A Hybrid Dependency Parsing Model Enhanced by Shared Feature

Yunzhe Yuan
School of Information Science and Technology
ShanghaiTech University
Shanghai, China
yuanyzh@shanghaitech.edu.cn

Zhiqiang Xie *
School of Information Science and Technology
ShanghaiTech University
Shanghai, China
xiezhq@shanghaitech.edu.cn

Abstract

Two major data-driven dependency approaches: graph-based and transition-based, adopt very different views of the parsing problem, each view having its own strengths and limitations. And the parsing accuracy of them can differ much for different corpus. In this project, we develop a neural network based hybrid model, trying to enhance the model-invariant features by encoding them in a Bidirectional LSTM (Bi-LSTM) layer. Besides, we integrate an online re-ranking classifier in the hybrid model to explore the potentiality of the hybrid idea further. Moreover, we test our model in a special tiny corpus, the result also suggests there may be a way to go through the learning task in small scale of datasets.

1. Introduction

Dependency Parsing is the task of recognizing a sentence and assigning a syntactic structure to it. The most widely used syntactic structure is the parse tree which can be generated using some parsing algorithms. These parse trees are useful in various applications like grammar checking or more importantly it plays a critical role in the semantic analysis stage. A parse tree for a sentence represents each word and its syntactic dependents through labeled directed arcs, as shown in figure 1.

Main data-driven approaches nowadays to dependency parsing can be briefly categorized into graph-based[10] and transition-based parsers[6]. Given an input sentence, a graph-based algorithm finds the highest scoring parse tree from all candidates, typically by factoring the graphs into their component arcs and perform parsing by searching for the tree with highest scoring, while a transition-based algorithm builds a parse tree by a sequence of actions, each action is made after evaluating the state of the transition system.

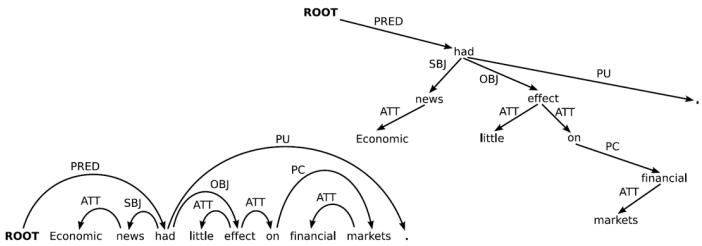

Figure 1. Dependency tree drawn in the standard way (left) and as a nested tree (right).

Theoretically, with the totally different background, these two approaches show us different properties:

. Graph-based models have a rather impoverished feature representation with a vary local scope, but they're globally trained and use exact inference algorithms (like Chu-Liu-Edmonds algorithms).

. Transition-based models have rich feature representations, as well as a defect called error propagation, which is introduced by local training and greedy inference algorithms.

Both models have achieved state-of-the-art accuracy for a wide range of corpus, as shown in the CoNLL shared tasks on dependency parsing. However, a detailed error analysis reveals important differences in the distribution of errors associated with the two models.[9]. In other words, utilize the strengths of these two approaches to improve the total accuracy

*indicates equal contribution

should be a bright way.

However, these two approaches adopt very different features. Regardless of the details of the structure of parsing models, the design of feature function is a crucial part for data-driven algorithms. Some early works have already tried out various kinds of feature engineering for both graph-based and transition-based models. Recently, encouraged by the enormous success of neural network framework in computer vision field, many parsing works deploy neural network to score and encode features. And we're looking for a way to represent the features inside these two models more coherently.

In this project, we introduce a simple and accurate model using Bidirectional LSTM (Bi-LSTM) feature representations [5] as our baseline. And construct our hybrid model based on the idea: encoding the features in Bi-LSTM. Each word is represented by its Bi-LSTM encoding, and the concatenation of a minimal set of such encoding is used to be the feature function, which is then passed to a non-linear scoring function: multi-layer perceptron (MLP). Moreover, we propose a shared Bi-LSTM layer which is trained jointly by two different evaluation cores (graph-based and transition based). This layer is designed for encoding and enhancing the model-invariant features from the word vector and the relations inside the whole sentence.

Aside from the shared layer, one simpler parser re-ranking idea[2] is introduced too. Like the auxiliary re-ranker which is trained to optimize for the downstream or extrinsic objective, we integrate a classifier to pick the better prediction core for different sentence by their encoded feature in Bi-LSTM.

We illustrate the basic components of our hybrid model in Section 2 and explain the use of them as well as our framework in Section 3. In Section 4, we show how we train the neural network based model and the corresponding testing result. Compared to our baseline, here shows some improvement in English corpus, from both the enhanced shared features and the online re-ranker. Furthermore, we test our hybrid model in a special tiny corpus: chatting records from Taobao, the accuracy compared to the baseline suggests there may be a bright way to go through the learning task in small scale of datasets. Finally, some defects in the design of the model and benchmark are talked about in Section 5.

2. Background & Related Work

2.1. Preliminaries

The dependency parsing task can be formulated as follow:

$$G = h(S, \Gamma, \lambda)$$

where G is the target parse tree, h is a search over the set of G, S represents the sentence, Γ is the rules we specified, λ is the parameter set.

As for data-driven parsing approaches, we mainly focus on the λ no matter what kind of form it is (perceptron, neural network or other machine learning methods).

2.2. Graph-based Parser

The graph-based dependency parsing idea comes from the graph theory. The parser parameterizes a model over smaller substructures to search the space of valid dependency graphs and produce the most likely one. The simplest parameterization is the arc-factored model that can be formulated as following:

$$h(S, \Gamma, \lambda) = \underset{G=(V,A)\in \mathbf{G}_s}{argmax} \sum_{(w_i, r, w_j)\in A} \lambda(w_i, r, w_j)$$

The problem is equivalent to finding the highest scoring directed spanning tree from a complete graph over the sentence. Some deterministic algorithms such as Chu-Liu-Edmonds algorithm perform well in solving it. Graph-based parsing is tractable in inference which is a prominent advantage to develop learning techniques that globally set parameters to maximize parsing performance on the training set[8].

2.3. Transition-based Parser

The transition-based parser uses a stack-based system, which can be regarded as a finite state machine with two kinds of transitions, shift and reduce. The system is initialized to a certain configuration and then finite transitions including shift and reduce can be applied to this system. It gets the terminal configuration after finite transitions and the parse tree is obtained.

A configuration of the finite state machine $c = (\sigma, \beta, T)$ consist of a stack σ, a buffer β and a set T. At first, σ and T are both empty, while $\beta = [w_1, w_2, \ldots, w_n, w_0]$. We use $\sigma|s_i|s\ldots|s_1|s_0]$ to denote items in the stack and $b_0|b_1|\ldots|b_i$ to denote those

Algorithm 1 Greedy transition-based parsing

Input: the sentence s, the score function SCORE()

$c \leftarrow \text{INITIAL}(s)$
while not $\text{TERMINAL}(c)$ **do**
    $\hat{t} \leftarrow \arg\max_{t \in \text{LEGAL}(c)} \text{SCORE}(\phi(c), t)$
    $c \leftarrow \hat{t}(c)$
return $tree(c)$

in the buffer. The terminal configuration contains an empty stack σ, a stack β with the only item w_0 or ROOT and the set T with arcs. We can build the parse tree from these arcs.

Three transitions can be applied to any configuration c, defined as:

$$\begin{aligned}
\text{SHIFT}(\sigma, b_0|\beta, T) &= (\sigma|b_0, \beta, T) \\
\text{LEFT}(\sigma|s_1|s_0, b_0|\beta, T) &= (\sigma|s_1, b_0|\beta, T \cup \{(b_0, s_0, l)\}) \\
\text{RIGHT}(\sigma|s_1|s_0, \beta, T) &= (\sigma|s_1, \beta, T \cup \{(s_1, s_0, l)\})
\end{aligned}$$

The first is the shift transition while the others are both reduce transitions. SHIFT moves the first item of the buffer b_0 to the stack. LEFT pops the top of the stack s_0 and attach it to the first item of the buffer b_0 with the dependency label l. RIGHT pops the top of the stack s_0 and attach it to its new top s_1 with the dependency label l.

We can use finite steps from the initial configuration to the terminate configuration. First, we consider that if we have an oracle who knows exactly whether SHIFT, LEFT or RIGHT should be applied to a certain configuration. However, the oracle usually does not exist. Then we can use a greedy algorithm that build a score function and choose the configuration with the maximum score as the next step. For each configuration, we define a feature extractor to extract a vector from the configuration for the convenience of calculation. Here is the algorithm:

This greedy algorithm works as a classifier, so we can use any classification algorithms. Kiperwasser's work[5] uses Bi-LSTM and below is the score function and the feature extractor.

$$\begin{aligned}
\phi(c) &= v_{s_2} \circ v_{s_1} \circ v_{s_0} \circ v_{b_1} \\
v_i &= BiLSTM(x_{1:n}, i)
\end{aligned}$$

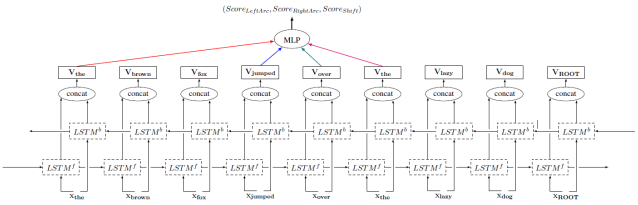

Figure 2. Transition-based Bi-LSTM

2.4. Bidirectional LSTM Feature Representations

Since graph-based and transition-based methods come from very different background and adopt very different feature representations, we're looking for a generic way to encode the model-invariant features. Thus, we introduce the Bi-LSTM feature extractor[5] to be the component of our hybrid model.

The Bi-LSTM comes from bidirectional recurrent neural network (Bi-RNN)[4], which is composed of two RNNs, RNN_F and RNN_R. One reads the sequence in its regular order, and the other reads it in reverse. Concretely, given a sequence of vector $x_{1:n}$, we get the vector representation of a word at the index i as this:

$$BiRNN_\theta(x_{1:n}, i) = RNN_F(x_{1:i}) \circ RNN_R(x_{n:i})$$

We can view the encoding of an item i as representing the item i together with a context of an infinite window around it. From Bi-RNN to Bi-LSTM, they share the same structure:

$$v_i = BiLSTM(x_{1:n}, i)$$

The feature function will concatenate a small number of Bi-LSTM vectors, and the feature function is model-variant. The resulting feature vectors are then scored using a multi-layer perceptron (MLP) with small number of hidden layers (usually one is enough), where the MLP is a non-linear function:

$$MLP_\theta(x) = W_2 \cdot tanh(W_1 \cdot x + b_1) + b_2$$

where $\theta = \{W_1, W_2, b_1, b_2\}$ are the parameters of the model.

The Bi-LSTM framework for transition-based and graph-based parsing are depicted in figure 2 and 3, respectively.

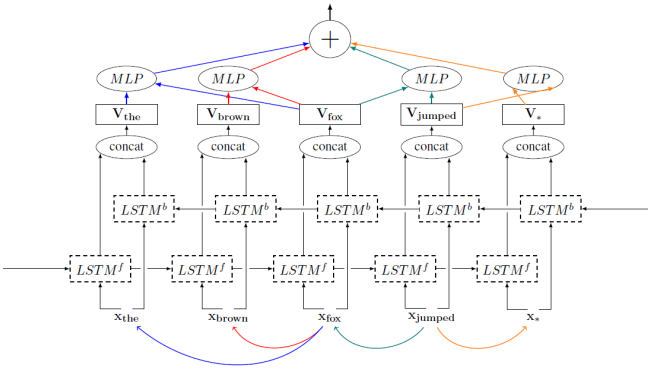

Figure 3. Graph-based Bi-LSTM

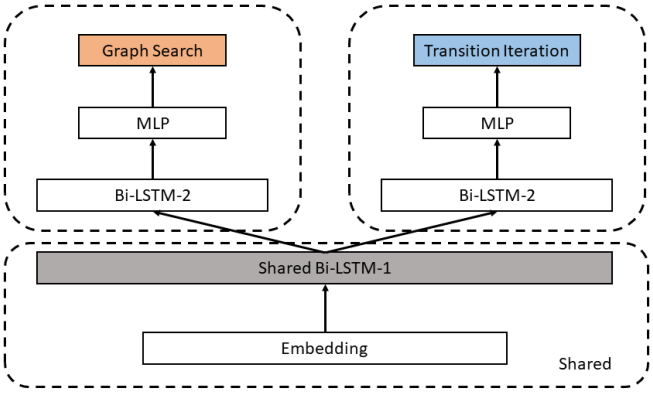

Figure 4. Shared Bi-LSTM encoding

2.5. Related Combination Effort

Some integrating trials have already been done from different views. By letting one model generate features for the other, Nivre and McDonald consistently improved accuracy for both models, resulting in a significant improvement[11]. By introducing a brand-new decoder, combining two parsers into one single system[12]. By designing new feature function and loss function, the parser can be trained better in jointly optimizing process.[3].

These works achieve considerable improvement as well telling many possible ways to do hybridization. However, here're less works based on neural network framework adapt the combination strategy. Encouraged by the success of multi-task learning, it may be a bright way to share something in training, that's what I'll illustrate in next section.

3. Methodology

3.1. Shared Model-invariant Features

We present the shared layer idea from a multi-task learning (MTL) view[7]. In contrast to the popular MTL methods who deploy the model in various kinds of tasks, we apply the similar idea in the single problem but two different paths to the solution. The basic idea of sharing the encoding can be expressed in a general conditional probability view:

$$\log p(y|x) = \sum_{j=1}^{m} \log p(y_i|y < j, x, s)$$

The basic idea of the hybrid model is depicted in figure 4. Note here two similar MLP with different weights are applied in the two different modules:

$$SCORE_\theta(x, t) = MLP_\theta(x)[t]$$

As for the graph-based core, we have this optimization goal:

$$\begin{aligned} parse(s) &= \underset{y \in \mathbf{Y}(s)}{argmax} \sum_{(h,m) \in y} score(\phi(s, h, m)) \\ &= \underset{y \in \mathbf{Y}(s)}{argmax} \sum_{(h,m) \in y} MLP(v_h \circ v_m) \\ v_i &= BiLSTM(x_{1:n}, i) \end{aligned}$$

where each of the tree scores is then calculated by activating the MLP on the arc representations. The entire loss can be viewed as the sum of multiple neural networks, which is sub-differentiable.

As for the transition-based core, we have this training goal:

$$\begin{aligned} max(0, 1 - \underset{t_o \in G}{max} MLP(\phi(c))[t_o] \\ + \underset{t_p \in A \backslash G}{max} MLP(\phi(c))[t_p]) \end{aligned}$$

where the loss comes from the difference between the gold transition and the selected transition. The gradients of the entire network (including the MLP, the individual BiLSTM and the shared BiLSTM) with respect to the sum of the losses are calculated using the backpropagation.

3.2. Online Re-ranking

Since we have obtained the model-invariant feature representations for words and sentence from the shared

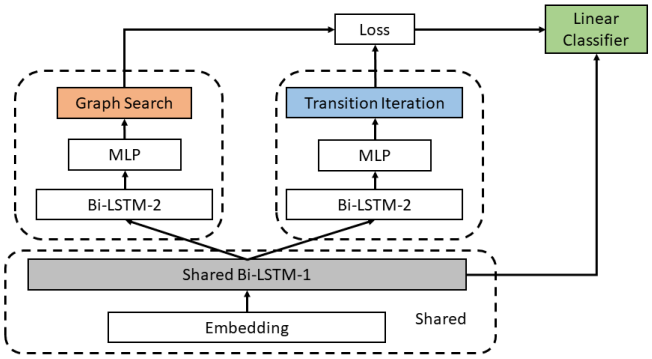

Figure 5. Online re-ranking

layer. It's natural to train an auxiliary re-ranker to select the fitted core. Different to the classical re-ranker [2], we feed the encoded features and the loss calculated by the two evaluation cores to the classifier, depicted in figure 5.

Treat the classifier as a function, we concatenate the vector representations of words to form the features of the sentence as the input. Concatenate the difference of the scores (higher score means a better choice) from two cores and pass it to the classifier as the target label.

The classifier is trained in the training process of the other components, but no paths are shared in the back-propagation. Thus, it won't affect the performance of either two cores.

4. Experiments & Result

4.1. Dataset & Metric

We evaluate our model on three datasets, UD-en, UD-zh and taobao. UD means Universal Dependencies, a framework for cross-linguistically consistent grammatical annotation and an open community effort with over 200 contributors producing more than 100 treebanks in over 60 languages (http://universaldependencies.org). Here taobao means a special tiny corpus produced by chatting between customers and shopkeepers.

The metric we adopt are mainly UAS (unlabeled attachment score) and LAS (labeled attachment score), which are the standard intrinsic parser metrics[1].

4.2. Hyperparameter Tuning

We performed a minimal hyper-parameter search with our hybrid model, but mainly adopt the hyper-parameters of the original Bi-LSTM work [5]. The hyper-parameters of the final networks used for the reported experiments are detailed in Table 1.

Word embedding dimension	100
POS tag embedding dimension	25
CPOS tag embedding dimension	25
Ontology (lemma) tag embedding dimension	25
Hidden units in MLP	100
Hidden units in MLP_{LBL}	100
Bi-LSTM layers	2
Bi-LSTM Dimensions (hidden/output)	125/125
dropout rate	0.33

Table 1. Hyper-parameters

4.3. Main Result

Table 2 lists the accuracies of our best parsing models, compared to the baseline we implemented.

It's clear that our model achieves some improvement in English corpus. We can see the improvement comes from both the shared layer as well classifier, and the shared layer contributes more.

The reason for the average performance in Chinese should be the embedding issue. The word embedding method we used here is not so friendly to Chinese, which causes the negative effect from the shared encoding layer.

4.4. Additional Result in Special Corpus

Besides, our model performs better in the taobao corpus. This corpus has some properties such as small scale, colloquial, lax in grammar. The positive result may suggest that the hybrid method can work better for this kind tiny dataset learning.

Actually, it's also intuitive that multi-task learning could be helpful to avoid the overfitting for dataset with small scale. And our hybridization idea is suited for under-fitting.

5. Conclusion & Further Work

We deployed two new ideas in the concise Bi-LSTM model, and achieved some improvement compared to our baseline. However, here're some defects in our model, which may suggest the room for improvement of our model.

1. train and test our model in some more widely used datasets (like PTB and CTB)

System	Method	UD-en		UD-zh		taobao	
		UAS	LAS	UAS	LAS	Accuracy 1	Accuracy 2
Baseline	graph	88.38	86.32	-	-	-	-
Baseline	transition	87.36	85.25	-	-	-	-
This work	hybrid + classifier	**88.85**	**86.66**	-	-	-	-
This work	hybrid + random	88.73	86.55	-	-	-	-
This work	hybrid + graph	88.65	86.26	-	-	-	-
This work	hybrid + transition	88.49	86.29	-	-	-	-
Baseline	graph	-	-	82.85	79.88	-	-
Baseline	transition	-	-	**83.28**	**80.55**	-	-
This work	hybrid + classifier	-	-	82.52	79.88	-	-
This work	hybrid + random	-	-	82.56	79.73	-	-
This work	hybrid + graph	-	-	82.27	79.18	-	-
This work	hybrid + transition	-	-	83.20	70.19	-	-
Baseline	graph	-	-	-	-	86.30%	87.02%
Baseline	graph + embedding	-	-	-	-	87.64%	89.18%
Baseline	graph + embedding + ontology	-	-	-	-	87.47%	89.12%
This work	hybrid + classifier	-	-	-	-	**88.74%**	88.58%
This work	hybrid + classifier + ontology	-	-	-	-	88.68%	88.92%
This work	hybrid + random + ontology	-	-	-	-	88.73%	88.87%
This work	hybrid + graph + ontology	-	-	-	-	88.59%	**89.51%**
This work	hybrid + transition + ontology	-	-	-	-	88.45%	89.21%

Table 2. Experiment Result.

2. introduce external embedding for a fully benchmark (especially for Chinese)

3. optimize some structure defects of the neural network for efficiency issue

4. try to find a better normalization method to feed the loss calculated by the two different evaluation algorithms to the classifier

6. Reference

References

[1] S. Buchholz and E. Marsi. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics, 2006.

[2] M. Collins and T. Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70, 2005.

[3] K. Hall, R. McDonald, J. Katz-Brown, and M. Ringgaard. Training dependency parsers by jointly optimizing multiple objectives. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1489–1499. Association for Computational Linguistics, 2011.

[4] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[5] E. Kiperwasser and Y. Goldberg. Simple and accurate dependency parsing using bidirectional lstm feature representations. *arXiv preprint arXiv:1603.04351*, 2016.

[6] S. Kübler, R. McDonald, and J. Nivre. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127, 2009.

[7] M.-T. Luong, Q. V. Le, I. Sutskever, O. Vinyals, and L. Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[8] R. McDonald, K. Lerman, and F. Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 216–220. Association for Computational Linguistics, 2006.

[9] R. McDonald and J. Nivre. Characterizing the errors of data-driven dependency parsing models. In *Pro-*

ceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL), 2007.

[10] R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 523–530, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[11] J. Nivre and R. McDonald. Integrating graph-based and transition-based dependency parsers. *Proceedings of ACL-08: HLT*, pages 950–958, 2008.

[12] Y. Zhang and S. Clark. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics, 2008.