Estimation of Word Representations Using Recurrent Neural Networks and Its Application in Generating Business Fingerprints

Kuan Fang KUANFANG@STANFORD.EDU

Abstract: Word vectors have been used as a useful feature in many NLP related tasks. In this project, we proposed a modified recurrent neural network algorithm to learn word representations from corpus in a state-transition manner. Our state-transition RNN could reach the results of state-of-the-art word embedding algorithms in terms of comparing Euclidean distance. With smaller hidden size, the state-transition RNN needs shorter time for training. As a testbench for our model, we trained our model on Yelp review corpus and implemented three different applications (business similarity, business search and business recommendation). Experimental results revealed that our state-transition RNN and our Business fingerprints generation algorithms achieved expected results.

1. Introduction

Word vectors have been an important and basic feature for a variety of natural language processing tasks. A reasonable set of word vectors should represent the similarity of words by computing the distance (e.g. cosine distance and Frobenius norm) of two different word vectors. Estimation algorithms of word representations such as skip-gram and GloVe have been implemented in recent years. The skip-gram is an efficient linear training model considering a small part of context words. And Glove is a log-bilinear model with a weighted least-squares objective considering global context. These existing models has shown good performance on word analogy tasks, but they have deficiencies such as considering only local context words or considering only the word-word co-occurrence in the text.

In this project, we aim to propose unsupervised models that can learn deeper relationships by using global context words in sentences and paragraphs. We will begin by implementing a modified recurrent neural network (RNN) language model as our baseline algorithm. Recurrent neural network has shown excellent performance in the language modeling task by taking the previous context words as input and predicting the target word. Since the word representation of the target word should be learned by context words before and after it, we attempt to implement a state-transition RNN to extract global contexts and even the semantics from the sentence.

With learned word vectors, we represent the each business fingerprint as a cluster of word vectors extracted from the reviews. And three implemented applications proved that these business fingerprints are useful in recommendation systems.

2. Prediction and Metrics

2.1. Learning Word Vectors

There is no direct way to evaluate the performance of trained word vectors. Before we implemented more sophisticated testbenches to test our word vectors on dataset like word-analogy problems we take the Perplexity (PPL) of the language modeling as a metric of our word vectors trained on State-Transition RNN.

2.2. Generating Business Fingerprints

Although a high performance of applications (e.g. business category prediction) is not a direct metric of fingerprint quality, a good fingerprint will be necessarily good to be used for the classification task. Based on this argument, we can formulate an optimization goal based on combinations of specific tasks. For example, one evaluation metric is to loop through all pairs of businesses and compute their distances, if the two businesses have many categories in common, we hope their distance is small; on the other hand if they do not have category in common, we want their distance to be large.

$$\min_{\theta} \sum_{(i,j)} (categorySim(i,j) - 0.5) \cdot distance(i,j) \quad (1)$$

3. Dataset

We trained our data on Penn Tree Bank data set and Yelp reviews corpus. The Penn Tree Bank data set is a common dataset in natural language processing. And

the Yelp reviews corpus is cleaned by ourselves from the Yelp Data Challenge data set.

4. Model

4.1. Word Representation Using Skip-gram Model

Skip-gram Model is a distributed word representation learning algorithm proposed by Mikolov et al in 2013. Skip-gram learn the word representation vectors based on a two-level neural network, taking the dot product of the word vector as input to predict the occurrence of words in a context window (usually 4 - 6 context words). In the training, Skip-gram maximize the log probability of context words.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{2}$$

The basic Skip-gram formulation defines $p(w_{t+j}|w_t)$ using the softmax function:

$$p(w_{t+j}|w_t) = \frac{\exp(v_{wO}^{\top} v_{wI})}{\sum_{w=1}^{W}\exp(v_w^{\top} v_{wI})} \tag{3}$$

We modified Google's word2vec toolkit implemented by Mikolov et al (Mikolov et al., 2013a) and trained Skip-gram on general text corpus and Yelp reviews corpus respectively. In terms of plugging the resulting word vectors into our recommendation model, we used cosine distance and Frobenius norm to compute the word distance respectively.

4.2. Word Representation Using State-Transition Recurrent Neural Network

We also proposed a recurrent neural network (RNN) based model to learn word representation. One of the disavantages of the word-word co-occurrence model is that they can not learn the word vector based on the semantics of the sentence. For example, "good" and "bad" has a close cosine distance after trained by Skip-gram. This is because these two words occurred in many similar contexts in the training data. Thus we turned to a modified RNN algorithm. Like existing RNN, we learn word vectors in an unsupervised manner taking the PPL of the language modeling as the objective. In stead of taking the one-hot word code as input, we model the semantics of the sentence using a dense state vector. Each entry of the state vector represents a feature of the semantics evolved with words. While each word maintains a transition matrix trans- from the input state into the output state, evolving

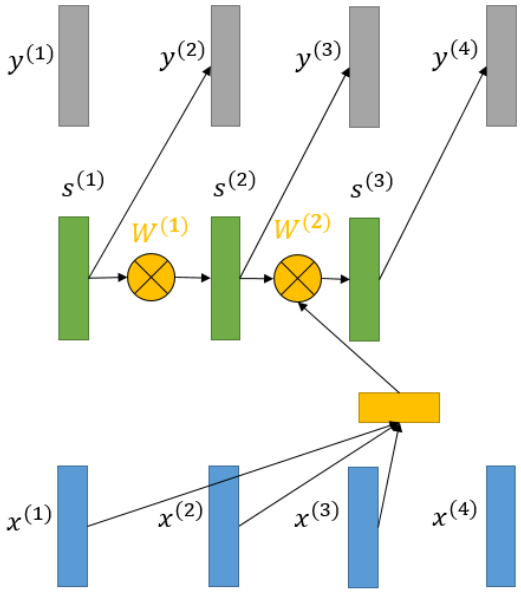


Figure 1. State Transition Reccurent Neural Network

the semantics of the sentence as the following words are read in one by one.

$$s^{(0)} = (1, 0, ..., 0)^{\top} \tag{4}$$
$$s^{(t)} = f(W^{(t)} \cdot s^{(t-1)} + b^{(t)}) \tag{5}$$

To train this RNN unsupervisely, a fixed matrix feed forwards the output state layer into the prediction layer, where the probability of the next word is represented by a one-hot vector.

$$y^{(t)} = f(V \cdot s^{(t)}) \tag{6}$$

The backpropagation steps of the model is as follows:

$$U^{(t)} := U^{(t)} + \alpha e_o^{(t)} \cdot s^{(t)\top} \tag{7}$$
$$W^{(t)} := W^{(t)} + \alpha e_h^{(t)} \cdot s^{(t-1)\top} \tag{8}$$

where we plug following error expressions in:

$$e_o^{(t)} = d^{(t)} - y^{(t)} \tag{9}$$
$$e_h^{(t)} = d_h(e_o^{(t)\top} V, t) \tag{10}$$
$$d_{hj}(x, t) = x s_j^{(t)}(1 - s_j^{(t)}) \tag{11}$$

We expect this algorithm could extract deeper relationship between words, thus generating more reasonable word vectors as long with lower PPL for the language modeling objective. We implemented our code based on Microsoft Research's RNN language modeling toolkit and modified almost all functions of it according to our needs.

5. Results

5.1. Unsupervised training of Word Vectors

We first test our model on Penn Tree Bank dataset, which is a hall mark in the natrual language processing field. We compare our results on State-Transition RNN with the state-of-art RNN language modeling implemented by Microsoft Research. Then we also trained our State-Transition RNN on the Yelp reviews dataset (cleaned by ourselves). Since Microsoft's RNN has much more parameters than we used, it could take much more time to converge.

For ST-RNN, we compared the PPL performance of different state sizes from 3 to 99 on Penn Tree Bank dataset. To speed up the training and prediction, we followed Mikolov's approach to categorize output words according to their frequencies in the training corpus. The results are shown in Figure. 2. We found that for Penn Tree Bank dataset, the optimal state size is around 30. For larger dataset like Yelp Data Challenge, the optimal state size is around 35.

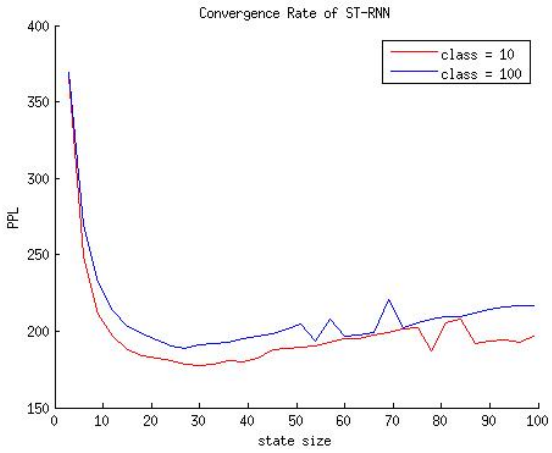


Figure 2. PPL Performance of Different State Sizes.

The size column below indicates the hidden size for RNN and state size for ST-RNN. Training State-Transition RNN took 15 hours on a research server in Stanford CS Department to converge. And the last row is ST-RNN trained on Yelp reviews.

Table 1. Training Skip-gram on general corpus

Model	Size	words/sec	Iters	PPL
RNN	100	20000	9	167.14
RNN	200	8000	9	153.62
ST-RNN	15	68000	8	188.31
ST-RNN	20	62000	8	183.78
ST-RNN	30	44000	8	177.34
ST-RNN (Y)	20	62000	7	117.80

Our State-Transition RNN achieves reasonable performance. We will add BPTT later and test our model on larger dataset.

5.2. Similar Word Vectors

We give some word examples here to compare the performance of two word embedding algorithms. Both examples are trained on the reviews data of Yelp Data Challenge which are cleaned by ourselves. The first row is the input word, and the rest are the closest words determined by cosine distance.

Table 2. Training State-Transition RNN on reviews

duck	burger	noodle	good
chicken	pizza	pasta	great
burrito	sandwich	tofu	wonderful
tofu	burrito	vegetable	decent
salmon	smoothie	seafood	bad
octopus	pho	ramen	fantastic

Table 3. Training Skip-gram on reviews

duck	burger	noodle	good
pork	hamburger	noodles	decent
squab	cheeseburger	tofu	great
chicken	burgers	noodlesi	solid
confit	burgeri	wonton	tasty
beef	buger	vermecelli	alright

Table 4. Training Skip-gram on general corpus

duck	burger	noodle	good
fethry	mcdonald	udon	bad
daffy	kfc	toppings	natured
amuck	mcdonalds	steamed	luck
eider	schalk	ramen	virtuous
coot	restaurant	kuroda	honest

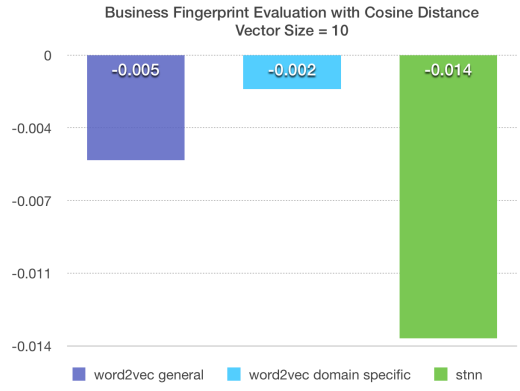
As we can see from these examples, training word vecters on domain-specific data (Yelp reviews in our case) has much better performance. Our word embedding algorithm is doing a good job, but it still cannot achieve the ideal result we want. It is still assigning antoynms closest vectors. We aspire to improve this issue in our following work.

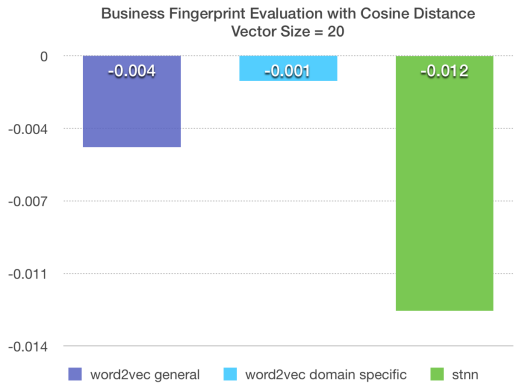
5.3. Business Similarity

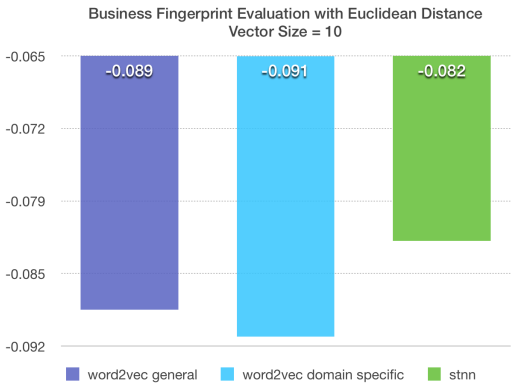
We tested our current model on a small data set of 10 categories each of which has 100 businesses, thus 1000 businesses in total. Note that for our evaluation method, smaller the costs are better our signatures are. We can see that use more feature words achieves lower cost, i.e. better signatures in terms of category consistency.

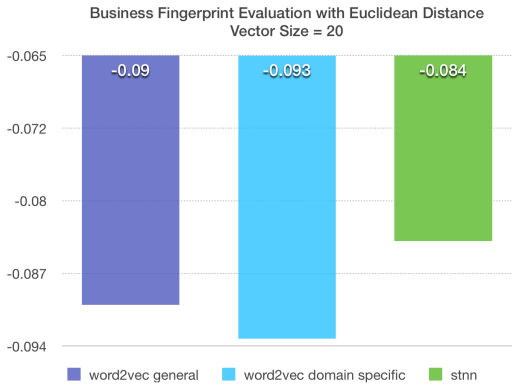
The ground truth is the labeling from the Yelp data. We count the number of co-existing words in the two restaurants as a measure of ground truth similarity, then computed the similarity based of business signature. We used both Euclidean distance and Cosine distance to measure it. We tried different length of feature words to concatenate together to form a signature. As shown by the below plot, the larger the number of words is chosen the more accurate the performance.

Note that the smaller the Euclidean distance means the larger Cosine distance, in both cases business fingerprint evaluation vector size = 20 using word2vec domain specific training method gives the best performance.









5.4. Business Search

Another application of building up business fingerprint is for semantic search. Traditional search is based on the appearance of a word, then building a inverted index. That is bad because if a word doesn't appear in a document but some similar word appears, it won't be captured. The word vector solved this problem in that it can find all the words that are close to the search word, making the search result more meaningful. It's no longer a word by word search, but semantic search. Here's a list of results returned by the search engine.

For example, in the query "morning light food", apparently the query wants to find a place to have breakfast. Although the query did not explicitly say breakfast, but "morning" and "lightfood" expressed this notion. By comparing the business signature the search engine found the Beach House Lounge which is exactly a place serving breakfast. And after digging into the reason why this restaurant get returned, it shows the word 'breakfast' has the closest distance 1.2018 to the query, so it gets returned.

Another example is the query "beer nightlife clubbing". The query intends to find a place to have some beer and enjoy the nightlife by clubbing. The business signature captures this underlying semantic and

returned the Tip Top Tavern Bar, which is exactly a night bar.

Query	Result
"morning light food"	Beach House Lounge
"beer nightlife clubbing"	Tip Top Tavern Bar
"fastfood"	Cheeseburger in Paradise
"healthy vegetarian"	Monty's Blue Plate Diner

5.5. Business Recommendation

The last application we built on top of the business signature is business recommendation, which is an item similarity based recommendation. The user specifies a restaurant he likes, then the system will return a restaurant that's similar to the specified restaurant. Basically the recommendation system is calculating the Euclidean distance between he business signatures and return the one with the smallest distance.

For example, the recommendation for "Uno Pizzeria and Grill" is "Benvenuto's Italian Grill", because they are both selling grills. Another example is the recommendation for "Flaming Wok" is "Gourmet House of Hong Kong", because they are both Chinese restaurants.

This business signature based recommendation system captures the intrinsic characteristics of a restaurant and provides a fair way to compare the similarity based on the vector distance. Overall the system provides a quantitative way to compare the businesses.

6. Discussion

The PPL of ST-RNNs approaches the PPL of traditional RNN, but ST-RNNs were trained at much faster speed. In the three business recommendation applications, ST-RNNs have similar performance with the Skip-gram when comparing Euclidean distance, while its performance is worse when comparing cosine distance. An explanation is that the Skip-gram algorithm contains an inner product in its model, so the algorithm is based on consine distance.

7. Conclusion

We designed state-transition RNN for word embedding. Comparing with traditional RNNs and the word2vec toolkit, our algorithm could approach the state-of-the-art results of word embedding and language modelling. With smaller state size, our model could be trained much faster than traditional RNNs. With the learned word vectors, we implemented three applications of business fingerprints generation as a testbed. With our word vectors, we could classify and search businesses in these applications as we expected.

8. Future Work

The back propagation through time (BPTT) algorithm did not work well for our ST-RNN. We assumed that this kind of neural networks with self-generated parameter matrices need other optimization algorithm to improve the performance. For example, we may need to use second order methods (e.g. Hessian-free optimization). In some related works, factorizing the parameter matrices to a product of several matrices might also help. We expected to explore these algorithms in our next step.

Acknowledgement

Some parts of this project also serves in my CS221 project: Learning Business Fingerprints from Texts. In my CS221 project, we have another two team members (Han Song and Charles Qi) working together. The word representation part was all done by myself. I also worked on the design of the three applications of generating business fingerprints.

References

Mikolov, Tomas, Karafiát, Martin, Burget, Lukas, Cernockỳ, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *INTERSPEECH*, pp. 1045–1048, 2010.

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

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pp. 3111–3119, 2013b.

Sutskever, Ilya, Martens, James, and Hinton, Geoffrey E. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024, 2011.

Taylor, Graham W, Hinton, Geoffrey E, and Roweis, Sam T. Two distributed-state models for generating high-dimensional time series. *The Journal of Machine Learning Research*, 12:1025–1068, 2011.