Bayesian Neural Word Embedding

Oren Barkan
Tel Aviv University

Abstract

Recently, several works in the domain of natural language processing presented successful methods for word embedding. Among them, the Skip-Gram (SG) with negative sampling, known also as word2vec, advanced the state-of-the-art of various linguistics tasks. In this paper, we propose a scalable Bayesian neural word embedding algorithm that can be beneficial to general item similarity tasks as well. The algorithm relies on a Variational Bayes solution for the SG objective and a detailed step by step description of the algorithm is provided. We present experimental results that demonstrate the performance of the proposed algorithm and show it is competitive with the original SG method.

Keywords: bayesian word2vec, bayesian skip-gram, neural word embedding, scalable bayesian word embedding, collaborative filtering, bayesian item2vec, item similarity, bayesian representation learning.

1 Introduction and Related Work

Recent progress in neural word embedding methods have advanced the state-of-the-art in the domain of natural language processing (NLP) [1]–[4]. These methods attempt to learn representations for words that capture semantic and syntactic relations between them. Specifically, Skip-Gram (SG) with negative sampling, known also as word2vec [4], set new records in various linguistic tasks and its applications have been extended to other domains beyond NLP such as computer vision [5], [6], Collaborative Filtering (CF) and item similarity [7].

In this paper, we propose a scalable Bayesian neural embedding algorithm that in principle, can be applied to any dataset that is given as sequences (sets) of words (items). Hence, the proposed method is not limited to the task of word embedding and applicable to general item similarity tasks as well. We provide a fully detailed, step by step algorithm, which is straightforward to implement and requires negligible amount of parameter tuning.

Bayesian methods for words representation are proposed in [8], [9]. Different from these methods, we propose a Variational Bayes (VB) [10] solution to the SG objective that is inspired by a recent work in the CF domain [11]. Therefore, we name our method Bayesian Skip-Gram (BSG). While the SG method maps words to vectors, BSG maps words to densities in a latent space. Furthermore, BSG provides a stable and robust behavior and requires less effort in hyperparameter tuning. This is in contrast to point estimate solutions [1], [4] that are more sensitive to singularities in the objective and often require significant amount of hyperparameter tuning. Moreover, it provides a confidence level in the embedding which is useful for long tail words.

The rest of the paper is organized as follows: Section 2 overviews the SG method. In Section 3, we provide the mathematical derivation of the BSG solution. In Section 4, we describe the BSG algorithm. In Section 5, we present experimental results on various datasets.

2 Skip-Gram with Negative Sampling

SG with negative sampling is a neural word embedding method that was introduced in [4]. The method aims at estimating words representation that captures semantic and syntactic relations between a word to its surrounding words in a sentence. Note that SG can be applied with hierarchical softmax [4], but in this work we refer to SG as SG with negative sampling. The rest of this section provides a brief overview of the SG method.
Given a sequence of words \((w_i)_{i=1}^{L}\) from a finite vocabulary \(W = \{w_i\}_{i=1}^{L}\), the SG method aims at maximizing the following objective

\[
\frac{1}{L} \sum_{i=1}^{L} \sum_{c \in \mathcal{C}(w_i) \cup \{0\}} \log p(w_{i,c} | w_i)
\]

where \(c\) is defined as the context window size and \(p(w_{i,c} | w_i)\) is the softmax function:

\[
p(w_{i,c} | w_i) = \frac{\exp(u_i^T v_c)}{\sum_{k \in I} \exp(u_i^T v_k)}
\]

where \(u_i \in U(\subset \mathbb{R}^m)\) and \(v_c \in V(\subset \mathbb{R}^m)\) are latent vectors that correspond to the target and context representations for the word \(w_i \in W\), respectively, \(I_w = \{1, \ldots, l\}\) and the parameter \(m\) is chosen empirically and according to the size of the dataset. Using Eq. (2) is impractical due to the computational complexity of \(\forall p(w_j | w_i)\), which is a linear function of the vocabulary size \(l\) that is usually in size of \(10^3 \sim 10^6\).

### 2.1 Negative Sampling

Negative sampling [4] is introduced in order to overcome the above computational problem by the replacement of the softmax function from Eq. (2) with

\[
p(w_j | w_i) = \sigma(u_i^T v_j) \prod_{k=1}^{N} \sigma(-u_i^T v_k)
\]

where \(\sigma(x) = 1/1 + \exp(-x)\), \(N\) is a parameter that determines the number of negative examples to be sampled per a positive example. A negative word \(w_i\) is sampled from the unigram distribution raised to the 3/4rd power \(p_{\text{uni}}^{3/4}(w)\), where \(p_{\text{uni}}(w)\) is defined as the number of times \(w\) appears in the entire corpus divided by the total length (in words) of the corpus. \(p_{\text{uni}}^{3/4}(w)\) was found to significantly outperform both the uniform and unigram distributions [4].

### 2.2 Data Subsampling

In order to overcome the imbalance between rare and frequent words, we discard each word \(w\) in a sentence according to the probability

\[
p(\text{discard} | w) = 1 - \sqrt{p / f(w)}, \quad \text{where} \quad f(w) \text{ is the frequency of the word } w \text{ and } p \text{ is a prescribed threshold. This procedure is reported to accelerate the learning process and to improve the representation of rare words [4].}
\]

### 2.3 Words Representation and Similarity

The latent representations \(U\) and \(V\) are learned by applying a stochastic gradient ascent with respect to the objective in Eq. (1).

It is worth noting that some versions of word embedding algorithms incorporates bias terms into Eq. (3) as follows

\[
p(w_j | w_i) = \sigma(u_i^T v_j + b_i + b_j) \prod_{k=1}^{N} \sigma(-u_i^T v_k - b_i - b_k).
\]

These biases often explain properties such as frequency of a word in the text [1], [3] or popularity of an item in the dataset [11]. In this work, we do not use biases, since in our experiments, their contribution was found to be marginal.

SG produces two different representations \(u_i\) and \(v_i\) for the word \(w_i\). Alternatively, we can use \(v_i\), the additive composition \(u_i + v_i\) or the concatenation \([u_i^T v_i^T]^T\). The last two options are reported to produce superior representation [12].

The similarity between a pair of words \(w_a, w_b\) is computed by applying cosine similarity to their corresponding representations \(u_a, u_b\) as follows

\[
sim(w_a, w_b) = u_a^T u_b / (\|u_a\| \|u_b\|).
\]

### 3 Bayesian Skip-Gram (BSG) - Theory

As described in Section 2, SG produces point estimates for \(U\) and \(V\). In this section, we propose a method for deriving full distributions for \(U\) and \(V\) (in this paper, we use the terms distribution and density interchangeably). We assume that each target and context random vectors are independent and have normal priors with a zero mean and diagonal covariance with a precision parameter \(\tau\) as follows

\[
\forall i \in I_w: p(u_i) = N_{u_i}(0, \tau^{-1} I) \quad \text{and} \quad p(v_i) = N_{v_i}(0, \tau^{-1} I).
\]

Note that different values of \(\tau\) can be set to the priors over \(U\) and \(V\). Furthermore, these hyperparameters can be treated as random variables and be learned from the data [10]. However, in
this work, we assume these hyperparameters are apriori given and identical.

We define \( C(w) \) as a multiset that contains the indices of the context words of \( w \) in the corpus. Let \( I_p = \{(i, j) \mid j \in C(w_i)\} \) and \( I_n = \{(i, j) \mid j \not\in C(w_i)\} \) be the positive and negative sets, respectively. Note that \( I_p \) is a multiset too and \( I_n \)'s size might be \(|W|\). Therefore, a conjugate relation between 

\[
\text{argmax}_{\theta} \quad \sum_{i,j \in I_p} \log(\theta_{ij}) - \sum_{i,j \in I_n} \log(\theta_{ij})
\]

is given by 

\[
\theta_{ij} = \frac{d_{ij}}{d_{ij} + d_{ij}}.
\]

Then, the likelihood of \( d_{ij} \) is given by 

\[
p(d_{ij} | u_i, v_j) = \sigma(d_{ij} u_i^T v_j).
\]

Note that when applied to multisets, the operator \( \cup \) is defined as the multiset sum and not as the multiset union.

The joint distribution of \( U, V \) and the data \( D \) is given by 

\[
p(U, V, D) = p(D | U, V) p(U) p(V) = \prod_{(i, j) \in I_p} p(d_{ij} | u_i, v_j) \prod_{i \in I_p} p(u_i) \prod_{v \in I_p} p(v_i) = \prod_{(i, j) \in I_p} \sigma(d_{ij} u_i^T v_j) \prod_{i \in I_p} N(u_i | 0, \tau^{-1} I) \prod_{v \in I_p} N(v_i | 0, \tau^{-1} I).
\]

### 3.1 Variational Approximation

We aim at computing the posterior \( p(U, V \mid D) \). However, direct computation is hard. The posterior can be approximated using MCMC approaches such as Gibbs sampling or by using VB methods. In this work, we choose to apply VB approximation [10], since it was shown to converge faster to an accurate solution, empirically.

Let \( \theta = U \cup V \), VB approximates the posterior 

\[
p(\theta \mid D)
\]

by finding a fully factorized distribution 

\[
q(\theta) = q(U, V) = q(U) q(V) = \prod_{i=1}^{j} q(u_i) q(v_i)
\]

that minimizes the KL divergence [10]

\[
D_{KL}(q(\theta) \| p(\theta \mid D)) = \int q(\theta) \log \frac{q(\theta)}{p(\theta \mid D)} d\theta.
\]

To this end, we define the following expression:

\[
L(q) = \int q(\theta) \log \frac{p(\theta, D)}{q(\theta)} d\theta = \int q(\theta) \log \frac{p(\theta \mid D)}{q(\theta)} d\theta + \int q(\theta) \log p(D) d\theta = (5)
\]

where the last transition in Eq. (5) is due to the fact \( q \) is a PDF. By rearranging Eq. (5) we get the relation 

\[
D_{KL}(q(\theta) \| p(\theta \mid D)) = \log p(D) - L(q),
\]

where we notice that \( \log p(D) \) is independent of \( q \). Hence, minimizing \( D_{KL}(q(\theta) \| p(\theta \mid D)) \) is equivalent to maximizing \( L(q) \). It was shown [10] that \( L(q) \) is maximized by an iterative procedure that is guaranteed to converge to a local optimum. This is done by updating each of \( q \)'s factors, sequentially and according to the update rule 

\[
q_{\theta_i} = \exp \left( \mathbf{E}_{q_{\theta_{-i}}} [\log p(\theta, D)] + \text{const} \right)
\]

where the update for \( q_{\theta_i} \) is performed by the replacement of \( u_i \) with \( v_i \) in Eq. (6).

Recall that the term \( p(\theta, D) = p(U, V \mid D) \) in Eq. (6) contains the likelihood \( p(D \mid U, V) \) from Eq. (4), which is a product of sigmoid functions of \( U \) and \( V \). Therefore, a conjugate relation between the likelihood and the priors does not hold and the distribution consequents from \( \mathbf{E}_{q_{\theta_i}} [\log p(\theta, D)] \) in Eq. (6) does not belong to the exponential family.

Next, we will show that by the introduction of additional parameters \( \xi \), we are able to bring Eq. (6) to a form that is recognized as the Gaussian distribution.

We start by lower bounding \( p(D \mid U, V) \) using the following logistic bound [13]

\[
\log \sigma(a) \geq \frac{a - \xi}{2} - \lambda(\xi)(a^2 - \xi^2) + \log \sigma(\xi)
\]

where \( \lambda(\xi) = \frac{1}{2\pi} \left( \sigma(\xi) - \frac{1}{2} \right) \). By applying the lower bound from (7) to \( \log p(D \mid \theta) \) we get

\[
\log p(D \mid \theta) \geq \log p_q(D \mid \theta) = \log p_q(D \mid U, V) = \sum_{(i, j) \not\in I_p} \frac{d_{ij} u_i^T v_j - \xi_{ij}}{2} - \lambda(\xi) (u_i^T v_j v_j^T u_i - \xi_{ij}) + \log \sigma(\xi)
\]

By using Eq. (8) we can bound \( L(q) \) as follows.
\[ L(q) \geq L_\eta(q) = \int q(\theta) \log \frac{p_\eta(\theta, D)}{q(\theta)} \ d\theta = \int q(\theta) \log \frac{p_\eta(D | \theta) p(\theta)}{q(\theta)} \ d\theta. \]

Furthermore, it was shown [13] that the bound in Eq. (6) is tight when
\[ \xi_{ij} = E_q[u_i^T v_j v_j^T u_i] = \text{var}(u_i^T v_j) + E_q[u_i^T v_j | E_q[v_j^T u_i]] = \text{var}(u_i^T v_j) + \mu_{ij}^T \mu_i \mu_j^T \mu_j, \]
where \( \mu_{ij} = E_q[v_j] \) and the last transition in Eq. (9) holds since \( u_i \) and \( v_j \) are independent. By assuming diagonal covariance matrices, the term \( \text{var}(u_i^T v_j) \) in Eq. (8) is computed by
\[ \text{var}(u_i^T v_j) = \sum_{t=1}^m \sigma_{\alpha_i}^2 \sigma_{\beta_j}^2 + \sigma_{\alpha_i}^2 \mu_{\alpha_i}^2 + \sigma_{\beta_j}^2 \mu_{\beta_i}^2. \]

Finally, by combining Eqs. (9) and (10) we get
\[ \xi_{ij} = \left( \sum_{t=1}^m (\sigma_{\alpha_i}^2 + \mu_{\alpha_i}^2) (\sigma_{\beta_j}^2 + \mu_{\beta_i}^2) \right). \]

Therefore, instead of maximizing \( L(q) \) we can maximize \( L_\eta(q) \) and this is done by replacing the term \( \log p(\theta, D) \) from Eq. (6) with \( \log p_\eta(\theta, D) \) as follows
\[ q^*_\eta = \exp \left( E_{q^*_\eta} [\log p_\eta(\theta, D)] + \text{const} \right). \]

By applying the natural logarithm to Eq. (12) we get
\[ \log q^*_\eta = E_{q^*_\eta} [\log p_\eta(\theta, D)] + \text{const} = E_{q^*_\eta} [\log p_\eta(D | U, V)] + E_{q^*_\eta} [\log p(U, V)] + c =
\[ u_i^T \sum_{j \in I_u} \left[ \frac{1}{2} d_{ij} \mu_j \right] \]

\[ -\frac{1}{2} u_i^T \left( \sum_{j \in I_u} \left[ 2\lambda(\xi_{ij}) E_q[v_j v_j^T] + \tau I \right] u_j + c_i \right), \]
with \( I_u = \{ j \mid (i, j) \in I_b \} \) and \( E_q[v_j v_j^T] = \mu_{ij} + \mu_{ij}^T \).

Note that in the last transition in Eq. (13) all the terms that are independent of \( u_i \) are absorbed to the \( c_i \) term.

By inspecting Eq. (13) we see that \( q^*_\eta \) is normally distributed with the natural parameters
\[ P_{\eta} = \sum_{j \in I_u} \left[ 2\lambda(\xi_{ij}) E_q[v_j v_j^T] + \tau I \right], \]
\[ r_{\eta} = \frac{1}{2} \sum_{j \in I_u} d_{ij} \mu_j, \]
where \( P_{\eta} = \sum_{u_i} \) is the precision matrix and \( r_{\eta} = \mu_{\eta} \) is the means times precision vector.

Note that the computation of \( q^*_\eta \)'s parameters is symmetric. Moreover, since the updates for \( \{q_{\eta}\}_{t=1} \) depends on \( \{q_{\eta}\}_{t=1} \) only and vice versa, they can be performed in parallel. This gives an alternating updates scheme that (under the assumption of a constant dataset) is guaranteed to converge to a local optimum [10]: First, update all \( \{q_{\eta}\}_{t=1} \) (in parallel), then update all \( \{q_{\eta}\}_{t=1} \) (in parallel) and repeat until convergence.

### 3.2 Stochastic Updates

Due to data sampling, the effective dataset changes between the iterations and the optimization becomes stochastic. Since we do not want to ignore the information from previous steps, we need to figure out a way to consider this information in our updates. A common practice is to perform updates in the spirit of the Robbins-Monro method [14]. This is performed by the introduction of an iteration dependent variable \( \beta^{(k)} \) that controls the updates as follows
\[ P_{\eta}^{(k)} = \beta^{(k)} P_{\eta} + (1 - \beta^{(k)}) P_{\eta}^{(k-1)}, \]
\[ r_{\eta}^{(k)} = \beta^{(k)} r_{\eta} + (1 - \beta^{(k)}) r_{\eta}^{(k-1)}. \]

In practice, this means that during the runtime of the algorithm we need to keep the results from the previous iteration.

Robbins and Monro shows several conditions for convergence, where one of them states that \( \beta^{(k)} \) needs to satisfy:
\[ \sum_{k=0}^{\infty} \beta^{(k)} = \infty \quad \text{and} \quad \sum_{k=0}^{\infty} (\beta^{(k)})^2 < \infty. \]

To this end, we suggest to use \( \beta^{(k)} = k^{-\gamma} \) with a decay parameter \( 0.5 < \gamma \leq 1 \) as this ensures the conditions in (15) hold. We further suggest to set \( \beta^{(1)} = 1 \) for the first few iterations, in order to avoid too early convergence. Specifically, in our implementation, we did not perform stochastic updates in the first 10 iterations.
4 The BSG Algorithm

In this section, we provide a detailed description of the BSG algorithm that is based on Sections 2 and 3. The algorithm is described in Fig. 1 and consists of three main stages. The first stage is an initialization, then the algorithm iterates between data sampling and parameter updates till a convergence criterion is met or number of iterations is exceeded.

4.1 Stage 1 - Initialization

The algorithm is given the following hyperparameters: the target representation dimension $m$, the number of maximum iterations $K$, the maximal window size $c_{\text{max}}$, the negative to positive ratio $N \in \mathbb{N}$, the subsampling parameter $\rho$, a stopping threshold $\epsilon$, the decay parameter $\gamma$ and the prior precision parameter $\tau$. As described in Section 3, different values of $\tau$ can be learned for $U$ and $V$, however in our implementation, we chose to use a unique parameter $\tau$.

In this stage, we further need to determine the effective set of words $W$ to learn representation for. This can be done by considering all words in the data that appear more than a prescribed number of times, or by considering the $l$ most popular words. In this work, we stick with the latter. Then, every word $w \not\in W$ is discarded from the data (step 1).

Step 2 initializes the target distributions $Q = \{q_\alpha, q_\beta \}_{\alpha=1}^c$ parameters. Specifically, the means are drawn from the multivariate standard normal distribution and the covariance matrices are set to identity.

In step 3, we compute $P_{\text{uni}}$ according to the description in Section 2.1, then we raise it to the $\frac{3}{4}$rd power.

Step 4 updates $k$ and $K$ according to $\kappa$. This ensures that stochastic updates are not performed in the first $\kappa$ iterations.

4.2 Stage 2 – Data Sampling

At the beginning of every iteration, we subsample the data (step 5.1) as described in Section 2.2. Then, we follow the description in Section 3: for each instance of the word $w_j$ in $T$, we sample a window size $c$ from the uniform discrete distribution over the set $\{1, \ldots, c_{\text{max}}\}$ and consider the $c$ words to the left and to the right of $w_j$ as context words for $w_j$. This results in a multiset $C(w_j)$ that contains the indices of the context words of $w_j$ (an index may appear multiple times). Then, we create a positive multiset of tuples $I_p = \{(i, j) \mid j \in C(w_i)\}$ (step 5.3).

Next, for each tuple $(i, j) \in I_p$, we sample $N$ negative examples $(i, z)$ such that $(i, z) \not\in I_p$. A negative word $w_i$ is sampled according to $p_{\text{uni}}^\text{neg}(w_i)$. We also update $I_u, I_v, I_z, d_\mu, d_v$ accordingly (step 5.4).

Note that the description of step 5.4 is naive. In practice, we suggest to save for each tuple a counter for the number of times it appears. This can be done by maintaining dictionary data structures that count positive and negative examples. This further avoids the need of maintaining $\{I_u, I_v\}_{\alpha=1}^c$ as multisets (list data structures) and replace them with set data structures.

4.3 Stage 3 – Parameter Updates

In this stage, we update the parameters of the distributions $Q = \{q_\alpha, q_\beta \}_{\alpha=1}^c$. The updates are performed first for $\{q_\alpha \}_{\alpha=1}^c$ (step 5.6) and then for $\{q_\beta \}_{\beta=1}^c$ in a symmetric manner (step 5.7). Moreover, each sequence of updates is performed in parallel. Note that step 5.6.1 involves the computation of $\xi_\mu, \lambda(\xi_\mu)$ and $E_q[v\xi^\mu]$ that are given by Eqs. (11), (7) and (13), respectively.

Due to data sampling the dataset is changed per iteration. Therefore, we apply stochastic updates (step 5.6.2). The stochastic updates are performed starting from the iteration $\kappa+1$. This is ensured by step 5.5.

A crucial point to notice is the computation of the mean and the covariance: first, we compute the covariance matrix by the inversion of the precision matrix (step 5.6.3). This is performed by using Cholesky decomposition. Then, we extract the mean (step 5.6.4). Finally, we set all the off diagonal values in $\Sigma_{\alpha, \beta}$ to zeros (step 5.6.5) while keeping $P_{\mu, \alpha}$ as is.
BSG Algorithm

**Input:**

- $m$ - target representation dimension
- $T$ - input text, given as sequence of sequences of words
- $\tau$ - prior precision
- $K$ - maximum number of iteration
- $c_{\text{max}}$ - maximal window size
- $l$ - number of most frequent words to be considered in the vocabulary
- $\rho$ - subsampling parameter
- $N$ - negative to positive ratio
- $\kappa$ - number of iterations to apply without performing stochastic updates (in the beginning)
- $\varepsilon$ - stopping threshold
- $\gamma$ - decay parameter.

**Output:**

$Q = \{q_u, q_v\}_{i=1}^I$ - parameters of the distributions $Q = \{q_u, q_v\}_{i=1}^I$

1. Create a set $W = \{w_i\}_{i=1}^l$ of the $l$ most frequent words in $T$ and discard all other words from $T$.
2. for $i \leftarrow 1$ to $l$
   2.1 $\mu_u \sim N(0, I) \ , \ \mu_v \sim N(0, I) \ , \ P_u \leftarrow I \ , \ P_v \leftarrow I$
3. Compute $p_{\text{uni}}^{3/4}$ over $W$ using $T$ as described in Section 2.1.
4. $k \leftarrow 1 - \kappa \ , \ K \leftarrow K - \kappa$ \hspace{1cm} // first $\kappa$ iterations are performed without stochastic updates
5. repeat
   5.1 $T_{\text{sub}} \leftarrow \text{Subsample } (T) \text{, as described in Section 2.2}$
   5.2 for $i \leftarrow 1$ to $l$
      5.2.1 $I_u \leftarrow \emptyset \ , \ I_v \leftarrow \emptyset$
      5.2.2. $P_{u}^{\text{prev}} \leftarrow P_{u} \ , \ P_{v}^{\text{prev}} \leftarrow P_{v} \ , \ r_{u}^{\text{prev}} \leftarrow r_{u} \ , \ r_{v}^{\text{prev}} \leftarrow r_{v}$ \hspace{1cm} // save values from the previous iteration
   5.3 Create $I_p$ based on $T_{\text{sub}}$ as described in Section 4.2 \hspace{1cm} // positive sampling
   5.4 for $(i, j)$ in $I_p$
      5.4.1. $I_u \leftarrow I_u \cup \{j\} \ , \ I_v \leftarrow I_v \cup \{i\} \ , \ d_{ij} \leftarrow 1$
   5.4.2. for $n \leftarrow 1$ to $N$ \hspace{1cm} // negative sampling
      5.4.2.1. Sample a negative word index $z$ according to $p_{\text{uni}}^{3/4}(w_z)$ s.t. $(i, z) \notin I_p$
      5.4.2.2. $I_u \leftarrow I_u \cup \{z\} \ , \ I_v \leftarrow I_v \cup \{i\} \ , \ d_{iz} \leftarrow -1$
   5.5 if $k > 0$ then $\beta \leftarrow k^{-7}$ \hspace{1cm} else $\beta \leftarrow 1$ \hspace{1cm} // stochastic updates condition
   5.6 parfor $i \leftarrow 1$ to $l$ \hspace{1cm} // parallel for loop
      5.6.1. Compute $P_u, r_{u}$ using Eq. (14)
      5.6.2. $P_u \leftarrow (1 - \beta)P_u + \beta P_{u}^{\text{prev}} \ , \ r_u \leftarrow (1 - \beta)r_u + \beta r_{u}^{\text{prev}}$
      5.6.3. $\Sigma_u \leftarrow P_u^{-1}$
      5.6.4. $\mu_u \leftarrow \Sigma_u r_{u}$
      5.6.5. $\Sigma_u \leftarrow \text{diag[diag[}\Sigma_u]}$
   5.7. Apply a symmetric version of step 5.6 to $\{q_u\}_{i=1}^I$ parameters
until $k > K$ or $\sum_{i=1}^{I}||q_{u,i} - q_{u,i}^{\text{prev}}||_2 < \varepsilon$ and $\sum_{i=1}^{I}||q_{v,i} - q_{v,i}^{\text{prev}}||_2 < \varepsilon$

Figure 1: The BSG algorithm
The algorithm stops if the convergence criterion is met or number of iterations is exceeded (last line).

4.4 Similarity Measures

BSG maps words to normal distributions. In this work, we choose to use the distributions \( \{ q_{\theta_i} \}_{i=1} \) for representing words.

The similarity between a pair of words \( w_i, w_j \) can be computed by the cosine similarity of their means \( \mu_{\theta_i}, \mu_{\theta_j} \). By using the covariance, a confidence level can be computed as well. To this end, we define a random variable \( y_{ij} = u_i^T u_j \). Though the distribution of \( y_{ij} \) is not normal, it has the following mean and variance

\[
\mu_{y_{ij}} = \mu_{\theta_i}^T \mu_{\theta_j}, \quad \sigma^2_{y_{ij}} = \text{tr}[\Sigma_{\theta_i} \Sigma_{\theta_j}] + \mu_{\theta_i}^T \Sigma_{\theta_i} \mu_{\theta_j} + \mu_{\theta_j}^T \Sigma_{\theta_j} \mu_{\theta_i}.
\]

(16)

Hence, we choose to approximate \( y_{ij} \)'s distribution with \( N(\mu_{y_{ij}}, \sigma^2_{y_{ij}}) \). Then, \( -\sigma^2_{y_{ij}} \) can be used as a confidence level of the similarity score.

BSG further enables the applications of other similarity types. For example, we can approximate \( p(d_{ij} = 1|D) \) by approximating the marginalization

\[
p(d_{ij} = 1|D) = \int p(d_{ij} = 1, u_i, u_j|D) du_i du_j = \\
\int p(d_{ij} = 1|u_i, u_j)p(u_i|D)p(u_j|D)du_i du_j = \\
\int \sigma(u_i^T u_j)q(u_i)q(u_j)du_i du_j = \int \sigma(y_{ij}) p(y_{ij}) dy_{ij} = \\
\sigma(\mu_{y_{ij}})/\sqrt{1+\sigma^2_{y_{ij}}/8}.
\]

(17)

where \( \mu_{y_{ij}} \) and \( \sigma^2_{y_{ij}} \) are given by Eq. (16) and the last three approximations follow from the VB approximation, Eq. (16) and [15], respectively.

Another option is to apply a similarity measure that is based on a symmetric version of the KL divergence between two multivariate normal distributions

\[
sim_{\text{symKL}}(q_{\theta_i}, q_{\theta_j}) = -D_{KL}(q_{\theta_i} \parallel q_{\theta_j}) - D_{KL}(q_{\theta_j} \parallel q_{\theta_i})
\]

(18)

where \( D_{KL}(q_{\theta_i} \parallel q_{\theta_j}) \) has the following closed form solution [4]

\[
D_{KL}(q_{\theta_i} \parallel q_{\theta_j}) = \frac{1}{2} \left\{ \log|\Sigma_{\theta_i}| - \log|\Sigma_{\theta_j}| + \\
(\mu_{\theta_i} - \mu_{\theta_j})^T \Sigma_{\theta_j}^{-1} (\mu_{\theta_i} - \mu_{\theta_j}) + tr[\Sigma_{\theta_j}^{-1} \Sigma_{\theta_j} - m] \right\}
\]

Note that the application of BSG to general item similarity tasks is straightforward. The only requirement is that the data is given in the same format. Specifically, every sentence of words in the data is replaced with a sequence of items. Moreover, if the items are given as sets (where order does not matter), the subsampling procedure in step 5.1 should be followed by shuffling of the sequences or by increasing the window size to the length of the sequence. This results in a Bayesian version of the item2vec method [7].

5 Experimental Setup and Results

In this section, we compare between the BSG and SG algorithms (for SG we used the \texttt{word2vec} implementation). The algorithms are evaluated on two different tasks: the word similarity task [16] and the word analogy task [4].

The word similarity task requires to score pairs of word according to their relatedness. For each pair, a ground truth similarity score is given. The similarity score we used for both models is the cosine similarity. Specifically, for BSG we observed no significant improvement, when applying the similarities from Eqs. (17) and (18), instead of the cosine similarity. In order to compare between the BSG and SG methods, we compute for each method the Spearman [17] rank correlation coefficient with respect to the ground truth.

The word analogy task is essentially a completion task: a bank of questions in the form of ‘\( w_a \) is to \( w_b \) as \( w_c \) is to ?’ is given, where the task is to replace ? with the correct word \( w_d \). The questions are divided to syntactic questions such as ‘onion is to onions as lion is to lions’ and semantic questions, e.g. ‘Berlin is to Germany as London is to England’.

The method we use to answer the questions is by reporting the word \( w_d \) that gives the highest cosine similarity score between \( u_{w_a} \) and \( u_{w_d} - u_{w_a} + u_{w_c} \). For the BSG and SG models we

\[1\] https://code.google.com/p/word2vec
used $\mu_w$ and $u_w$ as the representation for the word $w$, respectively.

## 5.1 Datasets

We trained both models on the corpus from [18]. In order to accelerate the training process, we limited our vocabulary to the 30K most frequent words in the corpus and discarded all other words. Then, we randomly sampled a subset of 2.8M ‘sentences’ that results in a total text length of 66M words.

The word similarity evaluation includes several different datasets: WordSim353 [16], SCWS [19], Rare Words (RW) [20], MEN [21] and SimLex999 [22]. The reader is referred to the references for further details about these datasets. For each combination of dataset and method, we report the Spearman rank correlation.

The word analogy evaluation dataset [4] consists of 14 distinct groups of analogy questions, where each group contains a different number of questions. Both models were evaluated on an effective set that contains 14122 questions (all questions that contains out-of-vocabulary words were discarded).

## 5.2 Parameters Configuration

The same parameters configuration was set for both systems. Specifically, we set the target representation dimension $m = 40$, maximal window size $c_{\text{max}} = 4$, subsampling parameter $\rho = 10^{-5}$, vocabulary size $l = 30000$ and negative to positive ratio $N = 1$. For BSG, we further set $\tau = 1$, $x = 10$ and $\gamma = 0.7$ (note that BSG is quite robust to the choice of $\gamma$ as long as $0.5 < \gamma \leq 1$). Both models were trained for $K = 40$ iterations. In order to mitigate the effect of noise in the results, we trained 10 different instances of BSG and SG and report the average score that is obtained for each entry in the tables.

## 5.3 Results

Table 1 presents the (average) Spearman rank correlation score (x100) obtained by BSG and SG on the word similarity task for various datasets.

Table 2 presents the (average) percentage of correct answers for each model per questions group on the word analogy task. We see that the models are competitive where BSG results in a better total accuracy. Examining the results from both tables, we notice that in most cases, BSG provides better results than SG. This might be explained by the fact that BSG leverages information from second moments as well.

Comparing our results with the literature [1], [4], we see that the scores obtained by both models are lower. This might be explained by several reasons: First, we use a much smaller corpus of 66M words vs. 6 – 30 billion in [1], [4]. Second, the target representation dimension we use is 40 vs. 100 – 600 in [1], [4]. Third, our models were trained for 40 iterations, where the models in [1] were trained for 50 – 100 iterations. Therefore, we believe that the performance of our models

### Table 1: A Comparison between BSG and SG on Various Word Similarity Datasets

| Method | WordSim353 [16] | SCWS [19] | RW [20] | MEN [21] | SimLex999 [22] |
|--------|-----------------|-----------|---------|----------|----------------|
| SG     | 58.6            | 59.4      | 51.2    | 60.9     | 26.4           |
| BSG    | 61.1            | 59.3      | 52.5    | 61.1     | 27.3           |

### Table 2: A Comparison between BSG and SG on the Word Analogy Task [4]

| Questions group name          | BSG (%) | SG (%) |
|-------------------------------|---------|--------|
| Capital common countries      | 62.4    | 59.5   |
| Capital world                 | 53.2    | 50.2   |
| Currency                      | 7.1     | 3.6    |
| City in state                 | 14.3    | 17.9   |
| Family                        | 55.7    | 63.3   |
| Adjective to adverb           | 20.1    | 15.2   |
| Opposite                      | 6.7     | 6.7    |
| Comparable                    | 47.2    | 43     |
| Superlative                   | 41      | 48.1   |
| Present participle            | 39      | 36.2   |
| Nationality adjective         | 88.9    | 82.3   |
| Past tense                    | 43.7    | 39.2   |
| Plural                        | 46.9    | 42.3   |
| Plural verbs                  | 29.4    | 29.1   |
| Total                         | 45.1    | 42.5   |
can be improved significantly by increasing the representation dimension, the number of iterations and the amount of training data. Yet, it is important to clarify that our main goal is to show that BSG is an effective word embedding method and provides competitive results when compared to the SG method.

6 Conclusion

In this paper, we introduced the BSG algorithm, which is a Bayesian version of the SG method. Specifically, BSG is based on a VB solution to the SG objective. We provide the mathematical derivation of the proposed solution as well as step by step algorithm that is straightforward to implement. We present experimental results on various linguistic tasks that show that BSG and SG are competitive.

In future we plan to investigate the application of BSG to general item similarity tasks [7].

References

[1] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global Vectors for Word Representation,” *EMNLP*, pp. 1532–1543, 2014.

[2] R. Collobert and J. Weston, “A unified architecture for natural language processing: Deep neural networks with multitask learning,” in *ICML*, 2008, pp. 160–167.

[3] A. Mnih and G. E. Hinton, “A Scalable Hierarchical Distributed Language Model” *NIPS*, pp. 1–8, 2008.

[4] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” *Nips*, pp. 1–9, 2013.

[5] A. Frome, G. Corrado, and J. Shlens, “Devise: A deep visual-semantic embedding model,” *NIPS*, pp. 1–11, 2013.

[6] A. Lazaridou, T. P. Nghia, and M. Baroni, “Combining Language and Vision with a Multimodal Skip-gram Model,” *Proceedings of Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL*, pp. 153–163, 2015.

[7] O. Barkan and N. Koenigstein, “Item2Vec: Neural Item Embedding for Collaborative Filtering,” *arXiv preprint arXiv:160304259*, 2016.

[8] L. Vilnis and A. McCallum, “Word Representations via Gaussian Embedding,” *ICLR*, p. 12, 2015.

[9] J. Zhang, J. Salwen, M. Glass, and A. Gliozzo, “Word Semantic Representations using Bayesian Probabilistic Tensor Factorization,” *EMNLP*, 2014.

[10] C. M. Bishop, *Pattern Recognition and Machine Learning*, vol. 4, no. 4, 2006.

[11] U. Paquet and N. Koenigstein, “One-class Collaborative Filtering with Random Graphs,” *Proceedings of the 22nd International Conference on World Wide Web*, pp. 999–1008, 2013.

[12] J. Garten, K. Sagae, V. Ustun, and M. Dehghani, “Combining Distributed Vector Representations for Words,” in *Proceedings of NAACL-HLT*, 2015, pp. 95–101.

[13] T. S. Jaakkola and M. I. Jordan, “A variational approach to Bayesian logistic regression models and their extensions,” *Aistats*, no. AUGUST 2001, 1996.

[14] H. Robbins and S. Monro, “A Stochastic Approximation Method,” *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[15] D. J. C. MacKay, “The Evidence Framework Applied to Classification Networks,” *Neural Computation*, vol. 4, no. 5, pp. 720–736, 1992.

[16] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin, “Placing search in context: the concept revisited,” *ACM Transactions on Information Systems*, vol. 20, no. 1, pp. 116–131, 2002.

[17] C. Spearman, “The proof and measurement of association between two things. By C. Spearman, 1904.,” *The American journal of psychology*, vol. 40, no. 3–4, pp. 441–471, 1987.

[18] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, “One billion word benchmark for measuring progress in statistical language modeling,” in *Proceedings of INTERSPEECH*, 2014, pp. 2635–2639.

[19] E. H. Huang, R. Socher, C. D. Manning, and A. Ng, “Improving word representations via global context and multiple word prototypes,” *ACL 2012*, pp. 873–882.

[20] M.T. Luong, R. Socher, and C. D. Manning, “Better Word Representations with Recursive Neural Networks for Morphology,” *CoNLL-2013*, pp. 104–113, 2013.

[21] E. Bruni, N. K. Tran, and M. Baroni, “Multimodal distributional semantics,” *Journal of Artificial Intelligence Research*, vol. 49, pp. 1–47, 2014.

[22] F. Hill, R. Reichart, and A. Korhonen, “SimLex-999: Evaluating Semantic Models with (Genuine) Similarity Estimation,” *Computational Linguistics*, vol. 41, no. 4, pp. 665–695, 2015.