Semantic role labeling using Recursive Neural Network

Tianshi Li, Baobao Chang

Key Laboratory of Computational Linguistics, Ministry of Education
School of Electronics Engineering and Computer Science, Peking University
Collaborative Innovation Center for Language Ability, Xuzhou 221009 China
lts_417@hotmail.com, chbb@pku.edu.cn

Abstract. Semantic role labeling (SRL) is an important NLP task for understanding the semantic of sentences in real-world. SRL is a task which assigns semantic roles to different phrases in a sentence for a given word. We design a recursive neural network model for SRL. On the one hand, comparing to traditional shallow models, our model does not dependent on lots of rich hand-designed features. On the other hand, different from early deep models, our model is able to add many shallow features. Further more, our model uses global structure information of parse trees. In our experiment, we evaluate using the CoNLL-2005 data and reach a competitive performance with fewer features.

1 Introduction

Semantic analysis of natural language is a basic task in natural language processing (NLP) that makes computer understand the semantic of the language automatically and always plays an important role in most NLP works. Semantic role labeling (SRL) is a sub-task of semantic analysis. Given a predicate of a sentence, the goal of SRL is to assign sematic roles to the constituents of the sentence with respect to the predicate.

Given a sentence s and a predicate (or a verb) in the sentence, the goal of SRL task is to find out all the phrases in the sentence which have semantic roles respect to the predicate. For example, consider the following sentence: *Jack opened the window*. In this sentence, *Jack* is the agent of the verb *opened*, and *the window* is the patient of the verb *opened*. Here agent and patient are two semantic roles to the verb *opened*. Attention that one phrase in the sentence could only have at most one semantic role to the given predicate. The earliest and most popular SRL annotation corpus is PropBank built by Martha Palmer et al [1]. In PropBank, the semantic roles are divided into two classes: core arguments and adjunctive arguments. Core arguments are consist of five subclasses: ARG0,…,ARG4. As the example sentence shown above, *Jack* is ARG0 and *the window* is ARG1. In recent years, the shared task of SRL as CoNLL-2005 usually gives the sentence and the predicate, the only goal is predicting the semantic roles of the given predicate.

SRL is a well-defined task explored by researchers over ten years. In the past decade, most researchers have been focusing on the SRL task mainly using shallow models. They extracted a lot of hand-designed features from the sentence and the parse tree of the sentence, and then put these features into a classifier. The researchers in previous works always used a two-step strategy [2]. The first step is to judge whether the constituent of the sentence is a semantic role of the predicate. The second step is distinguishing the different roles among the constituents which are classified as semantic role in the first step. Various machine learning method have been used in different SRL systems as SVM [3], decision tree [4], and log-linear model [5]. One of drawbacks of these traditional models is that they need a lot of hand-designed features and rich resources to reach high performance.

In recent years, deep learning is becoming more and more popular in NLP. Compare with traditional model, one significant advantage of deep learning is that deep neural model doesn’t need too many manual features. The network can learn this feature itself during training. Some researchers have already tried some deep learning method in SRL. Collobert et al. [6] used convolutional neural network instead of traditional model. They model SRL task using Time Delay neural networks and got competitive performance compared to the state-of-art traditional model. However, they did not model the structure information during model training.

This paper builds a novel recursive deep neural network model for SRL. SRL is a task based on a lot of syntax knowledge, especially parsed syntax tree. Previous works usually use this knowledge as features. We hope to excavate more information than previous works. By the convenience of deep neural network, we can directly use the parsed tree as our network. We expect that our model can learn more syntax knowledge during the network propagating on the parsed tree. For this reason, The recursive neural network is appropriate to our work. Our model is a global structure model which is different with Collobert. In their work, their model handled each word in the sentence one by one independently. In our model, when considering whether a phrase has semantic role, we use global structure information in the parse tree. Different from two-step method, our model directly gives whether a constituent is semantic role and which the role type. Although using a few features in out experiment, our model is still extendible that it can also add more traditional hand-designed features and gets a competitive performance.

Section 2 talks about our model’s architecture and how does our model work. Section 3 shows the experiment and the result. Section 4 shows some related works. Section 5 is the conclusion.

2 Recursive Neural Model

2.1 Basic Recursive Model

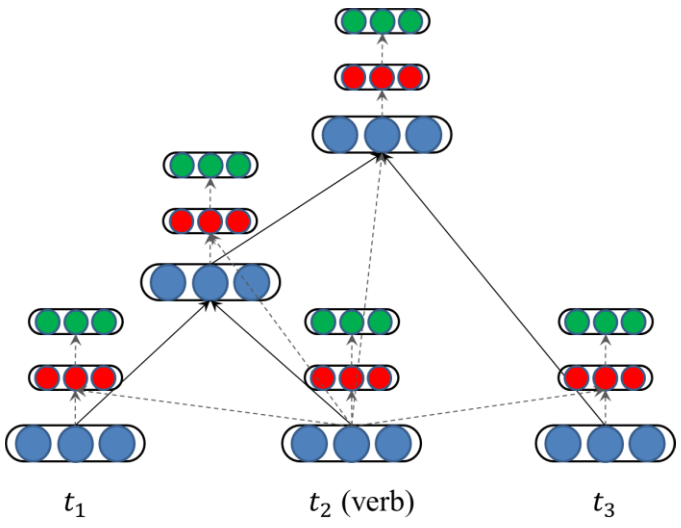

Fig. 1. Our basic recursive model for sentence *s*. The blue nodes are the parse tree nodes. Suppose t_2 is the verb.

Our basic model is shown as Figure 1. Each non-leaf node in the RNN tree has two children. Given a sentence *s* consist of words $t_1 t_2 \dots t_n$, x_i is the word embedding of t_i. Here $x_i \in \mathbb{R}^n$, where *n* is the dimension of the embedding.

First, we focus on the network of the parse tree. Let x_p be the embedding of a parent tree node, x_l be the embedding of the left sub tree node, x_r be the embedding of the right sub tree node. So x_p can be computed as follows:

$$x_p = f(W[x_l; x_r] + b) \tag{1}$$

where $W \in \mathbb{R}^{n\times 2n}$, $[c_1; c_2] \in \mathbb{R}^{2n\times 1}$, *b* is the bias vector and $b \in \mathbb{R}^n$. *f* is the activation function, $f = tanh(\cdot)$.

Second, considering that our model predict each node whether it has a semantic role, each tree node has an output layer. Consider the expandability of the network, we also add a hidden layer below output layer. Because each sentence has a given predicate, we add the predicate information into our hidden layer. The network is shown in Figure 2. Let x_i be the embedding of node *i*, $h_i \in \mathbb{R}^m$ be the hidden layer of node *i*, $o_i \in \mathbb{R}^d$ be the output layer of node *i*, x_{verb} be the embedding of the predicate node in the tree. Where *m* is the dimension of the hidden vector, *d* is the dimension of the output layer (*d* is also the number of semantic role types). h_i and o_i are computed as follows:

$$h_i = f(W_{hid}[x_i; x_{verb}] + b_{hid}) \tag{2}$$

where $W_{hid} \in \mathbb{R}^{m \times 2n}$, $[x_i; x_{verb}] \in \mathbb{R}^{2n}$ and $b_{hid} \in \mathbb{R}^m$.

$$o_i = f(W_{out} h_i + b_{out}) \tag{3}$$

where $W_{out} \in \mathbb{R}^{d \times m}$ and $b_{out} \in \mathbb{R}^d$.

Given the sentence s and semantic roles set y, $score(s, y, \theta)$ denotes the score of y in node i. Where θ is the parameter in the network. We directly use o_i as $score(s, y, \theta)$. We regard the role whose value is the max value among the dimension as the semantic role in this node. It is an intuitive way to predict all semantic roles in a sentence.

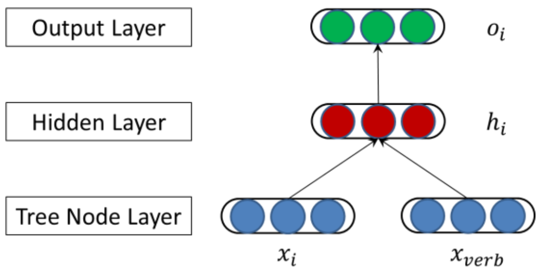

Fig. 2. The network of one tree node i. The blue ones are the Tree nodes, the red one is the hidden node, the green one is the output node. x_i is the embedding of tree node i, x_{verb} is the embedding of the predicate node, h_i is the embedding of the hidden layer of node i, o_i is the embedding of the output layer of node i.

2.2 Pruning Strategy

A sentence may contain several clauses, the roles of a given predicate are always in the same clause as the predicate is. So we need not take too much attention on the roles of other clauses. We use a pruning strategy similar as Xue and Palmer [5].

Pruning Strategy

Step 1: Designate the predicate as the current node and collect its sisters (constituents attached at the same level as the predicate) . If a sister is a PP, also collect its immediate children.

Step 2: Reset the current node to its parent and repeat Step 1 till it reaches the top level node.

Our strategy do not really discard the nodes which are cut off, we reserve their node embeddings computed as Equation (2). But only the nodes which are not cut off have the hidden layer and output layer after out pruning strategy. In this way, our model can not only filter redundancy information and accelerate the train speed but also use the whole parse tree information.

2.3 Add Parse Tree Feature

The baseline RNN model can predict correct role in each tree node. However, there is drawback in it. Consider the following case: given a sentence *s* and the predicate *v*. *t* is a word that appears twice in *s*. To distinguish these two same word, we set them t_1 and t_2. In this sentence, suppose t_1 has a semantic role but t_2 hasn't. When using our baseline model to predict the role of t_1 and t_2, the model use the same weight. Because t_1 and t_2 are the same word, they have same word embedding. According to Equation (3) and Equation (4), t_1 and $t_2$will finally have the same output after forward propagation. So our baseline model will predict them either the same semantic role or none role. To solve this problem, our baseline model add extra feature from the parse tree.

- Top node feature: Top node is the nearest common parent between the current tree node and the predict node. Top node feature is the embedding of the top node.

We add top node feature in the hidden layer. After adding the feature, the hidden layer vector is computed as following equation:

$$h_i = f(W_{hid}[x_i; x_{verb}; x_{top}] + b_{hid}) \quad (4)$$

where $W_{hid} \in \mathbb{R}^{m\times 3n}$, $[x_i; x_{verb}; x_{top}] \in \mathbb{R}^{3n}$.

Using this feature, we have different input in hidden layer and output layer between t_1 and t_2 in previous case. So our RNN model can distinguish the semantic roles between them now. Figure 3 shows the network after adding two features.

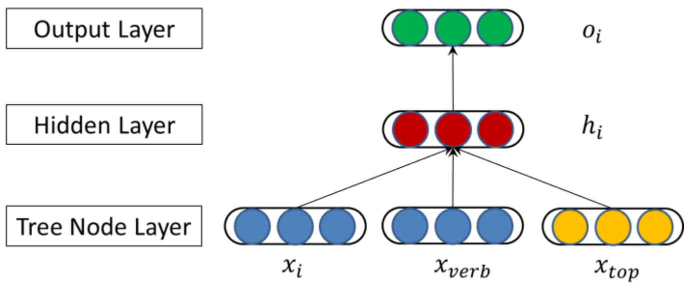

Fig. 3. The network after adding top feature. The yellow one is the feature embedding. x_{top} is the top node feature.

2.4 Distinguish Weight by POS Tag and Node Position

To improve basic model, we add more syntactic features in our RNN model. In this section, we distinguish the weights of the network by POS tag and node position.

POS Tag In our baseline model, when the network is propagating, the weight W on the parse tree is a sharing parameter. Although the purpose of sharing weight is that the network can learning some general features from each recursive substructure, our goal is predict different semantic roles in one sentence. So our model not only needs the global features but local features as well. Achieving this goal, weight of the parse tree is tied by POS tag. We write it in equation as follows:

$$x_p = f(W^{T_l} x_l + W^{T_r} x_r + b^{T_l} + b^{T_r}) \tag{5}$$

where $W^{T_l}, W^{T_r} \in \mathbb{R}^n$ are weight matrices depend on POS tag T_l and T_r. Here T_l is the POS tag of left sub-node and T_r is the POS tag of right sub-node. In this way, our model can learn global information by sharing weight in the same POS tag and learn local information using different weight between different POS tags.

Node Position Like the way handling weight W, hidden layer weight W_{hid} is tied by node position. Node position is a feature which represents whether the tree node is in the left, right or middle of the predicate node. Here let all predicate's ancestor nodes (include itself) be the middle of predicate node. For other node in the tree, if it is in the left subtree of any predicate's ancestor nodes, we set it is in the left of the predicate node; if it is in the right subtree of any predicate's ancestor nodes, we set it is in the right of the predicate node. Here the hidden layer equation is updated as follows:

$$h_i = f(W_{hid}^P [x_i; x_{verb}; x_{top}] + b_{hid}^P) \tag{6}$$

where $W_{hid}^P \in \mathbb{R}^{m \times 3n}$, P is the node position of x_i.

2.5 Decoding

When finding the optimal output of the whole tree, there is a problem: if a tree node is a real semantic role, its subtree node could never be any semantic roles. If we ignore this problem, considering the optimal output of each tree node independently, our model will make many wrong decisions. Algorithm 1 shows the decoding method. Consider there is a binary parse tree, we design two decode scores for each tree node. *decode score 1* is the sum of the score of the current node which is a semantic role (argument) and the score of its all subtree nodes are not arguments; *decode score 2* is the sum of the score that the current node which is not an argument and the score of its children node which chose their most likely assignments. So, the most likely assignment for a node is the one that corresponds to the maximum of these two decode scores. The most likely assignments for the whole tree are the ones that correspond to the maximum of the two decode scores of the root. We call this a global decoding algorithm because it decodes on whole parsed tree using some rules instead of decoding on each tree node independently.

Algorithm 1. GLOBALDECODING

```
Input: The root of the parse tree t for sentence s
Output: The most likely assignments for s
begin
  Let input regard as current node i
```

```
    Let l be the left son of i, r be the right son of i
    GLOBALDECODING(l)
    GLOBALDECODING(r)
    Compute *decode score 1* and *decode score 2*
    **if** i is the root
      **Return** the assignment of max{*decode score 1*, *de-
        code score 2*}
    **end**
  **end**
```

2.6 Training the RNN Model

We use the Max-Margin criterion to train our model following Pei et al. [7]. Given a sentence x_i, let $T(x_i)$ denote the tree node set of x_i and the correct semantic role set is y_i. For given node $p \in T(x_i)$, the correct semantic role for p is $y_{i,p}$. The parameter of out model is $\theta = \{a\ set\ of\ W, a\ set\ of\ b, a\ set\ of\ W_{hid}, a\ set\ of\ b_{hid}, W_{out}, b_{out}\}$. We first define a structured margin loss $\Delta(y_i, \hat{y})$ for predicting a semantic role set (contains none semantic role) $\hat{y}$ for a given correct semantic role set y_i in a tree:

$$\Delta(y_i, \hat{y}) = \sum_{p}^{|T(x_i)|} \kappa 1\{y_{i,p} \neq \widehat{y_p}\} \tag{7}$$

where $|T(x_i)|$ is the number of tree node in x_i and κ is a discount parameter.

The loss is proportional to the number of nodes with an incorrect role in the proposed role set, which increases the more incorrect the proposed role set is. We use $Y(x_i)$ to denote the set of all possible semantic role sets for the sentence x_i. For a given training instance (x_i, y_i), we search for the semantic role set with the highest score:

$$y^* = \arg \max_{\hat{y} \in Y(x_i)} S(x_i, \hat{y}, \theta) \tag{8}$$

where $S(x_i, \hat{y}, \theta)$ is computed as follows:

$$S(x_i, \hat{y}, \theta) = \sum_{p \in T(x_i)} score(x_i, \widehat{y_p}, \theta) \tag{9}$$

where $score(\cdot)$ is designed in Section 2.1.

The object of Max-Margin training is that the highest scoring role set is the correct one: $y^* = y_i$ and its score will be larger up to a margin to other possible role set $\hat{y} \in Y(x_i)$:

$$S(x_i, y_i, \theta) \geq S(x_i, \hat{y}, \theta) + \Delta(y_i, \hat{y}) \tag{10}$$

This leads to the objective function with regularized term for m training samples:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \max_{\hat{y} \in Y(x_i)} (S(x_i, \hat{y}, \theta) + \Delta(y_i, \hat{y}) - S(x_i, y_i, \theta)) + \frac{\lambda}{2} \|\theta\|^2 \tag{11}$$

By minimizing this object, the score of the correct role set y_i is increased and score of the highest score incorrect role set $\hat{y}$ is decreased.

There are many gradient based optimization methods to minimize the objective function. We chose Adaptive Sub-gradient Optimization [8] with minibatches. The parameter updates for the *i*-th parameter $\theta_{t,i}$ at time step *t* is as follows:

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i} \tag{12}$$

where α is the initial learning rate and $g_\tau \in \mathbb{R}^{|\theta_i|}$ is the sub-gradient at time step τ for parameter θ_i.

3 Experiment

3.1 Experiment Settings

We use the CoNLL-2005 data for training and evaluating out model. The data consists of sections of the Wall Street Journal part of the Penn TreeBank, with information predicate-argument structures extracted from the PropBank corpus. There are 64 different semantic roles in the data. CoNLL-2005 data takes sections 2-21 of WSJ data as training set, section 23 as testing set and section 24 as validation set. In addition, the test set of the shared task includes three sections of the Brown corpus (namely, ck01-03). CoNLL-2005 data uses Charniak parser to generate POS tags and full parses automatically. We used English Giga word corpus [9] for word embedding pre-training. To make the experiment result more comparable, we use auto-parsed syntax trees as input, just like other researchers do. Because the parsed syntax trees given in data set are multiway trees and our RNN model needs binary tree, we change the multiway tree to binary tree in a heuristic method. For each tree node *n* in the multiway tree, we simply stay its left child the same in the new tree as the multiway tree and make a new tree node *r* as *n*'s right child. Then let right sons of *n* in multiway tree become the son of *r*. We apply this method recursively in whole multiway tree from top to bottom. By this heuristic method, we reserve the structure and the span of the original multiway tree as much as possible.

There are many hyper-parameters in our model need be set before training our recursive neural network: the tree node embedding size $d = 50$, the hidden layer size $d_{hid} = 80$, the initial learning rate $\alpha = 0.08$, the discount parameter $\kappa = 0.1$, the regularized term parameter $\lambda = 0.0001$, the initial weight range of the whole network in $[-0.001, 0.001]$ and mini-batch size is 20.

3.2 Results and Analysis

Table 1 shows the final result of our experiment. Our RNN system reaches 71.53 F1 score in test set without pruning and 72.27 F1 score with pruning. Koomen et al. [10] get best performance in CoNLL-2005 and they reach 77.92 F1 score in test set. However, their best performance system uses six parse trees generated by Charniak parser.

They also report the performance using only one Charniak parse tree. It is more comparable to our system. To rich a high level performance, they uses lots of syntactic and parse tree features. But our RNN system only use three simple parse tree features. CNN is another deep network model presented by Collobert et al.. They reach 74.15 F1 score in test set, lower than Koomen et al. as well. LM is the language model they use for word embedding initialization. Their embedding dictionary contains 130,000 words trained by Wikipedia and Reuters corpus. Although our model also use pre-training word embedding, our embedding dictionary only contains 18,551 words trained by Giga word. Their system uses much more extra resources for pre-training than our system. They also report the performance without LM (70.99), is lower than our performance.

Because our network is based on the parsed syntax trees, the result is more dependent on the accuracies of the parser than other systems. It is one of the reason our result don't exceed the state of the art. Although our RNN model's performance is lower than Koomen et al. and CNN, our model use less features and extra resource and learn the global structure information by the RNN network itself to reach a competitive performance.

Table 1. Experiment result in for the CoNLL-2005 Test Set.

Approach	**F1**
Koomen et al. (six parse trees)	**77.92**
Koomen et al. (top Charniak parse tree only)	74.76
CNN	70.99
CNN + LM	74.15
Our RNN	71.53
Our RNN + Pruning	72.27

Our RNN system is faster than other two systems. Comparing to Koomen et al., our systems uses a simple architecture and fewer features. And our system does not rely on the output of other existing NLP system as well. Comparing to CNN system, our system has fewer network layers. CNN system has at least five layers and our system only have three layers including tree node layer, hidden layer and output layer. Although our system is a recursive network based on parse tree, an average parse tree level is about ten and the length of sentence is about fifteen, the cost of computing on the recursive parse tree is similar with the cost of convolution layer in CNN. There is one more important point that our system directly use the output of the network as the score to predict semantic roles. We do not need waste time using softmax layer the compute the possibility of each semantic role. Table 2 shows the runtime between our system and other systems.

Table 2. Runtime speed comparison between our system and other systems. We give the runtime in seconds for running on testing sets.

SRL System	**Time (s.)**
Koomen et al.	6253

CNN	51
Our RNN	**3**

4 Related Work

Recent years, many NLP researches achieve competitive performance using neural network. Collobert et al. [6] propose a unified neural network architecture and learning algorithm that can be applied to various natural language processing tasks including: part-of-speech tagging, chunking, named entity recognition, and semantic role labeling. Socher et al. [11] build Recursive Auto-encoder to learn the vector representation of the phrase. They also use recursive deep models for sentiment analysis [12]. Hashimoto et al. [13] use an averaged RNN model for semantic relation classification. Zhang et al. [14] use a RNN-based translation model for minimizing the sematic gap in embedding space.

Our research also use a recursive model as Socher and other researchers do. Different from other researchers' work, we use recursive model into a new task and our model is extendible for traditional features. Compare to Collobert et al., our word use a recursive model to handle SRL task and capture structure information of parse tree by global decoding.

5 Conclusion

In this paper, we used a novel recursive neural network model for SRL. Compare to traditional model, our model uses fewer features. Our model is also extensible for add more features and resources as talked in Section 3. Unlike other deep neural network models for SRL, our model import more global structure information from parse tree by our global decoding algorithm. We did an exploration on structure deep learning for SRL and got a competitive performance.

Acknowledgments. This work is supported by National Key Basic Research Program of China(2014CB340504) and National Natural Science Foundation of China(61273318).

References

1. Martha, P., Dan, G., & Paul, K. (2005). The Proposition Bank: A Corpus Annotated with Semantic Roles. Computational Linguistics Journal, 31, 1.
2. Pradhan, S., Hacioglu, K., Krugler, V., Ward, W., Martin, J. H., & Jurafsky, D. (2005). Support vector learning for semantic argument classification. Machine Learning, 60(1-3), 11-39.
3. Pradhan, S. S., Ward, W., Hacioglu, K., Martin, J. H., & Jurafsky, D. (2004, May). Shallow Semantic Parsing using Support Vector Machines. In HLT-NAACL (pp. 233-240).

4. Surdeanu, M., Harabagiu, S., Williams, J., & Aarseth, P. (2003, July). Using predicate-argument structures for information extraction. In Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1 (pp. 8-15). Association for Computational Linguistics.
5. Xue, N., & Palmer, M. (2004, July). Calibrating Features for Semantic Role Labeling. In EMNLP (pp. 88-94).
6. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. The Journal of Machine Learning Research, 12, 2493-2537.
7. Pei, W., Ge, T., & Baobao, C. (2014). Maxmargin tensor neural network for chinese word segmentation. In Proceedings of ACL.
8. Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. The Journal of Machine Learning Research, 12, 2121-2159.
9. Graff, D., & Chen, K. (2005). Chinese gigaword. LDC Catalog No.: LDC2003T09, ISBN, 1, 58563-58230.
10. Koomen, P., Punyakanok, V., Roth, D., & Yih, W. T. (2005, June). Generalized inference with multiple semantic role labeling systems. In Proceedings of the Ninth Conference on Computational Natural Language Learning (pp. 181-184). Association for Computational Linguistics.
11. Socher, R., Huang, E. H., Pennin, J., Manning, C. D., & Ng, A. Y. (2011). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In Advances in Neural Information Processing Systems (pp. 801-809).
12. Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013, October). Recursive deep models for semantic compositionality over a sentiment treebank. In Proceedings of the conference on empirical methods in natural language processing (EMNLP) (Vol. 1631, p. 1642).
13. Hashimoto, K., Miwa, M., Tsuruoka, Y., & Chikayama, T. (2013). Simple Customization of Recursive Neural Networks for Semantic Relation Classification. In EMNLP (pp. 1372-1376).
14. Zhang, J., Liu, S., Li, M., Zhou, M., & Zong, C. (2014). Mind the gap: Machine translation by minimizing the semantic gap in embedding space. Association for the Advancement of Artificial Intelligence.