Dependency Parsing With Deep Reinforcement Learning

Ying Shen
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
yshen2@cs.cmu.edu

Yiming Wang
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
yimingw2@cs.cmu.edu

Zhun Liu
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
zhunl@cs.cmu.edu

Abstract

In this project, we plan to improve the performance of transition-based dependency parsers using deep reinforcement learning. Previous work on transition-based dependency parsing mostly rely on greedy decoding at inference stage, and is prone to the error propagation problem, where one early error can lead the parser to diverge further and further away from the ground truth. We train a reinforcement learning agent using actor-critic algorithm to perform non-greedy decoding with transition-based parsers. This RL framework for non-greedy decoding for dependency parsing can be easily built on top of previous transition-based parsers, hence can benefit from previous parsing models. We perform experiments on the English Penn Treebank (PTB) datasets and our model achieve 90.63% unlabeled accuracy, which improves around 0.4% accuracy compared to the supervised neural dependency parser.

1 Introduction

Dependency parsing is one of the most fundamental tasks in the field of Natural Language Processing. It is widely used to facilitate further NLP research and applications. Recent research has shown that a number of NLP tasks involving natural language understanding benefits from dependency relationships (Levy and Goldberg [2014], Angeli et al. [2015], Bowman et al. [2016]). However, mistakes made by parsers can inject more error into downstream tasks and in turn hurt the performance. Hence building accurate dependency parsing models becomes increasingly vital for a range of NLP tasks that depend on it.

Previous work on dependency parsing mainly tackles this problem from two perspective: one approach views parsing as a structured prediction task, and uses advance search algorithms on graphs for exact inference; the other approach constructs a shift-reduce parser which greedily takes actions to produce the dependency arcs for each word incrementally. While the first approach is intriguing in its capability of exact inference, the transition-based shift-reduce parser attracts greater attention because of its decoding efficiency and almost comparable performance. Nevertheless, it still bears the problem of greedy decoding, which introduces the error propagation problem. For example, at any given step, if the parser diverges from the ground truth based on its current greedy choice, it diverges from the gold parse and can diverge further from there, since it's increasingly difficult for the parser

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

to predict subsequent correct sub-structures given the current incorrect parse. This may greatly affect the accuracy of parsing in some situations.

In this project, we propose to build reinforcement learning agent with transition-based shift-reduce parser which aims to learn non-greedy decoding by considering future rewards. Concretely, instead of using a feed-forward neural network alone to greedily decide which parsing action to take at each step, we train an RL agent using Advantage Actor Critic (A2C) algorithm to choose the "non-greedy best" action for the parser that maximizes not the current action likelihood, but the discounted future reward. In addition, this RL based method doesn't require a constructed oracle to provide the gold transitions at each step, and can be trained as long as we can define a reward for actions taken throughout the parsing episode. We demonstrate that our approach can achieve better performance over greedy-decoding shift-reduce parsers with marginally more computation complexity, and we claim that this RL framework for dependency parsing can be built upon previous shift-reduce parsers and reuse their pre-trained models easily.

2 Related Work

Deep reinforcement learning methods can be divided into two categories based on the role neural networks play in an RL agent. The first category is value-based learning where the network is trained to approximate the optimal value function, and actions are then taken using a greedy or epsilon greedy policy given the value function. The other category is policy-based learning where the network plays the role as the policy itself, emitting a probability distribution over actions given current state, which maximizes the expectation of reward. Both approaches have been explored for the task of dependency parsing, where a shift-reduce parser is viewed as an RL agent that decides to shift, emit arcs and reduce (depending on the different parser systems actions may vary) given current state of the sentence.

Transition-based Dependency Parsing An important class of dependency parsing models take the approach of transition-based dependency parsing. These parsers construct a shift-reduce parsing system, and turn the problem of parsing into predicting a sequence of actions taken to parse a given sentence. Such models also varies in the transition systems. Nivre and Scholz [2004] introduces an arc-standard parsing system, which maintains a stack and a buffer of words to keep track of the parsing progress, and uses three standard actions: `shift`, `left-arc` and `right-arc` to parse the sentence. On the other hand, arc-eager transition systems are also proposed (Nivre [2004]) exploiting the incrementality in parsing. However, arc-eager systems cannot ensure well-formedness and later Nivre and Fernández-González [2014] proposed modifications of this system to ensure well-formedness. In this work, since we're already using an RL agent to learn to take parsing actions which is very volatile in training, we choose the arc-standard system so as to reduce the chance of the agent to produce non-well-formed parses.

Neural Dependency Parsing As shift-reduce parsers turn parsing into a sequence of classification problems, it is reasonable to use neural networks as classifiers at each state. Neural networks was introduced for dependency parsing by Chen and Manning [2014] to tackle the problem of extensive manually designed features and computational complexity of feature extraction. The role of neural network is to predict the arc labels for each configuration given the concatenation of embeddings of word, POS-tags and arc labels for the current time step. A simple one-hidden layer neural networks and greedy decoding is used. As neural architectures can take in a large collection of combinations of features and still be able to learn generalizable representations for parsing task, Lei et al. [2014] proposed to use low-rank tensors to score the combinations of features efficiently using low-rank tensors. Dozat and Manning [2016] also extended this line of research by using a biaffine layer to compute attention over the the combinations of features. In this work, we plan to follow the system by Chen and Manning [2014], since it is efficient, achieves a decent accuracy, and uses greedy decoding in which an RL agent can improve upon. In our experiments we show that our RL agent with similar network architecture for policy network indeed learns to make decisions better than greedy decoding.

Reinforcement Learning Based Parsing It is intuitive that a shift-reduce parser can be viewed as an agent that takes actions in sequential manners, hence it can be trained as a reinforcement learning agent. For value-based learning approaches, Zhang and Chan [2009] approximate the state-action function $Q(s, a)$ by minimizing the free energy in a Restricted Boltzmann Machine. In their work,

SARSA (State-Action-Reward-State-Action) algorithm is used to update the parameters of the Q network. The action is then selected by the softmax scores computed by Boltzmann distribution. In their system, the immediate reward is defined by the Hamming Loss between partial generated tree and the expected tree.

Using this neural network as policy network, Le and Fokkens use the approximate policy gradient to further maximize the expected reward to find the best policy. The reward is given at the end of each parse by the number of correct labeled arcs. An approximation is used by sampling some number of trajectories when calculating the gradient.

3 Method

3.1 Transition-Based Dependency Parsing

The Transition-based dependency parsing aims to constructs a dependency tree by predicting a sequence of transitions given a sentence $s = w_0, w_1, ..., w_n$. There are many variants in terms of the transition-based approach. In this paper, we employ the **arc standard** transition system Nivre [2003], one of the most popular transition system.

In the **arc standard** transition system, the parsing configuration consists of triple $< \Sigma, B, A >$, where Σ represents a stack of the partially processed tokens, B represents a buffer of the unprocessed input tokens and A represents a set of dependency arcs that are created by the previous transitions. Denoting $\sigma_i (i = 1, 2, ...)$ as the $i^{\texttt{th}}$ top element on the stack, and $b_i (i = 1, 2, ...)$ as the $i^{\texttt{th}}$ element of the buffer, the arc eager transition system supports three types of transition operators:

LEFT$_l$ adds an arc $b_1 \xrightarrow{l} \sigma_1$ with label l and removes σ_1 from the stack.

RIGHT$_l$ adds an arc $\sigma_1 \xrightarrow{l} b_1$ with label l and removes b_1 from the stack.

SHIFT moves b_1 from the buffer to the stack.

The initial configuration for a sentence $s = w_0, w_1, ..., w_n$ is $\Sigma = [\texttt{ROOT}]$, $B = [w_0, w_1, ..., w_n]$, and $A = \emptyset$. The parser terminates when the buffer is empty, the stack contains the single node `ROOT` and A contains a parse tree.

3.2 Reinforcement Learning

In this section, to integrate Reinforcement Learning (RL) into the Transition-Based Dependency Parsing model, we reformulate the dependency parsing as Markov Decision Process (MDP, $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r)$) where $\mathcal{S}$ is the set of all possible states, $\mathcal{A}$ contains all possible actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the transition function and $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function.

In our model, a state corresponds to a configuration and the possible actions are the four possible transitions: **LEFT**, **RIGHT**, and **SHIFT**.

At each time-step t, given the current state s_t (configuration), the parser takes an action a_t following policy π and receives a reward r_t. The policy π defines the behavior of the parser by mapping states to a probability distribution over actions $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$.

Given a policy π, the state-action function can be defined as the expected accumulative reward R_t received after taking an action a_t in state s_t:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}[R_t | s_t, a_t] \tag{1}$$

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2}$$

where $\gamma \in [0, 1]$ is the discounted factor.

3.3 Actor-Critic Algorithm

The actor-critic is a widely used architecture which combine both state-value function and policy-gradient ideas as part of Critic and Actor, respectively Sutton et al. [2000]. The Actor can be viewed

as the approximated policy function with parameters θ, which produces the action given the current state. The Critic can be viewed as the approximated value function with parameters w that produces the estimated value given current state and action.

For the Actor, we use the similar neural network architecture proposed by Chen and Manning. The Actor works as a policy network $\pi_\theta(\cdot)$ mapping input states to probabilities of actions.

For the Critic, we use simply a densely connected neural network to approximate the value function. The Critic works as a value network V_ω mapping input states to a single value, i.e. evaluating the quality of the current policy by adapting the value function estimate.

We adopt the Advantage Actor-Critic algorithm here to provide a balance between bootstraping using the value function and using the full Monte-Carlo return. The Advantage Actor-Critic algorithm uses the advantage estimates rather than just discounted returns to allow the agent to determine not just how good its actions were, but how much better they turned out to be than expected.

The advantage function is defined as:

$$A_\omega(S_t) = R_t - V_\omega(S_t) \tag{3}$$

where R_t represents the Monte-Carlo estimated reward.

The Critic network therefore minimizes the mean–squared error between the Monte-Carlo estimated reward and the approximated value given the value function. The loss function of the Critic can be defined as:

$$L(\omega) = \frac{1}{T}\sum_{i=0}^{T-1}(R_t - V_\omega(S_t))^2 \tag{4}$$

We can then define the Actor's loss function as:

$$L(\theta) = \frac{1}{T}\sum_{i=0}^{T-1} A_\omega(S_t)\log\pi_\theta(A_t|S_t) \tag{5}$$

3.4 Parsing Environment

We specifically design a parsing environment called ParseEnv to attack the parsing problem. The environment should take a sentence as input for every movement. There are basically two functions in ParseEnv, one is **STEPS** which takes the current state S_t generated from the current parsing configuration $< \Sigma, B, A >$ as input and output $< S_{t+1}, R_t, F_t >$, which stands for next state, immediate reward and the episode ending situation of the sentence for the current step respectively. The other function in ParseEnv is **RESET**, which clears up all the predicted dependencies of the sentence and returns to its initial state with all the tokens on the buffer and only $ROOT$ label on the stack. The function returns the initial state of the sentence.

As for the reward functions, we specifically design the immediate reward r_t as the Hamming Loss between the partial tree and the expected tree. To be more specific, the reward is designed as :

$$r_t = -\sum_{i=1}^{T} 1[y_i \neq \hat{y}_i] \tag{6}$$

where y is the true output, $\hat{y}_i$ is the predicted output of the partial dependency tree and T is the sentence length.

4 Experimental Setup

4.1 Datasets

We conduct our experiments on the English Penn Treebank (PTB) dataset with standard splits. We use sections 2-21 for training, section 22 as development set and 23 as test set. The training, development and testing set contains $39,832$, $1,700$ and $2,416$ sentences, respectively.

4.2 Baselines

We compare the performance of our model with several baselines.

Neural Dependency Parser We use the Neural Dependency Parser by Chen and Manning as the supervised learning baseline to compared with our reinforcement learning method. In their model, a neural network works as a function mapping input features to probabilities of actions: $f_{NN} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. The neural network contains one input embedding layer, one hidden layer with a cube activation function, and one softmax output layer for modeling multi-class probabilities.

REINFORCE The REINFORCE model performs the policy-gradient learning using only the Actor network as the policy model. The Actor network adopts the similar structure as the Neural Dependency Parser. The loss function of the REINFORCE model is proportional to the product of the total discounted return G_t and $\log \pi_\theta(A_t|S_t)$.

4.3 Model Settings

For our Actor-Critic model, we set the learning rate for the policy model (Actor) and the value network (Critic) to be $1e-5$ and $2e-4$, respectively. The discount factor γ is set for the value of 0.99. We employ the Adam algorithm as the optimizer since it can automatically adjust the learning rate based on the statistics of the gradients it's observing. For each sentence we generate one episode from the environment and complete one update from the generated episode. The batch size we use here is the length of state-action pair from each generated episode.

For the Actor, we use the same neural structure from Chen and Manning as the policy network. We also apply the parameters of their model trained from the supervised learning method as the pretrained Actor and use our reinforcement learning method to improve its performance.

For the Critic, we employ an embedding layer for the one-hot discrete input for each state and 4 hidden layers with 128, 128, 64, 32 as the hidden size and RELU as the activation function. A last fully-connected layer is used to predict a scalar as the value for the current state.

We use different strategy for the generation of an episode between training and testing time. During training, we generate an episode from the **ParseEnv** by sampling from the action probabilities predicted by the policy network. On the other hand, during testing, the Actor select the most likely action given the probabilities of actions instead of sampling an action based on the probabilities, since unlike training an agent for game playing, there exists one unique correct action on each step for the transition based parsing algorithm.

The Actor network can sometimes produce illegal action for the given input states. For example, at the initial configuration of a sentence, the **LEFT** and **RIGHT** action will be considered as illegal since the agent tries to remove a token from the stack when the stack is empty. We consider two approaches to tackle the problem of illegal actions. One is to design a big negative reward for every illegal actions. The other is by limiting the set of actions when producing the most likely actions, which turns out to be much more effective than the previous approach. To eliminate illegal moves, we limit the set of actions $\mathcal{A}$ to only those that were legal and form the legal action set $Legal(\mathcal{A}) \subseteq \mathcal{A}$ for each step. The agent will then pick the most likely action a from $Legal(\mathcal{A})$.

5 Results

The performance of our model is evaluated by the unlabeled attachment score (UAS=the percentage of tokens with the correct head). Punctuation tokens are excluded from scoring.

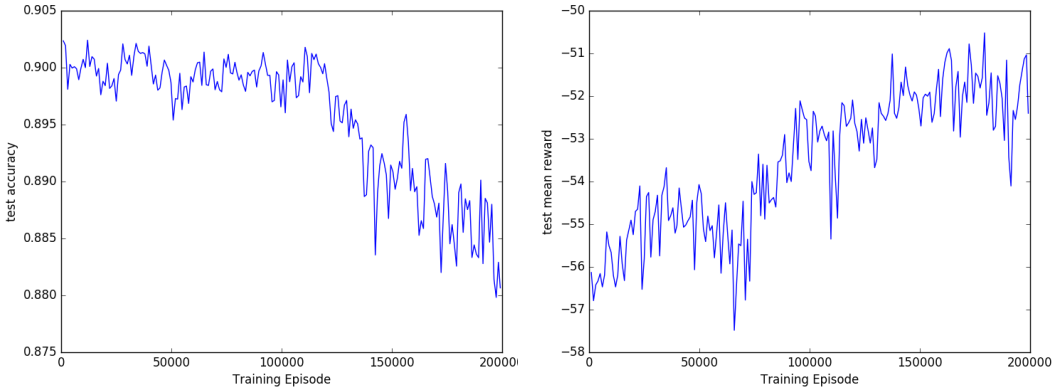
We compare the performance of our parser with Chen and Manning's supervised neural dependency parser and the REINFORCE parser as shown in Table 1. Compared to supervised Neural Dependency Parser and the REINFORCE method, our Advantage Actor-Critic model achieves around 0.4 percentage improvement, which is a decent improvement since for dependency parsing the basic supervised learning method can already achieve significant result.

Figure 1 and 2 shows the plot of the accuracy and reward during test time for REINFORCE and A2C method respectively. Both the accuracy and reward plots are produced every 1000 training episode. The y-axis of the accuracy plots are the unlabeled arc accuracy of the entire test set, while the y-axis of the mean reward plots are the average total rewards for each sentence.

Model	UAS
Neural Dependency Parser	90.2269
REINFORCE	90.2409
Advantage Actor-Critic	90.6338

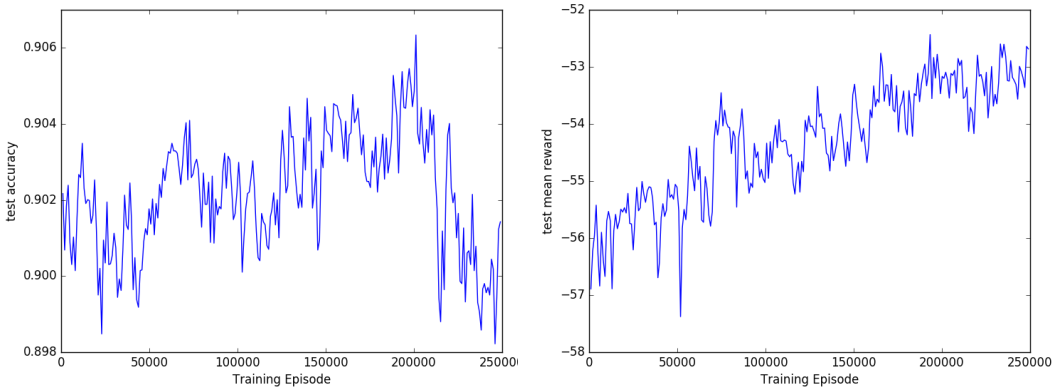
Table 1: Comparison of the unlabeled accuracy (UAS) between Neural Dependency Parser, REINFORCE model and Advantage Actor-Critic on the test set.

For REINFORCE method, the test accuracy is stable for 100 thousands episodes while the test mean reward is continuously increasing. For A2C method, both the test accuracy and the test mean reward is increasing for 200 thousands episodes. The test accuracy for both methods go down at the tail because of the divergence of reinforcement learning.



(a) Test accuracy (b) Test mean reward

Figure 1: The accuracy and mean reward for test set during training for REINFORCE method



(a) Test accuracy (b) Test mean reward

Figure 2: The accuracy and mean reward for test set during training for A2C method

6 Discussion

For reinforcement learning, the A2C algorithm yields better performance compared to the REINFORCE algorithm. We also find that overall the REINFORCE algorithm is more likely to main a certain stability than the A2C algorithm. We believe that this is due to the fact that the A2C algorithm uses a value function rather than the empirical returns to help reduce the variance at the cost of introducing bias where the different bias at each step can cause the algorithm to be more unstable.

Compared to supervised Neural based parser, both the REINFORCE method and the Advantage Actor-Critic method show their potential in improving the parsing performance. We argue that reinforcement learning framework can perform non-greedy decoding with transition-based parser by considering future rewards for actions taken throughout the parsing episode, while supervised Neural based parser only greedily maximizes the probabilities of the current action.

We notice that there is an accuracy drop at the tail of the plot, while the test mean reward is constantly increasing. We argue that this result from the structure of the reward function. The reward function we use is the negative Hamming Loss which compute the incorrectness of the partial tree compared to the true labels. In this situation, the negative reward gained from the false left/right arc action made at the early stage of the episode will continuously have effect for the immediate reward of the later parsing step. The loss essential forces the parser to finish the parsing as soon as possible. Therefore the parsing agent will gradually learn to perform the **SHIFT** action at the early step of the episode while perform the **LEFT/RIGHT** arc action at the later step of the episode. This phenomenon is depicted on the test mean reward plot with the continuous reward increasing because the agent learn to **SHIFT** before, while the **SHIFT** action will harm the performance, which is showed by the accuracy drop.

7 Future Work

Currently our agent's policy network takes only the current shift-reduce system state as input, and tries to make a decision based on the current state only. While this allows for looking back at the previous arcs (which is included in the state), it does not consider the actual action sequence that has been executed. This can probably lead to the problem mentioned in the previous section: the agent learns to be "shift-eager" in order to exploit the reward mechanism over choosing the right actions. Hence we think it is important for the model to be able to track its own action sequence. This may be addressed by integrating a recurrent network with attention over time steps as policy network.

References

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 344–354, 2015.

Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. *arXiv preprint arXiv:1603.06021*, 2016.

Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

Timothy Dozat and Christopher D Manning. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*, 2016.

Minh Le and Antske Fokkens. Tackling error propagation through reinforcement learning: A case of greedy dependency parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 677–687, 2017.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1381–1391, 2014.

Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 302–308, 2014.

Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Citeseer, 2003.

Joakim Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics, 2004.

Joakim Nivre and Daniel Fernández-González. Arc-eager parsing with the tree constraint. *Computational linguistics*, 40(2):259–267, 2014.

Joakim Nivre and Mario Scholz. Deterministic dependency parsing of english text. In *Proceedings of the 20th international conference on Computational Linguistics*, page 64. Association for Computational Linguistics, 2004.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

Lidan Zhang and Kwok Ping Chan. Dependency parsing with energy-based reinforcement learning. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 234–237. Association for Computational Linguistics, 2009.