ISSN 2395-8618

Optimize Hierarchical Softmax with Word Similarity Knowledge

Zhixuan Yang, Chong Ruan, Caihua Li, and Junfeng Hu

Abstract—**Hierarchical softmax is widely used to accelerate the training speed of neural language models and word embedding models. Traditionally, people believed that the hierarchical tree of words should be organized by the semantic meaning of words. However, Mikolov et al. showed that high quality word embeddings can also be trained by simply using the Huffman tree of words. To our knowledge, no work gives a theoretic analysis on how we should organize the hierarchical tree. In this paper, we try to answer this question theoretically by treating the tree structure as a parameter of the training objective function. As a result, we can show that the Huffman tree maximizes the (augmented) training function when word embeddings are random. Following this, we propose *SemHuff*, a new tree constructing scheme based on adjusting the Huffman tree with word similarity knowledge. Experiment results show that word embeddings trained with optimized hierarchical tree can give better results in various tasks.**

Index Terms—**Hierarchical Softmax, Word Embedding, Word Similarity Knowledge**

I. Introduction

Traditionally, words are treated as distinct symbols in NLP tasks. This treating has limitations especially when it is used with n-gram models. For example, if the size of the vocabulary is $|V|$, an n-gram language model will have $O(|V|^n)$ parameters. The curse of dimensions in the number of parameters leads to great difficulties on learning and smoothing the model. More advanced methods were proposed to address this problem. Word embedding, also known as distributed representations or word vectors, is among one of them. The key idea is to exploit the similarity between words. Word embedding maps words to vectors of real numbers in a low dimensional space. Similar words are mapped to close vectors while dissimilar words are mapped to vectors with longer distance. For example, the word *cat* may be mapped to (0.8, 0.7, $\cdots$), and *dog* may be mapped to (0.75, 0.77, $\cdots$), while the word *Paris* may be mapped to (-0.5, -0.9, $\cdots$).

Bengio et al. [1] proposed a neural language model that uses word embeddings as input features of the language model. The model also treats word embeddings as unknown parameters and learns them from data just like other parameters in the neural network. Such neural network methods were further improved in both efficiency and accuracy by many researchers in the past decade such as the log-bilinear model by Mnih & Hinton [14] and the skip-gram model by Mikolov et al. [11]. More efficient training algorithms were also introduced, including hierarchical softmax by Morin and Bengio [16], NCE (noise contrastive estimation) [4] [13], and negative sampling by Mikolov et al. [10], which is a further simplification of NCE. Simpler models and more efficient training algorithms allow learning accurate word embeddings from large-scale corpus.

People used to believed that the hierarchical tree used in hierarchical softmax should organize words by the semantic meaning of words. For example, Morin & Bengio [16] extracted the tree from the IS-A taxonomy in WordNet [12] and Mnih & Hinton [15] constructed the tree by repeatedly clustering word embeddings into a binary tree. However, Mikolov et al. showed that high quality word embeddings could also be learned by simply using the Huffman tree. To our knowledge, no work has given a theoretical analysis on how we should organize the hierarchical tree. Inspired by the analysis performed by Levy and Goldberg [9], we try to answer this question by treating the tree structure as a parameter of the training objective function. Following this, we show that the Huffman tree can maximize the augmented objective function when word embeddings are random. We can also explain more clearly why semantic related words should be placed together in the hierarchical softmax tree. Following the theoretical analysis, we propose **SemHuff**, a new hierarchical softmax tree constructing scheme based on adjusting the Huffman tree by rearranging nodes in same level. Our experiments show that SemHuff can improve the Huffman tree in all tasks and hierarchical softmax can outperform negative sampling in some situations.

The rest of this paper is organized in the following way: Section 2 analyzes hierarchical softmax by treating the tree structure as a parameter. Section 3 proposes SemHuff, our new tree constructing scheme. Section 4 presents our experiment results, which compares the negative sampling and the hierarchical softmax with different tree constructing schemes.

Manuscript received June 25, 2016.

Junfeng Hu is the corresponding author. (phone: 86-10-62765835 ext 103; fax: 86-10-62765835 ext 101; e-mail: hujf@pku.edu.cn)

Zhixuan Yang and Caihua Li are with the School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, P. R. China. (e-mail: {yangzx95, peterli}@pku.edu.cn).

Chong Ruan and Junfeng Hu are with Key Laboratory of Computational Linguistics, Ministry of Education, Institute of Computational Linguistics, School of Electronics Engineering and Computer Science, Peking University, P. R. China. (e-mail: {pkurc, hujf}@pku.edu.cn).

This work is supported by the National Natural Science Foundation of China. (grant No. M1321005, 61472017).

In the end, section 5 concludes this paper.

II. Analysis

A. Why Huffman Tree Works?

The skip-gram model predicts the probability over all words given one word in its context. The training objective function is the log likelihood:

$$L = \sum_{w_i \in Corpus} \sum_{w_j \in ctx(w_i)} \log p(w_j \mid w_i)$$

A linear classifier with |*V*| outputs is used to predict the conditional probability $p(w_j|w_i)$. In order to avoid an output layer with |*V*| outputs, which is very computationally expensive, hierarchical softmax organizes all words as a binary tree. Then, the conditional probability $p(w_j|w_i)$ of w_j is decomposed as the product of probabilities of going from the root of the tree to the leaf node representing w_j. When skip-gram is used with hierarchical softmax, the objective function can be written as:

$$L = \sum_{w_i \in Corpus} \sum_{w_j \in ctx(w_i)} \sum_{n \in path(w_j)} \log p(d(w_j, n) \mid n, w_i)$$

where $path(w_j)$ is the set of internal nodes on the path from the root of the tree to the leaf node representing w_j. And *d* (abbreviation for *direction*) is defined as:

$$d(w_j, n) = \begin{cases} 0, w_j \text{ is in the left subtree of } n \\ 1, w_j \text{ is in the left subtree of } n \end{cases}$$

The above objective function sums over every word and its context words in the corpus. We can group the summation by (w_i, w_j) pairs so that the equation can be written as:

$$L = \sum_{w_i \in V} \sum_{w_j \in V} \#(w_i, w_j) \sum_{n \in path(w_j)} \log p(d(w_j, n) \mid n, w_i)$$

where $\#(w_i, w_j)$ is the total number of observed (w_i, w_j) pairs in the corpus. We can divide the equation by the total number of all observed pairs, so that the count can be treated as probability:

$$L = \sum_{w_i \in V} \sum_{w_j \in V} p(w_i, w_j) \sum_{n \in path(w_j)} \log p(d(w_j, n) \mid n, w_i) \quad (1)$$

Then we change the order of the summations:

$$L = \sum_{w_j \in V} p(w_j) \sum_{n \in path(w_j)} \sum_{w_i \in V} p(w_i \mid w_j) \log p(d(w_j, n) \mid n, w_i)$$

where $p(w_i|w_j) = p(w_i, w_j) / p(w_j)$ is the conditional probability of w_i occurs in the context of w_j. When hierarchical softmax is used with the skip-gram model, $p(d(w_j, n)| n, w_i)$ is computed by a logistic regression:

$$p(d(w_j, n) \mid n, w_i) = \begin{cases} \sigma(\vec{n}^T \vec{w}_i), & \text{for } d(w_j, n) = 1 \\ 1 - \sigma(\vec{n}^T \vec{w}_i), & \text{for } d(w_j, n) = 0 \end{cases}$$

If word embeddings and classifier weights are independently random initialized from a uniform distribution with zero mean, the expectation of $p(d(w_j, n)| n, w_i)$ is 1/2. And the expectation of *L* with respect to word embeddings is:

$$E[L] = \sum_{w_j \in V} p(w_j) \sum_{n \in path(w_j)} \sum_{w_i \in V} p(w_i \mid w_j)\,(-1)$$

Since $p(w_i|w_j)$ is a distribution, it sums to 1:

$$E[L] = \sum_{w_j \in V} p(w_j) \sum_{n \in path(w_j)} (-1) = -\sum_{w_j \in V} p(w_j) \mid path(w_j) \mid$$

This is also the objective function of the Huffman tree.

From this result, we can see why the idea of using Huffman tree as the hierarchical softmax tree works in practice, since it maximizes the objective function when word embeddings are random. It also points out the structure of the hierarchical tree should consider the frequency of the words besides the meaning of the words.

B. Why Similar Words Should Be Placed Together?

The motivation of placing similar words in nearby positions in the tree is straightforward: the training samples for the classifier in each internal node will be more separable because similar words will have similar contexts by the distributional hypothesis of Harris [5]. We can explain this idea more clearly from the perspective of the training objective functions. It will also show some connections between the hierarchical softmax and decision trees.

In the training objective function (1), we can sums over all internal nodes first:

$$L = \sum_{n} \sum_{w_i, w_j \in V} p(w_i, w_j) \log p(d(w_j, n) \mid n, w_i)$$

If we assume the classifier in node *n* is perfectly trained, its estimation of the probability of going to the left subtree when the input word is w_i should be the ratio of training samples leading w_i to left, that is:

$$p(d = 0 \mid n, w_i) = \sum_{\substack{w_j \\ s.t.\ d(w_j, n) = 0}} p(w_j \mid w_i)$$

Hence an upper bound of the log-likelihood is:

$$L \le \sum_{n} \sum_{w_i} \sum_{k \in \{0,1\}} p(d{=}k, w_i | n) \log p(d{=}k | w_i, n) = -\sum_{n} \mathrm{H}_n[d|W]$$

where $\mathrm{H}_n[d \mid W]$ is the conditional entropy between the direction *d* and the word embedding *W* treated as a random variable, at internal node *n*. If we adopt a top-down splitting tree constructing scheme like the decision tree, the dividing criteria should be the conditional entropy. If we want to minimize the conditional entropy, placing words with similar contexts in the same subtree is likely to reduce the entropy.

III. SemHuff

In this section, we propose our hierarchical tree constructing scheme: *SemHuff*. Noticed that if we rearrange the nodes or subtrees of a Huffman tree within the same level, the resulting tree remains a Huffman tree, since all leaves keep their original depths unchanged. So our strategy is to rearrange nodes in the same level so that similar words can be placed together in the Huffman tree.

A. Description

Assuming we already have word similarity knowledge: S_{ij} is the similarity between word w_i and w_j. We generate an arbitrary Huffman tree from the corpus first. Then we adopt a bottom-up adjusting strategy: (1) from the bottom level to the top level, for level *i*, we calculate the similarity between all subtrees. The

ISSN 2395-8618

similarity of two subtrees is defined as the average similarity between theirs leaves (i.e. words). (2) Then we apply the weighted maximum matching algorithm to the similarity graph of subtrees. (3) Now, we rearrange these subtrees in level *i* according to the matching results. Matched subtrees are organized as siblings and subtrees without matching are paired arbitrarily. (4) go back to step (1) for level *i*-1 The algorithm is described by pseudo code in Figure 1.

```
function SemHuff(Corpus, S)
    $T$ ← GenerateHuffmanTree(Corpus)
    for d from Depth($T$) down to 1 do
        $N$ ← $\{s \mid$ subtree $s$ in level $d$ of $T\}$
        for $i$ in $N$ and $j$ in $N$ do
            $S^{d}_{i,j}$ ← Average($\{S_{u,v} \mid w_u \in i$ and $w_v \in j\}$)
        end for
        matching ← WeightedMaximumMatching($N$, $S^d$)
        for $i \in N$ do
            if $i$ is matched then
                rearrange $i$ and its match as siblings in level $d$ of $T$
            else
                randomly pick another unmatched subtree as its sibling
            end if
        end for
    end for
    return $T$
end function
```

Fig. 1. The pseudo code of SemHuff. The function takes the corpus and a similarity measure S between words as input and returns the adjusted Huffman tree.

The word similarity knowledge can be extracted from different kinds of language resources like WordNet [12] and PPDB [2] for the English language. For the Chinese language, we extract the similarity knowledge from the ontology generated by He et al. [6] in our experiments. The word similarity knowledge used by *SemHuff* is not limited to use external prior knowledge. It is also possible to use the word embeddings being trained by a bootstrapping process similar to Mnih and Hinton [15].

B. *Demonstration*

To illustrate our algorithm more clearly, a hand-crafted demo is provided for further investigation. We choose only 6 words and set the corresponding similarity matrix intuitively. Then the adjusting procedure is presented in Figure 2.

In the first step, we consider all possible matches among the leaves, *cat*, *dog*, *tree*, *grass* and find out the best match is obviously ⟨*cat*, *dog*⟩ and ⟨*tree*, *grass*⟩, so *cat* and *dog* are set as siblings and so are *tree* and *grass*. When it comes to the penultimate layer, the four nodes/subtrees to be matched are *plant*, {*dog*, *cat*}, *animal*, {*tree*, *grass*}. In this layer, the best matches are ⟨*animal*, {*dog*, *cat*}⟩ and ⟨*plant*, {*tree*, *grass*}⟩. Now we see that all similar words are grouped together. After this step, adjusting procedure stops, because no upper layer has more than two nodes.

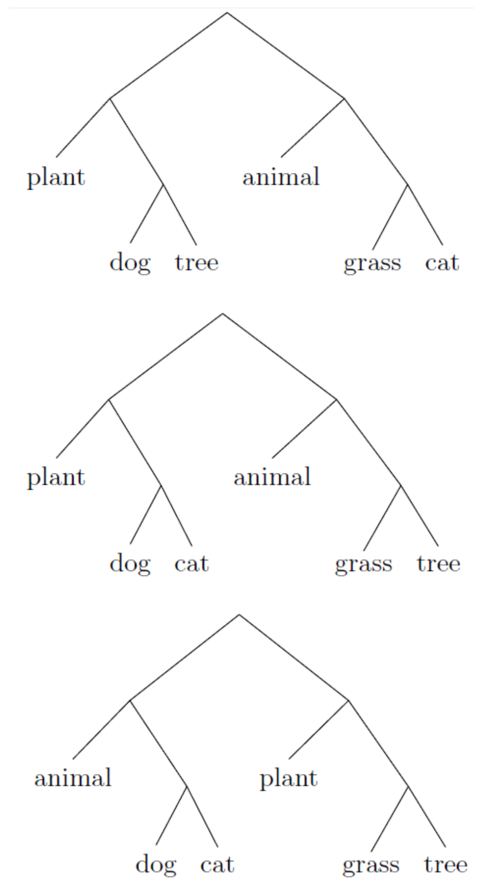


Fig. 2. Top: Huffman tree generated from corpus. Middle: Huffman tree after adjusted the bottom layer. Bottom: the final Huffman tree.

C. *Implementation Details*

Our implementation of *SemHuff* is modified from the original *word2vec* package. In our program, we use the highly efficient *Blossom V* package written by Kolmogorov & Vladimir [7] to perform weighted maximum matching. However, it's not practical to find the exact weighted maximum matching of a dense graph which has tens of thousands vertices and about a billion edges, since the complexity of the matching algorithm is $O(|E||V|^3)$. As a result, some approximation is made: for each word, we only keep its 30 most similar words and thus speed up the computation. With this method, adjusting can be done in less than 20 minutes for a Huffman tree of 80k leaves and 24 layers on a typical workstation.

We'd like to show a snippet of the adjusted Huffman tree in Figure 3. We see that similar words have been put together and it demonstrates the effectiveness of our adjusting algorithm.

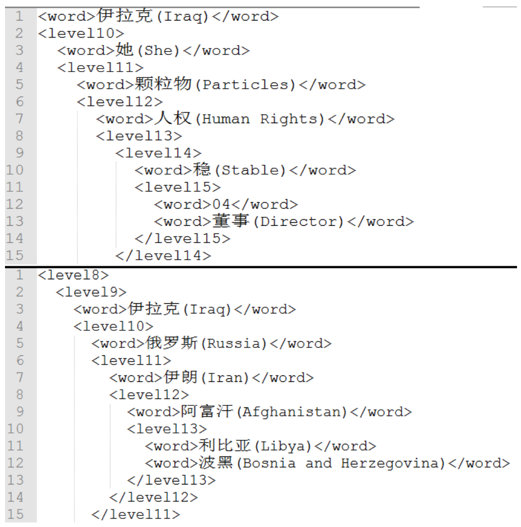
IV. Experiments

We conduct experiments to compare the performance of different hierarchical tree structures. Following Lai et al. [8], we evaluate word embeddings by tasks from two perspectives: semantic properties of word embeddings and using word embeddings as features for downstream NLP applications.

Our experiments were conducted on the Chinese language.

ISSN 2395-8618

The training corpus used is Xinhua News [3] of two years: 1997 and 2004. There are about 50 million tokens in the corpus and 80 thousand words occurring at least 5 times.



```
<word>伊拉克(Iraq)</word>
<level10>
  <word>她(She)</word>
  <level11>
    <word>颗粒物(Particles)</word>
    <level12>
      <word>人权(Human Rights)</word>
      <level13>
        <level14>
          <word>稳(Stable)</word>
          <level15>
            <word>04</word>
            <word>董事(Director)</word>
          </level15>
        </level14>
```

```
<level8>
  <level9>
    <word>伊拉克(Iraq)</word>
    <level10>
      <word>俄罗斯(Russia)</word>
      <level11>
        <word>伊朗(Iran)</word>
        <level12>
          <word>阿富汗(Afghanistan)</word>
          <level13>
            <word>利比亚(Libya)</word>
            <word>波黑(Bosnia and Herzegovina)</word>
          </level13>
        </level12>
      </level11>
```

Fig. 3. Top: Huffman tree generated from corpus. Bottom: Huffman tree after adjustment.

The following models are compared in our experiments:

a) HS-Huffman: hierarchical softmax with Huffman tree.

b) HS-SemHuff: hierarchical softmax with SemHuff. The similarity knowledge used is the ontology generated by the hierarchical clustering algorithm by He et al. [6].

c) NS: negative sampling with 7 negative samples.

All models are used with skip-gram and run 20 iterations over the entire corpus. The subsampling rate is set to 10^{-5}.

A. *Word Similarity Task*

We extract 3020 pairs of synonyms from *Tongyici Cilin* (available at http://ir.hit.edu.cn/demo/ltp/Sharing_Plan.htm), which is a manually built Chinese thesaurus. *Tongyici Cilin* comprises sets of Chinese synonyms. The synonym pairs are chosen from synonym sets whose size is not greater than 10, because large synonym set in *Tongyici Cilin* tends to be inconsistent.

For each extracted synonym pair ⟨A, B⟩, we measure the rank of B in a set of candidate words by the distance of its word embedding to the word embedding of A. The mean rank (MR) and mean reciprocal rank (MRR) is used to evaluate the quality of learned word embeddings. For MR, lower is better; while for MRR, higher is better. The result for different models in different settings is showed in Figure 4 and Figure 5.

The MR and MRR give consistent evaluation: NS > HS-SemHuff > HS-Huffman. Though, the MR value of several hundred is somehow counterintuitive. After investigation, the reason can be explained as follows. Only one word embedding is learned for a certain word, while it is always the case that one word has multiple meanings and usages. Take 团长 and 参谋长 for an example, 参谋长 means "a chief of staff in an army", while 团长 have many meanings. One of its meanings is "a regimental commander", which is used in the context of military affairs, so it may co-occur with 参谋长 frequently. Besides, 团长 can also be used to refer to the head of a delegation, a circus troupe, an opera troupe, and many other groups. This

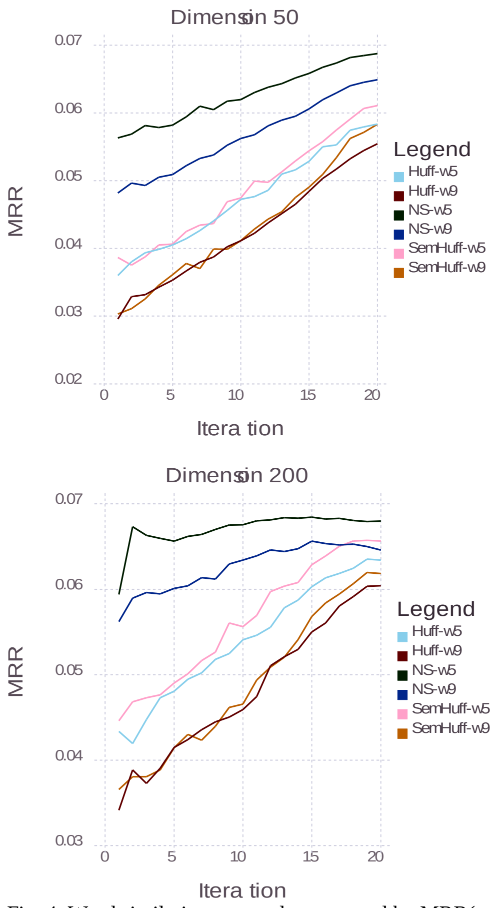


Fig. 4. Word similarity test results measured by MRR(mean reciprocal rank). The x-axis is the number of iterations over corpus during the training of the word embedding. The first row presents results for 50-dimensional vectors, while the second row is for 200-dimensional vectors.

asymmetry in word usage leads to the following phenomenon: for the word 参谋长, 团长 is its close neighbor, because 参谋长 is only used in military topics and 团长 is also salient in this context. But when we stand in the view of 团长, 参谋长 is not similar to 团长, because many other words such as 演员(actor), 代表(representative) will occupy the vicinity of 团长, and 参谋长 will get a rank worse than 1000. If we average a small number and a large number, say 10 and 1000, the result will be several hundred. What's more, the training corpus and the testing word pairs are not exactly in-domain, which contributes to this large rank, too.

B. *Analogy Task*

Our second evaluation method is the analogy task. If the relation between word A and B is similar to the relation between C and D, then *vector*(A) - *vector*(B) ≈ *vector*(C) - *vector*(D). Since we use Xinhua News as training data, information about Chinese cities and provinces should be attained. Thus, we choose 20 pairs of ⟨provincial capital city⟩:⟨province⟩ to

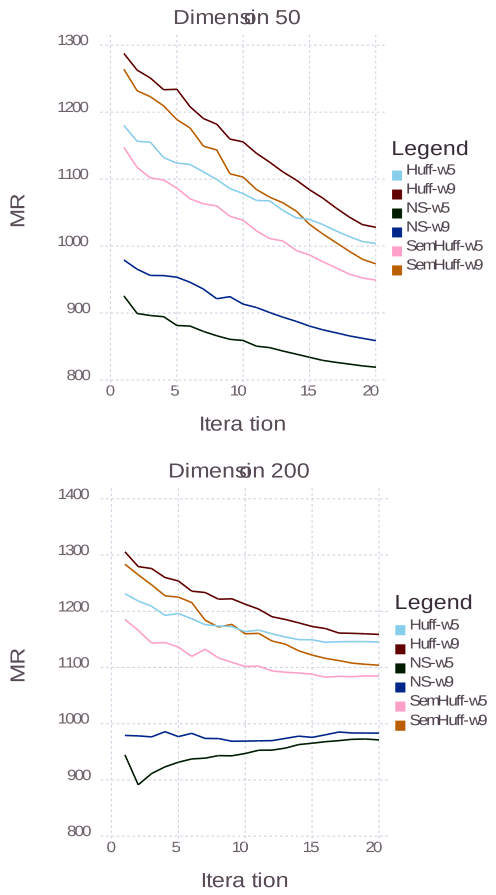


Fig. 5. Word similarity test results measured by MR(mean rank). The x-axis is the number of iterations over corpus during the training of the word embedding. The first row presents results for 50-dimensional vectors, while the second row is for 200-dimensional vectors.

generate $20^2 = 400$ analogy problems. In each problem, a city-province pair, say A and A', and another city, say B, are given, while the province whose capital city is B is unknown. Then, we search the word X from the whole vocabulary to fill in the blank such that the analogy identity *vector*(city A) – *vector*(province A') = *vector*(city B) – *vector*(X) is fitted as well as possible.

All models were trained for 20 iterations and the test is performed after every iteration. The best result for each model is showed in Table 1. The result is encouraging: word embeddings always give some province as the answer although we search for the word X among all words in the vocabulary. A precision of 75% can be achieved with our *SemHuff* model.

From these results, we see that NS is powerful for dimension 50 but is surpassed by hierarchical softmax models in the high dimensional case. Our *HS-SemHuff* model improves HS-Huffman significantly in the 50-dimensional case, and it outperforms HS-Huffman in every experimental setting, and gives the best result for dimension 200.

C. *POS Tagging Task*

The third task is using word embeddings as features for POS tagging. In this task, we use a very simple POS tagger: for a word *w*, we concatenate the word embeddings of words in the context windows of *w* as features.

TABLE I
ANALOGY TEST RESULTS

Model Name	Context Window Size	Word Vector Dimension	# of Correct Predictions
Huff	5	50	143
NS	5	50	**235**
SemHuff	5	50	217
Huff	9	50	161
NS	9	50	**254**
SemHuff	9	50	220
Huff	5	200	267
NS	5	200	194
SemHuff	5	200	**280**
Huff	9	200	294
NS	9	200	268
SemHuff	9	200	**300**

And a linear softmax classifier is trained on these features. This simple model is used because we think simpler models can better reflect the quality of input features. The results are showed in Figure 6.

In this task, NS performs better. As for two hierarchical softmax models, our *HS-SemHuff* is comparable with HS-Huffman in the low dimensional case and gives better results for dimension 200.

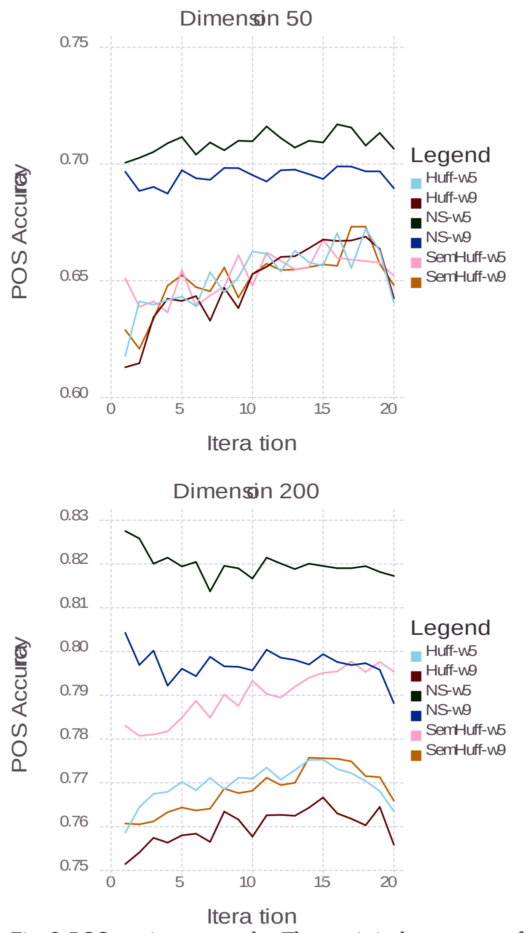


Fig. 6. POS tagging test results. The y-axis is the accuracy of the resulting POS tagger. The x-axis is the number of iterations over corpus during the training of word embeddings.

ISSN 2395-8618

V. Conclusion

In this paper, we seek to give some theoretical analysis on the widely used hierarchical softmax algorithm. By treating the tree structure as a parameter of the training objective function, we show that the reason why the common practice of using the Huffman tree works well is that the Huffman tree maximizes the objective function when word embeddings are random. We also show that the dividing criterion is the conditional entropy if we adopt a top-down splitting tree constructing scheme. Following the theoretical analysis, we propose *SemHuff*, a tree constructing scheme based on adjusting the Huffman tree. From the experiments, we show that negative sampling performs well in most tasks while hierarchical softmax performs better in high dimensional analogy task. And *SemHuff* further improves the original hierarchical softmax algorithm in all of our tasks.

In fact, a more natural idea is to directly optimize the training objective function with respect to the tree structure instead of adjusting a Huffman tree. However, the optimization over the space of all binary trees seems hard. We think some approximation or relaxation is necessary to solve this optimization problem. We leave this as future work for this research.

References

[1] Yoshua Bengio et al., “A Neural Probabilistic Language Model,” in: *The Journal of Machine Learning Research* 3 (2003), pp. 1137 – 1155.

[2] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch, “ PPDB: The Paraphrase Database,” in: *Proc. of NAACL-HLT,* Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 758 – 764.

[3] David Graff and Ke Chen, “Chinese Gigaword,” in: *LDC Catalog No.: LDC2003T09, ISBN* 1, 2005, pp. 58563 – 58230.

[4] M Gutmann and A Hyvärinen, “Noise-contrastive estimation: A new estimation principle for unnormalized statistical models,” in: *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 1 – 8.

[5] Zellig S Harris, “Distributional structure,” in: *Word* 10.2-3 (1954), pp. 146 – 162.

[6] Shaoda He et al., “Construction of Diachronic Ontologies from People’s Daily of Fifty Years,” in: *Proc. of the Ninth International Conference on Language Resources and Evaluation*, Edited by Nicoletta Calzolari (Conference Chair) et al. Reykjavik, Iceland: European Language Resources Association (ELRA), May 2014.

[7] Vladimir Kolmogorov, “Blossom V: A new implementation of a minimum cost perfect matching algorithm,” in: *Mathematical Programming Computation* 1.1, 2009, pp. 43 – 67.

[8] Siwei Lai et al., 2015. “How to Generate a Good Word Embedding?.” Available: http://arxiv.org/abs/1507.05523

[9] Omer Levy and Yoav Goldberg, “Neural Word Embedding as Implicit Matrix Factorization,” in: *NIPS*, 2014, pp. 1 – 9.

[10] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig, “Linguistic regularities in continuous space word representations,” in: *Proc. of NAACL-HLT*, June. 2013, pp. 746–751.

[11] Tomas Mikolov et al., “Efficient Estimation of Word Representations in Vector Space,” in: *Proc. of the International Conference on Learning Representations*, 2013, pp. 1–12.

[12] George A Miller, “WordNet: a lexical database for English,” in: *Communications of the ACM 38.11*, 1995, pp. 39–41.

[13] Andriy Mnih, “Learning word embeddings efficiently with noise-contrastive estimation,” in: *NIPS*, 2013, pp. 1–9.

[14] Andriy Mnih and Geoffrey Hinton, “Three new graphical models for statistical language modelling,” in: *Proc. of the 24th international conference on Machine learning*, 2007, pp. 641–648.

[15] Andriy Mnih and Geoffrey E Hinton, “A Scalable Hierarchical Distributed Language Model,” in: *NIPS*, 2008, pp. 1081–1088.

[16] Frederic Morin and Y Bengio, “Hierarchical probabilistic neural network language model,” in: *Proc. of the Tenth International Workshop on Artificial Intelligence and Statistics*, 2005, pp. 246–252.