Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Distributed Negative Sampling for Word Embeddings

Stergios Stergiou
Yahoo Research
stergios@yahoo-inc.com

Zygimantas Straznickas
MIT
zygi@mit.edu

Rolina Wu
University of Waterloo
rolina.wu@uwaterloo.ca

Kostas Tsioutsiouliklis
Yahoo Research
kostas@yahoo-inc.com

Abstract

Word2Vec recently popularized dense vector word representations as fixed-length features for machine learning algorithms and is in widespread use today. In this paper we investigate one of its core components, Negative Sampling, and propose efficient distributed algorithms that allow us to scale to vocabulary sizes of more than 1 billion unique words and corpus sizes of more than 1 trillion words.

Introduction

Recently, Mikolov et al (Mikolov et al. 2013; Mikolov and Dean 2013) introduced Word2Vec, a suite of algorithms for unsupervised training of dense vector representations of words on large corpora. Resulting embeddings have been shown (Mikolov and Dean 2013) to capture semantic word similarity as well as more complex semantic relationships through linear vector manipulation, such as vec("Madrid") - vec("Spain") + vec("France") $\approx$ vec("Paris"). Word2Vec achieved a significant performance increase over the state-of-the-art while maintaining or even increasing quality of the results in Natural Language Processing (NLP) applications.

More recently, novel applications of word2vec involving composite "words" and training corpora have been proposed. The unique ideas empowering the word2vec models have been adopted by researchers from other fields to tasks beyond NLP, including relational entities (Bordes et al. 2013; Socher et al. 2013), general text-based attributes (Kiros, Zemel, and Salakhutdinov 2014), descriptive text of images (Kiros, Salakhutdinov, and Zemel 2014), nodes in graph structure of networks (Perozzi, Al-Rfou, and Skiena 2014), queries (Grbovic et al. 2015) and online ads (Ordentlich et al. 2016). Although most NLP applications of word2vec do not require training of large vocabularies, many of the above-mentioned real-world applications do. For example, the number of unique nodes in a social network (Perozzi, Al-Rfou, and Skiena 2014), or the number of unique queries in a search engine (Grbovic et al. 2015), can easily reach few hundred million, a scale that is not achievable using existing word2vec systems.

Training for such large vocabularies presents several challenges. Word2Vec needs to maintain two d-dimensional vectors of single-precision floating point numbers for each vocabulary word. All vectors need to be kept in main memory to achieve acceptable training latency, which is impractical for contemporary commodity servers. As a concrete example, Ordentlich et al. (Ordentlich et al. 2016) describe an application in which they use word embeddings to match search queries to online ads. Their dictionary comprises $\approx$ 200 million composite words which implies a main memory requirement of $\approx$ 800 GB for $d = 500$. A complementary challenge is that corpora sizes themselves increase. The largest reported dataset that has been used was 100 billion words (Mikolov and Dean 2013) which required training time in the order of days. Such training speeds are impractical for web-scale corpus sizes or for applications that require frequent retraining in order to avoid staleness of the models. Even for smaller corpora, reduced training times shorten iteration cycles and increase research agility.

Word2Vec is an umbrella term for a collection of word embedding models. The focus of this work is on the Skip-Grams with Negative Sampling (SGNS) model that has been shown (Mikolov and Dean 2013) experimentally to perform better, especially for larger corpora. The central component of SGNS is Negative Sampling (NS). Our main contributions are: (1) a novel large-scale distributed SGNS training system developed on top of a custom graph processing framework; (2) a collection of approximation algorithms for NS that maintain the quality of single-node implementations while offering significant speedups; and (3) a novel hierarchical distributed algorithm for sampling from a discrete distribution. We obtain results on a corpus created from the top 2 billion web pages of Yahoo Search, which includes **1.066 trillion words** and a **dictionary of 1.42 billion words**. **Training time per epoch is 2 hours** for typical hyperparameters. To the best of our knowledge, this is the first work that has been shown to scale to corpora of more than 1 trillion words and out-of-memory dictionaries.

Related Work

Dense word embeddings have a rich history (Hinton, Mcclelland, and Rumelhart 1986; Elman 1990; Hinton, Rumelhart, and Williams 1985), with the first popular Neural Network Language Model (NNLM) introduced in (Bengio et

al. 2003). Mikolov introduced the concept of using word vectors constructed with a simple hidden layer to train the full NNLM (Mikolov 2008; Mikolov et al. 2009) and was the first work to practically reduce the high dimensionality of bag-of-words representations to dense, fixed-length vector representations. This model later inspired Mikolov's Word2Vec system (Mikolov and Dean 2013) which has since been implemented and improved by many researchers.

Single Node Implementations

The reference implementation of word2vec, later implemented into (Gensim 2016), (TensorFlow 2016) and (Medallia 2016), uses multithreading to increase training speed. Gensim improves its word2vec implementation three-fold by integrating BLAS primitives. Since SGNS word2vec maintains two matrices of size Words x Dimensions, limitations on main memory and single machine performance make it impossible for single-node implementations to scale up with dictionary and corpus size.

Distributed Implementations

Examples of distributed training systems that have been presented are (Deeplearning4j 2016), (MLLib 2016), Ji et al's (Ji et al. 2016) and Ordentlich et al's (Ordentlich et al. 2016). Deeplearning4j adopts GPUs to accelerate learning times. Spark MLLib's word2vec implementation uses an approach that is similar to Ji et al's (Ji et al. 2016). Both implementations replicate the word embeddings on multiple executors, which are then periodically synchronized among themselves. The input corpus is partitioned onto each executor which independently updates its local copy of the embeddings. This technique reduces training time, yet the memory required to store all vectors in all Spark executors makes this approach unsuitable for large dictionaries, although it addresses well the problem of large corpora.

Ordentlich et al. provide an alternative approach in (Ordentlich et al. 2016). Their implementation scales to large vocabularies using a Parameter Server (PS) framework to store the latest values of model parameters. The word embedding matrices are distributed over a predefined number k of worker nodes by assigning each node with $1/k$th of the columns (dimensions). The worker nodes then perform partial inner product sum operations on their corresponding segment, upon request by a separate computation node. This parameter server approach enabled the models to be trained over large vocabularies. Nevertheless, it is still hard to scale up the system since the number of executors is limited to the dimensionality of the word embeddings. In fact it only scales up to 10-20 computing nodes for typical model parameters, while the average throughput they obtained is $1.6 \cdot 10^6$ words per second because of this limitation.

Continuous Skip-gram Model

For each skip-gram (w_I, w_O) and a set of negative samples W_{neg}, Mikolov et al. (Mikolov and Dean 2013) define the Skip-gram Negative Sampling loss function as:

$$E = -\log \sigma(\mathbf{v}'^T_{w_O} \mathbf{v}_{w_I}) - \sum_{w_j \in W_{neg}} \log \sigma(-\mathbf{v}'^T_{w_j} \mathbf{v}_{w_I})$$

Algorithm 1 SGNS Word2Vec

function UPDATE(η)
  for all skip-grams w_I, w_O **do**
    $W_{neg} \leftarrow \{\}$
    for $i \in [1, N]$ **do**
      $W_{neg} \leftarrow W_{neg} \cup sample(P_n)$
    $\mathbf{g}_{w_O} \leftarrow (\sigma(\mathbf{v}'^T_{w_O} \mathbf{v}_{w_I}) - 1)\mathbf{v}_{w_I}$ ▷ get gradients
    $\mathbf{g}_{w_I} \leftarrow (\sigma(\mathbf{v}'^T_{w_O} \mathbf{v}_{w_I}) - 1)\mathbf{v}_{w_O}$
    for all $w_j \in W_{neg}$ **do**
      $\mathbf{g}_{w_j} \leftarrow \sigma(\mathbf{v}'^T_{w_j} \mathbf{v}_{w_I})\mathbf{v}'_{w_I}$
      $\mathbf{g}_{w_I} \leftarrow \mathbf{g}_{w_I} + \sigma(\mathbf{v}'^T_{w_j} \mathbf{v}_{w_I})\mathbf{v}'_{w_j}$
    $\mathbf{v}'_{w_O} \leftarrow \mathbf{v}'_{w_O} - \eta \cdot \mathbf{g}_{w_O}$ ▷ update vectors
    $\mathbf{v}_{w_I} \leftarrow \mathbf{v}_{w_I} - \eta \cdot \mathbf{g}_{w_I}$
    for all $w_j \in W_n$ **do**
      $\mathbf{v}'_{w_j} \leftarrow \mathbf{v}'_{w_I} - \eta \cdot \mathbf{g}_{w_j}$

where $\sigma(x) = \frac{1}{1+e^{-x}}$, and $\mathbf{v}_w$ and $\mathbf{v}'_w$ are the "input" and "output" vector representations for word w. Its gradients with respect to the word vectors are:

$$\frac{\partial E}{\partial \mathbf{v}'_{w_O}} = (\sigma(\mathbf{v}'^T_{w_O} \mathbf{v}_{w_I}) - 1)\mathbf{v}_{w_I}$$

$$\frac{\partial E}{\partial \mathbf{v}'_{w_j}} = \sigma(\mathbf{v}'^T_{w_j} \mathbf{v}_{w_I})\mathbf{v}_{w_I}$$

$$\frac{\partial E}{\partial \mathbf{v}_{w_I}} = (\sigma(\mathbf{v}'^T_{w_O} \mathbf{v}_{w_I}) - 1)\mathbf{v}_{w_O} + \sum_{w_j \in W_{neg}} \sigma(\mathbf{v}'^T_{w_j} \mathbf{v}_{w_I})\mathbf{v}'_{w_j}$$

The loss function is then minimized using a variant of stochastic gradient descent, yielding a simple learning algorithm that iterates over all the skip-grams, calculates the gradients of their loss functions and updates the values of their respective vectors, as well as the vectors of negative samples chosen for each skip-gram. A summary of the model is depicted in Algorithm 1.

Distributed Discrete Sampling

Central to SGNS is sampling from a noise distribution $P_n(w) : w \in W$ over the vocabulary W. Mikolov et al. (Mikolov and Dean 2013) experimentally showed that the unigram distribution $U(w)$ raised to the $3/4$-th power significantly outperformed both the $U(w)$ and uniform distributions. Negative Sampling lies at the core of the performance improvements obtained by Word2Vec as it allows for updating the loss function in $O(kd)$ instead of $O(|W|d)$, where k is the number of noise samples per skip gram and d the dimensionality of the embedding. Nevertheless, it remains the dominant time component of the algorithm.

It is desirable to be able to draw from the noise distribution in constant time. Word2Vec (Mikolov and Dean 2013) achieves this by using the following algorithm. It creates an array $\mathcal{A}$ (of size $|\mathcal{A}| \gg |W|$) in which it places word identifiers proportionally to their probability in $P_n(w)$ and draws

Algorithm 2 Alias Method

function BUILDALIAS($W, P_n(w)$)
  $\forall w \in W : S[w] = P_n(w) \cdot |W|$
  $\forall w \in W : A[w] = w$
  $T_L = \{w | P_n(w) < 1/|W|\}$
  $T_H = \{w | P_n(w) > 1/|W|\}$
  for $j \in T_L$ **do**
    $k \leftarrow POP(T_H)$ ▷ remove an element from T_H
    $S[k] = S[k] - 1 + S[j]$
    $A[j] = k$
    if $S[k] < 1$ **then**
      $T_L = T_L \cup \{k\}$
    else if $S[k] > 1$ **then**
      $T_H = T_H \cup \{k\}$
    $T_L = T_L \backslash \{j\}$
  return S, A
function SAMPLEALIAS(S, A)
  $u = Sample(\mathcal{U}\{0, |W|\}$ ▷ real sample $\in [0, |W|)$
  if S[$\lfloor u \rfloor$] $\leq u - \lfloor u \rfloor$ **then**
    return $A[\lfloor u \rfloor]$
  else
    return $\lfloor u \rfloor$

uniformly from $\mathcal{A}$. As a concrete example, assuming a vocabulary $W = \{A, B, C, D\}$ where $P_n(A) = 0.5, P_n(B) = 0.25, P_n(C) = 0.15, P_n(D) = 0.1$, array $\mathcal{A}$ might be:

$$[A,A,A,A,A,A,A,A,A,A, B,B,B,B,B, C,C,C, D,D].$$

This algorithm is approximate as probabilities in $P_n(w)$ are quantized to $1/|\mathcal{A}|$ and also requires significant memory resources for larger vocabularies (for instance, $|\mathcal{A}|/|W| > 100$ in the reference implementation.) In the following section we present an algorithm for drawing from $P_n(w)$ in constant time that only requires $O(|W|)$ space and $O(|W|)$ pre-processing time.

Alias Method

Let $P_n(w) : w = [0, .., |W| - 1]$ be a discrete distribution over W. We construct two arrays S, A, where $|S| = |A| = |W|$, initialized to $S[i] = P_n(i) \cdot |W|$ and $A[i] = i$. While there exists j such that $S[j] < 1$, select an arbitrary $k : S[k] > 1$ and set $A[j] = k$, $S[k] = S[k] - 1 + S[j]$. Entry j is never examined thereafter. Such a selection for k is always possible as $\sum_w S[w] = |W|$.

We will use arrays S, A to draw from $P_n(w)$ in constant time as follows: Let $u \in [0, |W|)$ be a draw from the uniform continuous distribution $\mathcal{U}\{0, |W|\}$, obtained in constant time. Then

$$q = \begin{cases} \lfloor u \rfloor & \text{if } S[\lfloor u \rfloor] > u - \lfloor u \rfloor \\ A[\lfloor u \rfloor] & \text{otherwise} \end{cases}$$

is a draw from $P_n(w)$, also obtained in constant time.

Let us examine the construction of arrays S, A for $P_n(A) = 0.5, P_n(B) = 0.25, P_n(C) = 0.15, P_n(D) = 0.1$ from our previous example. Arrays S, A are initialized to:

S	2.0	1.0	0.6	0.4
A	0	1	2	3

Let us first pick $j = 3$ and $k = 0$ since $S[3] < 1.0, S[0] > 1.0$. The arrays are updated to:

S	**1.4**	1.0	0.6	0.4
A	0	1	2	**0**

Following, we select $j = 2$ and $k = 0$ since $S[2] < 1.0, S[0] > 1.0$. Notice that $j = 3$ is no longer a valid selection as it has already been examined. The arrays are updated to:

S	**1.0**	1.0	0.6	0.4
A	0	1	**0**	0

To gain some insight on this algorithm, we observe that for each word w, either $S[w] = 1$ or $A[w] \neq w$, and that $P_n(w) = (S[w] + \sum_{w' \neq w, A[w']=w} 1 - S[w'])/|W|$. We also note that constructing the arrays S, A requires $O(|W|)$ time: Initially, all words are separated into three sets T_H, T_M, T_L, containing words with $P_n(w) > 1/|W|$, $P_n(w) = 1/|W|$ and $P_n(w) < 1/|W|$ respectively, in $O(|W|)$ time. The algorithm iterates over set T_L and "pairs" each word $j \in T_L$ with a word $k \in T_H$. If $S[k]$ is reduced to less than 1.0 then k is added to T_L. The algorithm terminates when $T_L = \emptyset$. The alias method was studied in (Walker 1974; 1977; Vose 1991; Marsaglia et al. 2004) and is presented in Algorithm 2.

Hierarchical Sampling

In a distributed context, where vocabulary W is (ideally) equi-partitioned onto p computation nodes, even assigning $O(|W|)$ space for sampling on each node may be wasteful. We present a hierarchical sampling algorithm that still allows for drawing from $P_n(w)$ in constant time, but only requires $O(p + |W|/p)$ space per partition and $O(p + |W|/p)$ preprocessing time. While the alias method is an improvement over the sampling algorithm in (Mikolov and Dean 2013), it still requires $O(|W|)$ memory, which as the dictionary size increases, may become prohibitively large. For instance, for a dictionary size of 1 billion words, an implementation of the alias method may require $16GB$ of memory on *every* computation node. In this section we propose a two-stage sampling algorithm that still allows to draw from $P_n(w)$ in constant time. However each partition only maintains information about the subset of the vocabulary it is responsible for (in addition to some low overhead auxiliary information.

Let W be the set of all words and $W_i, 1 \leq i \leq p$ be its p partitions such that $\bigcup_i W_i = W$ and $W_i \cap W_j = \emptyset$ for all $1 \leq i \neq j \leq p$. Let $c(u)$ be some positive function of the word u and T be a probability distribution of sampling a word u from the whole set W:

$$T(u) = \frac{c(u)}{\sum_{w \in W} c(w)} \tag{1}$$

For each partition i, let its word probability distribution be:

$$T_i(u) = \begin{cases} \frac{c(u)}{\sum_{w \in W_i} c(w)} & \text{if } u \in W_i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Consider a sampling process where first a partition k is sampled from a probability distribution P, then a word is

Algorithm 3 Hierarchical Sampling

function BUILDTABLES($c(u), W_i : i \in [1..p]$)
  $P(k) = \frac{\sum_{w \in W_k} c(w)}{\sum_{w \in W} c(w)}$
  $[S_P, A_P]$ =BUILDALIAS($[1..p], P(k)$)
  for $j \in [1..p]$ **do**
    $T_j(u) = \begin{cases} \frac{c(u)}{\sum_{w \in W_j} c(w)} & \text{if } u \in W_j \\ 0 & \text{otherwise} \end{cases}$
    $[S_j, A_j]$ =BUILDALIAS($W_j, T_j(u)$)
function SAMPLE($[S_P, A_P], [S_i, A_i] : i \in [1..p]$)
  k =SAMPLEALIAS(S_P, A_P)
  return SAMPLEALIAS(S_k, A_k)

sampled from the probability distribution T_k. Our goal is to choose P such that the total distribution of sampling a word using this process is equal to T.

Assume the word u is in partition k. Then the probability of sampling it is:

$$\sum_{i=1}^{N} P(i)T_i(u) = P(k)T_k(u) = P(k)\frac{c(u)}{\sum_{w \in W_k} c(w)}$$

We would like this to be equal to T. Then:

$$P(k)\frac{c(u)}{\sum_{w \in W_k} c(w)} = \frac{c(u)}{\sum_{w \in W} c(w)}$$

Therefore:

$$P(k) = \frac{\sum_{w \in W_k} c(w)}{\sum_{w \in W} c(w)}. \quad (3)$$

We perform hierarchical sampling as follows. First, we obtain an auxiliary probability distribution over the set of all partitions, as defined in Eq. 3. This distribution is made available to all computation nodes and is low-overhead as $p << |W|$. We construct arrays S_P, A_P as per the Alias method. For each partition i we additionally construct arrays S_i, A_i using the word distributions in Eq. 2. We can now sample from distribution $T(u)$ (Eq. 1) as follows: We first draw a partition j from $P(k)$ and subsequently draw a word w from $T_j(u)$. We observe that preprocessing is now $O(|W_i| + p)$ for partition i while drawing from $T(u)$ is still a constant-time operation. The algorithm is summarized in Algorithm 3.

Distributed SGNS

Preliminaries

We closely follow the reference SGNS implementation with respect to the conversion from an input corpus to the set of skip-grams. In particular, we reject input word w_i with probability:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

where t is a frequency cutoff, typically set to 10^{-5} and $f(w)$ are the unigram frequencies.

Each word w and its associated information is placed on a specific partition $M(h(w))$, determined by $h(w)$ which is the 128-bit md5sum of w. During each epoch, the input is subsampled and translated into a set of skip-grams (w_i, w_o). All skip-grams related to the input word w_i are then sent to partition $M(h(w_i))$. To support efficient subsampling, we maintain all word identifiers with frequency $> t$ on all partitions. This set of words is typically a very small subset of the vocabulary.

Considering dictionary words as nodes in a graph G and skip-grams as directed edges between them, SGNS can be seen as a distributed computation over G. Following, we provide a brief overview of our custom distributed graph processing framework on which we developed the algorithms presented in this work. We then present our three distributed algorithms for negative sampling, that differ only in the approach with which they obtain the negative samples and update the related word vectors. The learning phase of the epoch varies according to the negative sampling algorithm used and dominates the execution time.

Graph Processing Framework

Chronos is a proprietary in-memory / secondary-storage hybrid graph processing system, developed in C++11 on top of Hadoop MapReduce. Word-related state is maintained in memory, while the graph itself, specifically a serialization of the skip grams information, can either be stored in memory or on disk. Messages originating from a source process are transparently buffered before being sent to target processes, with buffering occurring separately for each target partition. Each message contains the related word embedding as its payload. The framework can optionally [de]compress buffers on-the-fly, transparently to the application. Communication is performed via a large clique of unidirectional point-to-point message queues. The system builds on the following low-level communication primitives: clique communication for graph level computations; and one-to-all and all-to-one communication for process-level computations. It implements loading and storing of vertex information through which it supports fault tolerance via check-pointing. Each superstep is executed as a sequence of three primitives: a one-to-all communication primitive that sends initialization parameters to all peers, a clique communication primitive that transmits edge messages, and an all-to-one communication primitive that gathers partition-level results from the peers to the master process.

Baseline Negative Sampling

In *Baseline Negative Sampling (BNS)*, each partition iterates over the skip-grams it maintains. For each skip-gram (w_I, w_O), it sends the input vector $\mathbf{v}_{w_I}$ to partition $p_0 = M(h(w_O))$, which is the partition that owns word w_O. Upon reception of $\mathbf{v}_{w_I}$, partition p_O updates output vector $\mathbf{v}'_{w_O}$ and sends back gradient $\mathbf{g}_{w_I}$ to the source partition for updating $\mathbf{v}_{w_I}$. BNS draws N samples $p_1, .., p_N$ from the partitions distribution. Subsequently, for each p_i it sends $\mathbf{v}_{w_I}$ to partition p_i. Upon reception, partition p_i draws a negative word w_{neg} from its own word distribution, updates $\mathbf{v}'_{w_{neg}}$ and sends back gradient $\mathbf{g}_{w_I}$ to the source partition

Algorithm 4 Baseline Negative Sampling

function PROCESSSKIPGRAM(w_I, w_O, η)
  $\mathbf{g}_{w_I} \leftarrow$ GETREMOTE($M(h(w_O)), \mathbf{v}_{w_I}, w_O$)
  for all $i \in [1, N]$ **do**
    $p \leftarrow$SAMPLEALIAS(S_P, A_P)
    $\mathbf{g}_{w_I} \leftarrow \mathbf{g}_{w_I} +$ GETNEGSAMPLE($p, \mathbf{v}_{w_I}$)
  $\mathbf{v}_{w_I} \leftarrow \mathbf{v}_{w_I} - \eta \cdot \mathbf{g}_{w_I}$
function GETREMOTE($\mathbf{v}_{w_I}, w_O$)
  $\mathbf{g}_{w_O} \leftarrow (\sigma(\mathbf{v}'^T_{w_O}\mathbf{v}_{w_I}) - 1)\mathbf{v}_{w_I}$
  $\mathbf{g}_{w_I} \leftarrow (\sigma(\mathbf{v}'^T_{w_O}\mathbf{v}_{w_I}) - 1)\mathbf{v}'_{w_O}$
  $\mathbf{v}'_{w_O} \leftarrow \mathbf{v}'_{w_O} - \eta \cdot \mathbf{g}_{w_O}$
  return $\mathbf{g}_{w_I}$
function GETNEGSAMPLE($\mathbf{v}_{w_I}$)
  $w_{neg} \leftarrow$ SAMPLEALIAS(S_k, A_k) ▷ k is partition id
  $\mathbf{g}_{w_{neg}} \leftarrow \sigma(\mathbf{v}'^T_{w_{neg}}\mathbf{v}_{w_I})\mathbf{v}_{w_I}$
  $\mathbf{g}_{w_I} \leftarrow \mathbf{g}_{w_I} + \sigma(\mathbf{v}'^T_{w_{neg}}\mathbf{v}_{w_I})\mathbf{v}'_{w_{neg}}$
  $\mathbf{v}'_{w_{neg}} \leftarrow \mathbf{v}'_{w_{neg}} - \eta \cdot \mathbf{g}_{w_{neg}}$
  return $\mathbf{g}_{w_I}$

for updating $\mathbf{v}_{w_I}$. The communication across partitions is depicted in Figure 1 while the algorithm is summarized in Algorithm 4. We observe that, while BNS closely adheres to the reference implementation, this comes at a cost of sending $2(N+1)$ vectors over the network, which may be prohibitively expensive for large corpora.

Single Negative Sampling

A reasonable approximation to BNS is to restrict sampling to pools of words whose frequencies still approximate the original distribution while being faster to learn from. We present the first such approximation, *Single Negative Sampling (SNS)*. SNS behaves similarly to BNS with respect to the output word w_O: it transfers input vector $\mathbf{v}_{w_I}$ to partition $p_0 = M(h(w_O))$, which subsequently updates output vector $\mathbf{v}'_{w_O}$ and sends back gradient $\mathbf{g}_{w_I}$ to the source partition for updating $\mathbf{v}_{w_I}$. However, it draws all negative samples for a given skip-gram from a single partition drawn from the partitions distribution. It is presented in Algorithm 5. We observe that SNS scales much better with N since it requires communicating 4 vectors per skip-gram over the network, a significant reduction from $2(N+1)$.

Target Negative Sampling

Target Negative Sampling (TNS) aims to further reduce the communication cost. We observe that, if negative samples are drawn from partition $p_0 = M(h(w_O))$ for skip-gram

CPU	2x Intel Xeon E5-2620
Frequency	2.5GHz (max)
RAM	64GB
Memory Bandwidth	42.6 GB/s (max)
Network	10Gbps Ethernet

Table 1: Cluster Node Configuration

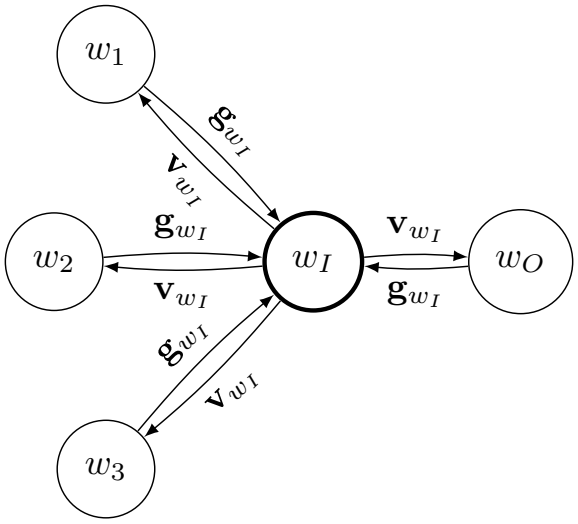

Figure 1: Baseline Negative Sampling Communication Pattern during processing of skip-gram (w_I, w_O), for 3 negative samples w_1, w_2, w_3.

(w_I, w_O), the gradient update to $\mathbf{v}_{w_I}$ that corresponds to the negative samples can be added to the gradient that p_0 is returning to the source partition. This allows for a communication pattern of 2 vectors per processed skip-gram. TNS is presented in Algorithm 6. Let us examine the set of skip-grams with the same input word w_I: $\{(w_I, w_{O_j})\}$. The possible partitions from which TNS will be drawing negative samples from is $R = \cup_j\{M(h(w_{O_j}))\}$. Considering that the number of partitions is typically much smaller than the dictionary size, R is likely a large subset of the set of all partitions R_A. However, it is not necessary a true random subset of R_A, while additionally it remains fixed across epochs (modulo any variation induced by subsampling.) Nevertheless, our experiments demonstrate that TNS performs well in practice.

Experiments

We collect experimental results on a cluster of commodity nodes whose configuration is depicted on Table 1.

We establish the quality of the embeddings obtained by

Model	Accuracy [%]		
	Semantic	Syntactic	Total
SGNS (Reference)	74.27	69.35	71.58
SG-BNS	73.10	66.16	69.31
SG-SNS	73.48	68.08	70.53
SG-TNS	74.72	66.13	70.03

Table 2: Accuracy Results on Google Analogy Dataset for Single-Epoch Trained Embeddings on the Composite Corpus for $d = 300$, $win = 5$, $neg = 5$, $t = 10^{-5}$, $min - count = 10$ and $threads = 20$. 350 partitions are used for BNS, SNS and TNS.

Algorithm 5 Single Negative Sampling

function PROCESSSKIPGRAM(w_I, w_O, η)
  $\mathbf{g}_{w_I} \leftarrow$ GETREMOTE($M(h(w_O))$, $\mathbf{v}_{w_I}, w_O$)
  $p \leftarrow$SAMPLEALIAS(S_P, A_P)
  $\mathbf{g}_{w_I} \leftarrow \mathbf{g}_{w_I} +$ GETNEGSAMPLES($p, \mathbf{v}_{w_I}$)
  $\mathbf{v}_{w_I} \leftarrow \mathbf{v}_{w_I} - \eta \cdot \mathbf{g}_{w_I}$
  %Statex
function GETREMOTE($\mathbf{v}_{w_I}, w_O$)
  $\mathbf{g}_{w_O} \leftarrow (\sigma(\mathbf{v}'^T_{w_O}\mathbf{v}_{w_I}) - 1)\mathbf{v}_{w_I}$
  $\mathbf{g}_{w_I} \leftarrow (\sigma(\mathbf{v}'^T_{w_O}\mathbf{v}_{w_I}) - 1)\mathbf{v}'_{w_O}$
  $\mathbf{v}'_{w_O} \leftarrow \mathbf{v}'_{w_O} - \eta \cdot \mathbf{g}_{w_O}$
  return $\mathbf{g}_{w_I}$
function GETNEGSAMPLES($k, \mathbf{v}_{w_I}$)
  $\mathbf{g}_{w_I} \leftarrow 0$
  for $j \in [1..N]$ **do**
    $w_j \leftarrow$SAMPLEALIAS(S_k, A_k) ▷ k is partition id
    $\mathbf{g}_{w_j} \leftarrow \sigma(\mathbf{v}'^T_{w_j}\mathbf{v}_{w_I})\mathbf{v}_{w_I}$
    $\mathbf{g}_{w_I} \leftarrow \mathbf{g}_{w_I} + \sigma(\mathbf{v}'^T_{w_j}\mathbf{v}_{w_I})\mathbf{v}'_{w_j}$=
    $\mathbf{v}'_{w_j} \leftarrow \mathbf{v}'_{w_j} - \eta \cdot \mathbf{g}_{w_j}$
  return $\mathbf{g}_{w_I}$

Model	Learning Time (sec)	Speedup
SGNS (Reference)	11507	1 x
SG-BNS	4378	2.6 x
SG-SNS	1331	8.7 x
SG-TNS	570	20.2 x

Table 3: Single-Epoch Learning Times on the Composite Corpus for $d = 300$, $win = 5$, $neg = 5$, $t = 10^{-5}$, $min - count = 10$ and $threads = 20$. 350 partitions are used for BNS, SNS and TNS.

our algorithms on a composite corpus comprising two news corpora[1][2], the 1 Billion Word Language Model Benchmark[3] (Chelba et al. 2013), the UMBC WebBase corpus[4] (Han et al. 2013) and Wikipedia[5], as constructed by the open-source reference word2vec implementation evaluation scripts.

We report quality results on the Google Analogy evaluation dataset (Mikolov et al. 2013) for the BNS, SNS and TNS algorithms, as well as the reference word2vec implementation, on Table 2. We observe that all three algorithms perform similarly to each other, slightly trailing the reference implementation. On Table 3, we report single-epoch learning times. Specifically, we report a 20X speedup over the reference implementation. We then present scalability

[1]http://www.statmt.org/wmt14/training-monolingual-news-crawl/news.2012.en.shuffled.gz

[2]http://www.statmt.org/wmt14/training-monolingual-news-crawl/news.2013.en.shuffled.gz

[3]http://www.statmt.org/lm-benchmark/1-billion-word-language-modeling-benchmark-r13output.tar.gz

[4]http://ebiquity.umbc.edu/redirect/to/resource/id/351/UMBC-webbase-corpus

[5]http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2

Algorithm 6 Target Negative Sampling

function PROCESSSKIPGRAM(w_I, w_O, η)
  $\mathbf{g}_{w_I} \leftarrow$ GETREMOTE($M(h(w_O))$, $\mathbf{v}_{w_I}, w_O$)
  $\mathbf{v}_{w_I} \leftarrow \mathbf{v}_{w_I} - \eta \cdot \mathbf{g}_{w_I}$
function GETREMOTE($k, \mathbf{v}_{w_I}, w_O$)
  $\mathbf{g}_{w_O} \leftarrow (\sigma(\mathbf{v}'^T_{w_O}\mathbf{v}_{w_I}) - 1)\mathbf{v}_{w_I}$
  $\mathbf{g}_{w_I} \leftarrow (\sigma(\mathbf{v}'^T_{w_O}\mathbf{v}_{w_I}) - 1)\mathbf{v}'_{w_O}$
  for $j \in [1..N]$ **do**
    $w_j \leftarrow$SAMPLEALIAS(S_k, A_k)
    $\mathbf{g}_{w_j} \leftarrow \sigma(\mathbf{v}'^T_{w_j}\mathbf{v}_{w_I})\mathbf{v}_{w_I}$
    $\mathbf{g}_{w_I} \leftarrow \mathbf{g}_{w_I} + \sigma(\mathbf{v}'^T_{w_j}\mathbf{v}_{w_I})\mathbf{v}'_{w_j}$
  $\mathbf{v}'_{w_O} \leftarrow \mathbf{v}'_{w_O} - \eta \cdot \mathbf{g}_{w_O}$
  for all $w_j \in W_{neg}$ **do**
    $\mathbf{v}'_{w_j} \leftarrow \mathbf{v}'_{w_j} - \eta \cdot \mathbf{g}_{w_j}$
  return $\mathbf{g}_{w_I}$

Query-Ads Dataset	Value
Dictionary Words	200 Million
Dataset Words	54.8 Billion
Dataset Size	411.5 GB
Partitions	1500
Learning Time	856 sec

Table 4: Query-Ads Dataset and Single-Epoch Learning Runtime for $d = 300$, $win = 5$ and $neg = 5$, $t = 10^{-5}$

results on two large datasets. The first dataset is obtained from (Ordentlich et al. 2016) and is related to mapping web queries to online ads. Results are shown on Table 4. The second dataset is obtained from 2 billion web pages and comprises more than 1 trillion words and more than 1.4 billion unique dictionary words. Results are shown on Table 5.

Conclusion

In this paper we proposed an efficient distributed algorithm for sampling from a discrete distribution and used it to optimize Negative Sampling for SGNS Word2Vec, allowing us to scale to vocabulary sizes of more than 1 billion words and corpus sizes of more than 1 trillion words. Specifically, our system learns from a web corpus of 1.066 trillion words on 1.42 billion vocabulary words in 2 hours.

Web Dataset	Value
Dictionary Words	1.42 Billion
Dataset Words	1.066 Trillion
Dataset Size	6021.6 GB
Partitions	3000
Learning Time	7344 sec

Table 5: Web Dataset and Single-Epoch Learning Runtime for $d = 300$, $win = 5$, $neg = 5$ and $t = 10^{-5}$

References

Bengio, Y.; Ducharme, R.; Vincent, P.; and Jauvin, C. 2003. A neural probabilistic language model. *Journal of Machine Learning Research* 3:1137–1155.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, 2787–2795.

Chelba, C.; Mikolov, T.; Schuster, M.; Ge, Q.; Brants, T.; and Koehn, P. 2013. One billion word benchmark for measuring progress in statistical language modeling. *CoRR* abs/1312.3005.

Deeplearning4j. 2016. http://deeplearning4j.org/word2vec.html.

Elman, J. L. 1990. Finding structure in time. *Cognitive science* 14(2):179–211.

Gensim. 2016. http://rare-technologies.com/word2vec-in-python-part-two-optimizing/.

Grbovic, M.; Djuric, N.; Radosavljevic, V.; Silvestri, F.; and Bhamidipati, N. 2015. Context-and content-aware embeddings for query rewriting in sponsored search. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 383–392. ACM.

Han, L.; Kashyap, A. L.; Finin, T.; Mayfield, J.; and Weese, J. 2013. UMBC_EBIQUITY-CORE: Semantic Textual Similarity Systems. In *Proceedings of the Second Joint Conference on Lexical and Computational Semantics*. Association for Computational Linguistics.

Hinton, G. E.; Mcclelland, J. L.; and Rumelhart, D. E. 1986. *Distributed representations, Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*. MIT Press, Cambridge, MA.

Hinton, G.; Rumelhart, D.; and Williams, R. 1985. Learning internal representations by back-propagating errors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* 1.

Ji, S.; Satish, N.; Li, S.; and Dubey, P. 2016. Parallelizing word2vec in shared and distributed memory. *arXiv preprint arXiv:1604.04661*.

Kiros, R.; Salakhutdinov, R.; and Zemel, R. S. 2014. Multimodal neural language models. In *ICML*, volume 14, 595–603.

Kiros, R.; Zemel, R.; and Salakhutdinov, R. R. 2014. A multiplicative model for learning distributed text-based attribute representations. In *Advances in Neural Information Processing Systems*, 2348–2356.

Marsaglia, G.; Tsang, W. W.; Wang, J.; et al. 2004. Fast generation of discrete random variables. *Journal of Statistical Software* 11(3):1–11.

Medallia. 2016. https://github.com/medallia/Word2VecJava.

Mikolov, T., and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*.

Mikolov, T.; Kopecky, J.; Burget, L.; Glembek, O.; et al. 2009. Neural network based language models for highly inflective languages. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, 4725–4728. IEEE.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T. 2008. Language models for automatic speech recognition of czech lectures. *Proc. of Student EEICT*.

MLLib, S. 2016. https://spark.apache.org/docs/latest/mllib-feature-extraction.html.

Ordentlich, E.; Yang, L.; Feng, A.; Cnudde, P.; Grbovic, M.; Djuric, N.; Radosavljevic, V.; and Owens, G. 2016. Network-efficient distributed word2vec training system for large vocabularies. *arXiv preprint arXiv:1606.08495*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 701–710. ACM.

Socher, R.; Chen, D.; Manning, C. D.; and Ng, A. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems*, 926–934.

TensorFlow. 2016. https://www.tensorflow.org/.

Vose, M. D. 1991. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on software engineering* 17(9):972–975.

Walker, A. J. 1974. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters* 8(10):127–128.

Walker, A. J. 1977. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)* 3(3):253–256.