Improvement of Reverse Dictionary by Tuning Word Vectors and Category Inference

Yuya Morinaga(✉) and Kazunori Yamaguchi(✉)

The University of Tokyo, 3-8-1 Komaba, Meguro-ku, Tokyo 153-8902, Japan
{morinaga,yamaguch}@graco.c.u-tokyo.ac.jp

Abstract. A reverse dictionary is a system that returns words based on user descriptions or definitions. OneLook Reverse Dictionary is a commercial reverse dictionary system constructed from existing dictionaries. Hill (2016) reported another reverse dictionary system was constructed from public dictionaries using word embeddings and that its performance was comparable to that of OneLook Reverse Dictionary at the time of the comparison. In this paper we report that, by selecting word vectors suitable for a reverse dictionary and combining Convolutional Neural Network text classification, we improved the reverse dictionary described by Hill. It is very significant that our model can automatically construct a reverse dictionary system from publicly available resources such that it obtains similar scores to those obtained with OneLook Reverse Dictionary in accuracy@100/1000. We also show that our model can be used as a filter to the OneLook Reverse Dictionary to improve its performance.

Keywords: Reverse dictionary · Concept search · Word embedding
Recurrent neural network · Convolutional neural network
Text classification · WordNet

1 Introduction

A dictionary maps a word to its definition while a *reverse dictionary* maps a description to the word specified by the description. Reverse dictionary applications include the tip-of-the-tongue problem [3] and the cross word problem [1].

One of the difficulties in developing a reverse dictionary is that we can not exhaustively enumerate descriptions for a word. For example, the definition of the word 'brother' in WordNet [4] is 'a male with the same parents as someone else', but a reverse dictionary should be able to map the description of 'son of my parents' to 'brother' also. To achieve this, a reverse dictionary should have some mechanism to calculate the similarity between unseen inputs and candidate words.

Several academic studies have proposed reverse dictionary models. Most of previous academic researches on English reverse dictionaries, such as [5,6], use hand-engineered features of sentences. On the other hand, [1] proposed reverse dictionary model based on Neural Language Model [7], which uses word embeddings as features. Reverse dictionary models of [1,6] are publicly available.

The Node-Graph Architecture proposed in [6] is a reverse dictionary that searches words in a graph where nodes are words and edges are the relations whose source nodes (words) appear in the descriptions of the target nodes (words). In this graph, the length of a path between nodes represents the distance between the nodes and the frequency of words represents the importance of the words. Outputs are the words that are closest to the important input words. This reverse dictionary was claimed to outperform OneLook Reverse Dictionary[1] when the vocabulary is limited to 3,000 words. However, the performance for a larger vocabulary is not available.

The Reverse Dictionary using Word Embeddings [1] (RDWE for short) can handle a larger vocabulary. This reverse dictionary converts the word vectors of word2vec for an input description into a vector by a linear transformation or a Recurrent Neural Network. This reverse dictionary was claimed to have the performance similar to that of OneLook Reverse Dictionary at the moment of [1] in an evaluation using the descriptions provided by users.

In this paper we improved the performance of the reverse dictionary through the use of two techniques. First, we generated word vectors more suited to the use of a reverse dictionary by using the dictionaries with non-ASCII characters removed. Second, following in the Watson [8] which is the Question Answering system employs category inference of answer, we employed a category information to eliminate similar but irrelevant vectors. We found that words belonging to different categories may have similar word vectors. This causes the reverse dictionary to provide incorrect words in irrelevant categories. This problem can be alleviated by filtering out words in the irrelevant categories.

Experiment results we obtained showed that the present OneLook Reverse Dictionary performs better than the reverse dictionary reported in [1] and outperformed RDWE including this work in some metrics. However, RDWE results are quite different from those returned by OneLook Reverse Dictionary, although the rate at which the target word is included in the top 100–1000 words is similar in both dictionaries. Accordingly, we were able to use RDWE as an output filter of OneLook Reverse Dictionary and in so doing we confirmed that search performance was improved.

The contributions of this paper are as follows:

- Improved RDWE accuracy by employing better word vectors
- Improved RDWE accuracy by employing category inference
- Improvement over a commercial reverse dictionary by employing RDWE as a filter.

The rest of this paper is organized as follows. The models we used are explained briefly in Sect. 2. Improvement and analysis on word vectors are explained in Sect. 3. We describe our proposed model in Sect. 4. Section 5 shows the experimental results we obtained. Section 6 concludes the paper with a summary of key points.

[1] http://onelook.com/reverse-dictionary.shtml.

2 Preliminary Definitions

In this section, we will briefly explain RDWE in Sect. 2.1 and a text classification model in Sect. 2.2.

We will denote the vector for word w as v_w and a description as $s = (w_1, w_2, ..., w_n)$.

2.1 Distributed Representation of Sentences

The RDWE outputs words sorted by rank determined by the cosine similarity between the vector of an input description and the vectors of the word. RDWE based on Neural Language Model [7].

Word vectors may be learned independently from a corpus or learned simultaneously in adjusting RDWE parameters. [1] reported that there was almost no difference in the results obtained for either case. In our work, we therefore used word vectors learned from a corpus.

word2vec Add (ADD). We call the model to use the sum of the word vectors of words in a description as *word2vec add model* (ADD for short). If word vectors are given, no learning is needed in ADD. In the work we describe in this paper, we used ADD as the baseline. In experiments we performed, we omitted word vectors of stop words in the summation although they were not omitted in [1]. This omission improves the accuracy.

Bag of Words (BOW). The *Bag of Words* distributed representation model (*BOW* for short) outputs a linearly transformed summation of the vectors of input words as follows. Here, D is the dimension of v, W is a matrix of $D \times D$, and A_j is a D dimensional vector.

$$\begin{aligned} A_1 &= W v_{w_1}, \\ A_{i(>1)} &= A_{i-1} + W v_{w_i} (= W \sum_{j=1}^{i} v_{w_j}). \end{aligned}$$

Because W is a linear transformation, A_n is equal to the summation of vectors of the words in a description multiplied by W. The order of words is neglected in this model. We learn W by the stochastic gradient method minimizing the cost between the vector thus calculated and the correct word vector available in the training data. As a cost function, the cosine distance and the rank loss function are used. The rank loss function is determined as follows.

$$\max(0,\ 0.1 - \cos(v_s, v_t) + \cos(v_s, v_{\text{rand}})),$$

where s is a description, t is the correct word, v_s is the vector for s, v_{rand} is a randomly selected vector not identical to v_t, and $\cos(a, b)$ is the cosine similarity function.

Recurrent Neural Network (RNN). *Recurrent Neural Network distributed representation model* (*RNN* for short) inputs words in a description one by one to calculate the next vector and outputs the final vector. Here, A_j is the vector after the j-th word in a description is processed.

$$\begin{aligned} A_1 &= \phi(Wv_{w_1} + b), \\ A_{i(>1)} &= \phi(UA_{i-1} + Wv_{w_i} + b), \end{aligned}$$

where W and U are $D \times D$ matrices, A_j is a D-dim vector, and $\tanh(x)$ is used as an activation function $\phi(x)$.

A_n is the final distributed representation representing the description $(w_1, w_2, ..., w_n)$. In this model, the order of words has an effect on the final output. W, U, and b are learned by the learning method and the cost function for RNN. LSTM (Long Short Term Memory) [9] is employed for long backpropagation.

2.2 Text Classification Model

Reverse dictionary can be regarded as a variety of Question Answering (QA) system, therefore QA system's technique can be applied to. IBM's Watson [8] is a well-known QA system that won against two of *Jeopardy*'s greatest champions in 2011. It classifies question, and judges appropriate answer type [10]. Watson's architecture is proprietary knowledge and specialized in *Jeopardy*, but still we can take similar approach by building another category inference system.

In the following, we will explain a text classification model we employed for our category inference system.

Convolutional Neural Network (CNN). *Convolutional Neural Network text classification model* (*CNN* for short) [2] is used to infer a category to which a word with the description belongs.

In CNN, C is the characteristic vector and p_i is the probability with which s belongs to category i for a given input s, $v_{w_{j:j+h-1}}$ is an $h \times D$ matrix of the concatenation of w_j to w_{j+h-1} of D-dim vectors, and W_i is a $h \times D$ matrix.

$$\begin{aligned} c_{i,j} &= \phi(W_i v_{w_{j:j+h-1}} + b_i), \\ c_i &= \max_j(c_{i.j}), \\ C &= (c_1, c_2, ..., c_k), \\ p_i &= \frac{\exp(c_i)}{\sum_m \exp(c_m)}. \end{aligned}$$

This transforms a word sequence into features and the features c_j are used to yield the probability of the i-th category by softmax.

We adjusted the parameters by using the stochastic gradient on the cost determined by the inferred probability and that in the training data. In order to avoid the overfitting, the dropout (masking W randomly when learning) was employed.

3 Word Vectors

3.1 Improving Word Vectors

The performance of RDWE depends on the word vectors it uses. The purpose of this section is to get word vectors suited to RDWE. We measured the suitability of word vectors by using ADD, i.e., the score obtained with the ADD reverse dictionary using the word vectors.

Note that we eliminated the stop words from the search space and overlooked stop words in the summation process. The target words were limited to those in the WordNet vocabulary.

There are two approaches to improve the word vectors. The first is to modify the corpus to use in the word2vec learning such as the tokenization of frequent phrases [11]. The second is to modify the model of word2vec such as the Multi-Sense Skip-gram model [12]. Because the second approach has potential problems with increased learning time and search space, we used the first approach.

Table 1. ADD scores. HIL uses word vectors in the Hill's distribution. AOC uses word vectors freshly learned from AOC.

Word vector	accuracy@1/10/100	Median
HIL	**0.02**/0.156/0.39	213
AOC	0.018/**0.192**/**0.454**	**143**

The results are shown in Table 1. The scores were calculated for the labeled data by users included in the Hill's data. We used the 23 GB text data collected from the Internet (including Wikipedia)[2] as the corpus.

HIL is the Hill's corpus including sentences with non-ASCII characters. The sentences are written in languages other than English and so they constitute noise for learning English word vectors. In total, 16 GB were left when we removed sentences which include non-ASCII characters from the corpus. We call the resulting corpus as *ASCII only corpus* (*AOC* for short).

Word vectors were learned from these corpora and their performances were measured by ADD. The test was conducted on the 500 samples obtained from WordNet. We used the accuracy@1/10/100, which is the rate at which the target word is in the top 1/10/100 word(s) and the median of the target word rank according to the rank ADD assigned for the word vectors.

The purpose of this learning of word vectors is to obtain word vectors suited for the reverse dictionary used for users' descriptions. Therefore, the selection of word vectors has to be done using users' descriptions. However because the number of available users' descriptions was small (200), all of them were used for the final evaluation and the selection of the word vectors was done by using

[2] Collected in the first part of https://github.com/svn2github/word2vec/blob/master/demo-train-big-model-v1.sh.

dictionary data. Each sample consists of a description and the word described by the description. We removed the part enclosed by parentheses because the enclosed part often includes the defined word. The results are shown in Table 1. The p-values of accuracy@1/10/100 and median were respectively 0.617, 0.067, 0.017, and 0.076. This means that our newly learned word vectors using AOC are statistically significantly better at accuracy@100.

3.2 Characteristics of Word Vectors

word2vec learns vectors of words from their context words. As a result, words with similar context have similar vectors in terms of the cosine similarity. However, they do not necessarily belong to the same 'category.' For example, 'bed' and 'asleep', 'cat' and 'meow', or 'mud' and 'muddy' have the similar context resulting in the similar word vectors. This is not desirable for word vectors in reverse dictionary applications because when the input sentence is something like 'an animal that...' it is obvious that the category of the target word is an animal (noun), but the search result may include 'meow' (verb).

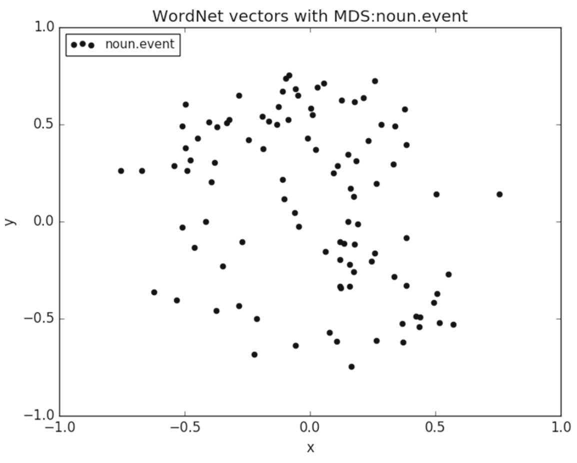
In the work we report in this paper, we employed the lexname in WordNet as a category. A lexname is one of 45 kinds of tags determined for each synset such as noun.animal, verb.emotion, and so on. Note that one word may belongs to multiple synsets and, as a result, have multiple lexnames.

Figure 1 shows the distribution of word vectors visualized in 2-dimensional MDS using the cosine distance between word vectors. From this figure, we see that the word vectors of these two categories overlap considerably. Roughly speaking, one third of the target words do not have any common categories with more than half of the nearest 100 words of the words. As this shows, a considerable number of words close to the calculated word vector fall in wrong categories.

4 Proposed Model

The distributed representation of the target word is calculated from the distributed representation of the words in an input sentence as in [1]. As shown in Sect. 3.2, words similar to the distributed representation may belong to wrong categories. By using the category of the target word, we can eliminate the obviously irrelevant words from the ranked result and improve the accuracy. Even though the category of the target word is not given, we can infer its category. In our work, we employed the CNN [2] for this.

The CNN result is the distribution on categories. If we use a small number of categories, the chance that the correct category is included is small. Once it is included the rank of the target word in the ranked result is high, but otherwise, we will miss the target word in the ranked list. Therefore, there is a trade-off between the precision and recall. We propose to control the trade-off by the estimated probability of selected categories. Let us assume that the probabilities of the target word t_q of an input sentence q belongs to the categories $c_1, c_2, ..., c_n$



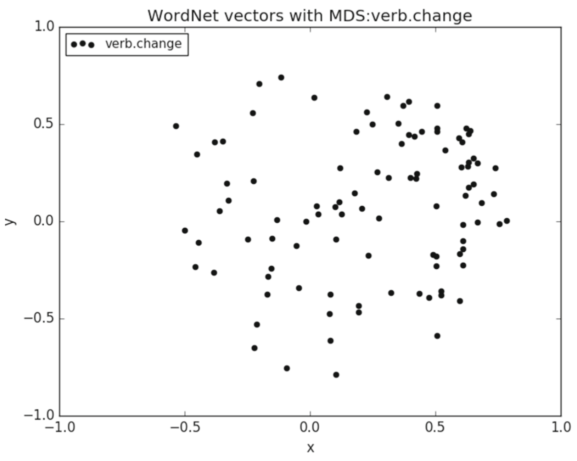


Fig. 1. 2-dimensional MDS of word vectors: 100 sample words each in noun.event (above) and verb.change (below).

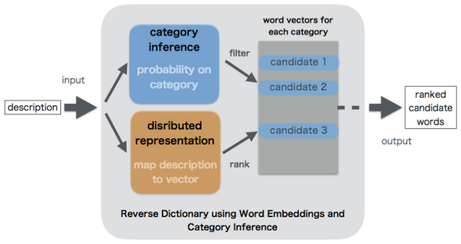

description
input
category inference
probability on category
disributed representation
map description to vector
filter
rank
word vectors for each category
candidate 1
candidate 2
candidate 3
ranked candidate words
output
Reverse Dictionary using Word Embeddings and Category Inference

Fig. 2. Reverse dictionary using word embeddings and category inference (RDWECI) (our proposal)

are $p_{c_1} \geq p_{c_2} \geq ... \geq p_{c_n}$[3]. Let us also assume that the probabilities belonging to categories are independent. Then, the probability that t_q does not belong to any of the top m categories is $\prod_{i=1}^{m}(1 - p_{c_i})$. By employing m_0 categories where $m_0 = \underset{m}{\arg\min} \prod_{i=1}^{m}(1 - p_{c_i}) < k$, we can keep this *failure rate* as low as k. Finally, we rank only words which are in the m_0 categories.

Our algorithm is summarized below.

Input: input sentence q, failure-rate k.
1. Calculate the distributed representation of v_q of q.
2. Calculate the probabilities $p_{c_1^q} \geq p_{c_2^q} \geq ... \geq p_{c_n^q}$ on categories with which the target word of q belong to the categories.
3. Determine the top m_0 categories $c_1^q, ..., c_{m_0}^q$ for k.
4. Rank words belonging to the categories $c_1^q, ..., c_{m_0}^q$ according to the cosine similarity with the word vector of the words and v_q.
Output: Top 1/10/100 word(s) in the ranked words.

The whole architecture is shown in Fig. 2. We call this model *RDWECI*.

5 Experiment

5.1 Data

Training Data. For training Hill's reverse dictionary model, we used the word and description pairs obtained from Wordnik API[4]. We accessed the word and description pairs from Wordnik API according to a word list distributed by Hill, removed remaining HTML tags such as "`<strong>`" from the obtained data. The results were about 540,000 word and description pairs and used in Sect. 5.2.

For training the text classification model, we used the lexname and definition pairs for each synset of WordNet. The results were about 120,000 category and definition pairs and is used in Sect. 5.3.

As to word vectors, we used HIL and AOC in Sect. 3.

Test Data and Evaluation Metrics. For the evaluation, we used the 200 word and description pairs in the data distributed by Hill.

For evaluation metrics, we used accuracy@1/10/100 and median.

Because we could search only a word whose category is defined in WordNet, our search space included the 90,000 words of the WordNet whose word vectors were available. This is broader than the 66,000 words space used by Hill.

Because about 80 words in the Hill's corpus are not included in AOC, the search space was slightly different between HIL and AOC.

For evaluating the text classification model, we used the above data and WordNet's seen and unseen data. The WordNet's data consists of the lexname

[3] In our experiment, $n = 45$.
[4] http://developer.wordnik.com.

and definition pairs for each synset of WordNet in Sect. 5.1. In calculating the accuracy, we considered the result was correct when the top category inferred was included in the categories to which one of the synsets of the word is assigned in WordNet. If the WordNet data is used, because the description defines a synset, we used the category of synset as the correct one. One definition in the WordNet corresponds to one synset having one category (lexname). Each of Hill's data is a pair of a word and its description, and the word may belong to multiple synsets and categories. Thus, for the evaluation using the Hill's data, we considered the result was correct if one of the categories to which the target word belongs is named.

5.2 Reverse Dictionary Model

Training. The reverse dictionary model is trained as Sect. 2.1. As to word vectors, we used those in HIL and AOC as in Sect. 3.1. We tried two types of networks: BOW and RNN, and two types of cost functions: cosine and rankloss. We used the implementation by Hill[5].

Evaluation. In this section, we conducted the evaluation of the trained reverse dictionary model by the data in Sect. 5.1. The results are shown in Figs. 3 and 4 in Sect. 5.4.

In general, the scores for AOC were better than those for HIL. As to networks and cost functions, BOW rankloss was the best. RNN was better in accuracy@1, but not so for other metrics. This may be the result of the overfitting by RNN which is more complex and flexible than BOW.

5.3 Text Classification Model

Training. As to CNN in Sect. 2.1 we used the implementation in the Harvard NLP[6]. As to word vectors, we used HIL and AOC as in Sect. 3.1.

Evaluation. The accuracy of the trained model was 93% (HIL)/94% (AOC) for the trained WordNet definitions, 83% (HIL)/84% (AOC) for the WordNet definitions not included in the training data, and 70% (HIL)/70% (AOC) for the Hill's evaluation data.

5.4 Evaluation of RDWECI

We evaluated RDWECI proposed in Sect. 4 by combining the models in Sects. 5.2 and 5.3. The results are shown in Figs. 3 and 4.

[5] https://github.com/fh295/DefGen2.
[6] https://github.com/harvardnlp/sent-conv-torch.

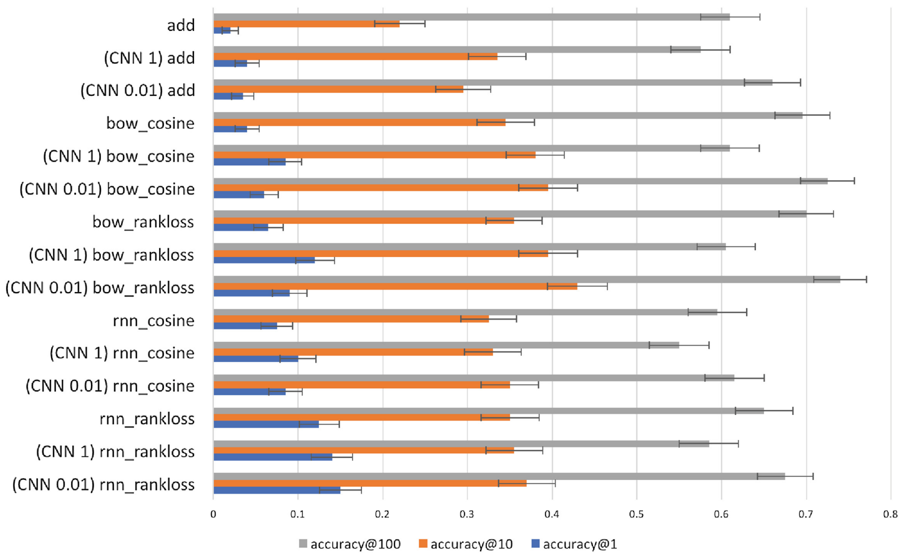


Fig. 3. Accuracy for various models and parameters for AOC. CNN is our model and others are Hill's model. Error bars show the standard deviations.

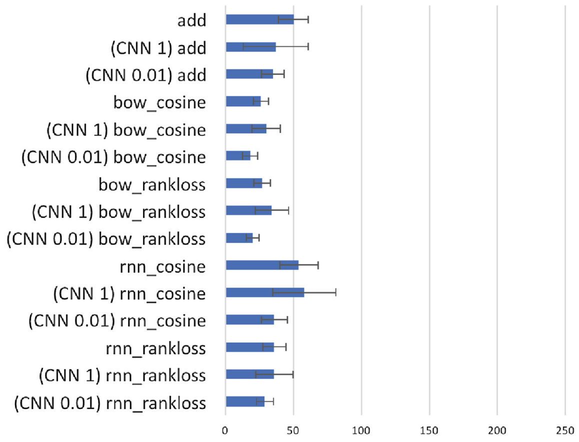


Fig. 4. Median for various combinations and parameters for AOC. CNN is our model and others are Hill's model. Error bars show the standard deviations.

For the failure-rate in Sect. 4, we tried 1 and 0.01. This failure-rate was chosen as follows. We tested the model with failure-rate = 1/0.5/0.4/0.3/0.2/0.1/0.05/0.01/0.001 using the dictionary data and the best score is obtained at failure-rate = 0.01. Therefore, in the following, we will use 0.01 as the failure-rate. For comparison, we use 1 as the failure-rate, because with failure-rate = 1 only the top category is used and this is the other extreme. In order to avoid overfitting, we didn't use the test data for this tuning.

In the realization of the Hill's experiment in [1], BOW rankloss using Hill's distributed data performed best resulting in accuracy@1/10/100 = 0.06/0.325/0.61 and median = 50. According to RDWECI, CNN 0.01 BOW rankloss AOC performed best marking accuracy@1/10/100 = 0.09/0.43/0.74 and median = 20. The difference between Hill's model and ours best are statistically significant in accuracy@10 ($p = 1.59 \times 10^{-2}$), accuracy@100 ($p = 2.20 \times 10^{-3}$), and median ($p = 4.40 \times 10^{-3}$).

5.5 OneLook Reverse Dictionary

Evaluation of OneLook Reverse Dictionary. OneLook Reverse Dictionary is the commercial reverse dictionary system. One can search words from their description from 1,061 dictionaries.

The score of the OneLook Reverse Dictionary was reported as accuracy@10/100 = 0.38/0.58[7] and median = 18.5, similar to the performance of Hill's BOW/RNN models.

We used data included in Hill's distribution in Sect. 5.1 for this and the next evaluation.

The scores of our RDWECI were higher than those of Hill's BOW/RNN model, and the OneLook Reverse Dictionary at the moment of [1].

We evaluated the score of the OneLook Reverse Dictionary on Oct.16, 2017. The results were accuracy@1/10/100 = 0.34/0.55/0.76 and median = 5 and accuracy@1/10/100 = 0.33/0.54/0.76 and median = 6.

OneLook Reverse Dictionary Enhanced by the Distributed Representation and Category Inference. OneLook Reverse Dictionary and RDWECI have different architectures and only 10–15% words are common in the top 1000 words lists of them. OneLook Reverse Dictionary performs well for accuracy@1/10 but RDWECI performs relatively well for accuracy@100/1000; the accuracy@100/1000 of OneLook Reverse Dictionary is 0.76/0.87 and that of RDWECI is 0.74/0.89.

From this observation, we constructed the following algorithm to combine the bests.

```
Input: Input sentence $q$.
1. Calculate the top 1,000 words $m$ by RDWECI for $q$.
2. Search the top 1,000 words $r$ by OneLook Reverse Dictionary for $q$.
3. Remove words from $r$ which are not in $m$.
Output: $r$
```

As shown in Table 2, the combination improved the score.

In this combination, the BOW and RNN did not perform any better than ADD. This may be because ADD is not so flexible in providing the target word at a higher rank, but it is robust enough to keep it in the top words list.

[7] accuracy@1 was not reported in [1].

Table 2. OneLook Reverse Dictionary Enhanced by the Distributed Representation and Category Inference (1) ADD, (2) CNN 0.01 add, (3) BOW rankloss, (4) CNN 0.01 BOW rankloss. All word vectors are from AOC.

Model	accuracy@1/10/100	Median
OneLook	0.34/0.55/0.76	5
(1) + OneLook	0.36/0.62/0.81	4
(2) + OneLook	**0.38**/**0.63**/**0.82**	**3**
(3) + OneLook	0.35/0.60/0.80	4
(4) + OneLook	0.36/0.61/**0.82**	**3**

6 Conclusion

In the work we described in this paper, we improved the accuracy of the reverse dictionary of [1] by selecting better word vectors and introducing category inference. Our best model performs similarly to OneLook Reverse Dictionary according to accuracy@1000. This is very significant because OneLook Reverse Dictionary is a commercial and dedicated system while our system is built using only category information as natural language knowledge and public dictionaries. We showed that we can also use RDWECI as a filter to improve the OneLook Reverse Dictionary. Various subjects for future work remain. First, we would like to use multi-sense word vectors in the reverse dictionary. By capturing more precise meanings of words by using such vectors in an input sentence, we may be able to improve reverse dictionary performance. Second, we would like to build a reverse dictionary model more suitable to be used as filter of OneLook Reverse Dictionary. As a filter to the OneLook Reverse Dictionary, our model with category inference shows only a marginal improvement to ADD. By employing a more suitable function to improve accuracy@1000, we may find a filter more suited to OneLook Reverse Dictionary.

References

1. Hill, F., Cho, K.H., Korhonen, A., Bengio, Y.: Learning to understand phrases by embedding the dictionary. Trans. Assoc. Comput. Linguist. **4**, 17–30 (2016)
2. Kim, Y.: Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1746–1751 (2014)
3. Schwartz, B.L., Metcalfe, J.: Tip-of-the-tongue (TOT) states: retrieval, behavior, and experience. Mem. Cogn. **39**(5), 737–749 (2011)
4. Miller, G.A.: Wordnet: a lexical database for english. Commun. ACM **38**(11), 39–41 (1995)
5. Shaw, R., Datta, A., VanderMeer, D., Dutta, K.: Building a scalable database-driven reverse dictionary. IEEE Trans. Knowl. Data Eng. **25**(3), 528–540 (2013)
6. Thorat, S., Choudhari, V.: Implementing a reverse dictionary, based on word definitions, using a node-graph architecture. In: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics, pp. 2797–2806 (2016)

7. Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C.: A neural probabilistic language model. J. Mach. Learn. Res. **3**(Feb), 1137–1155 (2003)
8. Ferrucci, D., et al.: Building Watson: an overview of the DeepQA project. AI Mag. **31**(3), 59–79 (2010)
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
10. Lally, A.: Question analysis: how Watson reads a clue. IBM J. Res. Dev. **56**(3.4), 2:1–2:14 (2012)
11. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, pp. 3111–3119 (2013)
12. Neelakantan, A., Shankar, J., Passos, R., Mccallum, A.: Efficient nonparametric estimation of multiple embeddings per word in vector space. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1059–1069 (2014)