Incrementally Learning the Hierarchical Softmax Function for Neural Language Models

Hao Peng,[†] **Jianxin Li,**[†] **Yangqiu Song,**[‡] **Yaopeng Liu**[†]
[†]Department of Computer Science & Engineering, Beihang University, Beijing 100191, China
[‡]Department of Computer Science & Engineering, Hong Kong University of Science and Technology, Hong Kong
{penghao,ljx,liuyp}@act.buaa.edu.cn yqsong@cse.ust.hk

Abstract

Neural network language models (NNLMs) have attracted a lot of attention recently. In this paper, we present a training method that can incrementally train the hierarchical softmax function for NNMLs. We split the cost function to model old and update corpora separately, and factorize the objective function for the hierarchical softmax. Then we provide a new stochastic gradient based method to update all the word vectors and parameters, by comparing the old tree generated based on the old corpus and the new tree generated based on the combined (old and update) corpus. Theoretical analysis shows that the mean square error of the parameter vectors can be bounded by a function of the number of changed words related to the parameter node. Experimental results show that incremental training can save a lot of time. The smaller the update corpus is, the faster the update training process is, where an up to 30 times speedup has been achieved. We also use both word similarity/relatedness tasks and dependency parsing task as our benchmarks to evaluate the correctness of the updated word vectors.

Neural network language models (NNLMs) (Bengio et al. 2003; Morin and Bengio 2005; Mnih and Hinton 2008; Turian, Ratinov, and Bengio 2010; Collobert et al. 2011; Mikolov et al. 2013a; 2013b; Levy and Goldberg 2014; Levy, Goldberg, and Dagan 2015) have attracted a lot of attention recently given their compact representation form and generalization property compared to the traditional lexical representations. It has been applied to many natural language processing tasks such as word similarity/relatedness and word analogy (Mikolov et al. 2013a; 2013b), working as features for part-of-speech tagging, chunking, named entity recognition, etc. (Turian, Ratinov, and Bengio 2010; Collobert et al. 2011). However, the above NNLMs only consider the static training corpus. There are a lot of applications, such as news and tweets text processing, requiring incremental update of the word vectors given the fact that the working domains are fast evolving. When the new corpus is relatively smaller than the old corpus, it will be much less efficient to retrain the word vectors with the combined corpus.

In this paper, we consider the problem of training the NNLMs given new corpora incrementally. In particular, we adopt the popular word2vec tool due to its simplicity and time efficiency, and comparable performance to other NNLMs (Mikolov et al. 2013a; 2013b; Levy, Goldberg, and Dagan 2015). To speedup the process of indexing and querying, word2vec employed two techniques called hierarchical softmax and negative sampling (Mikolov et al. 2013a; 2013b). Hierarchical softmax was first proposed by Mnih and Hinton (Mnih and Hinton 2008) where a hierarchical tree is constructed to index all the words in a corpus as leaves, while negative sampling is developed based on noise contrastive estimation (Gutmann and Hyvärinen 2012), and randomly samples the words not in the context to distinguish the observed data from the artificially generated random noise. It is empirically shown that hierarchical softmax performs better for infrequent words while negative sampling performs better for frequent words (Mikolov et al. 2013b). The reason is that hierarchical softmax builds a tree over the whole vocabulary, and the leaf nodes representing rare words will inevitably inherit their ancestors' vector representations in the tree, which can be affected by other frequent words in the corpus. Thus, we choose hierarchical softmax due to its good performance on rare words, which can benefit the further incremental training for new corpus.

When applying hierarchical softmax to NNLMs, there is a preprocessing step to build a hierarchical tree of words. When we incrementally incorporate more data, the hierarchical tree should be changed to reflect the change of data. For example, in word2vec, it uses the Huffman coding to construct the tree over the vocabulary because Huffman tree uses shorter codes for more frequent words with less visits to the leaves (Gutmann and Hyvärinen 2012), and results in faster training process. When frequencies of words change, Huffman tree will be changed. To handle this problem, we retain the paths from root to leaves of the old tree, and perform prefix matching between the old tree and the new tree. When updating the vectors for the matched ancestors of a leaf, we follow the original CBOW (Continuous Bag-of-Words) or Skip-gram algorithms in word2vec based on stochastic gradient ascent of log-likelihood. When updating the vectors for the different ancestors, we modify the old tree with stochastic gradient descent, while update the new tree with stochastic gradient ascent. In this way, we only modify all the nodes that need to be updated while retaining all the other nodes as the same as the ones trained based on the old corpus. Since the update process is independent for all the internal nodes, we also develop a parallel algorithm similar to word2vec, which makes our algorithm efficient when having more CPUs.

In the experiments, we show that using this updating procedure, we can get almost the same CBOW and Skip-gram models as fully re-trained ones. We check both individual vector's mean square error (MSE) and down-stream applications, word similarity/relatedness and dependency parsing, to prove the correctness of our model. In addition, we also provide a bound that can characterize our algorithm's difference from fully re-trained models.

Background

In this section, we introduce the background of CBOW and Skip-gram models based on the hierarchical softmax function. Suppose the tree has been built given a training corpus $\mathcal{W}$, where all the unique words in $\mathcal{W}$ will be the leaves.

The CBOW Model

In CBOW model, given a sequence of training words $w_1, w_2, \ldots, w_T$ in $\mathcal{W}$, the training objective is to maximize the average log-likelihood function $\sum_{w \in \mathcal{W}} \log P(w|\mathcal{C}(w))$, where w is a word and also refers a leaf node determined by a path from root in Huffman tree. Moreover, we denote $X_w = \sum_{-c \leq j \leq c, j \neq 0} w_j$ as the sum of the context vectors, and $2c$ is the size of training context centered at w. Let L^w be the length of the path, and p_i^w be the i-th node on the path from the root to w, and d_i^w be Huffman code of i-th node in path, where $d_i^w \in \{0, 1\}$ and $i \in \{2, \ldots, L^w\}$. In addition, we denote θ_i^w as the vector representation of the internal node p_i^w and $\theta_{L^w}^w$ is the vector representation of w. Then the log-likelihood can be re-written as:

$$\mathcal{J}_{\text{CBOW}} = \sum_{w \in \mathcal{W}} \log P(w|\mathcal{C}(w)) = \sum_{w \in \mathcal{W}} \sum_{i=2}^{L^w} \ell(w, i), \tag{1}$$

where we denote $\ell(w,i) = (1-d_i^w) \cdot \log[\sigma(X_w^T \theta_{i-1}^w)] + d_i^w \cdot \log[1 - \sigma(X_w^T \theta_{i-1}^w)]$, and $\sigma(x) = 1/(1 + \exp(-x))$. This function can be understood as using the current context to classify a word following the path from the root to the leaf of the word.

Using stochastic gradient ascent, the parameter vectors θ_{i-1}^w and word vectors in the context can be updated as follows:

$$\begin{aligned} &\theta_{i-1}^w := \theta_{i-1}^w + \eta[1 - d_i^w - \sigma(X_w^T \theta_{i-1}^w)] X_w \\ &\text{and} \quad v(\widetilde{w}) := v(\widetilde{w}) + \eta \sum_{i=2}^{L^w} \frac{\partial \ell(w,i)}{\partial X_w}, \end{aligned} \tag{2}$$

where $\widetilde{w} \in \mathcal{C}(w)$, $v(\widetilde{w})$ is the vector representation of the context word, and η is a degenerative learning rate.

The Skip-gram Model

Skip-gram model uses the current word to predict the context. The training objective of the Skip-gram model is to maximize the average log-likelihood function $\sum_{w \in \mathcal{W}} \log P(\mathcal{C}(w)|w) = \sum_{w \in \mathcal{W}} \sum_{u \in \mathcal{C}(w)} \log P(u|w)$. Here we have $P(u|w) = \prod_{j=2}^{L^u} P(d_j^u | v(w), \theta_{j-1}^u)$, where L^u is the length of the path from root to the leaf node representing word u, and θ_i^u is the vector representation of the i-th node in the path. The log-likelihood is further formulated as: $\mathcal{J}_{\text{SG}} = \sum_{w \in \mathcal{W}} \log P(\mathcal{C}(w)|w) = \sum_{w \in \mathcal{W}} \sum_{u \in \mathcal{C}(w)} \sum_{j=2}^{L^u} \ell(w, u, j)$, where we have $\ell(w, u, j) = (1 - d_j^u) \cdot \log[\sigma(v(w)^T \theta_{j-1}^u)] + d_j^u \cdot \log[1 - \sigma(v(u)^T \theta_{j-1}^u)]$, and again $v(w)$ and $v(u)$ are the vector representations of word w and word u respectively. This shows that w's context word u is predicted following the path on the tree from root to leaf given the vector representation of w: $v(w)$.

Applying stochastic gradient ascent, parameter vector θ_{j-1}^u and word vector are iteratively updated as: $\theta_{j-1}^u := \theta_{j-1}^u + \eta[1 - d_j^u - \sigma(v(w)^T \theta_{j-1}^u)] v(w)$ and $v(\widetilde{w}) := v(\widetilde{w}) + \eta \sum_{u \in \mathcal{C}(w)} \sum_{j=2}^{L^u} \frac{\partial \ell(w,u,j)}{\partial v(w)}$, where $\widetilde{w} \in \mathcal{C}(w)$, and η is the degenerative learning rate.

Learning Rate

The learning rate η is an important parameter for stochastic gradient iteration (Hegde, Indyk, and Schmidt 2015). In CBOW and Skip-gram models, an adaptive and segmented rate is set to be: $\eta = \eta_0(1 - \frac{\kappa}{\phi+1})$where η_0 is an initial value, ϕ is the number of tokens in corpus, and κ is the number of already trained words. The learning rate is governed by another parameter ρ, which controls the rate to decrease η after certain number of iterations, e.g., updating η after seeing every 10,000 words. In word2vec, a minimum value $\eta_{\min}$ is also set to enforce the update vectors based on the gradients.

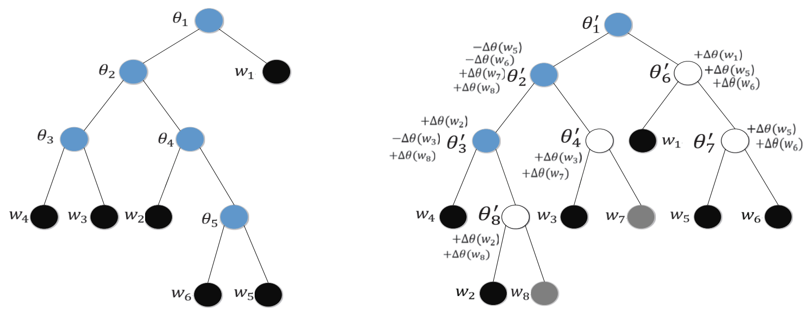
Incremental Training

We see from the above section that learning the word vectors involves not only the word vectors themselves, but also the internal nodes' vectors, which we have called parameters θ_i^w. When we see a new corpus in addition to the old corpus, we should re-build the hierarchical tree, e.g., the Huffman tree in our work, based on the combined corpus. Huffman tree is sensitive to the word frequency's distribution due to its nature of variable length representation based on the probability of occurrence of words (Huffman 1952).[1] Thus, when the tree changes, the vector representation of both internal and leaf nodes may be affected. If part of the tree remains the same, we can retain the structure as well as the vectors associated to the nodes, and distinguish the update between the parts from the old tree and the new tree.

Node Initialization and Inheritance

Suppose we have the old corpus $\mathcal{W}$ and the new corpus $\mathcal{W}' = \mathcal{W} \cup \Delta\mathcal{W}$. We can build the Huffman trees $\mathcal{T}$ and $\mathcal{T}'$ from both corpora respectively.

[1]More semantics related trees can be built (Mnih and Hinton 2008; Le et al. 2013), and the same problem of tree change should be handled.



(a) Original Huffman Tree (b) Updated Huffman Tree

Figure 1: The Figure 1(a) is $\mathcal{T}$ and the Figure 1(b) is $\mathcal{T}'$ provided the new corpus. The black leaf nodes represent inherited words vector. The blue internal nodes represent inherited parameters vector. The gray leaf node represents the new words shown in the new corpus. The white node represents the new internal nodes. The initialization of new internal nodes in $\mathcal{T}'$ are $\theta'_4 = 0$, $\theta'_6 = 0$, $\theta'_7 = 0$, $\theta'_8 = 0$. The initialization of new leaf nodes in $\mathcal{T}'$ are $v'(w_7) = \text{random}$, $v'(w_8) = \text{random}$. All the other nodes in $\mathcal{T}'$ are inherited from $\mathcal{T}$.

For the leaf nodes, if the word has been observed in the old corpus, we simply initialize the vector as the vector that has been trained. If the word is a new word, we randomly initialize it as a random vector:

$$v'(w) = \begin{cases} v(w), & w \in \mathcal{W} \\ \text{random}, & w \notin \mathcal{W} \end{cases}, \tag{3}$$

where $v(w)$ and $v'(w)$ are the vectors of word w for old and new trees respectively.

For the internal nodes, the Huffman code change of a word may affect only partial change of the path for that word. Along the path, each internal node owns one parameter vector. We distinguish the parameter vector $\theta_i^{w_1}$'s for word w_1 at i's position and $\theta_i^{w_2}$'s for word w_2 at i's position. When they are at the same i's position in the tree, then $\theta_i^{w_1} = \theta_i^{w_2}$. For example, in the left figure of Figure 1, $\theta_3^{w_2} = \theta_3^{w_6} = \theta_4$. Moreover, a word w_2 encoded as "0010" in the old tree $\mathcal{T}$ may be changed as "00010" in the new tree $\mathcal{T}'$. In this case, the matched prefix "00" remains the same, and corresponding matched internal nodes share the same structure in the new tree $\mathcal{T}'$ as the old tree $\mathcal{T}$. To make the prefix explicit, we denote L^w and L'^w as lengths of the code of word w in old and new trees respectively, e.g., $L^{w_2} = 4$ in $\mathcal{T}$ and $L'^{w_2} = 5$ in $\mathcal{T}'$ in Figure 1. We gather all internal vectors of leaf node w as set $\Omega(w) = \{\theta_i^w | i = 1, \cdots, L^w\}$ in tree $\mathcal{T}$. For the matched prefix, we use the existing parameter vector θ_i^w as the initialization for the new tree, while for the mismatched codes, we initialize them as zero vectors, as following:

$$\theta'_i = \begin{cases} \theta_i, & d'^w_i = d^w_i \\ 0, & \text{otherwise} \end{cases}, \tag{4}$$

where d'^w_i and d^w_i are the Huffman codes of internal nodes in the new and old trees respectively. Thus, the inherited internal nodes of leaf node w can be divided into common prefix matched substring $\Theta(w) = \{\theta_{i-1}^w | i = 2, \cdots, L_C^w + 1\}$ and other nodes $\Phi(w) = \{\theta_{i-1}^w | i = L_C^w + 2, \cdots, L^w\}$ in $\mathcal{T}$ and $\Phi'(w) = \{\theta_{i-1}^w | i = L_C^w + 2, \cdots, L'^w\}$ in $\mathcal{T}'$, where L_C^w is length of common prefix matched between old and new trees. Figure 1 also shows examples of inherited nodes and new nodes.

Model Updates

Given the inherited nodes and the new nodes by comparing the old and new trees, we also decompose the log-likelihood functions for CBOW and Skip-gram models based on the prefix matching results.

For CBOW model, we consider to factorize the log-likelihood function by aggregating the cost term $\ell(w, i)$ in Eq. (1).

$$\mathcal{J}'_{\text{CBOW}} = \sum_{w \in \mathcal{W}} \{ \sum_{i=2}^{L_C^w+1} + \sum_{i=L_C^w+2}^{L'^w} \} \ell(w, i) + \sum_{w \in \Delta\mathcal{W}} \sum_{i=2}^{L'^w} \ell(w, i) \tag{5}$$

Here we first split the training data to be $\mathcal{W}' = \mathcal{W} \cup \Delta\mathcal{W}$. For the words in $\mathcal{W}$, we factorize it based on the common Huffman codes and distinct codes. $\sum_{i=2}^{L_C^w+1}$ sums the codes that share the prefix with the old tree by word w. $\sum_{i=L_C^w+2}^{L'^w}$ sums inherited internal vector codes by other words and the zero initialization internal vector codes in the new tree. For the words in $\Delta\mathcal{W}$, we follow the original objective function of CBOW model.

Similarly, for Skip-gram model, the objective $\mathcal{J}'_{\text{SG}}$ is: $\sum_{w \in \mathcal{W}} \sum_{u \in \mathcal{C}(w)} \{ \sum_{j=2}^{L_C^u+1} + \sum_{L_C^u+2}^{L'^u} \} \ell(w, u, j) + \sum_{w \in \Delta\mathcal{W}} \sum_{u \in \mathcal{C}(w)} \sum_{j=2}^{L'^u} \ell(w, u, j)$.

To train a new set of word vectors, originally we need to re-scan and re-train the whole corpus $\mathcal{W}' = \mathcal{W} \cup \Delta\mathcal{W}$ based on stochastic gradient ascent method. Given the above factorization analysis of the objective function, we found that for the old corpus $\mathcal{W}$, we can apply the following trick to save a lot of training time.

Our goal is to find a new set of (local) optimal internal node vectors θ'^w_i and word vectors $v'(w)$ to approximate re-training. We first make an assumption that all the word vectors $v(w)$ are already (local) optimal and then further calibrate them. Then we perform stochastic gradient based optimization based on $\mathcal{W}$ and $\mathcal{W}'$ respectively.

When scanning the old corpus $\mathcal{W}$, we can update all the parameters while fixing the word vectors.[2] Denote all the parameters related to w as $\Theta(w) \cup \Phi'(w)$. We can see that for the $\Theta(w)$, the training process is the same as the original CBOW and Skip-gram models. For $\Phi'(w)$, since the tree structure has changed, for a certain internal node, some of the leaves (words) are still under it while the others has moved out. For example, in Figure 1, the word w_6 is now under θ'_6 and θ'_7 but moved out of θ_2, θ_4, and θ_5. To recover the parameters in the new tree so that the incremental training is as similar as the fully re-trained model when seeing w_6, we need to subtract the inherited gradients of θ'_2 related to w_6, and add the gradients to θ_6 and θ_7 (here θ'_4 is initialized as zero and θ'_5 is not inherited). Formally, for a word w, the CBOW update rule for the parameters in the new path from root to this word is as follows.

If $w \in \mathcal{T}', {\theta'}^w_{i-1} \in \Theta(w), i \in \{2, \dots, L^w_C + 1\}$, we have:

$$ {\theta'}^w_{i-1} := {\theta'}^w_{i-1}. \tag{6}$$

If $w \in \mathcal{T}', {\theta'}^w_{i-1} \in \Phi'(w), i \in \{L^w_C + 2, \dots, {L'}^w\}$, we have:

$$ {\theta'}^w_{i-1} := {\theta'}^w_{i-1} + \eta'[1 - d^w_i - \sigma(X^T_w {\theta'}^w_{i-1})]X_w. \tag{7}$$

If $w \in \mathcal{T}, \theta^w_{i-1} \in \Phi(w), i \in \{L^v_C + 2, \dots, L^v\}$, we have:

$$ {\theta'}^w_{i-1} := {\theta'}^w_{i-1} - \eta'[1 - d^w_i - \sigma(X^T_w {\theta'}^w_{i-1})]X_w. \tag{8}$$

Here retain the common prefix nodes, perform stochastic gradient ascent to the new nodes, and perform stochastic gradient descent to the old sub-tree path related to the word w. η' is the new learning rate.

For the update corpus $\Delta\mathcal{W}$, we simply perform the stochastic gradient ascent for both parameters and word vectors (e.g., in Eq. (2)). Thus, we can see that the most computational cost is saved by not updating word vectors in old corpus, and partially saved by adjusting (partially not updating) the parameters in old corpus. An illustration is shown in Figure 1. From the figure we can see that, we adjust the internal node to approximate the process of complete re-training.

Similarly for Skip-gram, if $u \in \mathcal{T}', {\theta'}^u_{j-1} \in \Theta(u), j \in \{2, \dots, L^u_C + 1\}$, we have: ${\theta'}^u_{j-1} := {\theta'}^u_{j-1}$. If $u \in \mathcal{T}', {\theta'}^u_{j-1} \in \Phi'(u), j \in \{L^u_C + 2, \dots, {L'}^u\}$, we have: ${\theta'}^u_{j-1} := {\theta'}^u_{j-1} + \eta'[1 - d^u_j - \sigma(v(w)^T {\theta'}^u_{j-1})]v(w)$. If $u \in \mathcal{T}, \theta^u_{j-1} \in \Phi(u), j \in \{L^v_C + 2, \dots, L^v\}$, we have: ${\theta'}^u_{j-1} := {\theta'}^u_{j-1} - \eta'[1 - d^u_j - \sigma(v(w)^T {\theta'}^u_{j-1})]v(w)$. For above equations, $u \in \mathcal{C}(w)$.

Theoretical Analysis

In this section, we present the theoretical analysis of our incremental learning algorithm using CBOW model.

Convergence Analysis

The log-likelihood function (1) is negative. Thus, maximizing the objective is bounded by zero. However, since in the objective function, it involves the dot product of the word vectors and parameters $X^T_w \theta^w_{i-1}$, this is a non-convex optimization problem. By using alternative optimization altering the word vectors and parameters, fixing one and optimizing the other is a convex problem. In our incremental learning process, the convergence of optimizing over the update corpus $\Delta\mathcal{W}$ is the same as original word2vec models. When optimizing over the old corpus, we assume the word vectors are already (local) optimal, and optimize the parameters over the new Huffman tree. For example for CBOW model, by checking the second order derivative of the parameters, we have $\nabla^2_{\theta'_{i-1}} = \sum_{w \in \mathcal{W}} \sum_{i=2}^{{L'}^w} -\sigma(X^T_w \theta'_{i-1}) X^T_w X_w$, where $\sigma(x) \in [0, 1]$ and $X^T_w X_w \geq 0$.

Compared to the original second order derivative over old corpus, we replace the summation term $\sum_{i=2}^{L^w}$ by $\sum_{i=2}^{{L'}^w}$. This is guaranteed by using both stochastic gradient ascent and descent in Eqs. (6)-(8) if we scan the old corpus in the same times, and thus the process is toward another local optimum. The Skip-gram model has the similar property.

Parameter Error Bound

Here we focus on the internal nodes that updated with stochastic gradient descent. In CBOW model, as shown in Eqs. (6)-(8), there are two parts affecting the final results.

First, the learning rate change is: $\Delta\eta = \eta_0 |\frac{\kappa}{\phi+1} - \frac{\kappa'}{\phi'+1}|$. Second, we first assume that the parameter vector can be bounded by ξ and then infer the bounded ξ in stochastic gradient descent. We also assume that the word vectors X^T_w can be bounded by a vector $\vec{\epsilon}_{X_w}$ where each element is ϵ. Since we have noted that the optimization process push the solution from on local optimum to another, based on first order Taylor expansion, we have $|\sigma(X^T_w({\theta'}^w_{i-1} + \xi) - \sigma(X^T_w {\theta'}^w_{i-1})| < \sigma(X^T_w {\theta'}^w_{i-1})(1 - \sigma(X^T_w {\theta'}^w_{i-1}))\xi < \xi$. Then we have the difference of the gradients:

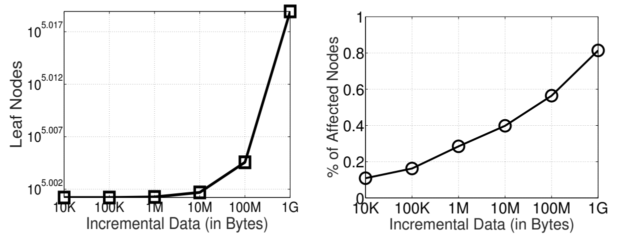
$$ |\Delta\mathcal{B}_{{\theta'}^w_{i-1}}| = |\sigma(X^T_w \theta^w_{i-1}) - \sigma(X^T_w {\theta'}^w_{i-1})||X_w| \preceq \xi \vec{\epsilon}_{X_w}, \tag{9}$$

where $\preceq$ denotes element-wise less than or equal to. If we denote $|\mathcal{B}| = |[1 - d^w_i - \sigma(X^T_w {\theta'}^w_{i-1})]X_w| \preceq \vec{\epsilon}_{X_w}$, then the parameter will be bounded as by aggregating the differences in Eq. (7): $\vec{\xi} = |\Delta\eta\mathcal{B} + \Delta\mathcal{B}\eta' + \Delta\eta\Delta\mathcal{B}| \preceq \Delta\eta\vec{\epsilon}_{X_w} + \eta'\xi\vec{\epsilon}_{X_w} + \Delta\eta\xi\vec{\epsilon}_{X_w}$, where $\vec{\xi}$ is a vector of the same value ξ. Then we solve ξ as:

$$ (1 - (\eta' + \Delta\eta)\epsilon)\xi \leq \Delta\eta\epsilon. \tag{10}$$

To have more concrete idea of the bound (10), for example, assume we have $\phi = 10^9$ trained with one billion words/tokens, and $\kappa = 10^6$ meaning that the vocabulary is with one million unique words. We augment the training data with $|\Delta\mathcal{W}| = 10^8$. Suppose we have $\eta_0 = 0.025$, which is the default setting in word2vec package. Then we have $\eta_{\min} \leq \eta \approx \eta' = \eta_0(1 - \frac{\kappa'}{\phi'+1}) \leq \eta_0$, and $\Delta\eta \approx 10^{-10}$. In addition, we assume that the elements of word vectors bounded by ϵ is in $[-5, 5]$. Thus, for η_0, we can compute Eq. (10) as $(1 - (0.025 + 10^{-10}) * 5)\xi \leq 10^{-10} * 5$, which means $\xi \leq 5.7 * 10^{-10}$. For $\eta_{\min} = \eta_0 * 0.0001$ as that in word2vec package, we have $\xi \leq 5.0 * 10^{-10}$. At most,

[2]We can also update the word vectors in the meantime, however, it will introduce more computational cost.



(a) Leaf node change. (b) Internal node change.

Figure 2: Leaf and internal nodes change with incremental corpus.

there can be half of the leaves changed from one side of the tree to the other side corresponding to completely reversing the order of frequencies. However, this will never happen in practice. Moreover, in practice, we found that on the top of the Huffman tree, more updates affected to perform stochastic gradient descent. However, it is also likely that the leaf will be in the same sub-tree even if it is moved from one successor to another. For example, in Figure 1, the move of w_2 and w_3 does not affect θ_2. For $n = 2 * 10^5$ which means 20% of the leaves moving out of a sub-tree, if we assume the error accumulates, then the error bound will be $2 * 10^5 * 5.7 * 10^{-10} = 1.1 * 10^{-4}$.

Experiments

In this section, we present the experiments to verify the effectiveness and efficiency of incremental training for hierarchical softmax function in NNLMs.

Training Time and Quality

We use the English Wikipedia as the source to train the NNLMs. We split the data into several sets. We use 2GB texts as the initial training corpus, which is the old corpus in previous sections. The 2GB data contains 474,746,098 tokens and 100,278 unique words. Then, we select 10KB, 100KB, 1MB, 10MB, 100MB, and 1GB as new update corpora to compare the performance of the algorithms. The number of words arises with new update corpus, as shown in Figure 2(a). For original global training, we combine the old and new corpora as a whole, and run the original CBOW and Skip-gram models. For the incremental training, we use the model trained based on the 2GB initial training corpus, and run our algorithm to update the Huffman tree as well as the parameters and word vectors. For all the experiments, we run with 10 CPU threads and generate word embeddings with 300 dimensions.

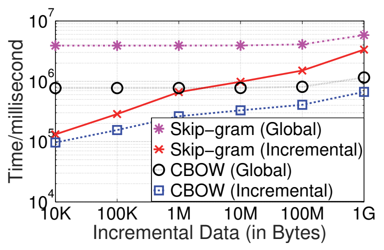
First, we check the percentage of Huffman tree change. In Figure 2(b) it shows the percentages of the internal nodes that are affected by the change of the tree. If an internal node is updated with stochastic gradient descent in Eq. (8), then we label it as affected. From the figure we can see that, there are more nodes affected when adding more training materials. The increase is not as much as the increase of total number of leaf nodes.

	Items	Global	Incremental	T-Reserved
CB	MSE	6.68×10^{-3}	6.66×10^{-3}	7.84×10^{-3}
	1-Cos	0.361	0.361	0.411
	E-Dis	1.415	1.413	1.543
SG	MSE	6.65×10^{-3}	6.68×10^{-3}	7.83×10^{-3}
	1-Cos	0.363	0.365	0.409
	E-Dis	1.413	1.415	1.533

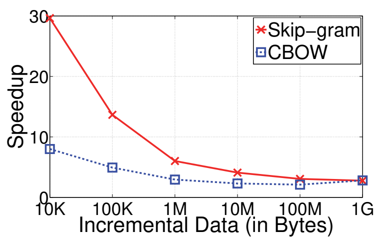
Table 1: Average of MSE (mean square error), 1-Cos (converting cosine similarity to dissimilarity), and E-Dis (Euclidean) on 2G+1G corpus. “T-Reserved” means that we reserve the old tree to perform a trivial incremental learning.

Then we check the training time and speedup using our incremental training algorithm. We show the training time results in Figure 3(a). It is shown that both CBOW and Skip-gram are linear to the training size. Since adding from 10KB to 100MB is relatively small compared to the original training size 2GB, the time curves for both CBOW and Skip-gram with global training is flat until with the 1GB additional training data. Moreover, we find that Skip-gram is in an order of magnitude slower than CBOW. By comparing CBOW and Skip-gram, we can see that for each context word, Skip-gram needs to update the parameters following the path from root to that word. Thus, the order comes from the number of window size used in the language model, which is five in all of our experiments. Furthermore, incremental training for both CBOW and Skip-gram benefits from the algorithm and faster than global training. Again, both scale linearly with the number of additional update corpus. The speedup results are shown in Figure 3(b). We can see that for smaller update corpus, the speedup is more significant. The Skip-gram model can have up to 30 times speedup with our incremental training algorithm, while CBOW model has up to 10 times speedup.

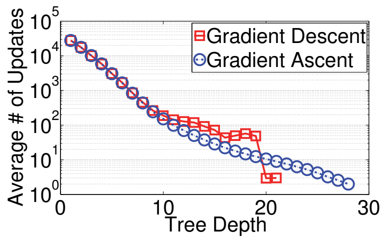
We also randomly selected 5,000 word vectors to test the difference of the word vectors. We use the mean square error (MSE) to evaluate the difference between two sets of word vectors. For global training, we run the same algorithm twice using different random initialization. For incremental training, we compare the incremental training results with the global training results. In Table 1, we compare the results. “**Global**” represents global training in all corpus. “**Incremental**” represents our incremental learning models, and “**T-Reserved**” is a trivial incremental learning by working with stochastic gradients on the new corpus with the old tree reserved. We can see that in general, the MSE of global training is better than incremental training. Nonetheless, they are of the same order of magnitude, which is around $10^{-3} \sim 10^{-2}$. “T-Reserved” is worst since it only uses the tree based on the old corpus. To further understand the MSE with the bound, we show the average number of gradient updates at each level of the Huffman tree for CBOW model in Figure 3(c), when scanning the 2GB old data. We can see that, the number of updates exponentially decreases when the depth of the tree increases. For top levels, the moves indicate that there are indeed a lot of nodes moved from one side to the other to trigger the change of parameter updates.



(a) Training time.



(b) Speedup.



(c) Average # of gradient updates for each depth (2GB+1GB).

Figure 3: Training performance of global and incremental training.

The order of change is at most 10^5. For deepest levels, there are no descents but only ascents, which indicates the new Huffman tree is deeper than the old one. Then for the 300 vectors, MSE will aggregate all the ξ of each dimension.

Word Similarity/Relatedness

Now we use the word similarity/relatedness evaluation benchmarks to evaluate the correctness of our incremental training algorithm. Specifically, we use the datasets collected by Faruqui and Dyer (Faruqui and Dyer 2014) which include MC-30, TR-3k, MTurk-287, MTurk-771, RG-65, RW-STANFORD (RW), SIMLEX-999, VERB-143, WS-353-ALL, WS-353-REL, WS-353-SIM, and YP-130.[3] We use cosine value to compute the similarities between words, and then rank the words similar/related to each other. The Spearman's rank correlation coefficient (Myers and Well. 1995) is used to check the correlation of ranks between human annotation and computed similarities. Due to the limited space, for incremental training, we show the average results trained over 1GB update data. From Figure 4 we can see that, the incremental training results are comparable and sometimes better than the global training results. "T-Reserved" is again the worst among the three methods we compared.

Dependency Parsing (DP)

We also test DP using different training methods. Different from the previous task, DP uses word embeddings as features, and train a model to predict the structural output of each sentence. Thus, it is more complicated than comparing the similarities between words. We use the CNN model (Guo et al. 2015) to train a model based on the word embeddings produces by our experiments. The data used to train and evaluate the parser is the English data in the CoNLL-X shared task (Buchholz and Marsi 2006). We follow (Guo et al. 2015; Upadhyay et al. 2016) to setup the training and testing processes, using the tools available online.[4] We train the model with 200,000 iterations, and set the parameters as distance of embedding to be 5, valency of embedding to be 5, and cluster of embedding to be 8 (Guo et al. 2015).

The results are shown in Figure 5. Both the labeled attachment score (LAS) and unlabeled attachment score (UAS) are reported. We can see that, the incremental training and global training are also similar, and it seems incremental training is a little bit better than global training. This again demonstrates that incremental training is comparable to global training, and saves a lot of training time. Moreover, the CBOW model and Skip-gram model perform similarly. This may be because the supervised learning model can eliminate the vector representation difference produced by different algorithms but with the same training corpus. Moreover, we found that there is a step of performance improvement at 10M new data. This may be because some important words are included starting from this data. However, for "T-Reserved", there is much less improvement when the step happens to the other two. This again demonstrates that the tree can affect the final embedding results. If we do not consider the tree change for incremental training, we may lose a lot of information provided by the update corpus.

Conclusion

In this paper, we present an incremental training algorithm for the hierarchical softmax function for CBOW and Skip-gram models. The results of the systematic evaluation and down-stream tasks show that our incremental training is significantly faster than global training, and has similar performance. Theoretical analysis also helped us better understand the performance of the incremental algorithm. The natural future work is to extend our approach to other advanced NNLMs beyond CBOW and Skip-gram such as dependency RNN (Mirowski and Vlachos 2015) and LSTM or deeper RNN models (Renshaw and Hall 2015).[5]

Acknowledgments

The corresponding author is Jianxin Li. This work is suppored by China 973 Fundamental R&D Program (No.2014CB340300), NSFC program (No.61472022, 61421003), SKLSDE-2016ZX-11, Beijing High-tech R&D Program (Grant No. Z161100000216128) and partly by the Beijing Advanced Innovation Center for Big Data and Brain Computing. Yangqiu Song was partially supported by China 973 Fundamental R&D Program (No.2014CB340304) and

[3]http://www.wordvectors.org/

[4]https://github.com/jiangfeng1124/acl15-clnndep

[5]Our system is publicly available at https://github.com/RingBDStack/incremental-word2vec

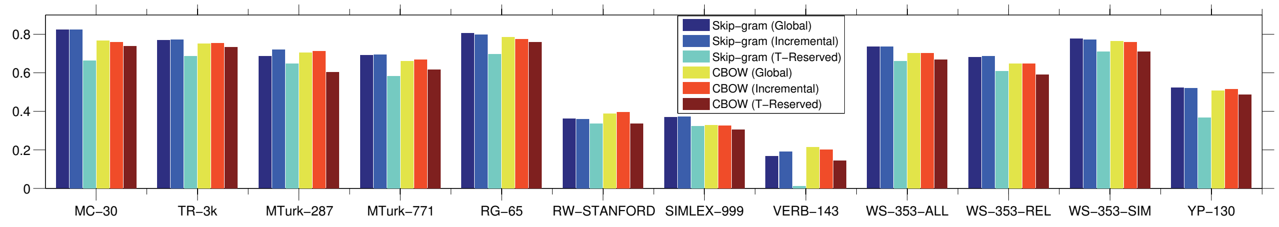
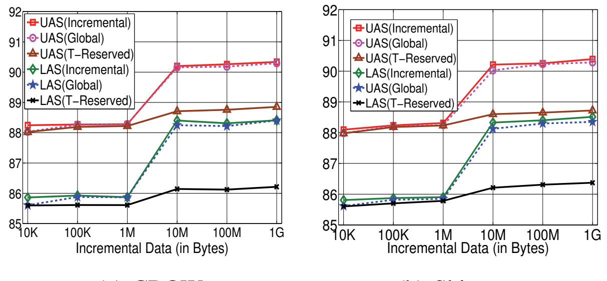


Figure 4: Comparison word embeddings for word similarity/relatedness benchmark datasets.



(a) CBOW (b) Skip-gram

Figure 5: Comparison of word embeddings for DP.

the LORELEI Contract HR0011-15-2-0025 with the US Defense Advanced Research Projects Agency (DARPA). Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. We also thank the anonymous reviewers for their valuable comments and suggestions that help improve the quality of this manuscript.

References

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

Buchholz, S., and Marsi, E. 2006. Conll-x shared task on multilingual dependency parsing. In *CoNLL*, 149–164.

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. P. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12:2493–2537.

Faruqui, M., and Dyer, C. 2014. Improving vector space word representations using multilingual correlation. In *EACL*, 462–471.

Guo, J.; Che, W.; Yarowsky, D.; Wang, H.; and Liu, T. 2015. Cross-lingual dependency parsing based on distributed representations. In *ACL*, 1234–1244.

Gutmann, M., and Hyvärinen, A. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research* 13:307–361.

Hegde, C.; Indyk, P.; and Schmidt, L. 2015. A nearly-linear time framework for graph-structured sparsity. In *ICML*, 928–937.

Huffman, D. A. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40(9):1098–1101.

Le, H. S.; Oparin, I.; Allauzen, A.; Gauvain, J.; and Yvon, F. 2013. Structured output layer neural network language models for speech recognition. *IEEE Trans. Audio, Speech & Language Processing* 21(1):195–204.

Levy, O., and Goldberg, Y. 2014. Neural word embedding as implicit matrix factorization. In *NIPS*, 2177–2185.

Levy, O.; Goldberg, Y.; and Dagan, I. 2015. Improving distributional similarity with lessons learned from word embeddings. *TACL* 3:211–225.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013a. Efficient estimation of word representations in vector space. *ICLR*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013b. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

Mirowski, P., and Vlachos, A. 2015. Dependency recurrent neural language models for sentence completion. In *ACL*, 511–517.

Mnih, A., and Hinton, G. E. 2008. A scalable hierarchical distributed language model. In *NIPS*, 1081–1088.

Morin, F., and Bengio, Y. 2005. Hierarchical probabilistic neural network language model. In *AISTATS*.

Myers, J. L., and Well., A. D. 1995. *Research Design & Statistical Analysis*. Routledge.

Renshaw, D., and Hall, K. B. 2015. Long short-term memory language models with additive morphological features for automatic speech recognition. In *ICASSP*, 5246–5250.

Turian, J.; Ratinov, L.-A.; and Bengio, Y. 2010. Word representations: A simple and general method for semi-supervised learning. In *ACL*, 384–394.

Upadhyay, S.; Faruqui, M.; Dyer, C.; and Roth, D. 2016. Cross-lingual models of word embeddings: An empirical comparison. In *ACL*.