Adjusting Word Embeddings by Deep Neural Networks

Xiaoyang Gao[1] and Ryutaro Ichise[2,3]
[1]*School of Electronics Engineering and Computer Science, Peking University, Beijing, China*
[2]*National Institute of Informatics, Tokyo, Japan*
[3]*National Institute of Advanced Industrial Science and Technology, Tokyo, Japan*

Keywords: NLP, Word Embeddings, Deep Learning, Neural Network.

Abstract: Continuous representations language models have gained popularity in many NLP tasks. To measure the similarity of two words, we have to calculate their cosine distances. However the qualities of word embeddings depend on the corpus selected. As for word2vec, we observe that the vectors are far apart to each other. Furthermore, synonym words with low occurrences or with multiple meanings are even further in distance. In these cases, cosine similarities are no longer appropriate to evaluate how similar the words are. And considering about the structures of most of the language models, they are not as deep as we supposed. "Deep" here refers to setting more layers in the neural network. Based on these observations, we implement a mixed system with two kinds of architectures. We show that adjustment can be done on word embeddings in both unsupervised and supervised ways. Remarkably, this approach can successfully handle the cases mentioned above by largely increasing most of synonyms similarities. It is also easy to train and adapt to certain tasks by changing the training target and dataset.

1 INTRODUCTION

To understand the meanings of words is the core task of natural language processing models. While still hard to compete with a human-like brain, many models successfully reveal certain aspects of similarity relatedness using distributed representation. These word embeddings are trained over large and unlabeled text corpus leveraging different kinds of neural networks (Bengio et al.(2003)Bengio, Ducharme, Vincent, and Jauvin; Collobert and Weston(2008); Mnih and Hinton(2009); Mikolov et al.(2011)Mikolov, Kombrink, Burget, Černockỳ, and Khudanpur) and have obtained much attention in many fields, such as part-of-speech tagging (Collobert et al.(2011)Collobert, Weston, Bottou, Karlen, Kavukcuoglu, and Kuksa), dependency parsing (Chen and Manning(2014)), syntactic parsing (Socher et al.(2013)Socher, Bauer, Manning, and Ng), etc.

One of the popular neural-network models is word2vec (Mikolov et al.(2013a)Mikolov, Chen, Corrado, and Dean; Mikolov and Dean(2013b)), in which words are embedded into a low-dimensional vector space, which corresponds to words in a way based on the distributional hypothesis: words that occur in similar context should have similar meanings. The hypothesis captures the semantic and syntactic relations between words, and learned vectors encode these properties. Both of the semantic and syntactic regularities can be revealed from linear calculation between word pairs: *boys - boy* $\approx$ *cars - car*, and *king - man* $\approx$ *queen - woman*. The means of measuring the similarity between the source word and the target word is to calculate the cosine distance. Bigger cosine similarity indicates that the target word is more similar to the source word than the others. After applying the model, the embedded vectors are spread in a large Euclidean space in which words are separated far from others, leading to sparse vectors. Traditional methods to measure the semantic similarity are often operated on the taxonomic dictionary WordNet (Miller(1995)) and exploit the hierarchy structure. (Menéndez-Mora and Ichise(2011)) proposed a new model which modifies the traditional WordNet-based semantic similarity metrics. But these methods do not consider the context of words, and it is hard to figure out the syntactic properties and linguistic regularities. A good idea is to combine WordNet with neural network models for deep learning.

Words that are synonyms denote that they mean nearly the same sense in the same language and they are interchangeable in certain contexts. For exam-

Gao X. and Ichise R.
Adjusting Word Embeddings by Deep Neural Networks.
DOI: 10.5220/0006120003980406
In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*, pages 398-406
ISBN: 978-989-758-220-2

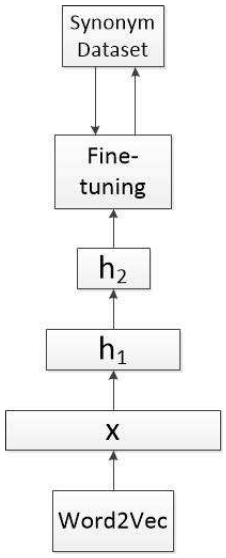

Figure 1: The framework of our system. We feed word representations to a deep neural network for pre-training, and then use learned parameters for a deep neural network which implements for fine-tuning. x is the input, h_1 and h_2 are the learned hidden representations from unsupervised model. Synonym dataset is used for supervised learning in fine-tuning model.

ple, *blush* and *flush* can be both interpreted as becoming red in the face when feeling embarrassed or shamed, but the cosine similarity between words is only around 0.2 when we obtain the vector by word2vec. It is also observed that word2vec doesn't recover another kind of relational similarity, for example, *regularize* and *regularise* are totally the same, but such word pair only gains around 0.7 cosine similarity which illustrate the sparse and scattered properties of word2vec representations. In this paper, we propose two different deep neural networks to compress the vectors and reduce the dimensionality in both supervised and unsupervised ways. Unsupervised deep neural networks are usually applied for pre-training and extracting meaningful features, and can achieve better performance comparing to the results without unsupervised learning process. Considering there is only one hidden layer in word2vec, we show that leveraging deep neural networks on learned vectors can lead to competitive and state-of-the-art results. By adjusting the elements in the vector, we found that using autoencoders can improve the similarity between synonyms, exhibiting the nature of this relation, making the cosine similarity more plausible to human perception. Without using other datasets like **WordNet** and regardless of the corpus used, stacked autoencoders can *automatically* enable the synonyms to get closer in the space.

A key insight of the work is that vectors from word2vec can be learned deeply by the neural network. Deep neural networks(DNNs) have achieve remarkable performances in many critical tasks comparing to traditional machine learning methods. (Krizhevsky et al.(2012)Krizhevsky, Sutskever, and Hinton) proposed a deep convolutional neural network which achieved record-breaking results on ImageNet classfication. (Dahl et al.(2012)Dahl, Yu, Deng, and Acero) also presented a novel DNN model with deep belief network for pre-training in speech recognition. The model reduced relative error and outperformed conventional models. Previous works inspire us to use DNNs on raw word representations. After adjustment, the model produces markedly different word representations and we find that the cosine similarities between most of the synonyms are improved. Together with a fine-tuning model and a novel loss function, all compressed word embeddings from hidden layers achieve state-of-the-art performances at recovering the synonym relation and deceasing distances between non-synonym word pairs. This result reflects the potential energy of autoencoders and the space for deep learning of the vectors from word2vec.

In this paper, we will introduce related work in Section 2, and present our deep learning model in Section 3. The experiment setup and results will be presented in Section 4, as well as the discussion. We will conclude our system and findings in Section 5.

2 RELATED WORK

Traditional language models reconstruct certain word co-occurrence matrix and represent words as high dimensional but sparse vectors, then reduce the dimension by matrix-factorization methods, such as SVD (Bullinaria and Levy(2007)), or LDA (Blei et al.(2003)Blei, Ng, and Jordan). Recently, dense vectors from neural network language models referred to as "word embedding" perform well in a variety of NLP tasks.

Neural network language models (NNLMs) "embed" words from large, unlabeled corpus into a low-dimensional vector space and outperform traditional models. Moreover, neural-network based models have been successfully applied in many specified fields, such as analogy answer tasks, named entity recognition, etc. In NNLMs, each word is presented as a k-dimension vector with real numbers, and two vectors that have a high cosine similarity result indicate that the corresponding words are semantically or syntactically related. Among all these models, word2vec which largely reduced the computational complexity gains the most popularity. The word embeddings of the model capture the *attributional similarities* for words that occur in the similar context. As for word2vec, the model increases the dot product of words that co-occur in the corpus, and this leads to the result that words that are semantic or syntactic related are closed to each other in the representation space. There are two structures for the model, each has its own characteristics and applicability in different cases. The continuous bag-of-words (CBOW) predict the target word by the window of words to the left as well as to the right of it. The Skip-gram model predicts the words in the window from current word.

However, we can find some problems in the model. Words in English always have multiple meanings. To deal with this issue, **WordNet**, a lexical ontology, distinguishes different meanings of one specific word by attributing it to different synsets. (Bollegala et al.(2015)Bollegala, Mohammed, Maehara, and Kawarabayashi) used a semantic lexicon and a corpus to learn word representations. Their method utilizes co-occurrence matrix and the lexical ontology instead of the neural network and outperforms both CBOW and Skip-gram by calculating the correlation coefficient between cosine similarities and benchmark datasets. Also there will be an intersection for senses of two words if they refer to the same meaning in the certain context. But because of different usages and writing habits, some words with the same meaning may occur in distinct contexts or appear infrequently in the text. For example, the cosine similarity for "let" and "allow" is smaller than human intuition judgement, and except "letting", "lets", the most similar word to "let" is "want". This indicates the sparse property of word embeddings learned by the model. Considering about two words, each word has a variety of meanings, but they share a few of them, in this case, they are widely spread in representation space by this model and far apart to each other. Moreover, one word can have different forms, for example, "regularize" can also be written in the form of "regularise", while in fact they are the same. However, when we use word representations to calculate the cosine similarities in these cases we get unreasonable scores which fail to accurately measure the similarity between words. To deal with these kinds of sparsity, and considering about the "depth" of NNLMs, we propose a deep neural network and this will be discussed later in the following section. Currently not so many works focus on addressing these problems.

3 PROPOSED APPROACH

We propose a novel deep neural network with pre-train process by using stacked autoencoders. Figure 1 illustrates the data flow of our architecture. First we process the input for pre-training to get latent parameters. After pre-training the raw input, we reduce the dimensionality to get better representations. The parameters of fine-tuning model are initialized by learned weights and biases. Both parts of the system leverage back propagation method for training. In the fine-tuning part we exhibit an innovative loss function which enables the model to increase the quality of similarities between synonym words as well as unrelated words.

3.1 Pre-train Model

An autoencoder (Rumelhart et al.(1985)Rumelhart, Hinton, and Williams; Hinton and Salakhutdinov(2006)) is a neural network for the purpose of reducing dimension, encoding original representations of the data in an unsupervised way. It is widely used for pre-training the data before classification tasks due to their strong dependency on the representation. By learning better representations,an auto-encoder reduces noise and extracts meaningful features, which improve performance of classifiers. (Kamyshanska and Memisevic(2015)) presents the potential energy of autoencoder using different activation functions and how different autoencoders achieve competitive performances in classification tasks. This inspires us to implement the network on word embeddings.

For an autoencoder, the network uses a mapping function f_θ to encode the input $\boldsymbol{x}$ into hidden representations $\boldsymbol{y}$, where $\theta = \{\boldsymbol{W}, \boldsymbol{b}\}$, $\boldsymbol{W}$ is the weight matrix and $\boldsymbol{b}$ is the bias term. Then the autoencoder decodes the hidden representations to reconstruct the input $\hat{\boldsymbol{x}}$ via $\boldsymbol{W}^T$ and another bias term for the output layer. h is the activation functions, *Sigmoid* and *ReLU* are frequently used.

$$\boldsymbol{y} = h(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{1}$$

$$\hat{\boldsymbol{x}} = h\left(\boldsymbol{W}^T h(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + \hat{\boldsymbol{b}}\right) \tag{2}$$

The goal of the autoencoder is to compare the reconstructed output to the original input and minimize the error so that output can be as close as possible to the input.

We leverage autoencoders to word embeddings learned by word2vec for the following three reasons:

1. We use the autoencoder to denoise and learn better representations.
2. Polysemous words spread widely in the representation space, far from their similar words, while

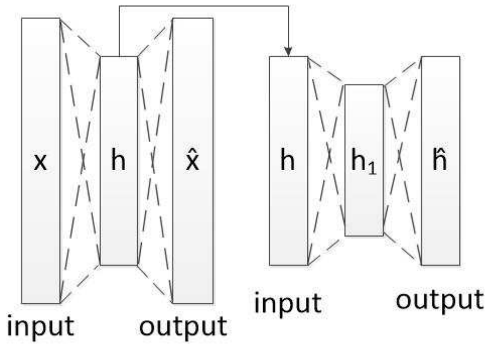

Figure 2: Stacked autoencoders with 2 hidden layers. Train the first autoencoder on the input to learn new features h. Then feed new features to the second autoencoder to obtain secondary features h_1.

autoencoder can compress the vectors and make them closer.

3. Representations of deeper hidden layers are relatively robust and steady to the input so we will reduce the dimensionality at least twice.

An advanced form of applying the reduction is stacked autoencoders. This neural network consists of multiple layers of sparse autoencoders in which the output of each layer is the input of the successive layer. The best way to train the stacked autoencoders is to use **greedy layer-wise training**. To do this, we first train one autoencoder for dimensionality reduction to obtain the weight matrix and bias. Then use the latent representation as the input for the next autoencoder. Follow this method in the subsequent layers, and finally a fine-tuning model is used to confirm and optimize the convergence of whole neural networks by the backpropagation method. In our system, we first use stacked autoencoders to obtain more robust representations than the corrupted input from word2vec as the pre-train step. Figure 2 shows the architecture of this step. Additionally, the activation function $f(x) = tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$ is used in the intermediate layers, and the output layer is also *tanh* because word representations should include positive as well as negative values.

In order to have the output as close as possible to the input, we use mean squared loss function

$$\boldsymbol{L} = \frac{1}{N}\sum_{i=1}^{N} ||\boldsymbol{x_i} - \boldsymbol{\hat{x}_i}||^2 \tag{3}$$

where x_i denotes the representation of a word in the corpus, and N is the total number of words. Using this function, the output will be as close as possible to the input.

3.2 Fine-tuning Model

The second part of our system is a novel fine-tuning model, constituted by a deep neural network. Unlike the fine-tuning model for autoencoders talked about in the previous section, we remove the symmetric structure of the network because we raise a new kind of loss function for learning deeper level representations. But the parameters are still initialized by learned weights and biases from stacked autoencoders for the purpose of fine-tuning. Figure 3 shows the architecture of the model.

We use the synonym dataset from WordNet to train the model. The dataset consists of words and their corresponding synsets IDs. If two words have the same synset ID, they denote the same concept. We iterate the whole dataset, each time we choose a word and find all the synonym words to it. At the same time, we randomly select 5 negative samples. Word *negative* means they are non-synonym instances to the present word. We utilize a innovative loss function for processing with word representations following the inspiration from (Huang et al.(2015)Huang, Heck, and Ji; Shen et al.(2014)Shen, He, Gao, Deng, and Mesnil)

$$\boldsymbol{L} = -\log \prod_{(w^+,w)} P\left(w^+|w\right) \tag{4}$$

$$P(w_j|w_i) = \frac{exp\left(sim(w_j, w_i)\right)}{\sum\limits_{w' \in E_i} exp\left(sim(w', w_i)\right)} \tag{5}$$

where w^+ denotes one related synonym word according to the current input instance in the dataset, and E_i is the set of instances which are not related or related to the input in any sense. We use one backpropagation method to minimize the loss to get optimal results, and in order to avoid overfitting, 10-fold cross-validation is implemented for determining model parameters after randomly sorting the vocabulary list. We set the first hidden layer to 150 neural units, and 100 for the next. On the top of the network, we output representations of positive and negative instances, and calculate the cosine similarity for word pairs so that we can measure the loss and update the parameters. By calculating the posterior probability and the loss, the model will increase the cosine similarities between synonym word pairs, and decrease those for non-synonym word pairs simultaneously. Also in this supervised way, we can save artificial efforts and obtain competitive or even state-of-the-art results comparing to word embeddings without learning by deep neural network.

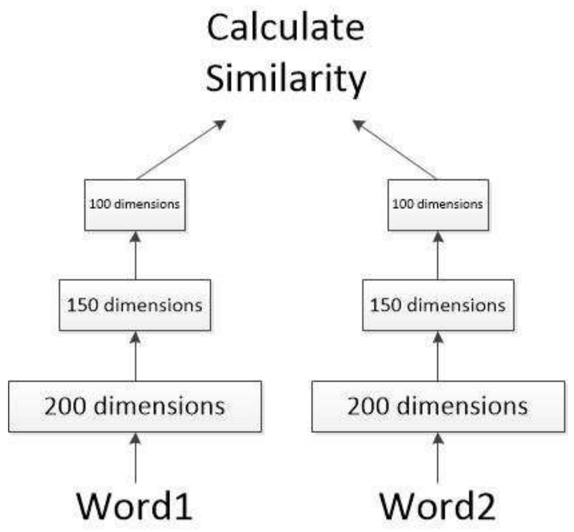

Figure 3: The structure of the fine-tuning model. Each time the input is the current word we come across in the dataset and we find a synonym of the word, we calculate the cosine similarity between them. We also choose a certain number of negative words which are not related to the current word.

4 EXPERIMENTS

We evaluate the learned representations of stacked autoencoders and the fine-tuning model, comparing to a variety of dimensions from word2vec without handling to show the performance of our system. For the experiment we use English Wikipedia dump collected in June of 2016. As processing, the corpus was lowercased and all punctuations are eliminated, phrases are separated so that words in them are treated independently. We set word2vec to apply CBOW instead of Skip-gram because there's little difference, negative sampling instead of hierarchy softmax. For parameters in the model, the size of dimension is 200 and the minimum count is 50. As a result we get 473926 words in total. Initially we reduced 200-dimension vectors learned by word2vec to 150 dimension. We use Adam algorithm which proposed by (Kingma and Ba(2014)) to minimize the loss function because of its computational efficiency. For the first autoencoder, the input is the original word representations from word2vec. After training to reach convergence, the first autoencoder produces dimensionality reduced representations and is used for the next autoencoder. By stacking the neural networks, we obtain embeddings of different dimensions. In this way we fed the ensemble of real value vectors which are of 150 dimension all the way to the secondary autoencoder to get 100 dimension results. We preserved weights and bias terms of each level for the initialization of the fine tuning model. We did not implement Mini-Batch Gradient Descent considering about the unique property of each word. The whole training process usually takes more than 10000 iterations and 12 hours on a single CPU. Meanwhile we dealt with 300-dimension embeddings for comparison since we found that the obtained results hold interesting trends and we want to see if the same case will happen for the same size of dimensionality, and to find out whether the depth influences the performance. Accordingly, vectors dimensionality of 200, 150 and 100 from word2vec are for comparison to our system's output latent representations. As for fine-tuning part, after filtering the instances in the dataset that do not occur in the corpus or are phrases, the total words for training the fine-tuning model is 55823 with 77538 different senses. Because we don't have test datasets for the work and in case of overfitting, we use 10-fold cross-validation to evaluate the average results about proportions between the number of synonym word pairs of which cosine similarities have been improved and the total number of pairs in the set.

Table 1: The empirical results for stacked autoencoders and the input is vectors of 200 dimension.

Dimensionality	150-ae	100-ae
200	73.54%	83.77%
150	49.00%	75.07%
100	27.50%	58.17%

Table 2: Results of the stacked autoencoders with 300 dimension vectors as input. In this experiment we compressed the vectors more times comparing to the previous one. The comparison between Table 1 shows that the deeper neural networks perform better.

Dimensionality	150-ae	100-ae
200	85.31%	88.11%
150	75.06%	83.87%
100	55.76%	73.46%

4.1 Experiment 1

We first evaluate the results of the pre-train model. The idea to use autoencoders is that we can get better vector representations which own less noise, more robust features and it can make related words closer. The input is vectors of 200 dimension from word2vec, and reduced the dimensionality to 150 by the first level autoencoder. Then we use learned representations as input for the second level to get dimensionality of 100. The evaluation is to calculate the average performances for test sets. We want to see how many synonym word pairs in the set whose cosine similarities have been improved. In this way we will calculate the proportion of improved word pairs and present the percentage in the following tables. To ensure the convergence, the iterating stops when we obtain the loss value as small as possible and it remains invariant for a period of time.

Table 1 shows some interesting findings. The row in Table 1 represents word embeddings of 200, 150 and 100 from word2vec respectively. The column stands for 150 and 100 dimension hidden representations after we used stacked autoencoders. We set the baseline as original cosine similarities, and calculate after we reduced the dimension, how many synonym word pairs' similarities have been improved. For 150 dimension vectors obtained from the autoencoder, 73.54% of synonym word pairs have been improved considering their cosine similarities, comparing to the raw input of 200 dimension. This demonstrates that after encoding, synonyms become closer in an unsupervised way. But for 150 dimension original representations, only 49% of word pairs have been improved which can be regarded as competitive results, and 27.50% for 100 dimension. The decreasing trend by virtue of the mechanism in word2vec, the smaller the dimensionality is, the higher cosine similarity results we get, however the representations still hold the deficiencies of the model. Then the secondary encoded vectors of 100 dimension give better results, 83.97% of word pairs have been improved, outperforming the previous one. Even we reduce the dimension twice, the encoded representations still hold the decreasing trend. As for 200 and 150 dimensions, hidden representations significantly promote the quality of similarity between synonym words without specific corpus in which the characteristics of synonyms in specific contexts are well preserved. Comparing to the same size, we still get competitive results, nonetheless, the 100 dimension latent representations perform better than 150 in the same case. To consolidate our idea and to exploit more of the strength of stacked autoencoders, we reduced 300 dimension to 150 dimension as well as 100 dimension by a 50 dimension drop for each reduction. This time we encoded the vectors 3 times and 4 times to get latent representations. In Table 2, the 150 and 100 dimension in column come from one different stacked autoencoders model and possess more reduction times comparing to hidden representations in the Table 1 column. After raising the reduction times, we can see the results become even better, and the reduction amount between adjacent layers is decreasing. The cosine similarities of the 150 and 100 dimension representations are greatly improved, and as for the same size of word2vec vectors, the results are beyond competitive. We can conclude that "the deeper, the better", deeper layer representations are more compact. In this way, we denoise so that related words get closer and the similarities of synonym words become higher, which seems more reasonable to human perception.

This pre-training method is an automatic learning process for the input, and the model develops better representations for synonym word pairs. We can argue that the stacked autoencoders is powerful to address the problem of ambiguity between synonym words and scattering vectors by reducing the dimension. This novel approach on dealing with word2vec vectors demonstrates that these vectors can still be learned by deep neural networks. It can be utilized to extract meaningful features in embeddings so that we can explore the properties held in them.

Table 3: Overall performances of fine-tuning model. The representations from two hidden layers are evaluated for computing the proportion between improved instances and the total number.

Dimensionality	150-bp	100-bp
200	94.37%	80.76%
150	92.36%	79.65%
100	88.91%	78.60%

4.2 Experiment 2

In order to fine-tune the parameters, we propose a deep synonym relatedness model to learn latent representations after pre-training. Following the architecture shown in Figure 3, we use 10-fold cross-validation and evaluate the average performances for learned representations from two hidden layer. The whole training process takes rougly 7 hours on a single machine. In Table 3, **150-bp** and **100-bp** represent learned hidden representations after using back-propagation method from fine-tuning model. The row in the table means the same as previous tables. In Table 3 we can see that, in 94.37% of the instances, the similarities of words have been improved, in comparison to dimensionalities of 200 and 150, and the results are also significantly better than deep stacked autoencoders. As for 100 dimension in fine-tuning model, it is only competitive to the same dimension from pre-train model. From the table we can see that after fine-tuning, the representations still hold the decreasing trend when comparing with the original vectors, the same behaviour with pre-training results. But the model improves the performance of latent representations on the same dimension and the smaller dimension. Considering about the 27.50% and 49.00% improvement in Table 1, we acquire 88.91% and 92.36% which are notable results. We can argue that even if we don't use autoencoders to encode and

Table 4: Examples of synonym words and the comparison between different dimensions of vectors.

Synonyms	200	150	100	150-bp	100-bp
regularize & regularise	0.661886	0.710118	0.702691	0.801452	0.999723
changeless & unalterable	0.500718	0.578448	0.640918	0.694430	0.918739
respectful & reverential	0.654054	0.683833	0.719712	0.742635	0.884301
repair & fix	0.452142	0.467059	0.474660	0.628190	0.997788
fortunately & fortuitously	0.395465	0.432642	0.428508	0.442118	0.963096
promoter & booster	0.043374	0.028916	-0.038873	0.240978	0.216803
curb & kerb	0.317527	0.346058	0.325593	0.618310	0.684300
dusk & nightfall	0.694591	0.723055	0.746611	0.804976	0.936354
awake & arouse	0.127879	0.134865	0.146026	0.397994	0.772240
run & consort	-0.189202	-0.221867	-0.267452	0.183840	0.269530

Table 5: Comparison between representations of two layer stacked autoencoders and fine-tuning model.

Dimensionality	150-bp	100-bp
150-ae	91.94%	79.45%
100-ae	86.95%	77.55%

compress the vectors, by using the fine-tuning model and the novel loss function, we can still promote the similarities between synonym words and decrease the similarities between non-synonym pairs. We use the similarities from word2vec as baselines and manually inspect similarities after fine-tuning(Table 4). *Run and consort* can mean the same when referring to hanging out with someone. The cosine similarity of two words are negative which makes no sense. After fine-tuning, the similarity is improved to more than 0.183840. *Arouse and awake* are interchangeable but they get poor 0.127879 cosine similarity results, however for 100 dimension representations it can get 0.772240 instead. *Regularize and regularise* are the same, the similarity is around 0.710118 because of the scattering and sparsity problem in word2vec, the fine-tuning result for 100 dimension is notably 0.999732, nearly 1. *Repair and fix* have only poor 0.474660 similarity, and the model improve it to 0.997788. These examples demonstrate that 100 dimension vectors after fine-tuning gain the largest improvement, better than 150 dimension. It seems reasonable because of the gradient vanishing. But we still found that for just a few instances, 100 dimension results are even lower than original vectors. This may because we should add some regularizers or smooth parameters.

The supervised learning method verifies the idea that deep neural networks can be applied on word2vec vectors for specific tasks by artificially adjusting. Because the model is for fine-tuning, so we will compare the results between autoencoders' vectors and fine-tuned vectors. We can see from Table 5 that by using the architecture without reducing dimensionality, the similarities between synonym words get further improvement. **150-bp** and **100-bp** stand for the hidden representations from the fine-tuning model, while **150-ae** and **100-ae** still represent the reduced vectors from hidden layers of stacked autoencoders. As for 150 dimension latent representations in stacked autoencoders, around 91.94% of synonym word pairs' similarities have been increased, indicating that the deep neural network is more effective because of supervised learning. But it is still uncertain why the performance of 100 dimension after back-propagation method is worse than 150 dimension. Maybe we can assume that the space of 100 dimension representations to be improved is not as much as 150 dimension, this also demonstrates that deeper unsupervised models have already denoise more and decrease the sparsity. Even without the initialization from pre-training process, if we incorporate more training data, the model can reform word embeddings with better quality. It is possible to utilize the neural network with appropriate data and more types of knowledge to learn representations that are suitable for certain tasks.

5 CONCLUSIONS AND FUTURE WORK

We have introduced a combined system with two kinds of neural networks to handle the word embeddings of word2vec. From the results we can say, stacked autoencoders are strong enough to encode and compact the vectors in the space. Due to different conditions for the used corpora and its restriction, word2vec will not produce word embeddings with good quality. Some synonyms are rarely seen in the text, so the similarities could be even negative values in word2vec. And some synonyms have not only a few same meanings but also other different senses. In this case, words are spread far apart in the space with extremely low cosine similarities. By using autoencoders in an unsupervised approach, we get more compact representations with less noise which automatically disambiguate the similarities between synonyms. The fine-tuning model also indicates the potential of representations to be learned by deep neural networks with carefully designed loss functions and knowledge graphs or lexical datasets. In this paper, we set the loss function in a softmax form with a dataset of WordNet synsets to calculate the posterior probability, this method improves cosine similarities between synonym words and decreases similarities of non-synonym ones. Unlike the idea in stacked autoencoders, by encoding word embeddings in a supervised way, the model only extracts useful semantic features for synonyms, and makes them closer.

Both of the models achieved significantly better performance than word2vec on measuring synonym relatedness, shed light on exploiting word embeddings in a supervised or unsupervised way. But these two models come up from different ideas and there is still something confuses us, the deeper stack autoencoders we use, the loss when converging will be bigger for each autoencoder in the network, we will keep studying on this phenomenon in the future to probe the features of autoencoders and word representations. For unsupervised learning, we plan to comprehensively evaluate the energy of autoencoders. We will explore the changes in linguistic regularities of latent representations, and discover the patterns of semantic and syntactic properties in embeddings. Autoencoder may be a good toolkit to clarify the meaning of opaque vectors. Our future work will also focus on disambiguating entities types by setting a classifier on the top layer of the network.

ACKNOWLEDGMENT

This work was partially supported by NEDO (New Energy and Industrial Technology Development Organization).

REFERENCES

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb): 1137–1155, 2003.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

Danushka Bollegala, Alsuhaibani Mohammed, Takanori Maehara, and Ken-ichi Kawarabayashi. Joint word representation learning using a corpus and a semantic lexicon. *arXiv preprint arXiv:1511.06438*, 2015.

John A Bullinaria and Joseph P Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior research methods*, 39 (3):510–526, 2007.

Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, 2014.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine learning*, pages 160–167. ACM, 2008.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, 2012.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Hongzhao Huang, Larry Heck, and Heng Ji. Leveraging deep neural networks and knowledge graphs for entity disambiguation. *arXiv preprint arXiv:1504.07678*, 2015.

Hanna Kamyshanska and Roland Memisevic. The potential energy of an autoencoder. *IEEE transactions on pattern analysis and machine intelligence*, 37(6):1261–1273, 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Raúl Ernesto Menéndez-Mora and Ryutaro Ichise. Toward simulating the human way of comparing concepts. *IEICE TRANSACTIONS on Information and Systems*, 94(7):1419–1429, 2011.

Tomas Mikolov and J Dean. Distributed representations of words and phrases and their compositionality. 2013b.

Tomas Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531. IEEE, 2011.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088, 2009.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 373–374. ACM, 2014.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 455–465, 2013.