Convolution Neural Network for Relation Extraction*

ChunYang Liu[1], WenBo Sun[2], WenHan Chao[2], and WanXiang Che[3]

[1] National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing
lcy@isc.org.cn
[2] The Institute of Intelligent Information Processing, Beihang University, Beijing, China
chaowenhan@buaa.edu.cn,
mike891212@gmail.com
[3] The Institute of Social Computing and Information Retreival, Harbin Institute of Technology
car@ir.hit.edu.cn

Abstract. Deep Neural Network has been applied to many Natural Language Processing tasks. Instead of building hand-craft features, DNN builds features by automatic learning, fitting different domains well. In this paper, we propose a novel convolution network, incorporating lexical features, applied to Relation Extraction. Since many current deep neural networks use word embedding by word table, which, however, neglects semantic meaning among words, we import a new coding method, which coding input words by synonym dictionary to integrate semantic knowledge into the neural network. We compared our Convolution Neural Network (CNN) on relation extraction with the state-of-art tree kernel approach, including Typed Dependency Path Kernel and Shortest Dependency Path Kernel and Context-Sensitive tree kernel, resulting in a 9% improvement competitive performance on ACE2005 data set. Also, we compared the synonym coding with the one-hot coding, and our approach got 1.6% improvement. Moreover, we also tried other coding method, such as hypernym coding, and give some discussion according the result.

Keywords: Relation Extraction, Convolution Network, Word Embedding, Deep Learning.

1 Introduction

Relation extraction focuses on semantic relations between two named entities in natural language text. Tree Kernel based method is a classic method for relation extraction, in which it need to parse the sentence to build tree kernel. However, parsing has very high complexity and it is hard to build a correct parse tree for a long sentence. In Addition, kernel method needs hand-engineering features. As for different tasks, we need to construct different kernel functions. Regarding these, an architecture which is not based on parse tree and can learn features is needed. Deep Neural Network architecture can achieve both two goals above.

* This research was supported by Research Fund for the Doctoral Program for Higher Education of China (New teacher Fund), Contract No. 20101102120016.

H. Motoda et al. (Eds.): ADMA 2013, Part II, LNAI 8347, pp. 231–242, 2013.

Recently, Deep Neural Network models have been applied to many NLP tasks, such as POS Tagging, Name Entity Recognition, Semantic Role Labeling and Sentiment Analysis. Collobert et al. developed the SENNA system that shares representation across task. The SENNA system integrates POS, NER, Language Model and SRL, which are all sequence labeling tasks. To learn feature vectors automatically, previous research usually use embedded representation of word as input layer of CNN. But this method neglects semantic meaning of words, regarding them unique words separately. In this work, we describe a new representation of word. We use a unique code to represent words with same semantic meaning, called synonym coding.

We propose a novel CNN architecture with synonym coding for relation extraction. In experiments, we compared our CNN architecture with the state of art Kernel methods. We also provide the performance of CNN, coding with word list. Result shows CNN with synonym coding outperformed word list coding by 1.6% on ACE 2005 dataset, and CNN architecture outperformed kernel method by nearly 15%, which is a significant improvement compared to kernel methods.

The rest of the article is as follows. First, we describe related work about CNN on natural language processing. We then detail our CNN architecture and synonym coding, followed by experiments that evaluate our method. Finally, we conclude with discussion of future work.

2 Related Work

Early approaches are usually based on patterns and rules, expressed by regular expression. The pattern methods assume all sentences in the same relation type sharing similar linguistic context. But in terms of wide variety of natural language forms in which a given relation may be expressed by multi ways, including syntactic, morphological and lexical variations. Pattern based methods cannot cover all language forms [1]. This causes very low recall value.

Several researchers have already attempted to build machine learning approaches for relation extraction. [2] presented a report of feature vectors based relation classification. The authors annotate words with rich features, including:

- Part-of-Speech
- The relation arguments
- The entity mention level
- The entity types
- Parse tree

In addition, they proposed to take advantage of parse tree without its structure information. However, entities usually have a long distance in relation extraction. When building parse tree, long range dependence will involve many unrelated words, which will sparse feature space. Parse tree structure provides grammar information of words. To fully using this information, [3] presented tree kernel method to consider structure of parse tree, and got slightly progress compared with approaches based on feature

vectors. [4] used dependency path, removing many syntactic nodes which have indirect dependency relation with target entities. This work produced a slightly improvement then that used full parse tree. But building parse tree and dependency tree is a very time-consuming processing, not only when training, but predicting.

Moreover, kernel method needs hard coding of features. To self-adapt different tasks, researchers import embedded word representation to NLP and obtain some success.

[5] described a lookup table processing and stack it on a multilayer perceptron for building language model, resulting in a 20% improvement on perplexity. Language model is an important build block of machine translation and speech recognition systems, so this improvement advanced the tasks involving language model. [6] proposed neural network architecture for machine translation, resulting in enhancing 2 point of BLEU score. In addition, [7] presented a unity CNN for basic NLP tasks and SLR, got the state-of-the-art performance. All methods above use a lookup table layer to learn words' feature by back-propagation. The input of lookup table is word indexes which are provided from a wordlist. Each word has a unique index, which means even words with similar semantic meaning are resolved separately, ignoring semantic information.

As for compositive applications, [8] trained a deep architecture, auto-encoder, for sentiment classifier, and surpassed the state-of-the-art. Auto-encoder [9] is one of the first processing of deep learning architecture. The goal of an auto-encoder is finding a best representation of input. For sentiment classifier, inputs are words, whose semantic meaning exactly indicates their sentiments. But for relation extraction, different words play unequal roles to identify a relation type. Some words are indicator of relation types, some are unrelated with relation. So, although auto-encoder for words can represent well in many domains, it can't provide enough specific information for relation extraction task, such as grammatical structure of sentences.

3 Convolution Network Architecture

Previous research on relation extraction focuses on how grammatical relation, such as parse tree, expresses semantic relation. Undeniably, parse tree and dependency tree performs well with large number of empirical features. For semantic information, WordNet provides us significant improvement on performance of relation extraction system. But building parse tree is a time consuming task. Moreover, hand-built features are rigid considering context.

Ideally, we want to build a relation extraction system without a time consuming parse tree. Meanwhile, the system should consider of grammatical structures.Also, we want to avoid hand-built features. Hence, we prefer to find an automatic weight learning approach to avoid time-consuming feature generation. We found that a deep neural network achieve both two goals.

To including grammatical information, we proposed convolution neural network (CNN) [10]. In image processing tasks, convolution is used to extract features over blocks. In language processing, CNN can be used to evaluate grammatical relations

between abutting words that probably constitute a phrase. So, CNN can replaces parse tree to provide grammatical information.

3.1 Word Embedding

Word embedding is a method to project words to vectors. The first step of word embedding is using One-hot coding. One-hot coding in NLP is a coding method that transfer word index to a binary code whose size is equal to size of wordlist for the convenience of computing. In one-hot code, only on position of index coded to 1, other positions are all 0.

One-hot coding method is formalized below:

Define word dictionary is D, the i^{th} word in D is w_i. Input sentence is S.

We construct an indice function to index words in the input sentence. s(i) refer to the i^{th} word in the input sentence. Given the notation above, we define one-hot coding of a word as an identity function:

$$C(w) = \{1_{w=w_i}\}_{i\in[0,\#D]} \tag{1}$$

Equation (1) means if and only if i^{th} word in D is equal to input word, element is 1. The other elements are all 0. Synonym coding uses a synonym list instead of wordlist. So, in synonym code, D_s represents the synonym dictionary. The i^{th} synonym cluster is noted by Syn_i. Synonym code is a vector of $\#D_s$ dimension, defined as:

$$SC(w) = (1_{w\in Syn_i})_{i\in[0,\#D_s]} \tag{2}$$

Equation (1) means if and only if i^{th} word in D is equal to input word, element is 1.

Although we have synonym list, the list can't hold all words. So if the input sentence has words beyond D_s. We add the words toD_s and each word composite a seperate synonym cluster. Since we have notations above, an input sentence of n words can be mapped into one-hot coding.

$$S = \{SC(s(i))\}_{i\in[1,n]} \tag{3}$$

3.2 Basic Architecture

The type of neural network we employ here is Convolution Neural Network[10]. Convolution Network has been used in one unity NLP architecture. The neural network concluded into 3 main building blocks:

- Input layer
- Convolution Layer
- Classic Neural Network Layers

Before inputting the sentence to the network, sentences will be mapped into synonym code. In terms of variant of sentences length, we define a window around P_1, P_2, whose size is noted as *wsz*. All words in this window will be coded. And the

distance of P_1, P_2 is noted as b. If b is larger than *wsz*, we set P_1 to left-most, and set value on the right-most position to B. If b is smaller than *wsz*, empty position will distribute around P_1, P_2 on average, and set those position to a sign X. Both B and X will be added to synonym list, participating in coding processing.

Input Layer

For the purpose that words weights can be learnt by back-propagation, we implement a Lookup Table Layer [11] as the input layer of CNN.

In lookup table layer$LT_W(\cdot)$, each word *w* will be mapped into a d-dimension space:

$$\mathrm{LT_W(s(i))} = \mathrm{W} \cdot \mathrm{SC(s(i))} = \mathrm{W_i} \tag{4}$$

where $W \in R^{d \times \#D_S}$is a matrix of parameters to belearnt.$W_i \in R^d$ is the i^{th} column of W. and d is the word feature vector size to be chosen by the user. In the first layer of our architecture an input sentence of n words is transformed into a series of vectors by applying lookup table to each word.

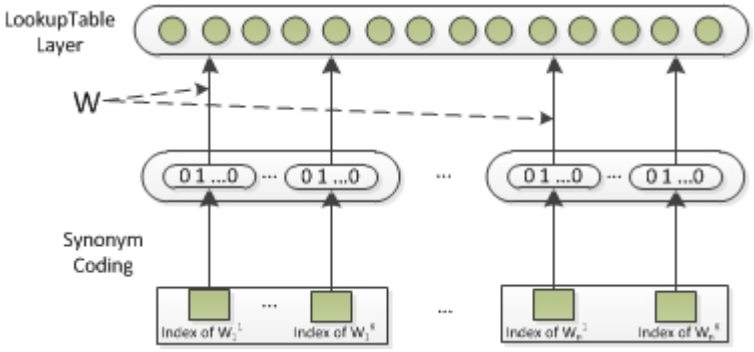

Fig. 1. Lookup Table Layer and Synonym Coding

In relation extraction, the mention level and type of entity are necessary information for classification. So in our experiments, we added Name Entity Type List and Mention Level List to code the name entity mention level and type. Two lists above will generate two new features.

When a word is decomposed into K features, it can be represented as a tuple

$$SC'(s(i)) = \{SC_1(s(i)), SC_2(s(i)), \ldots SC_K(s(i))\} \tag{5}$$

where $SC'(s(i)) \in D^1 \times D^2 \times \ldots \times D^K$, D^K is the dictionary for the k^{th} feature. Each feature will be mapped into a new vector space, so each feature need coding separately. To associate multi features, lookup table for each feature is defined as$LT_{W^k}(\cdot)$, with the parameters$W^k \in R^{d^k \times \#D^k}$where d^k is a user-specified vector size. A word s(i) is then mapped into a $\mathrm{d} = \sum_k d^k$ dimension vector by concatenating all lookup table outputs:

$$LT_{W^1,\ldots,W^K}(s(i))^T = \{LT_{W^1}(s(i)), LT_{W^2}(s(i)), \ldots, LT_{W^K}(s(i))\} \quad (6)$$

Convolution Layer

A convolution layer is typically used in a convolution network for vision tasks. When one want to get a more abstract feature, it's reasonable to ignore some inputs and select a subsequence and compute global weights.

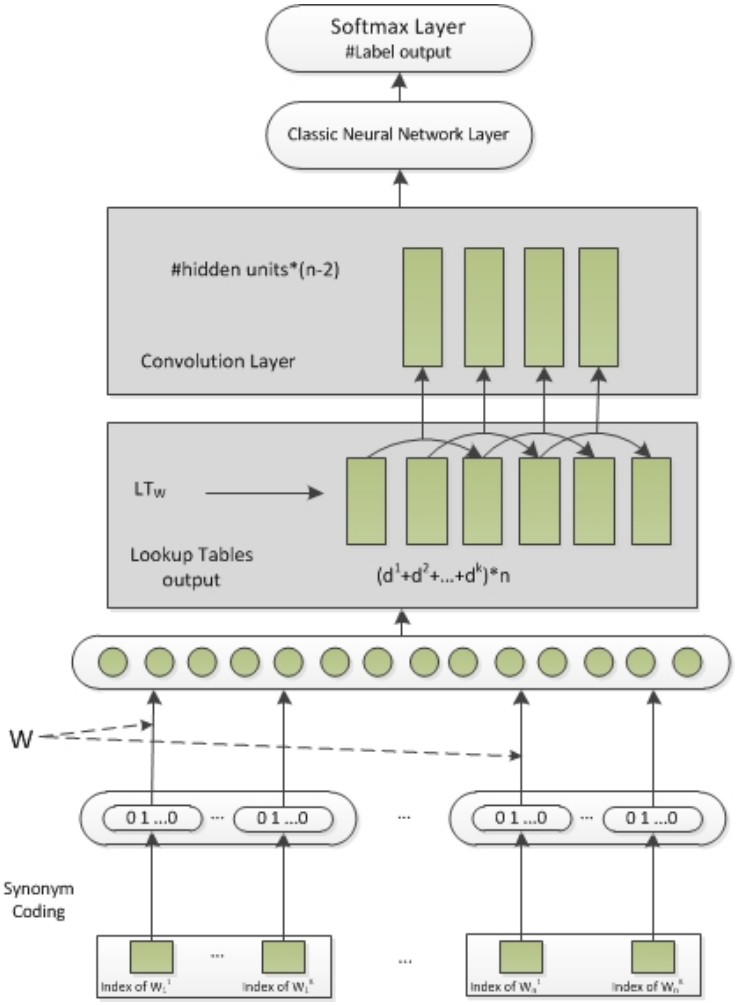

Fig. 2. Deep Neural Network Architecture

A typical convolution usually applied on 2D data, which can be visualized to an image. But in case of our sequential input, convolution layer of our CNN have a sequential kernel, which is an exception of typical convolution kernel. A convolution kernel maps a subsequence to a new vector space. The width of the kernel decides the

dimension of new vector space and step decides number of subsequence involved in convolution. A convolution computation on one subsequence is:

$$\text{Convolution}\left(s_{i,i+width-1}\right) = \sum_{j=1}^{width} L_j \cdot s_{i+j} \tag{7}$$

where $L_j \in R^{n \times d}$, $s_j \in R^d$. So the result convolution is a matrix of R^n.

Concatenating convolutions on all subsequence referring of lookup table layer's outputs, referring to (6), we'll get $\text{Convolution}\left(LT_W(s)\right)$

$$\left\{\text{Convolution}\left(LT_W(s)_{i,i+width-1}\right)\right\}_{i \in \{1,1+step,\ldots,wsz\}} \tag{8}$$

The dimension of each convolution result is number of hidden units. After convolution on whole sequence, it results in a matrix of$R^{hu \times (wsz-step)}$. We then add to the architecture a layer, which captures the most relevant features over the window by feeding CNN layers into a Max Layer, which takes the maximum over a row in (8) for each of the n_{hu} outputs.

3.3 General Deep Neural Network Architecture

Combining the lookup table layer and convolution layer, we'll get basic convolution network. The output of lookup table layer is the input of convolution layer. A max layer, which is applied to the output of convolution layer, reducing output dimension to number of hidden unit, is stack on the convolution layer. To classify relation types, we add a softmax layer to be output layer. Softmax Layer:

$$h_i(z) = \frac{\exp z_i}{\sum_j \exp z_j} \tag{9}$$

Where *i* indicates i^{th} output unit, and j refers to j^{th} input of softmax layer.

$$\sum_i h_i(z) = 1 \tag{10}$$

(9) and (10) allows us to interpret outputs as probabilities for each relation type prediction. The size of outputs is the number of classes of relation extraction.

Between convolution layer and output layer, some classic neural network layers can be added to get more abstract feature. The whole architecture can be summarized in Figure 2.

The whole network trained by a normal stochastic gradient decent with negative log-likelihood criterion.

4 Experiments

In this section, we compare the performance of our new CNN architecture with the performance of tree kernel method. To prove effectiveness of synonym coding, we also compare CNN with synonym coding input with that without synonym coding.

4.1 Date Set

The Automatic Content Extraction (ACE) Evaluation covers the Relation Detection track. ACE 2005 [12] dataset consists of 599 documents which are related with news, speech and email. As ACE 2005 defined, all relations are annotated to 7 major types and 19 subtypes. Our training set covers 6 major types: ART (686 instances), GEN-AFF (1280 instances), ORG-AFF (395 instances), PART-WHOLE (1009 instances), PER-SOC (465 instances), PHYS (469 instances), and 18 subtypes. The Metonymy type includes very small amount of instances, this is not enough to evaluate performance on classification for this type. So we abandoned it.

ACE training set provides entity mention level and entity type features, we extract them and generate their feature list. We also add Part-of-Speech feature generated by Stanford POS tagger [15]. So we finally have five feature lists: word list (after stemming), POS list, mention level list, entity major type list, entity subtype list.

4.2 Experiments Setup

As for the tree kernel classifier, we use Stanford parser [13] to obtain parse tree and dependency tree, and implemented all reported tree kernels by LibSVM [19].

For all kernels involved, we use same SVM parameters. We have conducted a 5-fold cross validation. In addition, we also extract 30% out of all training data to be test set.

4.3 Result

In the results, we report usual evaluation measure, Precision and Recall, comparing the performance with kernel-based method on unbalanced corpus.

Table 1. Precision, recall, and F-measure for 5-fold cross validation and test set. Here SPK denotes the Shortest Path Kernel [16], TDK denotes the Typed Dependency Kernel [17], CK denotes the Context-Sensitive Tree Kernel[18]

	5-fold cross validation			Test set		
	Precision	Recall	F1	Precision	Recall	F1
CNN	**0.868**	**0.875**	**0.872**	**0.837**	**0.839**	**0.838**
SPK	0.821	0.472	0.599	0.803	0.455	0.581
CK	0.812	0.658	0.727	0.796	0.641	0.710
DTK	0.822	0.702	0.758	0.811	0.688	0.744

The CNN outperforms the tree kernel methods by 9 points on F-measure, which is a significant improvement.

The state-of-the-art tree kernel gives a 74% F-measure, this is to say, kernel function is a good method to represent relation. But construction of kernel function need experiential coding for features. This cannot give the kernel function the best representation on given task. The performance of CNN on subtypes, tested by 5-fold cross validation, is showed in Table 2.

Table 2. Theperformance of CNN, SPK and TDK on subtypes

	Precision	Recall	F1
CNN	0.748	**0.748**	**0.748**
SPK	0.743	0.303	0.430
CK	**0.759**	0.549	0.637
DTK	0.755	0.599	0.668

Table 2 shows CNN has poor performance on subtypes, because 5 types in subtypes only have less than 20 instances. Obviously, this is not enough to train a classifier. The F-measures on 3 subtypes are only about 0.1. But when we evaluate performance on subtypes by F1, which considers the amount of different types, avoiding unbalanced dataset pulling down the performance, the F1 achieved **74.8%**, better than **66.8%** of DTK. This shows our architecture is effective on relation extraction.

ACE data set do not has enough words to train a word embedding layer for better semantic information, so we add synonym list in WordNet to provide more semantic information for words.

To generate synonym coding, we use Synonym List in WordNet 2.1 [14]. There are 8049 stemmed words in wordlist conducted from training set. After coding, the size of list reduces to 6741 words.

We compared CNN with synonym coding with CNN without that, performance showed in Table 3.

This result shows Synonym coding improve the performance by 1.6% than CNN without Synonym coding. The synonym coding method is proved to be effective here.

Table 3. Performance of CNN with synonym coding and that without synonym coding

	Precision	Recall	F1
Synonym	**0.837**	**0.839**	**0.838**
Non-Synonym	0.837	0.813	0.825

Variant of window size and feature dimension may influent performance of CNN, so we conducted another experiments on different wsz and d^k . The results are showed in table 4.

Table 4. Performance on different window size and different feature dimension

	wsz=7			wsz=15			wsz=21		
d^k	P	R	F1	P	R	F1	P	R	F1
15	0.769	0.762	0.765	0.823	0.818	0.821	0.835	0.837	0.836
50	0.766	0.785	0.767	0.825	0.822	0.824	0.837	0.839	0.838
100	0.769	0.756	0.762	0.816	0.807	0.812	0.807	0.802	0.804

On variant window size, result shows that larger *wsz* will get better performance. This appearance showed difference against dependency kernel, whose effect reduces unrelated words. In the dependency tree kernels reported in the first experiment, most of the node number on dependency path is around 5 or 6, smaller than the minimum *wsz* in CNN. This phenomenon can be explained by that words in CNN don't provide any grammar structure information, leading to no initial weight of words are given. But in tree kernel methods, words on dependency path indicate these words are more important than other words. So CNN needs more other words to learn which words are more effective. Besides, we also observed that the maximum feature dimension didn't give a better performance, but the middle one.

We also used other coding table except synonym table, such as, hypernym table from WordNet. We use three hierarchies to integrate words into their hypernym. The result shows in table 5.

Table 5. Performance using synonym table and hypernym table

	Precision	Recall	F1
Synonym	0.837	0.839	0.838
Hypernym-1	0.828	0.822	0.824
Hypernym-2	0.821	0.815	0.817
Hypernym-3	0.819	0.815	0.817

An interesting phenomenon is that as the semantic hierarchy going up, performance of CNN gets no enhance. So, we tried to cluster word feature vectors and got some interesting results.

Table 6. Sample of word clusters

Cluster36	Cluster88
Neichangshan, F-15e, RigobertoTiglao, Hezbollah	George W. Bush, High Court, Haditha, Al Anbar

In these three clusters, we can observe three different topics, which is showed by words' semantic meaning obviously. Here we spread out some words in cluster. In cluster1, most of words are about politics. And In cluster2, weapons are listed. In training processing, lookup table layer learn words vectors by back-propagation. After training, the words are clustered by their semantic meaning, like hypernym. But these clusters are more adaptive to the relation extraction task. Adding hypernym could be regarded as another hard coding comparing with self-learnt clusters. So Adding hypernym leads to negative effects on classification. Because of the limited corpus size, we didn't get more specific clusters, but we observe the trend of self-organization on semantic meaning.

In conclusion, CNN with Synonym coding give a better performance than the state-of-the art kernel method.

5 Discussion and Future Work

From the experiments above, we find that CNN have better performance on relation extraction. Synonym coding method is also an effective coding method to improve overall performance. Actually, only 16% words in wordlist are coded to synonym after synonym coding. In this trend, if we use a larger synonym coding list, which can code more words, the performance may improve.

Moreover, on a conceptual prospective, synonym coding is just a coding method to reduce input space before project words to vectors. Other coding method or word selection method can reduce input space too. For example, if we only consider the words on dependency path, which compose the advantage of dependency tree kernel with CNN architecture. Our future work will focus on finding more effective coding method and word selection.

Although CNN give us a better resolution on relation extraction, there are some problems about CNN architecture we should notice. The architecture doesn't give us a guide how to choose the feature dimension. Here in our method, a convolution layer is included. We also tried using linear layer to find the divergence with convolution layer. Result shows convolution layer products better performance, but we cannot find a way to explain this divergence. So for the future work, we will try to find some theoretical method to explain the differences when using variant layer and find a guild line to help us determine the feature dimension.

References

1. Grishman, R.: Information Extraction: Capabilities and Challenges. Lecture Notes of 2012 International Winter School in Language and Speech Technologies, Rovirai Virgili University (2012)
2. Kambhatla, N.: Combining lexical, syntactic, and semantic features with maximum entropy models for information extraction. In: The Companion Volume to the Proceedings of 42st Annual Meeting of the Association for Computational Linguistics, vol. 1, pp. 178–181 (2004)

3. Zelenko, D., Aone, C., Richardella, A.: Kernel methods for relation extraction. Journal Machine Learning Research 3, 1083–1106 (2003)
4. Culotta, A., Sorensen, J.: Dependency tree kernels for relation extraction. In: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, pp. 423–430 (2004)
5. Collobert, R., Weston, J.: Fast Semantic Extraction Using a Novel Neural Network Architecture. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pp. 560–567 (2008)
6. Schwenk, H., Rousseau, A., Attik, M.: Large, pruned or continuous space language models on a gpu-forstatistical machine translation. In: Workshop on the Future of Language Modeling for HLT (2012)
7. Collobert, R., Weston, J.: A unified architecture for natural language processing: Deep neural networks with multitask learning. In: ICML 2008 (2008)
8. Grolot, X., Bordes, A., Bengio, Y.: Domain adaptation for large-scale sentiment classification: A deep learning approach.In ICML 2011 (2011)
9. Bourlard, H., Kamp, Y.: Auto-association by multilayer perceptrons and singular value decomposition. Biological Cybernetics 59, 291–294 (1988)
10. LeCun, Y., Bengio, Y.: Convolutional Networks for Images, Speech, and Time-Series. The Handbook of Brain Theory and Neural Networks. MIT Press (1995)
11. Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C.: A Neural Probabilistic Language Model. Journal of Machine Learning Research 3, 1137–1155 (2003)
12. Walker, C., Strassel, S., Medero, J., Maeda, K.: ACE 2005 Multilingual Training Corpus. Linguistic Data Consortium, Philadelphia (2006)
13. Klein, D., Manning, C.: Accurate Unlexicalized Parsing. In: Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423–430 (2003)
14. Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
15. Toutanova, K., Klein, D., Manning, C., Singer, Y.: Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In: Proceedings of HLT-NAACL, pp. 252–259 (2003)
16. Bunescu, R., Mooney, R.: Proceedings of the 19th Conference on Neural Information Processing Systems (NIPS), Vancouver, BC (2005)
17. Reichartz, F., Korte, H., Paass, G.: Semantic Relation Extraction with Kernels Over Typed Dependency Trees. In: KDD 2010, Washington, DC (2010)
18. Zhou, G., Zhang, M., Ji, D., Zhu, Q.: Tree Kernel-based Relation Extractionwith Context-Sensitive Structured Parse Tree Information. In: EMNLP 2010, Prague, pp. 728–736 (2007)
19. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011)