A Comparative Study on Regularization Strategies for Embedding-based Neural Networks

Hao Peng,[*1] **Lili Mou,**[*1] **Ge Li,**[†1] **Yunchuan Chen,**[2] **Yangyang Lu,**[1] **Zhi Jin**[1]
[1]Software Institute, Peking University, 100871, P. R. China
{penghao.pku, doublepower.mou}@gmail.com,{lige, luyy11, zhijin}@sei.pku.edu.cn
[2]University of Chinese Academy of Sciences, chenyunchuan11@mails.ucas.ac.cn

Abstract

This paper aims to compare different regularization strategies to address a common phenomenon, severe overfitting, in embedding-based neural networks for NLP. We chose two widely studied neural models and tasks as our testbed. We tried several frequently applied or newly proposed regularization strategies, including penalizing weights (embeddings excluded), penalizing embeddings, re-embedding words, and dropout. We also emphasized on incremental hyperparameter tuning, and combining different regularizations. The results provide a picture on tuning hyperparameters for neural NLP models.

1 Introduction

Neural networks have exhibited considerable potential in various fields (Krizhevsky et al., 2012; Graves et al., 2013). In early years on neural NLP research, neural networks were used in language modeling (Bengio et al., 2003; Morin and Bengio, 2005; Mnih and Hinton, 2009); recently, they have been applied to various supervised tasks, such as named entity recognition (Collobert and Weston, 2008), sentiment analysis (Socher et al., 2011; Mou et al., 2015), relation classification (Zeng et al., 2014; Xu et al., 2015), etc. In the field of NLP, neural networks are typically combined with word embeddings, which are usually first pre-trained by unsupervised algorithms like Mikolov et al. (2013); then they are fed forward to standard neural models, fine-tuned during supervised learning. However, embedding-based neural networks usually suffer from severe overfitting because of the high dimensionality of parameters.

A curious question is whether we can regularize embedding-based NLP neural models to improve generalization. Although existing and newly proposed regularization methods might alleviate the problem, their inherent performance in neural NLP models is not clear: the use of embeddings is sparse; the behaviors may be different from those in other scenarios like image recognition. Further, selecting hyperparameters to pursue the best performance by validation is extremely time-consuming, as suggested in Collobert et al. (2011). Therefore, new studies are needed to provide a more complete picture regarding regularization for neural natural language processing. Specifically, we focus on the following research questions in this paper.

RQ 1: How do different regularization strategies typically behave in embedding-based neural networks?

RQ 2: Can regularization coefficients be tuned incrementally during training so as to ease the burden of hyperparameter tuning?

RQ 3: What is the effect of combining different regularization strategies?

In this paper, we systematically and quantitatively compared four different regularization strategies, namely penalizing weights, penalizing embeddings, newly proposed word re-embedding (Labutov and Lipson, 2013), and dropout (Srivastava et al., 2014). We analyzed these regularization methods by two widely studied models and tasks. We also emphasized on incremental hyperparameter tuning and the combination of different regularization methods.

Our experiments provide some interesting results: (1) Regularizations do help generalization, but their effect depends largely on the datasets' size. (2) Penalizing ℓ_2-norm of embeddings helps optimization as well, improving training accuracy unexpectedly. (3) Incremental hyperparameter tuning achieves similar performance, indicat-

[*]Equal contribution. [†]Corresponding author.

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 2106–2111,
Lisbon, Portugal, 17-21 September 2015.

ing that regularizations mainly serve as a "local" effect. (4) Dropout performs slightly worse than ℓ_2 penalty in our experiments; however, provided very small ℓ_2 penalty, dropping out hidden units and penalizing ℓ_2-norm are generally complementary. (5) The newly proposed re-embedding words method is not effective in our experiments.

2 Tasks, Models, and Setup

Experiment I: Relation extraction. The dataset in this experiment comes from SemEval-2010 Task 8.[1] The goal is to classify the relationship between two marked entities in each sentence. We refer interested readers to recent advances, e.g., Hashimoto et al. (2013), Zeng et al. (2014), and Xu et al. (2015). To make our task and model general, however, we do not consider entity tagging information; we do not distinguish the order of two entities either. In total, there are 10 labels, i.e., 9 different relations plus a default `other`.

Regarding the neural model, we applied Collobert's convolutional neural network (CNN) (Collobert and Weston, 2008) with minor modifications. The model comprises a fixed-window convolutional layer with size equal to 5, **0** padded at the end of each sentence; a max pooling layer; a $\tanh$ hidden layer; and a $\mathrm{softmax}$ output layer.

Experiment II: Sentiment analysis. This is another testbed for neural NLP, aiming to predict the sentiment of a sentence. The dataset is the Stanford sentiment treebank (Socher et al., 2011)[2]; target labels are `strongly/weakly positive/negative`, or `neutral`.

We used the recursive neural network (RNN), which is proposed in Socher et al. (2011), and further developed in Socher et al. (2012); Irsoy and Cardie (2014). RNNs make use of binarized constituency trees, and recursively encode children's information to their parent's; the root vector is finally used for sentiment classification.

Experimental Setup. To setup a fair comparison, we set all layers to be 50-dimensional in advance (rather than by validation). Such setting has been used in previous work like Zhao et al. (2015). Our embeddings are pretrained on the Wikipedia corpus using Collobert and Weston (2008). The learning rate is 0.1 and fixed in Experiment I; for RNN, however, we found learning rate decay helps to prevent parameter blowup (probably due to the recursive, and thus chaotic nature). Therefore, we applied power decay (Senior et al., 2013) with power equal to -1. For each strategy, we tried a large range of regularization coefficients, $10^{-9}, \cdots, 10^{-2}$, extensively from underfitting to no effect with granularity 10x. We ran the model 5 times with different initializations. We used mini-batch stochastic gradient descent; gradients are computed by standard backpropagation. For source code, please refer to our project website.[3]

It needs to be noticed that, the goal of this paper is not to outperform or reproduce state-of-the-art results. Instead, we would like to have a fair comparison. The testbed of our work is two widely studied models and tasks, which were not chosen on purpose. During the experiments, we tried to make the comparison as fair as possible. Therefore, we think that the results of this work can be generalized to similar scenarios.

3 Regularization Strategies

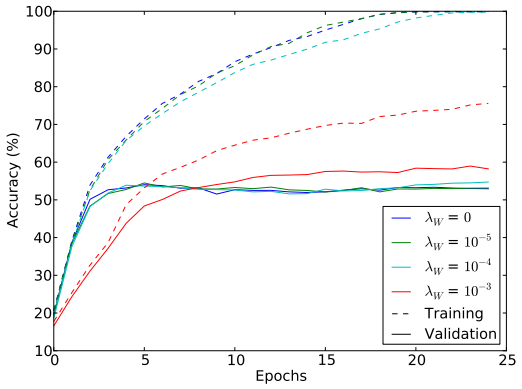
In this section, we describe four regularization strategies used in our experiment.

- Penalizing ℓ_2-norm of weights. Let E be the cross-entropy error for classification, and R be a regularization term. The overall cost function is $J = E + \lambda R$, where λ is the coefficient. In this case, $R = \|W\|^2$, and the coefficient is denoted as λ_W.
- Penalizing ℓ_2-norm of embeddings. Some studies do not distinguish embeddings or connectional weights for regularization (Tai et al., 2015). However, we would like to analyze their effect separately, for embeddings are sparse in use. Let Φ denote embeddings; then we have $R = \|\Phi\|^2$.
- Re-embedding words (Labutov and Lipson, 2013). Suppose Φ_0 denotes the original embeddings trained on a large corpus, and Φ denotes the embeddings fine-tuned during supervised training. We would like to penalize the norm of the difference between Φ_0 and Φ, i.e., $R = \|\Phi_0 - \Phi\|^2$. In the limit of penalty to infinity, the model is mathematically equivalent to "frozen embeddings," where word vectors are used as surface features.
- Dropout (Srivastava et al., 2014). In this strategy, each neural node is set to 0 with a predefined dropout probability p during training; when testing, all nodes are used, with activation multiplied by $1 - p$.

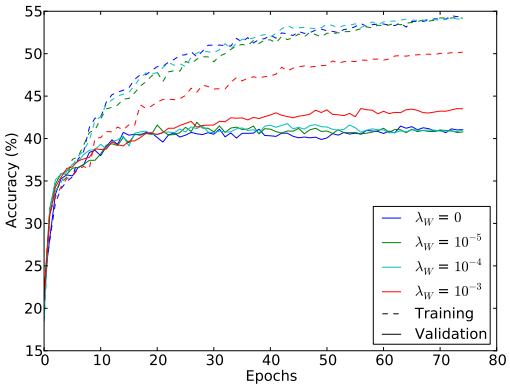
[1] http://www.aclweb.org/anthology/S10-1006

[2] http://nlp.stanford.edu/sentiment/

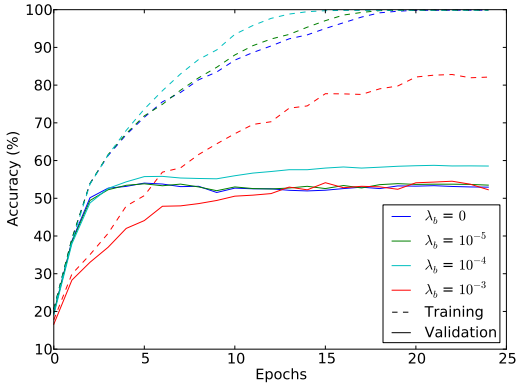
[3] https://sites.google.com/site/regembeddingnn/



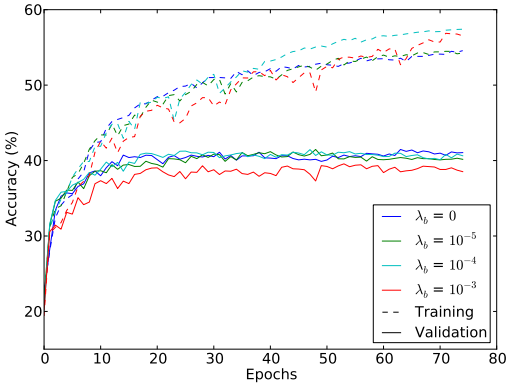
(a) Penalizing weights in Experiment I.



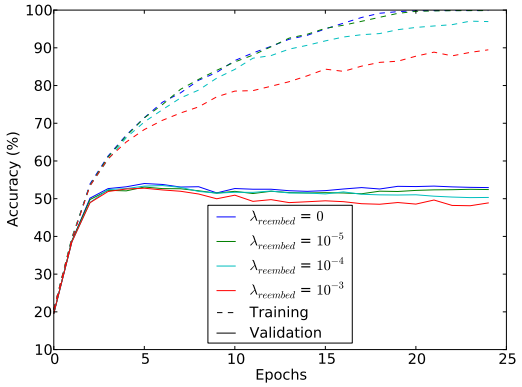
(b) Penalizing weights in Experiment II.



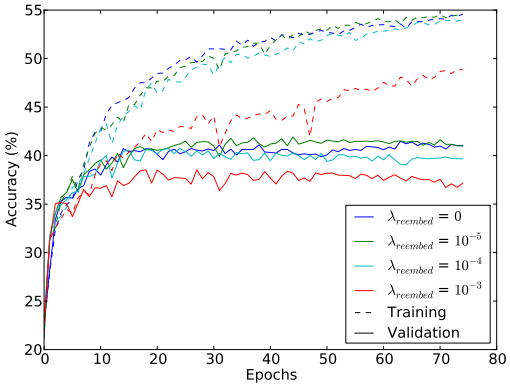
(c) Penalizing embeddings in Experiment I.



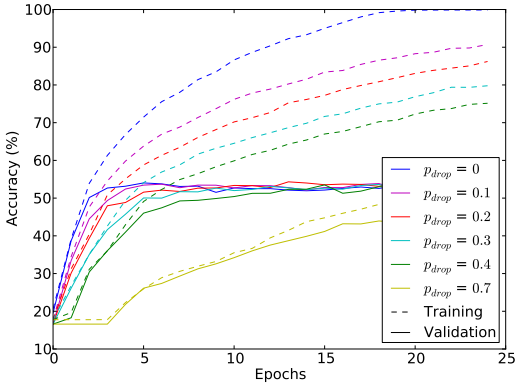
(d) Penalizing embeddings in Experiment II.



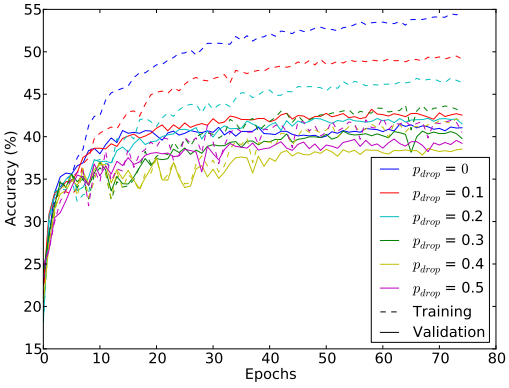
(e) Re-embedding words in Experiment I.



(f) Re-embedding words in Experiment II.



(g) Applying dropout in Experiment I. $p = 0.5, 0.6$ are omitted because they are similar to small values.



(h) Applying dropout in Experiment II.

Figure 1: Averaged learning curves. Left: Experiment I, relation extraction with CNN. Right: Experiment II, sentiment analysis with RNN. From top to bottom, we penalize weights, penalize embeddings, re-embed words, and drop out. Dashed lines refer to training accuracies; solid lines are validation accuracies.

4 Individual Regularization Behaviors

This section compares the behavior of each strategy. We first conducted both experiments without regularization, achieving accuracies of $54.02 \pm 0.84\%$, $41.47 \pm 2.85\%$, respectively. Then we plot in Figure 1 learning curves when each regularization strategy is applied individually. We report training and validation accuracies through out this paper. The main findings are as follows.

- Penalizing ℓ_2-norm of weights helps generalization; the effect depends largely on the size of training set. Experiment I contains 7,000 training samples and the improvement is 6.98%; Experiment II contains more than 150k samples, and the improvement is only 2.07%. Such results are consistent with other machine learning models.

- Penalizing ℓ_2-norm of embeddings unexpectedly helps optimization (improves training accuracy). One plausible explanation is that since embeddings are trained on a large corpus by unsupervised methods, they tend to settle down to large values and may not perfectly agree with the tasks of interest. ℓ_2 penalty pushes the embeddings towards small values and thus helps optimization. Regarding validation accuracy, Experiment I is improved by 6.89%, whereas Experiment II has no significant difference.

- Re-embedding words does not improve generalization. Particularly, in Experiment II, the ultimate accuracy is improved by 0.44, which is not large. Further, too much penalty hurts the models in both experiments. In the limit λ_{reembed} to infinity, re-embedding words is mathematically equivalent to using embeddings as surface features, that is, freezing embeddings. Such strategy is sometimes applied in the literature like Hu et al. (2014), but is not favorable as suggested by the experiment.

- Dropout helps generalization. Under the best settings, the eventual accuracy is improved by 3.12% and 1.76%, respectively. In our experiments, dropout alone is not as useful as ℓ_2 penalty. However, other studies report that dropout is very effective (Irsoy and Cardie, 2014). Our results are not consistent; different dimensionality may contribute to this disagreement, but more experiments are needed to confirm the hypothesis.

5 Incremental Hyperparameter Tuning

The above experiments show that regularization generally helps prevent overfitting. To pursue the best performance, we need to try out different hyperparameters through validation. Unfortunately, training deep neural networks is time-consuming, preventing full grid search from being a practical technique. Things will get easier if we can incrementally tune hyperparameters, that is, to train the model without regularization first, and then add penalty.

In this section, we study whether ℓ_2 penalty of weights and embeddings can be tuned incrementally. We exclude the dropout strategy because its does not make much sense to incrementally drop out hidden units. Besides, from this section, we only focus on Experiment I due to time and space limit.

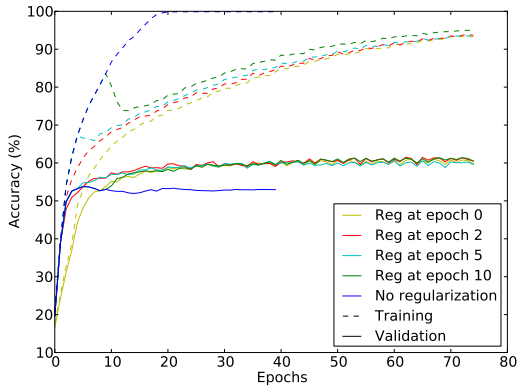
Before continuing, we may envision several possibilities on how regularization works.

- (On initial effects) As ℓ_2-norm prevents parameters from growing large, adding it at early stages may cause parameters settling down to local optima. If this is the case, delayed penalty would help parameters get over local optima, leading to better performance.
- (On eventual effects) ℓ_2 penalty lifts error surface of large weights. Adding such penalty may cause parameters settling down to (a) almost the same catchment basin, or (b) different basins. In case (a), when the penalty is added does not matter much. In case (b), however, it makes difference, because parameters would have already gravitated to catchment basins of larger values before regularization is added, which means incremental hyperparameter tuning would be ineffective.

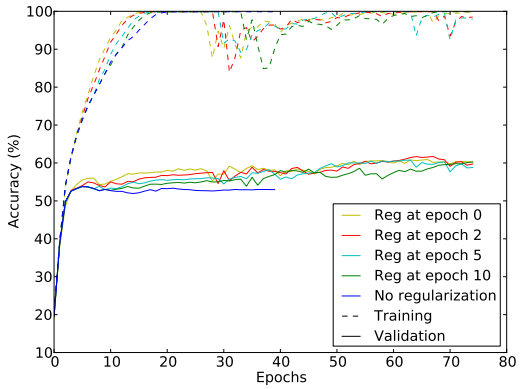
To verify the above conjectures, we design four settings: adding penalty (1) at the beginning, (2) before overfitting at epoch 2, (3) at peak performance (epoch 5), and (4) after overfitting (validation accuracy drops) at epoch 10.

Figure 2 plots the learning curves regarding penalizing weights and embeddings, respectively; baseline (without regularization) is also included.

For both weights and embeddings, all settings yield similar ultimate validation accuracies. This shows ℓ_2 regularization mainly serves as a "local" effect—it changes the error surface, but parameters tend to settle down to a same catchment basin. We notice a recent report also shows local optima



(a) Incrementally penalizing ℓ_2-norm of weights.



(b) Incrementally penalizing ℓ_2-norm of biases.

Figure 2: Tuning hyperparameters incrementally in Experiment I. Penalty is added at epochs 0, 2, 5, 10, respectively. We chose the coefficients yielding the best performance in Figure 1. The controlled trial (no regularization) is early stopped because the accuracy has already decreased.

	λ_{W}			
λ_{embed}	0	10^{-4}	$3{\cdot}10^{-4}$	10^{-3}
0	54.02	57.88	59.96	61.00
10^{-5}	54.94	57.82	60.68	62.05
$3{\cdot}10^{-5}$	55.68	61.02	**64.00**	**63.15**
10^{-4}	60.91	**64.00**	**63.07**	60.56
$3{\cdot}10^{-4}$	58.92	61.33	59.85	42.93
10^{-3}	54.77	56.43	54.05	16.50

Table 1: Accuracy in percentage when we combine ℓ_2-norm of weights and embeddings (Experiment I). Bold numbers are among highest accuracies (greater than peak performance minus 1.5 times standard deviation, i.e., 1.26 in percentage).

	λ_W			λ_{embed}		
p	10^{-4}	$3{\cdot}10^{-4}$	10^{-3}	10^{-5}	$3{\cdot}10^{-5}$	10^{-4}
0	57.88	**59.96**	**61.00**	54.94	55.68	**60.91**
1/6	58.36	59.36	43.42	58.49	59.59	**60.00**
2/6	58.22	**60.00**	16.60	59.34	**60.08**	59.61
3/6	58.63	59.73	16.60	59.59	**59.98**	58.82
4/6	56.43	54.63	16.60	56.76	59.19	56.64
5/6	38.07	16.60	16.60	49.79	53.63	49.75

Table 2: Combining ℓ_2 regularization and dropout. Left: connectional weights. Right: embeddings. (p refers to the dropout rate.)

may not play an important role in training neural networks, if the effect of parameter symmetry is ruled out (Breuel, 2015).

We also observe that regularization helps generalization as soon as it is added (Figure 2a), and that regularizing embeddings helps optimization also right after the penalty is applied (Figure 2b).

6 Combination of Regularizations

We are further curious about the behaviors when different regularization methods are combined.

Table 1 shows that combining ℓ_2-norm of weights and embeddings results in a further accuracy improvement of 3–4 percents from applying either single one of them. In a certain range of coefficients, weights and embeddings are complementary: given one hyperparameter, we can tune the other to achieve a result among highest ones.

Such compensation is also observed in penalizing ℓ_2-norm versus dropout (Table 2)—although the peak performance is obtained by pure ℓ_2 regularization, applying dropout with small ℓ_2 penalty also achieves a similar accuracy. The dropout rate is not very sensitive, provided it is small.

7 Discussion

In this paper, we systematically compared four regularization strategies for embedding-based neural networks in NLP. Based on the experimental results, we answer our research questions as follows. (1) Regularization methods (except re-embedding words) basically help generalization. Penalizing ℓ_2-norm of embeddings unexpectedly helps optimization as well. Regularization performance depends largely on the dataset's size. (2) ℓ_2 penalty mainly acts as a local effect; hyperparameters can be tuned incrementally. (3) Combining ℓ_2-norm of weights and biases (dropout and ℓ_2 penalty) further improves generalization; their coefficients are mostly complementary within a certain range. These empirical results of regularization strategies shed some light on tuning neural models for NLP.

Acknowledgments

This research is supported by the National Basic Research Program of China (the 973 Program) under Grant No. 2015CB352201 and the National Natural Science Foundation of China under Grant No. 61232015. We would also like to thank Hao Jia and Ran Jia.

References

Yoshua Bengio, Réjean Ducharme, P. Vincent, and C. Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Thomas M. Breuel. 2015. The effects of hyperparameters on sgd training of neural networks. *arXiv preprint arXiv:1508.02788*.

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

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

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Proceedings of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing*.

Kazuma Hashimoto, Makoto Miwa, Yoshimasa Tsuruoka, and Takashi Chikayama. 2013. Simple customization of recursive neural networks for semantic relation classification. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems*.

Ozan Irsoy and Claire Cardie. 2014. Deep recursive neural networks for compositionality in language. In *Advances in Neural Information Processing Systems*.

Alex Krizhevsky, Ilyz Sutskever, and Geoffrey Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*.

Igor Labutov and Hod Lipson. 2013. Re-embedding words. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2)*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*.

Andriy Mnih and Geoffrey Hinton. 2009. A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems*.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *Proceedings of International Conference on Artificial Intelligence and Statistics*.

Lili Mou, Hao Peng, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. 2015. Tree-based convolution: A new neural architecture for sentence modeling. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (to appear)*.

Andrew Senior, Georg Heigold, Marc'aurelio Ranzato, and Ke Yang. 2013. An empirical study of learning rates in deep neural networks for speech recognition. In *Proceedings of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing*.

Richard Socher, Jeffrey Pennington, Eric Huang, Andrew Ng, and Christopher D. Manning. 2011. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, pages 1929–1958.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*.

Yan Xu, Lili Mou, Ge Li, Yunchuan Chen, Hao Peng, and Zhi Jin. 2015. Classifying relations via long short term memory networks along shortest dependency paths. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (to appear)*.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. 2014. Relation classification via convolutional deep neural network. In *Proceedings of Computational Linguistics*.

Han Zhao, Zhengdong Lu, and Pascal Poupart. 2015. Self-adaptive hierarchical sentence model. In *Proceedings of Intenational Joint Conference in Artificial Intelligence*.