform exact (full) batch learning. In contrast to SGD, batch learning does not involve any sampling procedure and computes the gradient over all training samples. As such, it can easily converge to a better optimum in a more stable way. Nevertheless, a well-known

*The first two authors contributed equally to this paper and share the first-authorship.

difficulty in applying full batch learning lies in the expensive computational cost for large-scale data. Taking the word embedding learning as an example, if the vocabulary size is $|V|$, then evaluating the loss function and computing the full gradient takes $O(|V|^2k)$ time, where k is the embedding size. This high complexity is unaffordable in practice, since $|V|^2$ can easily reach billion level or even higher.

In this paper, we introduce AllVec, an exact and efficient word embedding method based on full batch learning. To address the efficiency challenge in learning from all training samples, we devise a regression-based loss function for word embedding, which allows fast optimization with memorization strategies. Specifically, the acceleration is achieved by reformulating the expensive loss over all negative samples using a *partition* and a *decouple* operation. By decoupling and caching the bottleneck terms, we succeed to use all samples for each parameter update in a manageable time complexity which is mainly determined by the positive samples. The main contributions of this work are summarized as follows:

- We present a fine-grained weighted least square loss for learning word embeddings. Unlike GloVe, it explicitly accounts for all negative samples and reweights them with a frequency-aware strategy.
- We propose an efficient and exact optimization algorithm based on full batch gradient optimization. It has a comparable time complexity with SGD, but being more effective and stable due to the consideration of all samples in each parameter update.
- We perform extensive experiments on several benchmark datasets and tasks to demonstrate the effectiveness, efficiency, and convergence property of our AllVec method.

2 Related Work

2.1 Skip-gram with Negative Sampling

Mikolov et al. (2013a,b) proposed the skip-gram model to learn word embeddings. SG formulates the problem as a predictive task, aiming at predicting the proper context c for a target word w within a local window. To speed up the training process, it applies the negative sampling (Mikolov et al., 2013b) to approximate the full softmax. That is, each positive (w, c) pair is trained with n randomly sampled negative pairs (w, w_i). The sampled loss function of SG is defined as

$$L_{wc}^{SG} = \log \sigma(U_w \tilde{U}_c^T) + \sum_{i=1}^{n} \mathbf{E}_{w_i \sim P_n(w)} \log \sigma(-U_w \tilde{U}_{w_i}^T)$$

where U_w and $\tilde{U}_c$ denote the k-dimensional embedding vectors for word w and context c. $P_n(w)$ is the distribution from which negative context w_i is sampled.

Plenty of research has been done based on SG, such as the use of prior knowledge from another source (Kumar and Araki, 2016; Liu et al., 2015a; Bollegala et al., 2016), incorporating word type information (Cao and Lu, 2017; Niu et al., 2017), character level n-gram models (Bojanowski et al., 2016; Joulin et al., 2016) and jointly learning with topic models like LDA (Shi et al., 2017; Liu et al., 2015b).

2.2 Importance of the Sampling Distribution

Mikolov et al. (2013b) showed that the unigram distribution raised to the 3/4th power as $P_n(w)$ significantly outperformed both the unigram and the uniform distribution. This suggests that the sampling distribution (of negative words) has a great impact on the embedding quality. Furthermore, Chen et al. (2018) and Guo et al. (2018) recently found that replacing the original sampler with adaptive samplers could result in better performance. The adaptive samplers are used to find more informative negative examples during the training process. Compared with the original word-frequency based sampler, adaptive samplers adapt to both the target word and the current state of the model. They also showed that the fine-grained samplers not only speeded up the convergence but also significantly improved the embedding quality. Similar observations were also found in other fields like collaborative filtering (Yuan et al., 2016). While being effective, it is proven that negative sampling is a biased approximation and does not converges to the same loss as the full softmax — regardless of how many update steps have been taken (Bengio and Senécal, 2008; Blanc and Rendle, 2017).

2.3 Count-based Embedding Methods

Another line of research is the count-based embedding, such as GloVe (Pennington et al., 2014). GloVe performs a biased MF on the word-context co-occurrence statistics, which is a common approach in the field of collaborative filtering (Koren, 2008). However, GloVe only formulates the loss on positive entries of the co-occurrence matrix, meaning that negative signals about word-context co-occurrence are discarded. A remedy solution is LexVec (Salle et al., 2016a,b) which integrates negative sampling into MF. Some other methods (Li et al., 2015; Stratos et al., 2015; Ailem et al., 2017) also use MF to approximate the word-context co-occurrence statistics. Although predictive models and count-based models seem different at first glance, Levy and Goldberg (2014) proved that SG with negative sampling is implicitly factorizing a shifted pointwise mutual information (PMI) matrix, which means that the two families of embedding models resemble each other to a certain degree.

Our proposed method departs from all above methods by using the full batch gradient optimizer to learn from all (positive and negative) samples. We propose a fast learning algorithm to show that such batch learning is not "heavy" even with tens of billions of training examples.

3 AllVec Loss

In this work, we adopt the regression loss that is commonly used in count-based models (Pennington et al., 2014; Stratos et al., 2015; Ailem et al., 2017) to perform matrix factorization on word co-occurrence statistics. As highlighted, to retain the modeling fidelity, AllVec eschews using any sampling but optimizes the loss on all positive and negative word-context pairs.

Given a word w and a symmetric window of win contexts, the set of positive contexts can be obtained by sliding through the corpus. Let c denote a specific context, M_{wc} be the number of co-occurred (w, c) pairs in the corpus within the window. $M_{wc} = 0$ means that the pair (w, c) has never been observed, i.e. the negative signal. r_{wc} is the association coefficient between w and c, which is calculated from M_{wc}. Specifically, we use r^+_{wc} to denote the ground truth value for positive (w, c) pairs and a constant value r^-(e.g., 0 or -1) for negative ones since there is no interaction between w and c in negative pairs. Finally, with all positive and negative pairs considered, a regular loss function can be given as Eq.(1), where V is the vocabulary and S is the set of positive pairs. α^+_{wc} and α^-_{wc} represent the weight for positive and negative (w, c) pairs, respectively.

$$\begin{aligned} L = & \underbrace{\sum_{(w,c)\in S} \alpha^+_{wc}(r^+_{wc} - U_w \tilde{U}_c^T)^2}_{L_P} \\ & + \underbrace{\sum_{(w,c)\in (V\times V)\backslash S} \alpha^-_{wc}(r^- - U_w \tilde{U}_c^T)^2}_{L_N} \end{aligned} \tag{1}$$

When it comes to r^+_{wc}, there are several choices. For example, GloVe applies the log of M_{wc} with bias terms for w and c. However, research from Levy and Goldberg (2014) showed that the SG model with negative sampling implicitly factorizes a shifted PMI matrix. The PMI value for a (w, c) pair can be defined as

$$PMI_{wc} = log\frac{P(w,c)}{P(w)P(c)} = log\frac{M_{wc}M_{**}}{M_{w*}M_{*c}} \tag{2}$$

where '*' denotes the summation of all corresponding indexes (e.g., $M_{w*} = \sum_{c\in V} M_{wc}$). Inspired by this connection, we set r^+_{wc} as the positive point-wise mutual information (PPMI) which has been commonly used in the NLP literature (Stratos et al., 2015; Levy and Goldberg, 2014). Sepcifically, PPMI is the positive version of PMI by setting the negative values to zero. Finally, r^+_{wc} is defined as

$$r^+_{wc} = PPMI_{wc} = \max(PMI_{wc}, 0) \tag{3}$$

3.1 Weighting Strategies

Regarding α^+_{wc}, we follow the design in GloVe, where it is defined as

$$\alpha^+_{wc} = \begin{cases} (M_{wc}/xmax)^\rho & M_{wc} < xmax \\ 1 & M_{wc} \geq xmax \end{cases} \tag{4}$$

As for the weight for negative instances α^-_{wc}, considering that there is no interaction between w and negative c, we set α^-_{wc} as α^-_c (or α^-_w), which means that the weight is determined by the word itself rather than the word-context interaction. Note that either $\alpha^-_{wc} = \alpha^-_c$ or $\alpha^-_{wc} = \alpha^-_w$ does not influence the complexity of AllVec learning algorithm described in the next section. The design of α^-_c is inspired by the frequency-based oversampling scheme in skip-gram and missing data reweighting in recommendation (He et al., 2016). The intuition is that a word with high frequency is more likely to be a true negative context word if there is no observed word-context interactions. Hence, to effectively differentiate the positive and negative examples, we assign a higher weight for the negative examples that have a higher word frequency, and a smaller weight for infrequent words. Formally, α_{wc}^- is defined as

$$\alpha_{wc}^- = \alpha_c^- = \alpha_0 \frac{M_{*c}^\delta}{\sum_{c \in V} M_{*c}^\delta} \tag{5}$$

where α_0 can be seen as a global weight to control the overall importance of negative samples. $\alpha_0 = 0$ means that no negative information is utilized in the training. The exponent δ is used for smoothing the weights. Specially, $\delta = 0$ means a uniform weight for all negative examples and $\delta = 1$ means that no smoothing is applied.

4 Fast Batch Gradient Optimization

Once specifying the loss function, the main challenge is how to perform an efficient optimization for Eq.(1). In the following, we develop a fast batch gradient optimization algorithm that is based on a *partition* reformulation for the loss and a *decouple* operation for the inner product.

4.1 Loss Partition

As can be seen, the major computational cost in Eq.(1) lies in the term L_N, because the size of $(V \times V) \setminus S$ is very huge, which typically contains over billions of negative examples. To this end, we show our first key design that separates the loss of negative samples into the difference between the loss on all samples and that on positive samples[1]. The loss partition serves as the prerequisite for the efficient computation of full batch gradients.

$$L_N = \sum_{w \in V} \sum_{c \in V} \alpha_c^- (r^- - U_w \tilde{U}_c^T)^2 - \sum_{(w,c) \in S} \alpha_c^- (r^- - U_w \tilde{U}_c^T)^2 \tag{6}$$

By replacing L_N in Eq.(1) with Eq.(6), we can obtain a new loss function with a more clear structure. We further simplify the loss function by merging the terms on positive examples. Finally, we achieve a reformulated loss

$$\begin{aligned} L = & \underbrace{\sum_{w \in V} \sum_{c \in V} \alpha_c^- (r^- - U_w \tilde{U}_c^T)^2}_{L_A} \\ & + \underbrace{\sum_{(w,c) \in S} (\alpha_{wc}^+ - \alpha_c^-)(\Delta - U_w \tilde{U}_c^T)^2 + C}_{L_{P'}} \end{aligned} \tag{7}$$

where $\Delta = (\alpha_{wc}^+ r_{wc}^+ - \alpha_c^- r^-)/(\alpha_{wc}^+ - \alpha_c^-)$. It can be seen that the new loss function consists of two components: the loss L_A on the whole $V \times V$ training examples and $L_{P'}$ on positive examples. The major computation now lies in L_A which has a time complexity of $O(k|V|^2)$. In the following, we show how to reduce the huge volume of computation by a simple mathematical decouple.

[1]The idea here is similar to that used in (He et al., 2016; Li et al., 2016) for a different problem.

4.2 Decouple

To clearly show the *decouple* operation, we rewrite L_A as $\widetilde{L}_A$ by omitting the constant term $\alpha_c^-(r^-)^2$. Note that u_{wd} and $\tilde{u}_{cd}$ denote the d-th element in U_w and $\tilde{U}_c$, respectively.

$$\begin{aligned} \widetilde{L}_A = & \sum_{w \in V} \sum_{c \in V} \alpha_c^- \sum_{d=0}^{k} u_{wd} \tilde{u}_{cd} \sum_{d'=0}^{k} u_{wd'} \tilde{u}_{cd'} \\ & - 2r^- \sum_{w \in V} \sum_{c \in V} \alpha_c^- \sum_{d=0}^{k} u_{wd} \tilde{u}_{cd} \end{aligned} \tag{8}$$

Now we show our second key design that is based on a *decouple* manipulation for the inner product operation. Interestingly, we observe that the summation operator and elements in U_w and $\tilde{U}_c$ can be rearranged by the commutative property (Dai et al., 2007), as shown below.

$$\begin{aligned} \widetilde{L}_A = & \sum_{d=0}^{k} \sum_{d'=0}^{k} \sum_{w \in V} u_{wd} u_{wd'} \sum_{c \in V} \alpha_c^- \tilde{u}_{cd} \tilde{u}_{cd'} \\ & - 2r^- \sum_{d=0}^{k} \sum_{w \in V} u_{wd} \sum_{c \in V} \alpha_c^- \tilde{u}_{cd} \end{aligned} \tag{9}$$

An important feature in Eq.(9) is that the original inner product terms are disappeared, while in the new equation $\sum_{c \in V} \alpha_c^- \tilde{u}_{cd} \tilde{u}_{cd'}$ and $\sum_{c \in V} \alpha_c^- \tilde{u}_{cd}$ are "constant" values relative to $u_{wd} u_{wd'}$ and u_{wd} respectively. This means that they can be pre-calculated before training in each iteration. Specifically, we define $p_{dd'}^w$, $p_{dd'}^c$, q_d^w and q_d^c as the pre-calculated terms

$$\begin{aligned} p_{dd'}^w &= \sum_{w \in V} u_{wd} u_{wd'} & q_d^w &= \sum_{w \in V} u_{wd} \\ p_{dd'}^c &= \sum_{c \in V} \alpha_c^- \tilde{u}_{cd} \tilde{u}_{cd'} & q_d^c &= \sum_{c \in V} \alpha_c^- \tilde{u}_{cd} \end{aligned} \tag{10}$$

Then the computation of $\tilde{L}_A$ can be simplified to $\sum_{d=0}^{k} \sum_{d'=0}^{k} p_{dd'}^w p_{dd'}^c - 2r^- q_d^w q_d^c$.

It can be seen that the time complexity to compute all $p_{dd'}^w$ is $O(|V|k^2)$, and similarly, $O(|V|k^2)$ for $p_{dd'}^c$ and $O(|V|k)$ for q_d^w and q_d^c. With all terms pre-calculated before each iteration, the time complexity of computing $\tilde{L}_A$ is just $O(k^2)$. As a result, the total time complexity of computing L_A is decreased to $O(2|V|k^2 + 2|V|k + k^2) \approx O(2|V|k^2)$, which is much smaller than the original $O(k|V|^2)$. Moreover, it's worth noting that our efficient computation for $\widetilde{L}_A$ is strictly equal to its original value, which means AllVec does not introduce any approximation in evaluating the loss function.

Finally, we can derive the batch gradients for

u_{wd} and $\tilde{u}_{cd}$ as

$$\frac{\partial L}{\partial u_{wd}} = \sum_{d'=0}^{k} u_{wd'} p^c_{dd'} - \sum_{c\in I_w^+} \Lambda \cdot \tilde{u}_{cd} - r^- q^c_d$$
$$\frac{\partial L}{\partial \tilde{u}_{cd}} = \sum_{d'=0}^{k} \tilde{u}_{cd'} p^w_{dd'} \alpha_c^- - \sum_{w\in I_c^+} \Lambda \cdot u_{wd} - r^- \alpha_c^- q^w_d \tag{11}$$

where I_w^+ denotes the set of positive contexts for w, I_c^+ denotes the set of positive words for c and $\Lambda = (\alpha^+_{wc} - \alpha^-_c)(\Delta - U_w \tilde{U}_c^T)$. Algorithm 1 shows the training procedure of AllVec.

Algorithm 1 AllVec learning

```
Input: corpus $\boldsymbol{\Gamma}$, $win$, $\alpha_0$, $\delta$, $iter$, learning rate $\eta$
Output: embedding matrices $U$ and $\tilde{U}$
 1: Build vocabulary V from $\boldsymbol{\Gamma}$
 2: Obtain all positive $(w,c)$ and $M_{wc}$ from $\boldsymbol{\Gamma}$
 3: Compute all $r^+_{wc}$, $\alpha^+_{wc}$ and $\alpha^-_c$
 4: Initialize $U$ and $\tilde{U}$
 5: for $i = 1, ..., iter$ do
 6:     for $d \in \{0, .., k\}$ do
 7:         Compute and store $q^c_d$                 $\triangleright O(|V|k)$
 8:         for $d' \in \{0, .., k\}$ do
 9:             Compute and store $p^c_{dd'}$         $\triangleright O(|V|k^2)$
10:         end for
11:     end for
12:     for $w \in V$ do
13:         Compute $\Lambda$                          $\triangleright O(|S|k)$
14:         for $d \in \{0, .., k\}$ do
15:             Update $u_{wd}$                        $\triangleright O(|S|k + |V|k^2)$
16:         end for
17:     end for
18:     Repeat 6-17 for $\tilde{u}_{cd}$              $\triangleright O(2|S|k+2|V|k^2)$
19: end for
```

4.3 Time Complexity Analysis

In the following, we show that AllVec can achieve the same time complexity with negative sampling based SGD methods.

Given the sample size n, the total time complexity for SG is $O((n+1)|S|k)$, where $n+1$ denotes n negative samples and 1 positive example. Regarding the complexity of AllVec, we can see that the overall complexity of Algorithm 1 is $O(4|S|k + 4|V|k^2)$.

For the ease of discussion, we denote $\bar{c}$ as the average number of positive contexts for a word in the training corpus, i.e. $|S| = \bar{c}|V|$ ($\bar{c} \geq 1000$ in most cases). We then obtain the ratio

$$\frac{4|S|k + 4|V|k^2}{(n+1)|S|k} = \frac{4}{n+1}(1 + \frac{k}{\bar{c}}) \tag{12}$$

where k is typically set from 100 to 300 (Mikolov et al., 2013a; Pennington et al., 2014), resulting in $k \leq \bar{c}$. Hence, we can give the lower and upper bound for the ratio:

$$\frac{4}{n+1} < \frac{4|S|k+4|V|k^2}{(n+1)|S|k} = \frac{4}{n+1}(1+\frac{k}{\bar{c}}) \leq \frac{8}{n+1} \tag{13}$$

The above analysis suggests that the complexity of AllVec is same as that of SGD with negative sample size between 3 and 7. In fact, considering that $\bar{c}$ is much larger than k in most datasets, the major cost of AllVec comes from the part $4|S|k$ (see Section 5.4 for details), which is linear with respect to the number of positive samples.

5 Experiments

We conduct experiments on three popular evaluation tasks, namely word analogy (Mikolov et al., 2013a), word similarity (Faruqui and Dyer, 2014) and QVEC (Tsvetkov et al., 2015).

Word analogy task. The task aims to answer questions like, "a is to b as c is to _?". We adopt the Google testbed[2] which contains $19,544$ such questions in two categories: semantic and syntactic. The semantic questions are usually analogies about people or locations, like "king is to man as queen is to _?", while the syntactic questions focus on forms or tenses, e.g., "swimming is to swim as running to _?".

Word similarity tasks. We perform evaluation on six datasets, including MEN (Bruni et al., 2012), MC (Miller and Charles, 1991), RW (Luong et al., 2013), RG (Rubenstein and Goodenough, 1965), WS-353 Similarity (WSim) and Relatedness (WRel) (Finkelstein et al., 2001). We compute the spearman rank correlation between the similarity scores calculated based on the trained embeddings and human labeled scores.

QVEC. QVEC is an intrinsic evaluation metric of word embeddings based on the alignment to features extracted from manually crafted lexical resources. QVEC has shown strong correlation with the performance of embeddings in several semantic tasks (Tsvetkov et al., 2015).

We compare AllVec with the following word embedding methods.

- SG: This is the original skip-gram model with SGD and negative sampling (Mikolov et al., 2013a,b).
- SGA: This is the skip-gram model with an adaptive sampler (Chen et al., 2018).

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

Table 1: Corpora statistics.

Corpus	Tokens	Vocab	Size
Text8	17M	71K	100M
NewsIR	78M	83K	500M
Wiki-sub	0.8B	190K	4.0G
Wiki-all	2.3B	200K	13.4G

- GloVe: This method applies biased MF on the positive samples of word co-occurrence matrix (Pennington et al., 2014).
- LexVec: This method applies MF on the PPMI matrix. The optimization is done with negative sampling and mini-batch gradient descent (Salle et al., 2016b).

For all baselines, we use the original implementation released by the authors.

5.1 Datasets and Experimental Setup

We evaluate the performance of AllVec on four real-world corpora, namely Text8[3], NewsIR[4], Wiki-sub and Wiki-all. Wiki-sub is a subset of 2017 Wikipedia dump[5]. All corpora have been pre-processed by a standard pipeline (i.e. removing non-textual elements, lowercasing and tokenization). Table 1 summarizes the statistics of these corpora.

To obtain M_{wc} for positive (w, c) pairs, we follow GloVe where word pairs that are x words apart contribute $1/x$ to M_{wc}. The window size is set as $win = 8$. Regarding α^+_{wc}, we set $xmax = 100$ and $\rho = 0.75$. For a fair comparison, the embedding size k is set as 200 for all models and corpora. AllVec can be easily trained by AdaGrad (Zeiler, 2012) like GloVe or Newton-like (Bayer et al., 2017; Bradley et al., 2011) second order methods. For models based on negative sampling (i.e. SG, SGA and LexVec), the sample size is set as $n = 25$ for Text8, $n = 10$ for NewsIR and $n = 5$ for Wiki-sub and Wiki-all. The setting is also suggested by Mikolov et al. (2013b). Other detailed hyper-parameters are reported in Table 2.

5.2 Accuracy Comparison

We present results on the word analogy task in Table 2. As shown, AllVec achieves the highest total accuracy (Tot.) in all corpora, particularly in smaller corpora (Text8 and NewsIR). The reason is that in smaller corpora the number of positive (w, c) pairs is very limited, thus making use of negative examples will bring more benefits. Similar reason also explains the poor accuracy of GloVe in Text8, because GloVe does not consider negative samples. Even in the very large corpus (Wiki-all), ignoring negative samples still results in sub-optimal performance.

Our results also show that SGA achieves better performance than SG, which demonstrates the importance of a good sampling strategy. However, regardless what sampler (except the full softmax sampling) is utilized and how many updates are taken, sampling is still a biased approach. AllVec achieves the best performance because it is trained on the whole batch data for each parameter update rather than a fraction of sampled data.

Another interesting observation is AllVec performs better in semantic tasks in general. The reason is that our model utilizes global co-occurrence statistics, which capture more semantic signals than syntactic signals. While both AllVec and GloVe use global contexts, AllVec performs much better than GloVe in syntactic tasks. We argue that the main reason is because AllVec can distill useful signals from negative examples, while GloVe simply ignores all negative information. By contrast, local-window based methods, such as SG and SGA, are more effective to capture local sentence features, resulting in good performance on syntactic analogies. However, Rekabsaz et al. (2017) argues that these local-window based methods may suffer from the topic shifting issue.

Table 3 and Table 4 provide results in the word similarity and QVEC tasks. We can see that AllVec achieves the best performance in most tasks, which admits the advantage of batch learning with all samples. Interestingly, although GloVe performs well in semantic analogy tasks, it shows extremely worse results in word similarity and QVEC. The reason shall be the same as that it performs poorly in syntactic tasks.

5.3 Impact of α^-_c

In this subsection, we investigate the impact of the proposed weighting scheme for negative (context) words. We show the performance change of word analogy tasks on NewsIR in Figure 2 by tuning α_0 and δ. Results in other corpora show similar trends thus are omitted due to space limitation.

[3]http://mattmahoney.net/dc/text8.zip

[4]http://research.signalmedia.co/newsir16/signal-dataset.html

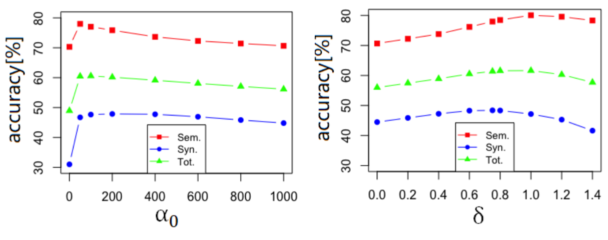
[5]https://dumps.wikimedia.org/enwiki/

Table 2: Results ("Tot." denotes total accuracy) on the word analogy task.

Corpus	Text8						NewsIR					
	para.			Sem.	Syn.	Tot.	para.			Sem.	Syn.	Tot.
SG	1e-4	8	25	47.51	32.26	38.60	1e-5	10	10	70.81	47.48	58.10
SGA	6e-3	-	-	48.10	33.78	39.74	6e-3	-	-	71.74	48.71	59.20
GloVe	10	15	1	45.11	26.89	34.47	50	8	1	78.79	41.58	58.52
LexVec	1e-4	25	-	51.87	31.78	40.14	1e-5	10	-	76.11	39.09	55.95
AllVec	350	0.75	-	56.66	32.42	**42.50**	100	0.8	-	78.47	48.33	**61.57**
	Wiki-sub						Wiki-all					
SG	1e-5	10	5	72.05	55.88	63.24	1e-5	10	5	73.91	61.91	67.37
SGA	6e-3	-	-	73.93	56.10	63.81	6e-3	-	-	75.11	61.94	67.92
GloVe	100	8	1	77.22	53.16	64.13	100	8	1	77.38	58.94	67.33
LexVec	1e-5	5	-	75.95	52.78	63.33	1e-5	5	-	76.31	56.83	65.48
AllVec	100	0.75	-	76.66	54.72	**64.75**	50	0.75	-	77.64	60.96	**68.52**

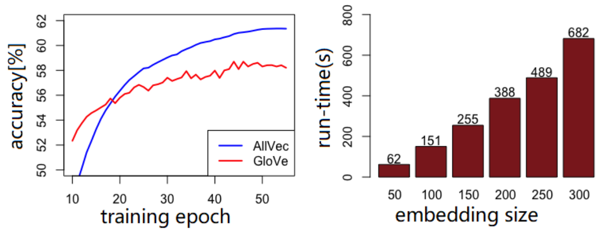
The parameter columns (para.) for each model are given from left to right as follows. SG: subsampling of frequent words, window size and the number of negative samples; SGA: λ (Chen et al., 2018) that controls the distribution of the rank, the other parameters are the same with SG; GloVe: $xmax$, window size and symmetric window; LexVec: subsampling of frequent words and the number of negative samples; AllVec: the negative weight α_0 and δ. Boldface denotes the highest total accuracy.

Figure 2(a) shows the impact of the overall weight α_0 by setting δ as 0.75 (inspired by the setting of skip-gram). Clearly, we observe that all results (including semantic, syntactic and total accuracy) have been greatly improved when α_0 increases from 0 to a larger value. As mentioned before, $\alpha_0 = 0$ means that no negative information is considered. This observation verifies that negative samples are very important for learning good embeddings. It also helps to explain why GloVe performs poorly on syntactic tasks. In addition, we find that in all corpora the optimal results are usually obtained when α_0 falls in the range of 50 to 400. For example, in the NewIR corpus as shown, AllVec achieves the best performance when $\alpha_0 = 100$. Figure 2(b) shows the impact of δ with $\alpha_0 = 100$. As mentioned before, $\delta = 0$ denotes a uniform value for all negative words and $\delta = 1$ denotes that no smoothing is applied to word frequency. We can see that the total accuracy is only around 55% when $\delta = 0$. By increasing its value, the performance is gradually improved, achieving the highest score when δ is around 0.8. Further increase of δ will degrade the total accuracy. This analysis demonstrates the effectiveness of the proposed negative weighting scheme.



(a) (b)

Figure 2: Effect of α_0 and δ on NewsIR.



(a) (b)

Figure 3: Convergence and runtime.

5.4 Convergence Rate and Runtime

Figure 3(a) compares the convergence between AllVec and GloVe on NewsIR. Clearly, AllVec exhibits a more stable convergence due to its full batch learning. In contrast, GloVe has a more dramatic fluctuation because of the one-sample learning scheme. Figure 3(b) shows the relationship between the embedding size k and runtime on NewsIR. Although the analysis in Section 4.3 demonstrates that the time complexity of AllVec is $O(4|S|k + 4|V|k^2)$, the actual runtime shows a near linear relationship with k. This is because $4|V|k^2/4|S|k = k/\bar{c}$, where $\bar{c}$ generally ranges from $1000 \sim 6000$ and k is set from 200 to 300 in practice. The above ratio explains the fact that $4|S|k$ dominates the complexity, which is linear

Table 3: Results on the word similarity task.

Corpus	Text8						NewsIR					
	MEN	MC	RW	RG	WSim	WRel	MEN	MC	RW	RG	WSim	WRel
SG	.6868	.6776	.3336	.6904	.7082	.6539	.7293	.7328	.3705	.7184	.7176	.6147
SGA	.6885	.6667	.3399	**.7035**	.7291	.6708	**.7409**	.7513	.3797	.7508	.7442	.6398
GloVe	.4999	.3349	.2614	.3367	.5168	.5115	.5839	.5637	.2487	.6284	.6029	.5329
LexVec	.6660	.6267	.2935	.6076	.7005	.6862	.7301	**.8403**	.3614	**.8341**	.7404	**.6545**
AllVec	**.6966**	**.6975**	**.3424**	.6588	**.7484**	**.7002**	.7407	.7642	**.4610**	.7753	**.7453**	.6322
	Wiki-sub						Wiki-all					
SG	**.7532**	.7943	.4250	.7555	.7627	**.6563**	.7564	.8083	.4311	.7678	**.7662**	.6485
SGA	.7465	.7983	.4296	.7623	**.7715**	.6560	**.7577**	.7940	.4379	.7683	.7110	.6488
GloVe	.6898	.6963	.3184	.7041	.6669	.5629	.7370	.7767	.3197	.7499	.7359	.6336
LexVec	.7318	.7591	.4225	.7628	.7292	.6219	.7256	**.8219**	.4383	.7797	.7548	.6091
AllVec	.7155	**.8305**	**.4667**	**.7945**	.7675	.6459	.7396	.7840	**.4966**	**.7800**	.7492	**.6518**

Table 4: Results on QVEC.

Qvec	Text8	NewsIR	Wiki-sub	Wiki-all
SG	.3999	.4182	.4280	.4306
SGA	.4062	.4159	**.4419**	.4464
GloVe	.3662	.3948	.4174	.4206
LexVec	**.4211**	.4172	.4332	.4396
AllVec	**.4211**	**.4319**	.4351	**.4489**

Table 5: Comparison of runtime.

Model	SI	Iter	Tot.
SG-3	259s	15	65m
SG-7	521s	15	131m
SG-10	715s	15	179m
AllVec	388s	50	322m

SG-n represents n negative samples for skip-gram, SI represents the runtime for a single iteration, and Tot. denotes the total runtime. All models are of embedding size 200 and trained with 8 threads.

with k and $|S|$.

We also compare the overall runtime of AllVec and SG on NewsIR and show the results in Table 5. As can be seen, the runtime of AllVec falls in the range of SG-3 and SG-7 in a single iteration, which confirms the theoretical analysis in Section 4.3. In contrast with SG, AllVec needs more iterations to converge. The reason is that each parameter in SG is updated many times during each iteration, although only one training example is used in each update. Despite this, the total run time of AllVec is still in a feasible range. Assuming the convergence is measured by the number of parameter updates, our AllVec yields a much faster convergence rate than the one-sample SG method.

In practice, the runtime of our model in each iteration can be further reduced by increasing the number of parallel workers. Although baseline methods like SG and GloVe can also be parallelized, the stochastic gradient steps in these methods unnecessarily influence each other as there is no exact way to separate these updates for different workers. In other words, the parallelization of SGD is not well suited to a large number of workers. In contrast, the parameter updates in AllVec are completely independent of each other, therefore AllVec does not have the update collision issue. This means we can achieve the embarrassing parallelization by simply separating the updates by words; that is, letting different workers update the model parameters for disjoint sets of words. As such, AllVec can provide a near linear scaling without any approximation since there is no potential conflicts between updates.

6 Conclusion

In this paper, we presented AllVec, an efficient batch learning based word embedding model that is capable to leverage all positive and negative training examples without any sampling and approximation. In contrast with models based on SGD and negative sampling, AllVec shows more stable convergence and better embedding quality by the all-sample optimization. Besides, both theoretical analysis and experiments demonstrate that AllVec achieves the same time complexity with the classic SGD models. In future, we will extend our proposed all-sample learning scheme to deep learning methods, which are more expressive than the shallow embedding model. Moreover, we will integrate prior knowledge, such as the words that are synonyms and antonyms, into the word embedding process. Lastly, we are interested in exploring the recent adversarial learning techniques to enhance the robustness of word embeddings.

Acknowledgements. This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative. Joemon M.Jose and Xiangnan He are corresponding authors.

References

Melissa Ailem, Aghiles Salah, and Mohamed Nadif. 2017. Non-negative matrix factorization meets word embedding. In *SIGIR*, pages 1081–1084.

Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. 2017. A generic coordinate descent framework for learning from implicit feedback. In *WWW*, pages 1341–1350.

Yoshua Bengio and Jean-Sébastien Senécal. 2008. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks*, pages 713–722.

Guy Blanc and Steffen Rendle. 2017. Adaptive sampled softmax with kernel based sampling. *arXiv preprint arXiv:1712.00527*.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Danushka Bollegala, Mohammed Alsuhaibani, Takanori Maehara, and Ken-ichi Kawarabayashi. 2016. Joint word representation learning using a corpus and a semantic lexicon. In *AAAI*, pages 2690–2696.

Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. 2011. Parallel coordinate descent for l1-regularized loss minimization. *arXiv preprint arXiv:1105.5379*.

Elia Bruni, Gemma Boleda, Marco Baroni, and Nam-Khanh Tran. 2012. Distributional semantics in technicolor. In *ACL*, volume 1, pages 136–145.

Shaosheng Cao and Wei Lu. 2017. Improving word embeddings with convolutional feature learning and subword information. In *AAAI*, pages 3144–3151.

Long Chen, Fajie Yuan, Joemon M Jose, and Weinan Zhang. 2018. Improving negative sampling for word representation using self-embedded features. In *WSDM*, pages 99–107.

Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. 2007. Co-clustering based classification for out-of-domain documents. In *SIGKDD*, pages 210–219.

Manaal Faruqui and Chris Dyer. 2014. Community evaluation and exchange of word vectors at wordvectors. org. In *ACL*, pages 19–24.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *WWW*, pages 406–414.

Guibing Guo, SC Ouyang, and Fajie Yuan. 2018. Approximating word ranking and negative sampling for word embedding. In *IJCAI*.

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

Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, pages 549–558.

Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. 2012. Improving word representations via global context and multiple word prototypes. In *ACL*, pages 873–882.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hervé Jégou, and Tomas Mikolov. 2016. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD*, pages 426–434.

Abhishek Kumar and Jun Araki. 2016. Incorporating relational knowledge into word representations using subspace regularization. In *ACL*, volume 2, pages 506–511.

Wenqiang Lei, Xuancong Wang, Meichun Liu, Ilija Ilievski, Xiangnan He, and Min-Yen Kan. 2017. Swim: A simple word interaction model for implicit discourse relation recognition. In *IJCAI*, pages 4026–4032.

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

Huayu Li, Richang Hong, Defu Lian, Zhiang Wu, Meng Wang, and Yong Ge. 2016. A relaxed ranking-based factor model for recommender system from implicit feedback. In *IJCAI*, pages 1683–1689.

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 *IJCAI*, pages 3650–3656.

Quan Liu, Hui Jiang, Si Wei, Zhen-Hua Ling, and Yu Hu. 2015a. Learning semantic word embeddings based on ordinal knowledge constraints. In *ACL*, volume 1, pages 1501–1511.

Yang Liu, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. 2015b. Topical word embeddings. In *AAAI*, pages 2418–2424.

Thang Luong, Richard Socher, and Christopher Manning. 2013. Better word representations with recursive neural networks for morphology. In *CoNLL*, pages 104–113.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

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

George A Miller and Walter G Charles. 1991. Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28.

Yilin Niu, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2017. Improved word representation learning with sememes. In *ACL*, volume 1, pages 2049–2058.

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

Navid Rekabsaz, Mihai Lupu, Allan Hanbury, and Hamed Zamani. 2017. Word embedding causes topic shifting; exploit global context! In *SIGIR*, pages 1105–1108.

Herbert Rubenstein and John B Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.

Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Alexandre Salle, Marco Idiart, and Aline Villavicencio. 2016a. Enhancing the lexvec distributed word representation model using positional contexts and external memory. *arXiv preprint arXiv:1606.01283*.

Alexandre Salle, Marco Idiart, and Aline Villavicencio. 2016b. Matrix factorization using window sampling and negative sampling for improved word representations. *arXiv preprint arXiv:1606.00819*.

Fabrizio Sebastiani. 2002. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47.

Bei Shi, Wai Lam, Shoaib Jameel, Steven Schockaert, and Kwun Ping Lai. 2017. Jointly learning word embeddings and latent topics. In *SIGIR*, pages 375–384.

Karl Stratos, Michael Collins, and Daniel Hsu. 2015. Model-based word embeddings from decompositions of count matrices. In *ACL*, volume 1, pages 1282–1291.

Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Guillaume Lample, and Chris Dyer. 2015. Evaluation of word vector representations by subspace alignment. In *EMNLP*, pages 2049–2054.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *ACL*, pages 384–394.

Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. 2016. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In *CIKM*, pages 227–236. ACM.

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.