PAPER

Relation Extraction with Deep Reinforcement Learning

Hongjun ZHANG[†], ***Nonmember***, **Yuntian FENG**[†a)], ***Student Member***, **Wenning HAO**[†], **Gang CHEN**[†], ***and*** **Dawei JIN**[†], ***Nonmembers***

SUMMARY In recent years, deep learning has been widely applied in relation extraction task. The method uses only word embeddings as network input, and can model relations between target named entity pairs. It equally deals with each relation mention, so it cannot effectively extract relations from the corpus with an enormous number of non-relations, which is the main reason why the performance of relation extraction is significantly lower than that of relation classification. This paper designs a deep reinforcement learning framework for relation extraction, which considers relation extraction task as a two-step decision-making game. The method models relation mentions with CNN and Tree-LSTM, which can calculate initial state and transition state for the game respectively. In addition, we can tackle the problem of unbalanced corpus by designing penalty function which can increase the penalties for first-step decision-making errors. Finally, we use Q-Learning algorithm with value function approximation to learn control policy π for the game. This paper sets up a series of experiments in ACE2005 corpus, which show that the deep reinforcement learning framework can achieve state-of-the-art performance in relation extraction task.

key words: *relation extraction, deep reinforcement learning, CNN, Tree-LSTM*

1. Introduction

Information extraction [1] is to extract entities, relations, events and other factual information of specified types from natural language text automatically and then output structured information, which is one of the most important applications in the field of natural language processing. Relation extraction is the kernel technique of information extraction, which identifies predefined relations between target named entity pairs in text.

In the last decade, relation extraction research has been dominated by two methods. One is feature method. Kambhatla [2] employed Maximum Entropy models to combine diverse lexical, syntactic and semantic features for extracting relations. Sun et al. [3] presented a simple semi-supervised relation extraction system with large-scale word clustering. The other one is kernel method. Culotta and Sorensen [4] detected and classified relations between entities using this kernel within a Support Vector Machine. Bunescu and Mooney [5] designed a kernel method that used an even smaller part of the dependency structure for relation extraction. Nguyen et al. [6] explored the use of innovative kernels based on syntactic and semantic structures for a target relation extraction task. These methods attempt to improve the relation extraction performance by enriching the feature sets from linguistic analysis and knowledge resources. However, the clear drawback is that linguistic analysis and knowledge resources are hand-designed and often performed by existing natural language processing modules.

With the rise of deep learning techniques more and more researchers have applied CNN and RNN in relation extraction task, which achieve good performance. Deep learning based method uses only word embeddings as network input, which can extract hidden structure from relation mentions automatically and capture global information over long distances. It models relations between target named entity pairs while reducing the workload of hand-made feature engineering greatly. Nguyen et al. [7] introduced a convolutional neural network for relation extraction that took advantages of multiple window sizes for filters and pre-trained word embedding as an initializer on a non-static architecture to improve the performance. Miwa et al. [8] stacked bidirectional tree-structured LSTM-RNNs on bidirectional sequential LSTM-RNNs to extract entities and relations between them.

Relation extraction task always comes with a very unbalanced corpus, where there are no predefined relations between most named entity pairs. Therefore, deep learning method cannot effectively extract relations from the unbalanced corpus by equally dealing with each relation mention. It is also the main reason why the performance of relation extraction is significantly lower than that of relation classification. Relation classification aims to classify the semantic relations between pairs of nominal in a sentence. It often comes with a balanced dataset where the number of non-relations is comparable to predefined relations.

At present, many researchers have begun to apply deep reinforcement learning in some text processing tasks. Deep reinforcement learning can generate deeper representation of states and actions in the environment, and then learn mappings from states to actions, which enable chosen actions to get the greatest reward from the environment. Guo et al. [9] introduced a novel schema for sequence to sequence learning with a Deep Q-Network (DQN), which decoded the output sequence iteratively. He et al. [10] introduced a novel architecture for reinforcement learning with deep neural networks designed to handle state and action spaces character-

Manuscript received November 6, 2016.
Manuscript revised March 14, 2017.
Manuscript publicized May 17, 2017.
[†]The authors are with PLA University of Science and Technology, Nanjing, Jiangsu, 210007 China.
a) E-mail: fengyuntian2009@live.cn (Corresponding author)
DOI: 10.1587/transinf.2016EDP7450

ized by natural language, as found in text-based games.

Inspired by above researches, this paper designs a deep reinforcement learning framework for relation extraction. We consider relation extraction task as a two-step decision-making game. The first step is to judge whether target named entity pairs contain predefined relations initially. The second step is to classify contained relations as concrete types. CNN can be applied to extract deeper features from relation mentions, since it is good at capturing key information in the sentence. The features represent the state of first-step decision, namely initial state s_1 in the environment. We use Tree-LSTM to extract deeper features from relation mentions again, because semantic features that relation extraction task requires mostly exist in the structure of dependency tree. The features represent the state of second-step decision, namely transition state s_2 in the environment. Besides, LSTM component can also solve the vanishing gradient problem in traditional recursive neural network. Although two-step decision-making game is not multi-step continuous decision-making problem which reinforcement learning is good at solving, the deep reinforcement learning framework can combine two deep learning models perfectly. In addition, we design penalty function to increase the penalties for first-step decision-making errors, which can take care of the problem of unbalanced corpus. Finally, we use Q-Learning algorithm with value function approximation to learn control policy π for the game. This paper sets up a series of experiments in ACE2005 corpus, which show that the deep reinforcement learning based method can achieve state-of-the-art performance in relation extraction task.

To the best of our knowledge, there has been no work on employing deep reinforcement learning for relation extraction task so far. This paper is the first attempt to fill in that gap and provides a good thinking way for future research in this area. In the following, we define the task in Sect. 2 and present the deep reinforcement learning framework in Sect. 3. We detail an extensive evaluation in Sect. 4 and finally conclude in Sect. 5.

2. Task Definition

To define relation extraction task, this section analyzes a sentence which is randomly selected from ACE2005 corpus, as shown in Table 1. The sentence in Table 1 contains two relation mentions, of which the relation types are both "PHYS" and the subtypes are both "Located". The target named entity pair in the first relation mention are "teams of nurses and doctors" and "packed emergency rooms", and the target named entity pair in the second relation mention are "the wounded" and "packed emergency rooms". The named entity types of "teams of nurses and doctors" and "the wounded" are both "PER", and their subtypes are both "Group". The named entity type of "packed emergency rooms" is "FAC", and its subtype is "Subarea-Facility".

Relation extraction task is to extract relations between target named entity pairs in the sentence. Table 1 shows

Table 1 A sentence in ACE2005 corpus

teams of nurses and doctors were seen in packed emergency rooms attending to the wounded	
Relation Example 1	Type="PHYS" Subtype="Located"
teams of nurses and doctors	Type="PER" Subtype="Group"
packed emergency rooms	Type="FAC" Subtype="Subarea-Facility"
Relation Example 2	Type="PHYS" Subtype="Located"
the wounded	Type="PER" Subtype="Group"
packed emergency rooms	Type="FAC" Subtype="Subarea-Facility"

that there may be many named entity pairs in a sentence, but there are no predefined relations between most named entity pairs. Therefore, relation extraction task should include two steps: (1) judge whether target named entity pair contains certain predefined semantic relation; (2) classify the relation as a concrete type.

The second step is usually separate to study as a task, and that is relation classification. The performance of relation extraction is significantly lower than that of relation classification. The main reason is that the corpus which relation extraction task comes with is extremely unbalanced, where most named entity pairs do not contain predefined relations. Therefore, the first step of relation extraction task is crucial. That is what many relation extraction methods attach little attention to, and those methods do not design strategy that singling out the first step. We consider relation extraction task as a two-step decision-making game, and two decisions perform two steps of relation extraction task separately. More importantly, the reinforcement learning method that we introduce in this paper can reward or penalize results of each decision. If first-step decision leads to errors, we will increase penalties to tackle the problem of unbalanced corpus.

3. Deep Reinforcement Learning Framework

This section describes a deep reinforcement learning framework, which is capable of combing two deep learning models effectively for relation extraction task. Firstly, we use CNN to model key information in relation mentions and abstract initial state in the framework. Secondly, we use Tree-LSTM to model dependency information in relation mentions and abstract transition state in the framework. Then we introduce the method of pre-training two deep learning models. Finally, we use Q-Learning algorithm with value function approximation to train the entire framework.

3.1 Framework Description

We build a deep reinforcement learning framework for relation extraction task, as shown in Fig. 1. The environment of relation extraction task is input text. It is a sentence with target named entity pair. There are three states in the environment: state1, state2, and end, as denoted by $S = s_1, s_2, s_e$. There are many actions to take, such as action1, action2, action3, action4 and so on, as denoted by $A = a_1, a_2, a_3, a_4 \cdots$. It is difficult to find some measures to represent states of input text directly, so we use two deep learning models to abstract deeper features of input text, and then generate initial

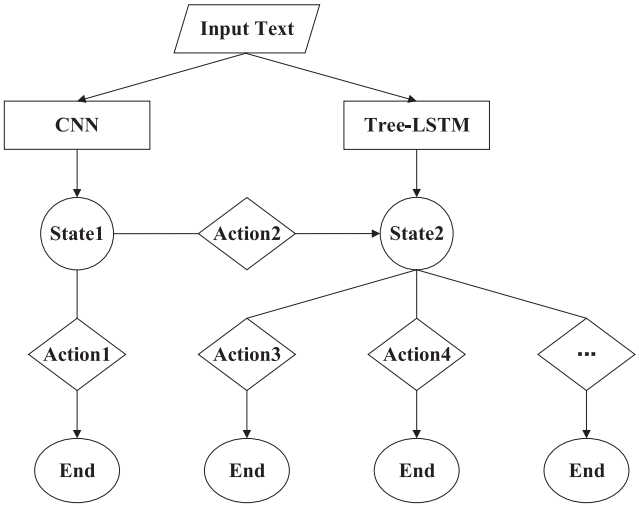

Fig. 1 Deep reinforcement learning framework

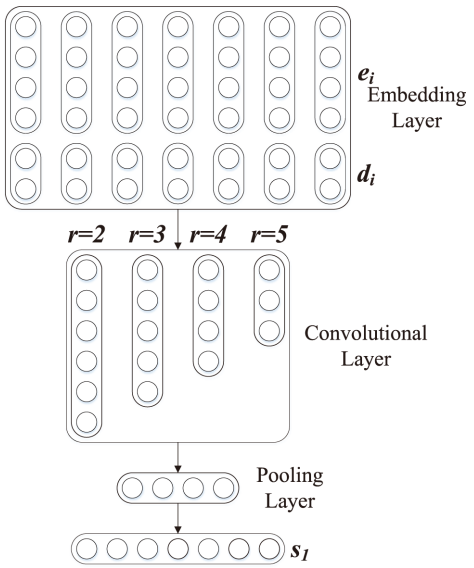

Fig. 2 The structure of CNN

state s_1 and transition state s_2. An agent can interfere with the environment constantly by taking action set A at state set S. a_1 and a_2 can be taken at s_1. a_1 represents to judge that there are no predefined relations in target named entity pair. After taking a_1, end state s_e will be reached. a_2 represents to judge that there is certain predefined relation in target named entity pair. After taking a_2, transition state s_2 will be reached. $a_3, a_4 \cdots$ can be taken at s_2. $a_3, a_4 \cdots$ represent to judge the relation contained in target named entity pair to be different types respectively.

In addition, while interfering with the environment, some state transitions will occur, such as (s_1, a_1, r_1, s_e), (s_1, a_2, r_2, s_2), (s_2, a_3, r_3, s_e), (s_2, a_4, r_4, s_e) and so on. (s_1, a_2, r_2, s_2) represents that an agent takes a_2 at s_1, reaches s_2 and then gets reward r_2 from the environment. After (s_1, a_1, r_1, s_e) occurs, if there are no relations in target named entity pair, then set $r_1 = 10$; if there is certain predefined relation in target named entity pair, then set $r_1 = -20$ to penalize first-step decision-making error. After (s_1, a_2, r_2, s_2) occurs, set $r_2 = 5$. After (s_2, a_3, r_3, s_e) occurs, if the type of the relation contained in target naming entity pair corresponds to the judgment of a_3, then set $r_3 = 10$; if does not, then $r_3 = -10$; if there is not certain relation in target named entity pair, then set $r_3 = -20$ to penalize second-step decision-making error. After (s_2, a_4, r_4, s_e) occurs, the condition of reward and penalty is the same as that of (s_2, a_3, r_3, s_e).

3.2 Generate Initial State by CNN

Due to the special structure sharing local weighs, convolutional neural network (CNN) has unique advantage in local feature extraction. Its structure may more possibly accord with real biological neural network, reducing the complexity of neural network. Therefore, we use CNN to represent relation mention as a feature vector with fixed size of dimensions, which is initial state s_1 in the deep reinforcement learning framework.

The above CNN can be divided into three layers: (1) Embedding Layer, it replaces traditional discrete vector with real-valued vector to represent each word in relation mention, and is the input of CNN. (2) Convolutional Layer, it uses filters to extract important features from the input. (3) Pooling Layer, it uses pooling function to determine the most relevant features in convolutional layer. The structure of CNN is shown in Fig. 2.

3.2.1 Embedding Layer

Embedding layer converts discrete features of each word in relation mention into continuous features, which can be the input of CNN. Due to the structure of neural network that can only receive the input with fixed size, we must trim long sentences and complete short sentences to make them have the same length.

We use only word embeddings to represent each word in relation mention, which reduce the workload of handmade feature engineering. We use the word embedding that has been trained by predecessors to initialize word embedding table. By querying the table, we can convert the i-th word in relation mention into its word embedding e_i. In relation extraction task, some feature embedding d_i is necessary, such as part of speech feature, named entity type feature and named entity location feature. d_i is a real-valued vector which is initialized randomly. e_i and d_i can be considered as parameters to tune when training the model, and be optimized constantly until reaching the most effective state. And then we concatenate word embedding and some feature embedding together to generate a complete feature vector $x_i = [e_i, d_i]$, which represents the i-th word in relation mention. Therefore, we finally use matrix $X = [x_1, x_2, \ldots, x_n]$ to represent features of input text as the input of CNN, where n is the length of input text.

3.2.2 Convolutional Layer

Each nerve cell in the convolutional layer only connects with local nerve cells in embedding layer, so we can extract local features and mine correlation information between words. In Fig. 2, there are four kinds of filters with different widths in convolutional layer, which are 2, 3, 4 and 5 respectively. The filters with different widths can extract features at different levels of abstraction from embedding layer. For example, the width of a filter is 3, as denoted by $r = 3$, which represents that each nerve cell in convolutional layer connects with feature vectors x_{i-1}, x_i, x_{i+1} of three adjacent words in embedding layer. In essence, the filter $f_{r=3}$ is weigh matrix when executing a convolution on feature vectors x_{i-1}, x_i, x_{i+1}. Finally, filter f_r extracts local feature vector $C_r = [c_1, c_2, \ldots, c_{n-r+1}]$, where $c_i = g(\sum_{j=0}^{r-1} f_{j+1}^T x_{j+i}^T + b)$.

In above equation, g denotes a non-linear function, and b denotes a bias term. The feature vector C_r is abstract representation of input matrix X and global representation of wider word space.

3.2.3 Pooling Layer

In the structure of CNN, a convolution layer is usually followed by a pooling layer. The basic principle of pooling layer is to perform aggregation calculation of local feature vector C_r that each filter generates, which ensures the invariance of absolute location and further extracts features from convolution layer. The most common pooling functions are max() and $avg()$. max() can get the most important features in C_r, while $avg()$ can get the average features in C_r. We use max() to calculate the maximum pooling score $c_{\max}^{f_r} = \max\{c_1, c_2, \ldots, c_{n-r+1}\}$ and then generate the maximum pooling score vector $P = [c_{\max}^{f_2}, c_{\max}^{f_3}, c_{\max}^{f_4}, c_{\max}^{f_5}]$. Finally, we perform non-linear transform on the maximum pooling score vector P, and then get s_1. s_1 is deeper features generated by CNN, and is also initial state in the deep reinforcement learning framework.

3.3 Generate Transition State by Tree-LSTM

Long short-term memory (LSTM) [11] is a variant of recursive neural network (RNN), and can solve vanishing gradient problem of RNN. We use Tree-LSTM [12] to extract long-distance information in the structure of dependency tree. The structure computes every father node with neighboring child nodes recursively until reaching the root of the tree. So it can represent relation mention in a bottom-up way, capture more comprehensive semantic information and generate transition state in the deep reinforcement learning framework. Now Tree-LSTM has been proposed and applied in many natural language processing tasks, such as sentiment analysis, relation classification, question answering system and so on.

Table 2 The dependency parsing result of a relation mention

teams of nurses and doctors were seen in packed emergency rooms attending to the wounded
nsubjpass(seen-7, teams-1), case(nurses-3, of-2), nmod:of(teams-1, nurses-3), cc(nurses-3, and-4), nmod:of(teams-1, doctors-5), conj:and(nurses-3, doctors-5), auxpass(seen-7, were-6), root(ROOT-0, seen-7), case(rooms-11, in-8), amod(rooms-11, packed-9), compound(rooms-11, emergency-10), nmod:in(seen-7, rooms-11), acl(rooms-11, attending-12), case(wounded-15, to-13), det(wounded-15, the-14), nmod:to(attending-12, wounded-15)

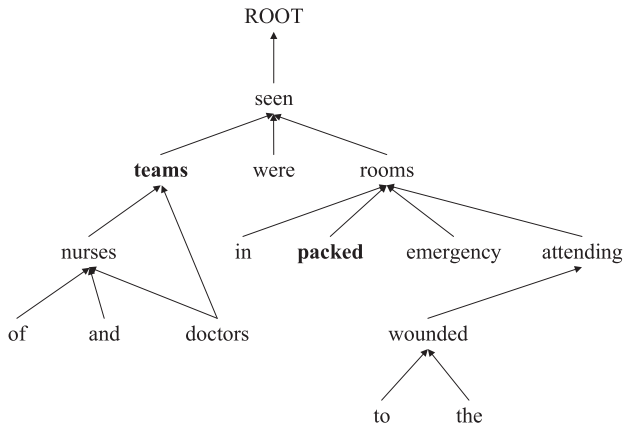

Fig. 3 The dependency tree of a relation mention

3.3.1 Dependency Parsing

We perform dependency parsing on a relation mention, and generate a dependency tree. Table 2 is parsing result, and the tags before brackets represent the relationships between word pairs. For example, nsubjpass represents passive nominal subject. We construct all word pairs into tree structure, and then we can get a dependency tree, as shown in Fig. 3. In Fig. 3, "teams" and "packed" represent the heads of target named entity pair in relation mention.

Traditional Sep-LSTM is good at capturing long-distance information in linear structure, while useful information for relation extraction task mostly exists in the structure of dependency tree. In Fig. 3, it is far from "seen" to "rooms" in linear structure. The two words are separate by many unrelated words or preposition structure, but they are in the direct hyponymy of dependency tree. The characteristic of dependency tree is very useful for mining relations between words, so we use dependency tree as the skeleton of Tree-LSTM, which can capture core dependency relations in target named entity pairs.

3.3.2 Tree-LSTM

Tree-LSTM that we construct is shown in Fig. 4. To extract core dependency relations in target named entity pair, we only use the part between two named entity heads to construct the model. Moreover, for the convenience of implementation, dependency trees should have the same depth and width. We prune the dependency trees or pad them with special characters.

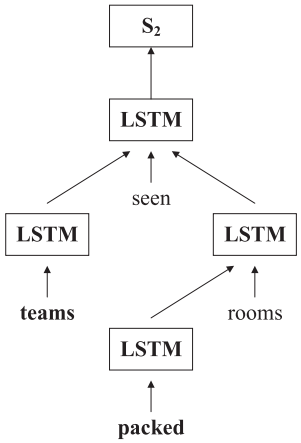

Fig. 4 The structure of Tree-LSTM

Each LSTM component in Tree-LSTM only receives the input from one word. As with embedding layer of CNN, LSTM component uses feature vector $x_j = [e_j, d_j]$ to represent corresponding input word. Each LSTM component contains a memory block. Memory block adds three kinds of gates on the basis of hidden node of RNN: input gate, forget gate and output gate, as denoted by i_j, f_j and o_j. Meanwhile, memory block contains one or more memory cells. Memory cell maintains a cell state c_j, which can retain long-term history information. The data sources of each LSTM component include feature vector x_j of corresponding input word, feedback feature h_{jk} of child node of current node and stored value c_k in the memory cell of child node. Current node uses different forget gate f_{jk} to control different child. The calculation equations of above procedure are shown as follows:

$$i_j = \sigma\left(W^{(i)}x_j + \sum_{k\in C(j)} U^{(i)}h_{jk} + b^{(i)}\right)$$

$$f_{jk} = \sigma\left(W^{(f)}x_j + \sum_{k\in C(j)} U^{(f)}h_{jk} + b^{(f)}\right)$$

$$o_j = \sigma\left(W^{(o)}x_j + \sum_{k\in C(j)} U^{(o)}h_{jk} + b^{(o)}\right)$$

$$u_j = \tanh\left(W^{(u)}x_j + \sum_{k\in C(j)} U^{(u)}h_{jk} + b^{(u)}\right)$$

$$c_j = i_j \odot u_j + \sum_{k\in C(j)} f_{jk} \odot c_k$$

$$h_j = o_j \odot \tanh(c_j)$$

Where W and U both denote weigh matrixes, and b denote bias vectors. All the child nodes of one node share weight matrix U. Their superscripts have the meaning as the name suggests. In addition, σ denotes logistic function, and $\odot$ denotes element-wise multiplication.

In Tree-LSTM, we make computation in a bottom-up way and start from leaf nodes. We compute father node by combining child node with its neighboring nodes recursively until reaching root node, which is common ancestor of all the words in relation mention. The common ancestor is usually a verb. Then we perform non-linear transform on output vector to get s_2. s_2 is deeper features that Tree-LSTM generates finally, and is also transition state in the deep reinforcement learning framework.

3.4 Pre-Training of Two Deep Learning Models

We process all relation mentions to divide into two types. The first type means that relation mention contains predefined relations; the second type means that relation mention does not contain predefined relations. Then we can train parameters of CNN on the two types of relation mentions. In addition, we divide the first type of relation mentions into different relation types. Then we can train parameters of Tree-LSTM on the different types of relation mentions. Finally, we pass deeper feature vector s_1 extracted by CNN and deeper feature vector s_2 extracted by Tree-LSTM to standard neural network separately, and use softmax function to generate conditional probability distribution $P(Y|X)$. $P(Y|X)$ represents conditional probability of the type Y conditioned on input feature X of relation mention, so we can assign the type that have the highest conditional probability to relation mention.

During pre-training two deep learning models, we define a tagging vector T for each relation mention. If a relation mention falls into the i-th type, then the i-th element in the vector T is 1 and other elements are all 0. To train the parameters of models, we use stochastic gradient descent to optimize the cross entropy errors between Y and T. For each relation mention, we define objective function

$$\min_{\theta}\left(-\sum_{j} T_j \log(Y_i)\right)$$

Where θ denoted unknown parameters. The pre-training is to minimize the objective function by stochastic gradient descent.

3.5 Q-Learning Algorithm with Value Function Approximation

In the deep reinforcement learning framework, we use two deep learning models to represent initial state and transition state respectively. The state represented by CNN is denoted as $s_1 = CNN(\mathbf{x}; \boldsymbol{\theta}_1)$, and the state represented by Tree-LSTM is denoted as $s_2 = Tree(\mathbf{x}; \boldsymbol{\theta}_2)$, where $\mathbf{x}$ denotes input text that contains target named entity pair, and $\boldsymbol{\theta}_1$ denotes the parameters of CNN, and $\boldsymbol{\theta}_2$ denotes the parameters of Tree-LSTM.

The action-value function $Q(s, a)$ represents future long-term reward obtained by taking action a at state s. The aim of reinforcement learning is to learn the most optimal action-value function $Q(s, a)$ by maximizing cumulative reward and determine the policy about how to take actions. However, the state space of input text is contiguous, so it

is impracticable to maintain $Q(s,a)$ value for each state-action pair. Therefore, in order to learn action-value function $Q(s,a)$ directly, we use neural network model to approximate $Q(s,a)$. It can be denoted as follow:

$$Q(s,a) = MLP(\phi(\mathbf{x};\boldsymbol{\theta}),a;\boldsymbol{\eta})$$

Where $\phi(\mathbf{x};\boldsymbol{\theta})$ denotes state vector generated by deep learning models, and $\boldsymbol{\eta}$ denotes the parameters of neural network models. After pre-training the deep learning models, the parameters $\boldsymbol{\theta}$ will be known.

We train the parameters $\boldsymbol{\eta}$ on state-action pairs, and make value function approximation. To make value function approximate real value function Q^{π} as closely as possible, we measure the degree of approximation with the least squares error:

$$E_{\boldsymbol{\eta}} = E_{s;a\sim\pi}[(Q^{\pi}(s,a) - Q_{\boldsymbol{\eta}}(s,a))^2]$$

Where $E_{s;a\sim\pi}$ denotes the expectation on state-action pairs sampled by the policy π.

To minimize the error, we use gradient descent method with RMSprop [13] to take negative derivative of the error:

$$-\frac{\partial E_{\boldsymbol{\eta}}}{\partial \boldsymbol{\eta}} = E_{s;a\sim\pi}\left[2(Q^{\pi}(s,a) - Q_{\boldsymbol{\eta}}(s,a))\frac{\partial Q_{\boldsymbol{\eta}}(s,a)}{\partial \boldsymbol{\eta}}\right]$$

The Update rule of the parameters is:

$$\boldsymbol{\eta} = \boldsymbol{\eta} + \alpha(Q^{\pi}(s,a) - Q_{\boldsymbol{\eta}}(s,a))\frac{\partial Q_{\boldsymbol{\eta}}(s,a)}{\partial \boldsymbol{\eta}}$$

Then we use Q-Learning algorithm and replace real value function Q^{π} with estimated value function. It is denoted as follow:

$$Q^{\pi}(s,a) = \frac{1}{T}r + \frac{T-1}{T}Q^{\pi}(s',a')$$

$$\boldsymbol{\eta} = \boldsymbol{\eta} + \alpha\left(\frac{1}{T}r + \frac{T-1}{T}Q_{\boldsymbol{\eta}}(s',a') - Q_{\boldsymbol{\eta}}(s,a)\right)\frac{\partial Q_{\boldsymbol{\eta}}(s,a)}{\partial \boldsymbol{\eta}}$$

Where (s',a') is the state-action pair of next time.

We use Q-Learning algorithm with value function approximation to learn control policy π for relation extraction task. The detailed flow of the algorithm is shown in Algorithm 3.1.

After learning the value function $Q(s,a)$, we will take the action that has the highest Q-value $Q_{\boldsymbol{\eta}}(s,a'')$ to maximize expectancy reward. During each epoch, the parameters will be undated to reduce the error between estimated value $Q_{\boldsymbol{\eta}}(s,a)$ and expected value $Q^{\pi}(s,a)$.

Algorithm 3.1 Relation Extraction with Q-Learning Algorithm with Value Function Approximation

Input:
Environment E;
Action Space A;
Input Text and Relation Type $<\mathbf{x},y>$;
Update Step α.

Output:
Policy π.

Procedure:
$\boldsymbol{\eta} = \mathbf{0}$;
Pre-training CNN and Tree-LSTM;
for each $<\mathbf{x},y>$ **do**
extract deeper features from $\mathbf{x}$ by CNN and Tree-LSTM respectively, and generate s_1 and s_2.
end for
for $epoch = 1,2,\cdots$ **do**
for each $<\mathbf{x},y>$ **do**
for $t = 1,2$ **do**
r, s' = the reward and transition state generated by taking action $\pi(s)$ in E;
$a' = \pi(s')$;
then perform gradient descent step;
$E_{\boldsymbol{\eta}} = E_{s;a\sim\pi}[(Q^{\pi}(s,a) - Q_{\boldsymbol{\eta}}(s,a))^2]$;
$\boldsymbol{\eta} = \boldsymbol{\eta} +$
$\alpha(\frac{1}{T}r + \frac{T-1}{T}Q_{\boldsymbol{\eta}}(s',a') - Q_{\boldsymbol{\eta}}(s,a))\frac{\partial Q_{\boldsymbol{\eta}}(s,a)}{\partial \boldsymbol{\eta}}$;
$\pi(s) = \arg\max_{a''} Q_{\boldsymbol{\eta}}(s,a'')$;
$s = s'$, $a = a'$.
end for
end for
end for

4. Experiments

4.1 Experiment Environment

We do experiments with mobile workstation HP ZBook 17, of which CPU is Intel(R) Core(TM) i7-4900MQ CPU @ 2.80GHz and graphics card is NVDIA Quadro K5100M. We set up deep learning programming environment Python2.7+Theano+Cuda7.5 under Ubuntu operating system to construct CNN and Tree-LSTM. On this basis, we realize the deep reinforcement learning framework with Python directly.

4.2 DataSet

We evaluate our method on ACE 2005 multilingual training corpus. We obtain ACE 2005 from Linguistic Data Consortium (LDC), of which LDC catalog number is LDC2006T06 and isbn is 1-58563-376-3. Experiments are performed on the English data of ACE 2005, of which data sources include 20% Newswire (NW), 20% Broadcast News (BN), 15% Broadcast Conversation (BC), 15% Weblog (WL), 15% Usenet Newsgroups/Discussion Forum (UN) and 15% Conversational Telephone Speech (CTS). There are seven types of named entities in the corpus, which are anno-

Table 3 Impact of different filter strategies

Filter strategy	F (%)
3	79.10
4	78.77
3 and 4	79.76
2, 3 and 4	80.42
2, 3, 4 and 5	80.75

tated by "PER", "ORG", "LOC", "GPE", "FAC", "VEH" and "WEA". Between those named entities, there are seven relation types, which are annotated by "PHYS", "PART-WHOLE", "PER-SOC", "ORG-AFF", "ART", "GEN-AFF" and "METONYMY".

To reduce noise and compare with predecessors' experiment results, we process the corpus in the same way with predecessors [7]. We remove all the relation mentions whose distances between two named entity heads are greater than 15. Then we transform 80% of the corpus as training set and the remaining 20% as testing set. If there is predefined relation between target named entity pairs in a relation mention, then it is a positive relation mention; if not, it is a negative relation mention. After truncation, there are 8365 positive relation mentions and 79147 negative relation mentions in the corpus. It is consistent with the view in this paper that it is an unbalanced corpus for relation extraction task.

4.3 Pre-Training

We use the publicly available word embedding Glove [14] whose size is 300, and initialize word embedding table with Glove. In addition, we set the size of speech feature embedding and named entity type feature embedding to 50, and set the size of named entity location feature embedding to 5. Those feature embedding tables are initialized randomly, in which the elements range in value from −0.25 through 0.25. During pre-training deep learning models, all the elements in feature embedding tables are viewed as the parameters in the networks, and updated constantly until reaching the most effective state. The sizes of hidden nodes are all set to 100. We perform dropout [15] on hidden nodes, and the rate of dropout is 0.5. During each iteration, the corpus is divided into many batches, and we can only process one batch at a time. A batch contains many relation mentions, and the size of batch is 30. Meanwhile, the constraint of max-norm regularization is equal to 3, and we perform non-linear transform with function ReLU in the networks. Next, we will use cross-validation to determine the hyper-parameters of the models.

We divide all the relation mentions into two types, and pre-train the parameters of CNN on the two types of relation mentions by batch gradient descent (BGD). Our task is binary classification of relation mentions. The experiment evaluates different filter strategies, and the result is shown in Table 3. The result shows that multi-filter strategies are much better than single-filter strategies, and multi-filter strategy employing the filters 2, 3, 4 and 5 simultaneously can get the best F-value. The experiment also evaluates the impact of the size of convolutional layer on the experiment result, as shown in Fig. 5. The result shows that CNN can perform best when the size of convolutional layer is 150. Too large or too small size of convolutional layer may result in a disturbing effect on the experiment result. If the size of convolutional layer is too large, the network will be more complex, which may lead to over-fitting; if the size of convolutional layer is too small, it will be not enough to learn proper features. Therefore, in the rest of experiments we will employ the filters 2, 3, 4 and 5 simultaneously, and set the size of convolutional layer to 150.

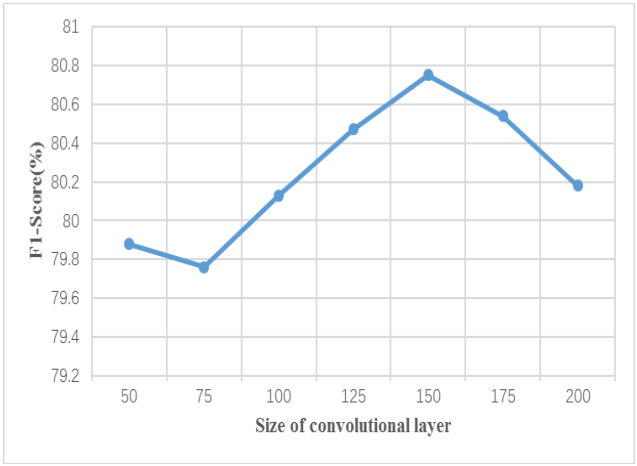

Fig. 5 Impact of different sizes of convolutional layer

Table 4 Impact of different syntactic parsing strategies

Syntactic Parsing Strategy	F (%)
Constituency Parser	82.2
Dependency Parser	82.9

We select all the relation mentions that contain different types of predefined relations, and train the parameters of Tree-LSTM on the different types of relation mentions. Our task is fine-grained classification over seven relation types. We use Stanford PCFG Parser [16] to produce constituency parses of each relation mention. The generated constituency tree is the skeleton to construct Constituency Tree-LSTM. In addition, we use Stanford Neural Network Dependency Parser [17] to produce dependency parses of each relation mention. The generated dependency tree is the skeleton to construct Dependency Tree-LSTM. The experiment evaluates the impact of different syntactic parsing strategies on the experiment result, as shown in Table 4. The result shows that Tree-LSTM constructed by dependency tree performs better than that constructed by constituency tree for fine-grained classification. Therefore, in the rest of experiments we will use dependency tree to construct Tree-LSTM.

4.4 Training the Whole Framework

We use gradient descent method with RMSprop [13] to train the entire deep reinforcement learning framework. Learning rate is 0.0005, and discount rate is 0.95. The experiment

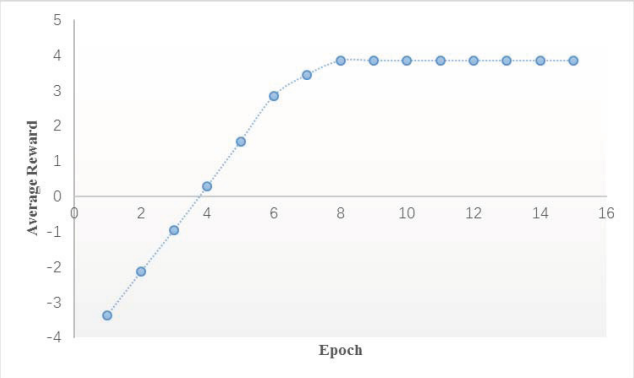

Fig. 6 Impact of different epoch numbers on average reward

Table 5 Performance of different methods for relation extraction task

Method	P (%)	R (%)	F (%)
CNN	68.35	53.33	59.91
Tree-LSTM	69.21	54.49	60.97
Deep Reinforcement Learning	68.09	65.83	66.94
Deep Reinforcement Learning+	69.25	66.41	67.80

evaluates the impact of different epoch numbers on average reward, and the result is shown in Fig. 6. At the beginning of training, the reward is so limited, because the actions have close to a random chance. The reward begins to increase and seems to be stabilizing as epoch number is increasing, which means the agent has learnt how to perform relation extraction task to earn positive reward. After the eighth epoch, the reward will remain basicly stable. Therefore, in the rest of experiments we will compute the deep reinforcement learning framework with eight epochs.

Besides, we use CNN and Tree-LSTM to perform relation extraction task directly, and compare their results with the result of deep reinforcement learning based method. The experiment results are shown in Table 5. This is a 5-fold cross validation experiment and all the comparisons are significant at 0.05 level. The results show that Tree-LSTM performs better than CNN for relation extraction task, and the performance of deep reinforcement learning based method is much better than that of using either deep learning model separately, especially in recall score. If we tune the parameters of two pre-trained deep learning models while training the entire framework, the performance will be further improved. The best performance corresponds to "Deep Reinforcement Learning+" in Table 5, and its precision, recall and F-value are 69.25%, 66.41% and 67.80% respectively. "Deep Reinforcement Learning+" makes training time of the whole framework increase significantly, but relation extraction task is not time sensitive. Therefore, in the rest of experiments we will use the method "Deep Reinforcement Learning+" to train the entire framework.

4.5 Comparing to the State-of-the-Art

At present, deep learning techniques achieve the best performance in relation extraction task. Nguyen et al. [18] proposed to combine the traditional feature-based method, the convolutional and recurrent neural networks to simultaneously benefit from their advantages. In addition, Nguyen et al. used different methods to construct and assemble CNNs and RNNs: VOTE-BIDIRECT, STACK-FORWARD and VOTE-BACKWARD, where the performance of STACK-FORWARD is best. STACK-FORWARD is to use the stacking method to combine CNN and forward RNN. We perform a paired t-test between F-value of our method and the baseline methods. Our proposed method significantly outperforms Nguyen's method at 0.05 level. The detail results are shown in Table 6. Compared with STACK-FORWARD our method raises recall and F-value to a certain extent, and the improvements are 0.12% and 0.03% respectively. Our method not only can absorb the advantages of deep learning methods, but also can tackle the problem of unbalanced corpus, meanwhile the performance has improved to a certain extent. The reason of low precision is that we do not use external knowledge base and the external parsing module that we use may result in some errors.

Table 6 Comparison with the State-of-the-art

Method	P (%)	R (%)	F (%)
VOTE-BIDIRECT	71.30	62.40	66.55
STACK-FORWARD	69.32	66.29	67.77
VOTE-BACKWARD	70.79	64.02	67.23
Our Method	69.25	66.41	67.80

Table 7 Performance breakdown per relation for "Deep Reinforcement Learning+" and "Deep Learning" on the testing set

Relation	F (%)		
	Deep Learning	Deep Reinforcement Learning	Change
PHYS	54.75	56.83	2.08
PART-WHOLE	68.99	78.22	9.23
PER-SOC	60.76	68.29	7.53
ORG-AFF	76.48	82.86	6.38
ART	48.64	58.42	9.78
GEN-AFF	55.10	60.18	5.08
METONYMY	62.07	70.80	8.73
All	60.97	67.80	6.83

4.6 Analysis

In this subsection, we will analyze the role of deep reinforcement learning in relation extraction. By comparing extracted relations by "Deep Reinforcement Learning+" and "Deep Learning" (Tree-LSTM), we find the influence of our method varies from different relation types. Table 7 shows the F-value of each relation type by the above two methods on the testing set.

We can see that our method generally has a positive impact on all the relation types. With only using the deep learning method, the "ART" and "GEN-AFF" relations are easily confused, but our method can better distinguish these two relation types. However, the effect on the "PHYS" relations is not obvious. The main reason is that errors occur during relation classification stage rather than relation detection stage.

Our method preforms in two stages: relation detection and relation classification. Deep learning methods mainly

focus on the second stage, and do not specialize in the first stage. However, our method is superior in dealing with the first stage, and succeeds the advantages of deep learning method for relation classification stage. The experiment results show that deep reinforcement learning can improve the performance of relation extraction. It is especially salient for the "ART" and "GEN-AFF" relations. Since these relations are few in number and the relation detection stage is particularly important.

Compared with previous methods, our proposed method can detect relation mentions reliably. For example, there are 4 entities "You", "you", "wannabe" and "herself" in the following sentence: "You can see the video here if you can stand to watch a young no-talent wannabe continue to humiliate herself in public.". In fact, there are no relations in the sentence. Previous methods mistakenly extract some relations in the sentence, while our method can make good judgment.

5. Conclusions

In this paper, we consider relation extraction task as a two-step decision-making game, and design a deep reinforcement learning framework to combine two deep learning models perfectly. To be specific, we use CNN and Tree-LSTM to model relation mentions separately and calculate internal states in the framework. More importantly, we design penalty function to tackle the problem of unbalanced corpus. We perform a series of experiments in ACE2005 corpus, and the results show that compared with predecessors' research our method raises recall and F-value to a certain extent. How to define more reasonable penalty function and increase training speed of the entire framework will become a study emphasis in further research.

Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive comments.

References

[1] M.T. Pazienza, "Information extraction: A multidisciplinary approach to an emerging information technology," Lecture Notes in Computer Science, vol.1299, Springer, Heidelberg, 1997.
[2] N. Kambhatla, "Combining lexical, syntactic and semantic features with maximum entropy models for extracting relations," Proc. ACL 2004 on Interactive poster and demonstration sessions, pp.22, 2004.
[3] A. Sun, R. Grishman, and S. Sekine, "Semi-supervised relation extraction with large-scale word clustering," Proc. 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, Association for Computational Linguistics, Portland, pp.521–529, Oregon, USA, DBLP, June 2011.
[4] A. Culotta and J. Sorensen, "Dependency tree kernels for relation extraction," Meeting of the Association for Computational Linguistics, Barcelona, Spain, DBLP, pp.423–429, July 2004.
[5] R.C. Bunescu and R.J. Mooney, "A shortest path dependency kernel for relation extraction," Conference on Human Language Technology and Empirical Methods in Natural Language Processing, Association for Computational Linguistics, pp.724–731, 2005.
[6] T.-V.T. Nguyen, A. Moschitti, and G. Riccardi, "Convolution kernels on constituent, dependency and sequential structures for relation extraction," Conference on Empirical Methods in Natural Language Processing, pp.1378–1387, 2009.
[7] T.H. Nguyen and R. Grishman, "Relation extraction: Perspective from convolutional neural networks," Proc. NAACL-HLT, pp.39–48, 2015.
[8] M. Miwa and M. Bansal, "End-to-end relation extraction using LSTMs on sequences and tree Structures," Proc. 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp.1105–1116, 2016.
[9] H. Guo, "Generating text with deep reinforcement learning," Computer Science, vol.40, no.4, pp.1–5, 2015.
[10] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf, "Deep reinforcement learning with a natural language action space," Meeting of the Association for Computational Linguistics, pp.1621–1630, 2016.
[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol.9, no.8, pp.1735–1780, 1997.
[12] K.S. Tai, R. Socher, and C.D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," Proc. 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), vol.5, no.1, pp.1556–1566, 2015.
[13] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," COURSERA: Neural Networks for Machine Learning, vol.4, no.2, 2012.
[14] J. Pennington, R. Socher, and C.D. Manning, "Glove: Global Vectors for Word Representation," Conference on Empirical Methods in Natural Language Processing, no.14, pp.1532–1543, 2014.
[15] G.E. Hinton, N. Srivastava, A. Krizhevsky, et al, "Improving neural networks by preventing co-adaptation of feature detectors," Computer Science, vol.3, no.4, pp.212–223, 2012.
[16] D. Klein and C.D. Manning, "Accurate unlexicalized parsing," Proc. 41st Annual Meeting on Association for Computational Linguistics-Volume 1, Association for Computational Linguistics, pp.423–430, 2003.
[17] D. Chen and C.D. Manning, "A fast and accurate dependency parser using neural networks," Conference on Empirical Methods in Natural Language Processing, pp.740–750, 2014.
[18] T.H. Nguyen and R. Grishman, "Combining neural networks and log-linear models to improve relation extraction," Computer Science, 2015.

Hongjun Zhang is male, born in 1963, professor, PhD supervisor in PLA University of Science and Technology. His main research fields are data engineering, combat effectiveness evaluation and combat simulation. E-mail: jsnjzhj@263.net

Yuntian Feng is male, born in 1990, doctor candidate in PLA University of Science and Technology. His main research fields are deep learning and natural language processing. E-mail: fengyuntian2009@live.cn

Wenning Hao is male, born in 1971, professor, master supervisor in PLA University of Science and Technology. His main research fields are massive high-dimensional data reduction and combat effectiveness evaluation. E-mail: hwnbox@163.com

Gang Chen is male, born in 1974, professor in PLA University of Science and Technology. His main research fields are data engineering, combat effectiveness evaluation and combat simulation. E-mail: gchen1027@sina.com

Dawei Jin is male, born in 1979, associate professor in PLA University of Science and Technology. His main research fields are data engineering, combat effectiveness evaluation and combat simulation. E-mail: dave_nj@163.com