A Shift-Reduce Dependency Parser Based on Reinforcement Learning

Wei-dong WEN1, Ni Li2, Zhen LI3, Ao-xue DONG3, Qiang XU3 and Chen CHEN4

1School of Computer Science, Wuhan University, Wuhan, China
2China Electric Power Research Institute (State Key Laboratory of Power Grid Environmental Protection), Wuhan, China
3State Grid Jixi Electric Power Company, Jixi, China
4Data Processing Center of China Construction Bank, Wuhan, China

*Corresponding author

Keywords: Dependency parsing, Natural language processing, Reinforcement learning, Shift-Reduce.

Abstract. Dependency parsing is a task of extracting the relationships between words in a sentence. Researchers have achieved great success in transition-based and graph-based methods recently. However, there are problems of error propagation and high time complexity. This paper proposes dependency parser based on reinforcement learning to improve transition-based parser. We regard the actions in transition-based method as RL agent’s actions. Meanwhile, we introduce a BACK action, so that the agent can get back to the previous state after entering the error state. We use the Universal Dependencies dataset to make experiments on different language. Experiments show that this method effectively solves the problem of error propagation of transition-based method and improve the accuracy significantly.

Introduction

The aim of dependency parsing is to analysis the language structure by the relation between words. The relationship between the components can be explicitly expressed as a syntactic diagram visually, as shown in Fig. 1. Dependency parsing is important in the field of natural language processing, because it can help other tasks, such as coreference resolution [1], semantic analysis [2], machine translation [3], information extraction [4] and so on.

![Figure 1. An example of syntactic diagram.](image)

The approaches can be roughly grouped into two major categories: graph-based method and transition-based method. The graph-based method assumes that any two elements in a sentence have a relation with a certain probability. It trains a function to evaluate the scores of the subtrees and obtains the optimal spanning tree in the dynamic programming process. The time complexity is usually O(n^3). The transition-based method, which is also called shift-reduce parsing, reads sentences from left to right and sets a stack and a buffer queue as working area. The operators will take actions to change the stack and the buffer according to the feature of stack and buffer. Complete dependencies will be explored after several actions. The time complexity of this method is about O(n), but it always suffers from the error propagation because any error action will make the rest relation not reliable.
Deep Reinforcement Learning is one of the most concerned areas in the field of artificial intelligence in recent years. It always tries and fails just like human and animals. It focuses on the interaction with the environment, and optimizes the decision using the feedback signal given by the environment. Reinforcement learning has prospects in solving complex optimization decision problems since it does not need supervised labels. The process of reinforcement learning is a process of testing and evaluation. The agent selects and executes an action in the environment. After the environment accepts the action, it changes to a reward signal and feeds it back to the Agent. The Agent then selects the follow-up action according to the reward signal. The basic framework for reinforcement learning is shown in Figure 2.

Figure 2. The basic framework for reinforcement learning.

Reinforcement learning has achieved excellent results in many fields, but the main applications are still concentrated in the fields of games, robots, and autonomous driving. In order to solve the error propagation problem of shift-reduce method, this paper makes an improvement, which transforms the dependency parsing task into a sequence of selections of actions. Besides, this paper introduces a “BACK” mechanism, which help the agent get back to the previous step, thus improving the accuracy of dependency parsing.

Related Work

The foundations for dependency grammars were laid down by Tesniere [5]. This increase in attention is partially due to community efforts such as the 2006 and 2007 CoNLL Shared Tasks [6], but also since the dependency grammar framework easily lends itself to languages with free word order. As a result, annotated resources are readily available for a diverse, large set of languages.

Dependency parsing algorithms generally fall within two major classes; graph-based and transition-based. Graph-based algorithms consider the complete set of possible dependency graphs and transition-based algorithms iteratively construct the dependency graph by processing the sentence sequentially. While they both have strengths and weaknesses, in this work we will only consider transition-based algorithms.

Chen et al. [7] made the first successful attempt at incorporating deep learning into a dependency parser. Their approach involves using a feedforward network classifier to make parsing actions; at each step, the network takes as input the concatenation of the word, tag, and label embedding for words in critical positions and puts them through a multilayer perceptron (MLP) that assigns a probability to each action the parser can take. At each step, the parser takes the most probable action, updating the stack and buffer accordingly.

The principal limitation of the Chen et al. [7] parser is that it doesn’t have access to the entire sentence when making each parsing action, and a number of other approaches have modified or augmented the Chen et al. [7] parser to address this. Dyer et al. [8] and Ballesteros et al. [9] replaced the input to the feedforward network with the output of LSTMs over the stack, buffer, and previous actions.

Dyer et al. [10] proposed a technique for learning representations of parser states in transition-based dependency parsers, whose primary innovation is a new control structure for sequence-to-sequence neural networks. Stenetorp [11] introduced a compositional vector framework for transition-based dependency parsing algorithms that can capture semantic relations between phrases. It applied Recursive Neural Network (RNN) models to transition-based dependency parsing and then demonstrate how it can be applied to three previously proposed transition-based algorithms.
Weiss et al. [12] and Andor et al. [13] achieved state of the art performance by instead augmenting it with a beam search and a CRF loss so that the model can avoid committing to partial parses that later evidence might reveal to be incorrect.

**RL Model**

There are five important components in reinforcement learning. This section merges the parts of dependency parsing into reinforcement learning, and describes each component in the process of training.

**Environment**

Environment consists of a stack S and a queue B. It stores word nodes and generates corresponding operations according to the instructions made by the Action. Initially, the stack S contains only one virtual root node. B indicates the queue Buffer that stores the unprocessed node. Initially, the entire sentence is stored in B, and B\(_i\) represents the i-th node in B.

The functions of Environment can be divided into two categories, shift and reduce. Shift refers to selecting the top element from the unprocessed queue B to insert into the stack S and waiting to be processed. Reduce is to determine whether there is dependency relation between the top two elements in the stack S. If it exists, the relationship will be output and the child nodes will be deleted. After several shift and reduced executed, the sentence can be converted from the initial state to the final state, where all dependencies are output and S = [root], B = [].

**Actions**

Action sends instructions to the environment. There are four actions in this model. LEFT: S\(_i-1\) is a leaf node, and is one child node of S\(_i\). Then the relationship of S\(_i-1\) \(\rightarrow\) S\(_i\) will be recorded and the leaf node S\(_i-1\) will be removed from S. RIGHT: S\(_i+1\) is a leaf node and S\(_i\) is its parent node. Then the relationship of S\(_i\) \(\rightarrow\) S\(_i+1\) will be recorded and the leaf node S\(_i+1\) will be removed from S. SHIFT: There is no dependency and no action will be taken. BACK: Roll back to the state of the previous step. This article gives a sample sentence, and shows the steps to execute the action gradually in Figure 3.

![Figure 3. Steps for shift-reduce.](image)

Figure 3 shows an absolutely correct action path, but in fact it is inevitable that where is always a wrong action be generated, resulting in error propagation in the subsequent environment. For example, it is an error situation when the first two elements have no dependency and the buffer queue B is empty. Therefore, the paper sets the fourth action “Back”, which can roll the environment back to the previous step, ensuring that at least one action can be legally executed under any circumstances.

**State**

State is a series of feature vectors extracted from Stack and Buffer in the environment. Words and part-of-speech are stored in Stack and Buffer. In this section, word vectors are represented as EMB\(_w\) \(\in R^{N_w \times d}\), and part-of-speech vectors are represented as EMB\(_p\) \(\in R^{N_p \times d}\). We choose the first three items in stack and buffer as state features. If there are less than three items, Null is added, as shown in Eq. (1) (2) (3)

\[
X_w = [S_1.w, S_2.w, S_3.w, B_1.w, B_2.w, B_3.w]
\]
\[
X_p = [S_1.p, S_2.p, S_3.p, B_1.p, B_2.p, B_3.p]
\]  
(2)

\[
X = [X_w, X_p]
\]
(3)

**Reward**

The rewards, which can be positive, negative, or zero, is produced by Environment when executing actions. This paper does not supervise train data by labels, rather rewarding the final results of the action execution. For any one of the action sequences \(A_i\), if the final complete relation was generated after actions were executed, the action will get reward of +1. If \(A_i\) is not executable, the action will get reward of -1 because there must be at least one error action in the action sequence. For the other situation, the action will get reward of 0 because we can’t know whether \(A_i\) is correct.

**Agent**

The reinforcement learning agent is composed of multiple layers of neural networks. The input is the feature vector of the current state, and the output is the probability of each action. In this paper, we establish two fully-connected neural networks with the same structure and independent parameters. \(X\) represents the input feature vectors, followed by the full-connected layers and the sigmoid activation function, as shown in Eq. (4)(5).

\[
h = \text{sigmoid}(W_2 \text{sigmoid}(W_1 X + b_1) + b_2)
\]
(4)

\[
q = \text{softmax}(W_q h)
\]
(5)

**Train**

**Overview**

Environment generates the initial state based on the word vectors and part-of-speech vectors in Stack and Buffer. State vectors will be put into the first neural network, and the probability distribution of the four actions will be output after the process of hidden layers. However, the accuracy of the neural network is not guaranteed at the beginning of training. In order to find more possible paths, the agent sets a small probability \(\epsilon\) and randomly generates a probability distribution of four actions. The Stack and Buffer in the Environment will be updated by the probability distribution of the four actions and the output of the neural network. If the final state is reached, which means \(S = [\text{root}]\) and \(B = []\), the reward will be +1. If an illegal action is performed, the reward will be -1. The reward for other cases is 0 and the process will continue till the reward is 1 or -1.

**Experience Replay**

It makes the model effect discount if the data is dropped after trained once. The RL agent introduces Experience Replay mechanism, which stores the train data of the agent at each time as a memory sequence. The environment will generate the updated state’ and reward, and \(e_i = (\text{state}, \text{action}, \text{reward}, \text{state}')\) to a limit-length queue reply buffer. In the future of training, a batch of data can be randomly extracted from the experience queue for training. The Experience Replay mechanism increases the efficiency of data usage by resampling historical data.

**Target Network**

The data generated by the environment is unstable. In order to eliminate the fluctuation caused by the data difference, this paper introduces the target network, which has the same structure and different parameters with the first neural network. During the training process, a batch of data is randomly extracted from the reply buffer, and the loss value is calculated from the output of the target network, and then the parameters of the first network is updated by gradient descent. The parameters of the first network are copied to the target network every several epochs.

The definition of the loss function \(L_i(\theta_i)\) for epoch \(i\) is expressed as Eq.(6):
\[ L_i(\theta_i) = E_{s,a \sim \rho(\cdot)} \left[ (y_i - Q(s,a; \theta_i))^2 \right] \]  

(6)

In Eq.(6), \( \rho(\cdot) \) indicates the probability distribution of actions in a given environment. \( y_i \) represents the value of the Q function in i-th epoch, as shown in Eq.(7).

\[ y_i = E_{s' \sim \epsilon} \left[ r + \gamma \max Q_2(s', a'; \theta_{i-1}) | s, a \right] \]  

(7)

In Eq.(7), \( r \) is the reward value of the environment, and \( \gamma \) is the discount factor. The goal of learning depends on the weight of the network. The formula for updating the network weight is shown in Eq.(8)

\[ \Delta \theta_i L_i(\theta_i) = E_{s,a \sim \rho(\cdot); s' \sim \epsilon} \left[ (r + r \max Q_2(s', a'; \theta_{i-1}) - Q(s,a; \theta_i)) \Delta Q(s,a; \theta_i) \right] \]  

(8)

The detailed structure is shown in Figure 4 and Algorithm 1.

---

**Figure 4. Structure of RL.**

---

**Algorithm 1 RL Dependency Parsing Algorithm**

1. Initialize reply buffer queue
2. Initialize \( \theta \) for \( Q_1 \), \( \theta_2 \) for \( Q_2 \)

for \( i = 1, T \)

1. Random choose a sentence
2. Initialize state

for \( i = 1, T \)

1. Generate an action by \( Q \) with probability \( 1 - \epsilon \),
2. OR generate an action randomly with probability \( \epsilon \)
3. \( a \leftarrow \max(ACTIONS) \)
4. Execute a
5. reply buffer \( \leftarrow (state, action, reward, state') \)
6. Random choose a batch from reply buffer

if \( state' \) is final state then

1. \( y_t = r_{t+1} \)
2. else
3. \( y_t = r_{t+1} + \gamma \max(Q_2(s', a')) \)
4. end if
5. optimize \( (y_t - Q(s,a))^2 \)
6. \( \theta \leftarrow \theta_2 \)
7. end if
8. end for
9. end for
Experiment

Overview

This paper uses English and Chinese corpus provided by Universal Dependency as the data set. Besides, this paper uses the external large corpus to train the word2vec vectors and the part of the speech vectors. The whole model is implemented by pytorch. The input to the multi-layer network is the vector features represented by state, and the output is the value of each action. This experiment uses the Adam optimizer to initialize parameters in a random method with a learning rate of 0.001. In some researches, the word vectors are randomly initialized and trained as model parameters, while in this paper, we compare the random initialization method with the pre-training method.

Evaluation

There are two general evaluation methods for dependency parsing, which are unlabeled attachment score (UAS) and labeled attachment score (LAS). The UAS only evaluates whether there is a dependency between two elements, and the latter considers the classification of relationship. The RL model in this paper only discriminates whether there is a relationship between two adjacent elements, so we use UAS as evaluation method.

Results

In order to demonstrate the performance of the method, we compare out RL model with some researches’ on the UAS scores, as shown in Table 1.

<table>
<thead>
<tr>
<th>Method</th>
<th>Word vectors</th>
<th>English Dataset</th>
<th>Chinese Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stenetorp, P. et al. [5]</td>
<td>pretrain</td>
<td>92.2</td>
<td>91.8</td>
</tr>
<tr>
<td>Stenetorp, P. et al. [5]</td>
<td>random</td>
<td>91.9</td>
<td>90.7</td>
</tr>
<tr>
<td>Ballesteros, M. et al.[6]</td>
<td>pretrain</td>
<td>93.8</td>
<td>91.5</td>
</tr>
<tr>
<td>Ballesteros, M. et al.[6]</td>
<td>random</td>
<td>93.1</td>
<td>91.0</td>
</tr>
<tr>
<td>Dozat, T. et al. [7]</td>
<td>pretrain</td>
<td>91.8</td>
<td>88.6</td>
</tr>
<tr>
<td>Dozat, T. et al. [7]</td>
<td>random</td>
<td>92.6</td>
<td>88.9</td>
</tr>
<tr>
<td>Dyer, C. et al.[8]</td>
<td>pretrain</td>
<td>93.1</td>
<td>91.8</td>
</tr>
<tr>
<td>Dyer, C. et al.[8]</td>
<td>random</td>
<td>91.7</td>
<td>90.3</td>
</tr>
<tr>
<td>OUR’s</td>
<td>pretrain</td>
<td>94.1</td>
<td>93.6</td>
</tr>
<tr>
<td>OUR’s</td>
<td>random</td>
<td>93.7</td>
<td>93.0</td>
</tr>
</tbody>
</table>

According to the results above, the method of reinforcement learning + pre-training word vector achieves 94.1% UAS accuracy in English dataset. The Chinese accuracy is slightly lower than the accuracy of English dataset. In addition, the accuracy of using randomized word vectors is generally lower than the accuracy of using pre-trained word vectors.

Action Analysis

Since there is a Back action, the length of generated action sequence may be longer than the sentence. The table 2 shows the number of actions to reach the final state for different sentences. Obviously, the required number of actions is higher than the shift-reduce based parser, and the average number increases as the length of the sentence increases, which indicates that the longer sentence is easier to get invoked into the loop of Back action and other action.
Table 2. Number of Actions.

<table>
<thead>
<tr>
<th>Length of sentence</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of actions</td>
<td>30</td>
<td>68</td>
<td>123</td>
<td>228</td>
<td>410</td>
</tr>
<tr>
<td>Average number of actions</td>
<td>3</td>
<td>3.4</td>
<td>4.1</td>
<td>5.7</td>
<td>8.2</td>
</tr>
</tbody>
</table>

At the beginning of the training, the agent always walks out of the wrong path, so it goes into a loop and then executes the state of Back so that it can only generate only few relations. As the steps of training increases, the agent can move forward. As shown in Figure 5, the abscissa represents the number of steps of training, and the ordinate represents the proportion of the number of relationships generated in a maximum of 300 actions to the number of all relationships in the sentence. Obviously, this ratio is rising to 1 gradually.

Figure 5. Ratio of completed relationships.

Conclusion

This paper proposes a new dependency parsing method based on the shift-reduce method. It combines reinforcement learning and regards the three actions in shift-reduce as the agent actions. Besides, a BACK action is introduced to get back to the previous state, reducing the error propagation problems.

Experiments show that the method described above can get higher accuracy than that in shift-reduce based method in dependency parsing task. However, we find it always costs too much time because the BACK action can get it into a loop in some parts of the sentence. The reason we analysis is that a single reward function is not powerful. In the next step, we will discuss the effect of different reward function on the reinforcement learning effect.

Acknowledgements

The paper was supported by the Science and Technology Project of State Grid Corporation of China (Research on Public Opinion and Risk Tracking Countermeasure in Electromagnetic Environment of Power Grid, No. GY71-18-009).

References


