6.864/6.806 Project Report:

Reinforcement Learning Approach for Transition-Based Parsing

Yuqing Zhang and Kalki Seksaria

December 11, 2015

Abstract

We evaluate the feasibility of a reinforcement learning (RL) approach for transition-based parsing. In transition-based parsing, the parser modifies a sentence configuration (stack, buffer, arcs) in discrete steps until termination. At each step, the parser must take a well-defined action (left-arc, right-arc, shift). In our RL framework, the parser learns its policy, a mapping from each sentence configuration to a probability distribution over the next action, by responding to an environmental reward. We first build a toy parser whose policy is characterized by a log-linear model with few (1641) features. We find that even with a local reward function, the toy parser can be trained with RL to achieve ∼63.6% unlabeled arc score (UAS) on the Universal Dependencies test set for English. We further show that by incorporating a global reward function, the toy parser achieves ∼72.1% UAS. This boost produced by the global reward is promising – it suggests that the ability of RL to train long-term decision making might be useful. Finally, we attempt to extend our RL framework to Chen & Manning's (2014) neural parser, which was originally trained on a per-arc (local) basis.

1 Introduction

1.1 Problem Formulation

In this project we analyze a reinforcement learning (RL) approach for transition-based parsing. A transition-based parser modifies a sentence configuration in discrete steps until completion. A sentence configuration consists of a stack, a buffer, and the arcs that have been drawn. At each step, the parser must choose from a set of well-defined actions – left-arc, right-arc, and shift.

In a reinforcement learning approach, the parser learns its policy by responding to an environmental reward. The parser's policy is a mapping from each sentence configuration to a probability distribution over the next action. The main advantage of RL is its ability to train long-term decision making by incorporating a global reward function. A global reward scores the entire sentence, while a local reward is only scores individual transitions. In transition-based parsing, each transition has long-term consequences for subsequent transitions – each arc influences the following arcs. By using a global reward, RL frames sentence parsing as a structured prediction problem, and this approach might overcome the limitations of a per-arc (local) training objective.

Our goal is to apply a RL training framework to Chen & Manning's (2014) neural parser [1], which was originally trained on a per-arc (i.e. local) basis. We hypothesize that by training the parser as a structured prediction model (i.e. RL with global reward) we can improve its performance.

1.2 Two-Model Experimental Setup

Parsing is a computationally heavy task that can require hundreds of thousands of features and long training times for state-of-the-art performance. The computational complexity makes it difficult to develop a new

learning framework directly on the neural parser. Due to this, we set up two different models in our project pipeline.

We first implement a simple toy parser that we train via RL. The toy parser serves as a sandbox for idea development. We take the successful ideas from the toy parser and attempt to extend them to the neural parser. This pipeline allows our idea generation to iterate more quickly.

2 Data & Evaluation Benchmarks

Our parsers work with the Universal Dependencies v1.1 data sets for English. We adhere to the given partitions of training, development, and test sets. For the sake of reducing computation complexity, we focus on unlabeled arc score (UAS) as our evaluation metric. It requires more computational time to train a model to produce high labeled arc scores (LAS), because labeled dependency parsing is a more complex problem.

Compared to other methods, the main strength of RL is its ability to train long-term decision making. This strength can only be realized with a global reward function – it is impossible to make long-term decisions by optimizing a local (per-arc) reward. In the toy parser, we use a local reward as a benchmark and determine whether a global reward outperforms it. If not, then reinforcement learning unlikely to be a fruitful idea.

Because our ultimate goal is to apply an RL framework to Chen & Manning's neural parser, the original neural parser serves as a natural benchmark. We first train the neural parser with the original framework. Then, we train the neural parser with our RL framework and compare the performance.

3 Reinforcement Learning Framework

In this section we provide a high-level outline of our reinforcement learning framework, which is applied to both our toy parser and our modified neural parser. Our RL framework consists of two components: "oracle" training and a policy gradient algorithm. The policy gradient algorithm is the main engine of the training, but it can be quite slow when the parameters θ are far from optimal. The purpose of the oracle training is parameter initialization – it pushes the parameters in a reasonable direction so that the policy gradient algorithm can be effective. Figure 1 outlines the training framework in pseudocode.

Next, we describe the oracle training in more detail. For each training sentence, we can iteratively look at the gold arcs and determine an optimal action (left-arc, right-arc, or shift). We take the optimal action and push θ in the direction that increases the probability of the optimal action, given the configuration. Oracle training harbors no notion of continuity across transitions – each configuration state is seen as a separate training pair (configuration, optimal action). Oracle training can very quickly push θ to a reasonable value, but it alone might not achieve optimal performance for the reasons below:

- Especially when θ is high-dimensional, oracle training encounters only a small subset of the feature space. It only encounters configuration states that are along the path of optimal actions, which is presumably a tiny fraction of all possible configuration states.
- In oracle training, we look ahead and force the model to take actions that are we know are long-term optimal. As a result, the model is only learns to take the best myopic action and is unaware of the long-term consequences of an incorrect action. The model might under-perform on unseen test sentences.

Step 2 in Figure 1 outlines our policy gradient algorithm, adapted from [2]. In the toy parser we experiment with both local and global rewards, while in the modified neural parser we apply only a global reward. Local rewards are determined only by individual transitions, e.g. the correctness of a drawn arc. Global rewards are determined by the quality of an entire parsed sentence. When the reward is positive (negative), θ is updated to encourage (discourage) the choices that led to the reward.

1. Initialize θ with "Oracle" training
2. **for** i in $1, 2, 3, \ldots$ NumIter **do**
   for Sentence S in TrainSet **do**
   config $\leftarrow$ InitConfig(S), $\Delta_{\text{global}} \leftarrow \mathbf{0}$
   while NotDone(config) **do**
   trans $\sim$ Policy(config; θ)
   $\text{r}_{\text{local}} \leftarrow$ LocalReward(trans), $\Delta_{\text{local}} \leftarrow$ LocalDelta(config, trans)
   $\theta \leftarrow \theta + \text{r}_{\text{local}} \times \Delta_{\text{local}}$ (toy-only)
   $\Delta_{\text{global}} \leftarrow \Delta_{\text{global}} + \Delta_{\text{local}}$
   config $\leftarrow$ UpdateConfig(config, trans)
   $\text{r}_{\text{global}} \leftarrow$ GlobalReward(config)
   $\theta \leftarrow \theta + \text{r}_{\text{global}} \times \Delta_{\text{global}}$

Figure 1: High-level outline of the RL framework for the toy and neural parsers.

4 Toy Parser

4.1 Model Characterization

Our toy parser is a simple log-linear model that predicts three classes: left-arc, right-arc, and shift (it only draws unlabeled arcs). It consists of the following 1641 features based on the stack and buffer. Let s_i denote the i-th word on stack and b_i denote the i-th word in the buffer.

- 615 unigram word features. For each of s_1, s_2, and b_1, we maintain a one-hot vector (length 205) indicating whether the word is one of the 205 most frequent words.
- 54 unigram coarse part of speech (CPOS) features. For each of s_1, s_2, and b_1, we maintain a one-hot vector (length 18) indicating the CPOS of the word.
- 972 bigram CPOS features. For each of $s_1 * s_2$, $s_1 * b_1$, and $s_2 * b_1$, we maintain a one-hot vector (length 324) indicating the combinations of the CPOS of the two words.

4.2 Local vs. Global Reward

In the toy parser, we first determine if a global reward indeed outperforms a local reward. The main purpose of RL is to learn long-term planning by optimizing a global reward. If a global reward is unable to outperform a local one with the toy parser, then RL is unlikely to be fruitful with the neural parser. Below we discuss the reward functions and their results, applied in the algorithm outlined in Figure 1.

Our local reward, below, provides feedback to each arc that is drawn. The asymmetry in the reward function for correct versus incorrect arcs might unintuitive at first glance. When we encourage (discourage) one action, we effectively discourage (encourage) the other two actions because the probability distribution must sum to 1. When we draw a correct arc, we want to strongly encourage the taken action and discourage the other two. However, we do not have the same certainty in the parameter updates when we draw an incorrect arc. Even though we know that the taken action was incorrect, we do not know which of the alternatives should have been taken instead. Since we do not want to strongly encourage *both* of the two alternative actions, we take a relatively smaller step in the parameter updates. The local reward is scaled by a 0.01 per-transition step size.

$$\text{Local Reward} = \begin{cases} +1.0 & \text{if correct arc} \\ -0.5 & \text{if incorrect arc} \\ 0 & \text{o.w.} \end{cases}$$

Our global reward, below, provides feedback after an entire sentence is parsed. Accuracy refers to the percentage of arcs in the sentence that were drawn correctly. One challenge in defining a global reward

function is reducing bias towards the actions that happened to be sampled. For example, consider an intuitive but ineffective global reward Global Reward = Accuracy. Since the accuracy is always non-negative, with this global reward we always reward the series of sampled actions. This is undesirable – when the accuracy is extremely low, we would rather push θ *away* from many of the sampled actions. To resolve this we introduce a threshold. Only when the total sentence accuracy is above this threshold are we confident enough in the series of sampled actions to encourage the entire sequence.

Furthermore, we emphasize the importance of a nonnegative reward. Suppose we instead adopt a global reward Global Reward = Accuracy − Threshold that can be negative. When Accuracy < Threshold, we push θ away from *each* action in the sampled series. This is problematic because although Accuracy < Threshold, many specific actions in the sequence might still be desirable. By pushing θ away from all of the actions, we would worsen the model. To avoid this problem we design a nonnegative reward. The global reward is scaled by a 0.1 per-sentence step size.

$$\text{Global Reward} = \begin{cases} \text{Accuracy} & \text{if Accuracy} > \text{Threshold} \\ 0 & \text{o.w.} \end{cases}$$

After parameter initialization via oracle training, we perform 20 iterations of RL. We pass through the entire training set once per iteration. In each iteration, the value of Threshold is the training UAS achieved in the previous iteration. Figure 2 graphs the toy parser performance with the local reward alone, the global award alone, and both rewards together. We see that reinforcement learning with the global reward clearly outperforms the local reward, and combining the two improves results even further. On the test set, the local reward achieves 63.6% UAS, the global reward achieves 71.3% UAS, and the combination achieves 72.1% UAS. The boost associated with incorporating the global reward is promising – it suggests that the ability of RL to train long-term planning might be useful.

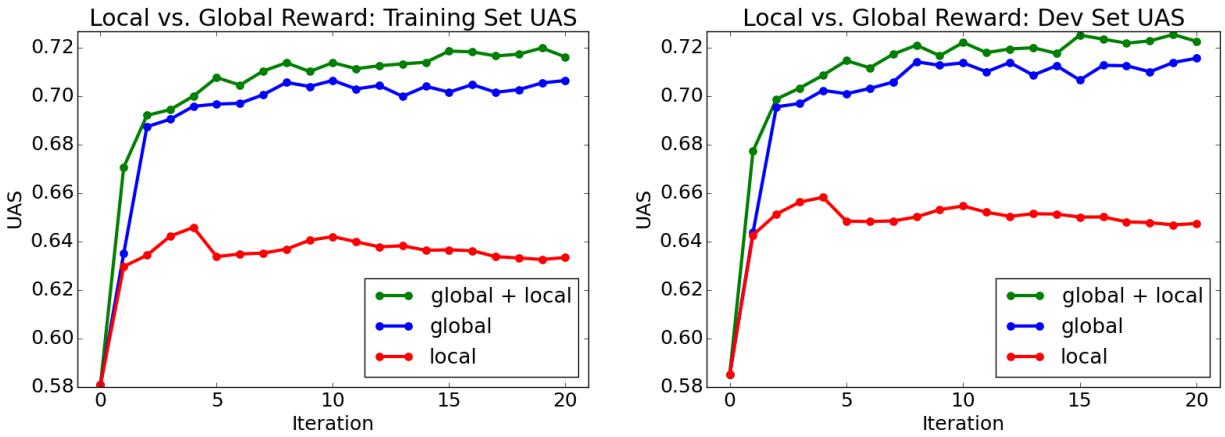

Figure 2: Local vs. global reward, training (left) and development (right) performance.

4.3 Level-weighted Global Reward

Next, we experiment with a more complicated global reward. In the global reward outlined in Section 4.2, each arc in the gold tree is assigned equal importance. One might instead hypothesize that arcs closer to the root are more important. When a head → child arc is drawn incorrectly, it might have negative long-term consequences – it might lead to incorrect child → grandchild arcs.

Consider the gold dependency tree for a sentence. We propose a level-weighted global reward in which the arcs in Level N are given α times the importance of the arcs in Level $N-1$, for some decay factor $0 \leq \alpha \leq 1$. The root arc is in Level 0. The global reward presented in Section 4.2 is equivalent to a level-weighted global

reward with $\alpha = 1.0$. We explicitly write the level-weighted global reward below.

$$\text{Level-Weighted Global Reward} = \frac{\sum_{N \geq 0} \alpha^N \sum_{x \in L(N)} \delta(x)}{\sum_{N \geq 0} \alpha^N |L(N)|},$$

where $L(N)$ is the set of arcs in the N-th level of the gold tree and $\delta(x)$ indicates whether arc x exists in the parsed sentence.

Figure 3 summarizes the performance of the level-weighted global reward for $\alpha = 1.0, 0.9, 0.8, 0.7, 0.6, 0.5$. It appears that our intuition is incorrect – the performance steadily worsens as α is decreased from 1.0. We conclude that the level-weighted global reward is ineffective.

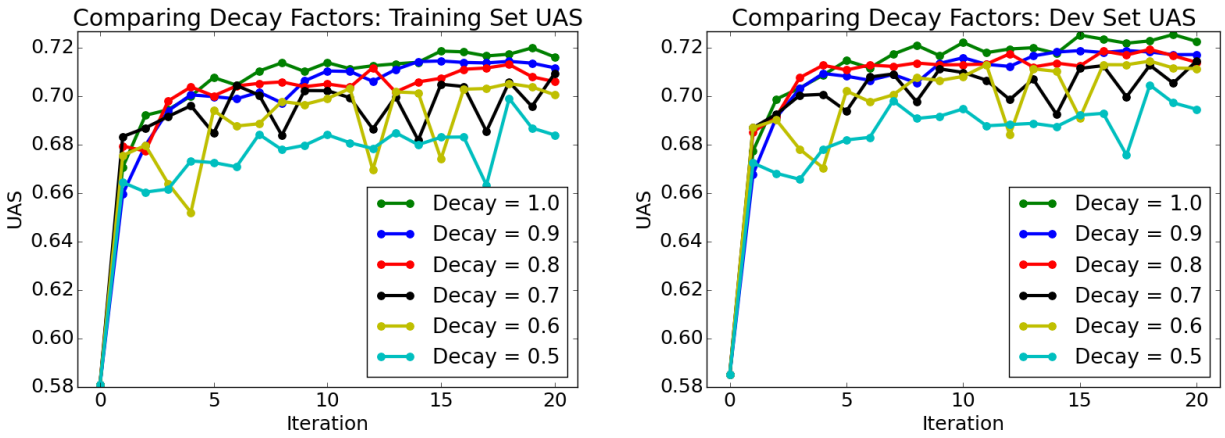

Figure 3: Comparing level-weighted decay factors, training (left) and development (right) performance.

5 Modified Chen & Manning Parser

5.1 Model Characterization

The original model [1] is a transition-based dependency parser that uses a neural network to decide what transition to make. It is designed to draw labeled arcs, but we train it for unlabeled parsing only. The classifier consists of a neural network with 3 layers – an input layer, a hidden layer, and a softmax layer. In our RL framework, we view the softmax output as a policy. The model uses embeddings for the words and POS tags of elements in the sentence configuration. The word embeddings are pre-trained inputs to the training framework. For both the original model and our RL model, we use the 300-dimensional word embeddings (word2vec) published by Google[1] for all words that appear in the training set. To match the dimensionality used by Chen & Manning, we collapse the 300-dimensional Google embeddings into 50-dimensions using PCA. We keep the other hyperparameters - the Adagrad learning rate, the number of hidden units, and regularization coefficients - at their original values [1].

5.2 RL Framework Details

The original model is learned exclusively via oracle training (described in Section 3). Section 3 provides a high-level outline of our reinforcement learning framework, and here we discuss details specific to the modified neural parser.

Like in the toy model, we first initialize the model parameters via oracle training. Instead of running oracle training until convergence, we only run oracle training until the model achieves 60% UAS on the dev

[1] `https://code.google.com/p/word2vec/`

1. Initialize θ with "Oracle" training until a UAS of 60%.
2. **for** i in $1, 2, 3, ...$ NumIter **do**
   for Sentence S in TrainSet **do**
   goldTree $\leftarrow$ GoldTree(S), currentTree $\leftarrow$ Parse(S)
   DefaultAccuracy $\leftarrow$ Score(goldTree, currentTree)
   if DefaultAccuracy $<$ threshold(parser Dev UAS) **then**
   config $\leftarrow$ InitConfig(S), $\Delta_{\text{global}} \leftarrow \mathbf{0}$
   while NotDone(config) **do**
   trans $\sim$ Policy(θ; config)
   $\Delta_{\text{global}} \leftarrow \Delta_{\text{global}} + \eta \nabla_\theta log P(trans|config)$
   config $\leftarrow$ UpdateConfig(config, trans)
   SampleAccuracy $\leftarrow$ Score(goldTree, Tree(config))
   if SampleAccuracy $>$ DefaultAccuracy **then**
   $\text{r}_{\text{global}} \leftarrow ScaleFactor * (SampleAccuracy - DefaultAccuracy)$
   $\theta \leftarrow \theta + \text{r}_{\text{global}} \times \Delta_{\text{global}}$
   end if
   end if

Figure 4: RL algorithm outline for the modified neural parser.

set. We chose 60% somewhat arbitrarily – if we run too few iterations of oracle training, θ might be far from optimal and our RL framework might be extremely slow. If we run oracle trianing for too many iterations, we might constrain ourselves to a local optimum identified by oracle training. Additionally, it would become more difficult to compare our RL framework against the original training methods.

After oracle training, we run many iterations of RL. In our RL formulation, a training example consists of a sentence along with its gold tree. We first parse the sentence with the current model, taking the highest-probability transition at each step. If the parsed tree is already highly accurate, then it is skipped, since there is likely little improvement for RL to make. Otherwise, we sample a history (sequence of configuration states and transitions), treating the softmax layer as a non-deterministic policy. This parse tree is scored against the gold tree and a reward is computed. Based on that reward, the parameters are updated to encourage or discourage the sequence of actions that generated the reward. We only reward improved accuracies and do not update the parameters when we see a worse tree. This is because improved trees have a much stronger signal compared to worse trees.

We define the accuracy of a parsed sentence as the percentage of correct arcs. The accuracy threshold that determines whether a sentence is skipped increases across the RL iterations – it is determined by the parser performance on the dev set during a previous iteration. One advantage of introducing this threshold is a reduction in computation time. If the parser is able to parse a sentence accurately by taking the highest-probability actions, it is unlikely that a randomly sampled sequence of actions will outperform it. By skipping these sentence altogether we can save the time required to sample a history and calculate its reward.

We provide pseudocode in Figure 4, which is a detailed adaptation of Figure 1. We experimented with other sampling frameworks, e.g. sampling from the policy 95% of the time and sampling a random action 5% of the time, but they did not improve performance.

5.3 Global Reward Function

We initially experimented with both local and global reward functions. Afterwards, however, we decided to adopt only a global reward to avoid confounding factors. The original parser is trained with a local objective function, and adopting only a global reward allows us to more easily evaluate the merits of RL. Unfortunately, we believe that relying solely on a global reward decreases the rate of learning and requires more training iterations.

Below is our reward function. SampleAccuracy is the sentence accuracy achieved by sampling from the

policy, while DefaultAccuracy is the sentence accuracy achieved by taking the highest-probability transitions. We only update the parameters if the sampled history outperforms the history generated by the highest-probability transitions. See Section 4.2 for a discussion of desirable properties of a global reward function. The reward is scaled by a 0.05 per-sentence step size.

$$\text{Reward} = \begin{cases} \text{SampleAccuracy} - \text{DefaultAccuracy} & \text{if SampleAccuracy} > \text{DefaultAccuracy} \\ 0 & \text{o.w.} \end{cases}$$

5.4 Results

Figure 5 summarizes development data set performance over a number of iterations. The original neural parser is trained with 5,000 iterations of oracle training. The RL-modified neural parser is trained with 20,000 iterations of RL after oracle training to 60% UAS. At the end of the iterations, the original neural parser achieves ∼86%, while the modified parser achieves ∼83% on the test set.

While the RL-modified parser underperforms the original parser, it is unclear if additional iterations would allow the RL-modified parser to perform better. In Figure 5, the performance of the original parser (left) appears to have converged, while the performance of the modified parser (right) appears as if it would improve with additional iterations. Ideally, we would be able to train both models with at least 100,000 iterations, but then the training times would exceed a week.

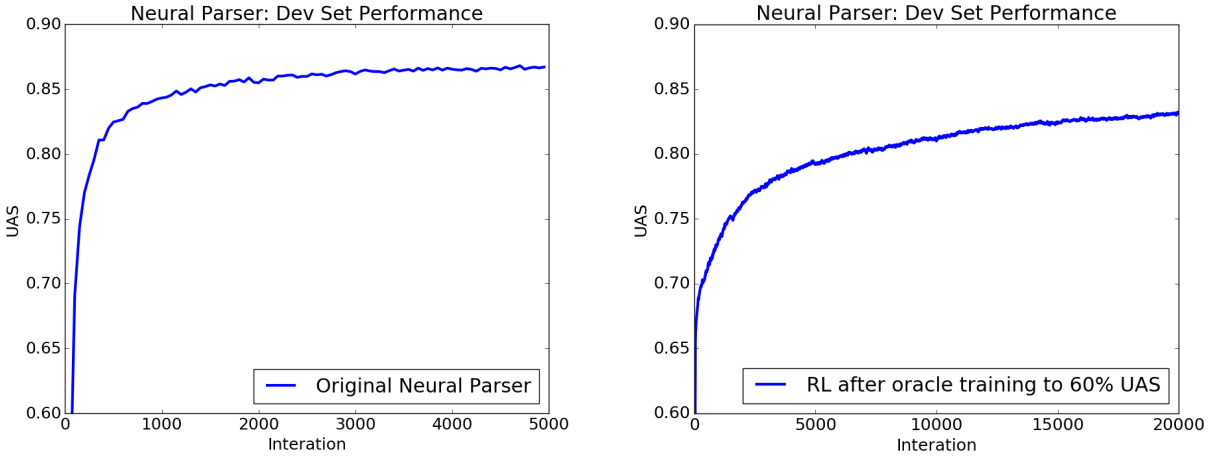

Figure 5: Comparing the original neural parser and the RL neural parser on the dev set.

We were able to try several variants (reward functions, etc.) of our RL framework, but the day-long training times were limiting. Since the performance of our modified performance improves steadily over the iterations, many aspects of our RL framework appear to be functional (but likely not optimal). One reason for the slow learning rate of RL is be that when the parser is already parsing sentences well, a random sample from the policy is most likely worse than taking the highest-likelihood transitions. In such instances, no parameter updates occur. This can be contrasted with oracle updates, where the parameters are updated with every training example.

6 Conclusions

Table 1 summarizes the performance of various models on the test set. We conclude that reinforcement learning is likely a feasible approach to transition-based parsing. Although our RL-modified neural parser is unable to outperform the original neural parser, it is competitive and might outperform the original parser with additional iterations. Furthermore, we see that a global reward greatly outperforms a local reward in our toy parser, a sign that the ability of RL to train long-term decision making might be useful.

Test Set Results (UAS)		
Toy Parser	Local Reward	63.6%
Toy Parser	Global + Local Reward	72.1%
Neural Parser	Original (5000 iterations)	86.0%
Neural Parser	RL (20000 iterations)	83.0%

Table 1: The results of both parsers for UAS on the test set.

Another key advantage of RL is its ability to search more of the feature space. In reinforcement learning, we sample histories at random, encountering configuration states that are unreachable in oracle training. However, there exists a fundamental trade-off between global optimization and training time – random sampling allows us to encounter a wider range of configuration states, but many randomly selected transitions are likely to be sub-optimal.

Improvements upon our reinforcement learning framework will almost surely involve modifications to the reward function. A superior reward function might be able to vastly improve training times. Additionally, incorporating a local reward function to our framework would likely improve performance, as we saw in the toy model. However, the performance of such a framework is more difficult to meaningfully compare against the original model.

7 Contributions

We found a clean separation of responsibility that allowed us to work effectively in parallel. Yuqing took charge of the toy parser implementation. He developed and experimented with different reinforcement learning frameworks, some of which were promising enough to extend to the modified neural parser. Kalki took charge of the neural parser. He studied Chen & Manning's codebase and implemented our reinforcement learning framework within it. This extension of our toy parser framework to the neural parser involved additional adjustments. Near the project deadline, we consolidated our efforts to the neural parser and presentation of the material (poster and report).

References

[1] Danqi Chen and Christopher D Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. *Proceedings of EMNLP 2014*

[2] S.R.K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. 2009. Reinforcement Learning for Mapping Instructions to Actions. *Proceedings of ACL, 2009*