Multi-Task Semantic Dependency Parsing with Policy Gradient for Learning Easy-First Strategies

Kurita, Shuhei; Søgaard, Anders

Published in:
Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics

DOI:
10.18653/v1/P19-1232

Publication date:
2019

Document version
Publisher's PDF, also known as Version of record

Document license:
CC BY

Citation for published version (APA):
Kurita, S., & Søgaard, A. (2019). Multi-Task Semantic Dependency Parsing with Policy Gradient for Learning Easy-First Strategies. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (pp. 2420-2430). Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1232

Multi-Task Semantic Dependency Parsing with Policy Gradient for Learning Easy-First Strategies

Shuhei Kurita
Center for Advanced Intelligence Project
RIKEN
Tokyo, Japan
shuhei.kurita@riken.jp

Anders Søgaard
Department of Computer Science
University of Copenhagen
Copenhagen, Denmark
soegaard@di.ku.dk

Abstract

In Semantic Dependency Parsing (SDP), semantic relations form directed acyclic graphs, rather than trees. We propose a new iterative predicate selection (IPS) algorithm for SDP. Our IPS algorithm combines the graph-based and transition-based parsing approaches in order to handle *multiple* semantic head words. We train the IPS model using a combination of multi-task learning and task-specific policy gradient training. Trained this way, IPS achieves a new state of the art on the SemEval 2015 Task 18 datasets. Furthermore, we observe that policy gradient training learns an easy-first strategy.

1 Introduction

Dependency parsers assign syntactic structures to sentences in the form of trees. Semantic dependency parsing (SDP), first introduced in the SemEval 2014 shared task (Oepen et al., 2014), in contrast, is the task of assigning *semantic* structures in the form of directed acyclic graphs to sentences. SDP graphs consist of binary semantic relations, connecting semantic predicates and their arguments. A notable feature of SDP is that words can be the semantic arguments of multiple predicates. For example, in the English sentence: “The man went back and spoke to the desk clerk” – the word “man” is the subject of the two predicates “went back” and “spoke”. SDP formalisms typically express this by two directed arcs, from the two predicates to the argument. This yields a directed acyclic graph that expresses various relations among words. However, the fact that SDP structures are directed acyclic graphs means that we cannot apply standard dependency parsing algorithms to SDP.

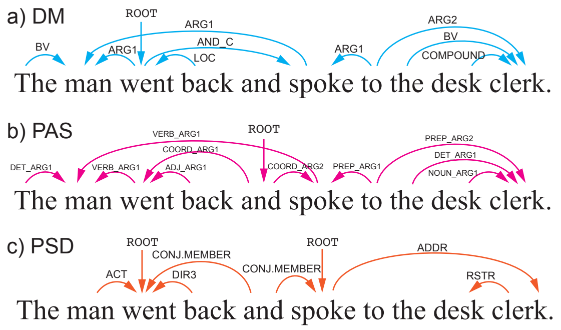

Figure 1: Semantic dependency parsing arcs of DM, PAS and PSD formalisms.

Standard dependency parsing algorithms are often said to come in two flavors: transition-based parsers score transitions between states, and gradually build up dependency graphs on the side. Graph-based parsers, in contrast, score all candidate edges directly and apply tree decoding algorithms for the resulting score table. The two types of parsing algorithms have different advantages (McDonald and Nivre, 2007), with transition-based parsers often having more problems with error propagation and, as a result, with long-distance dependencies. This paper presents a compromise between transition-based and graph-based parsing, called *iterative predicate selection* (IPS) – inspired by head selection algorithms for dependency parsing (Zhang et al., 2017) – and show that error propagation, for this algorithm, can be reduced by a combination of multi-task and reinforcement learning.

Multi-task learning is motivated by the fact that there are several linguistic formalisms for SDP. Fig. 1 shows the three formalisms used in the shared task. The DELPH-IN MRS (DM) formalism derives from DeepBank (Flickinger et al., 2012) and minimal recursion semantics (Copestake et al., 2005). Predicate-Argument Structure (PAS) is a formalism based on the Enju HPSG parser (Miyao et al., 2004) and is generally considered slightly more syntactic of nature than the

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 2420–2430
Florence, Italy, July 28 - August 2, 2019. ©2019 Association for Computational Linguistics

other formalisms. Prague Semantic Dependencies (PSD) are extracted from the Czech-English Dependency Treebank (Hajič et al., 2012). There are several overlaps between these linguistic formalisms, and we show below that parsers, using multi-task learning strategies, can take advantage of these overlaps or synergies during training. Specifically, we follow Peng et al. (2017) in using multi-task learning to learn representations of parser states that generalize better, but we go beyond their work, using a new parsing algorithm and showing that we can subsequently use reinforcement learning to prevent error propagation and tailor these representations to specific linguistic formalisms.

Contributions In this paper, (i) we propose a new parsing algorithm for semantic dependency parsing (SDP) that combines transition-based and graph-based approaches; (ii) we show that multi-task learning of state representations for this parsing algorithm is superior to single-task training; (iii) we improve this model by task-specific policy gradient fine-tuning; (iv) we achieve a new state of the art result across three linguistic formalisms; finally, (v) we show that policy gradient fine-tuning learns an easy-first strategy, which reduces error propagation.

2 Related Work

There are generally two kinds of dependency parsing algorithms, namely transition-based parsing algorithms (McDonald and Nivre, 2007; Kiperwasser and Goldberg, 2016; Ballesteros et al., 2015) and graph-based ones (McDonald and Pereira, 2006; Zhang and Clark, 2008; Galley and Manning, 2009; Zhang et al., 2017). In graph-based parsing, a model is trained to score all possible dependency arcs between words, and decoding algorithms are subsequently applied to find the most likely dependency graph. The Eisner algorithm (Eisner, 1996) and the Chu-Liu-Edmonds algorithm are often used for finding the most likely dependency trees, whereas the AD^3 algorithm (Martins et al., 2011) is used for finding SDP graphs that form DAGs in Peng et al. (2017) and Peng et al. (2018). During training, the loss is computed after decoding, leading the models to reflect a structured loss. The advantage of graph-based algorithms is that there is no real error propagation to the extent the decoding algorithms are global inference algorithm, but this also means that reinforcement learning is not obviously applicable to graph-based parsing. In transition-based parsing, the model is typically taught to follow a gold transition path to obtain a perfect dependency graph during training. This training paradigm has the limitation that the model only ever gets to see states that are on gold transition paths, and error propagation is therefore likely to happen when the parser predicts wrong transitions leading to unseen states (McDonald and Nivre, 2007; Goldberg and Nivre, 2013).

There have been several attempts to train transition-based parsers with reinforcement learning: Zhang and Chan (2009) applied SARSA (Baird III, 1999) to an Arc-Standard model, using SARSA updates to fine-tune a model that was pre-trained using a feed-forward neural network. Fried and Klein (2018), more recently, presented experiments with applying policy gradient training to several constituency parsers, including the RNNG transition-based parser (Dyer et al., 2016). In their experiments, however, the models trained with policy gradient did not always perform better than the models trained with supervised learning. We hypothesize this is due to credit assignment being difficult in transition-based parsing. Iterative refinement approaches have been proposed in the context of sentence generation (Lee et al., 2018). Our proposed model explores multiple transition paths at once and avoids making risky decisions in the initial transitions, in part inspired by such iterative refinement techniques. We also pre-train our model with supervised learning to avoid sampling from irrelevant states at the early stages of policy gradient training.

Several models have been presented for DAG parsing (Sagae and Tsujii, 2008; Ribeyre et al., 2014; Tokgöz and Gülsen, 2015; Hershcovich et al., 2017). Wang et al. (2018) proposed a similar transition-based parsing model for SDP; they modified the possible transitions of the Arc-Eager algorithm (Nivre and Scholz, 2004b) to create multi-headed graphs. We are, to the best of our knowledge, first to explore reinforcement learning for DAG parsing.

3 Model

3.1 Iterative Predicate Selection

We propose a new semantic dependency parsing algorithm based on the head-selection algorithm for syntactic dependency parsing (Zhang et al.,

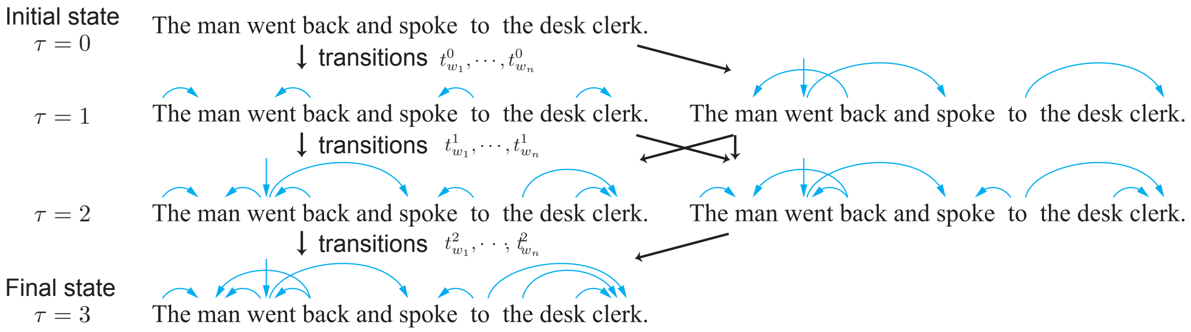
Initial state
$\tau = 0$
The man went back and spoke to the desk clerk.
transitions $t^0_{w_1}, \cdots, t^0_{w_n}$
$\tau = 1$
The man went back and spoke to the desk clerk.
The man went back and spoke to the desk clerk.
transitions $t^1_{w_1}, \cdots, t^1_{w_n}$
$\tau = 2$
The man went back and spoke to the desk clerk.
The man went back and spoke to the desk clerk.
transitions $t^2_{w_1}, \cdots, t^2_{w_n}$
Final state
$\tau = 3$
The man went back and spoke to the desk clerk.

Figure 2: Construction of semantic dependency arcs (DM) in the IPS parsing algorithm. Parsing begins from the initial state and proceeds to the final state following one of several paths. In the left path, the model resolves adjacent arcs first. In contrast, in the right path, distant arcs that rely on the global structure are resolved first.

2017). Head selection iterates over sentences, fixing the head of a word w in each iteration, ignoring w in future iterations. This is possible for dependency parsing because each word has a unique head word, including the root of the sentence, which is attached to an artificial root symbol. However, in SDP, words may attach to multiple head-words or *semantic predicates* whereas other words may not attach to any semantic predicates. Thus, we propose an iterative predicate selection (IPS) parsing algorithm, as a generalization of head-selection in SDP.

The proposed algorithm is formalized as follows. First, we define transition operations for all words in a sentence. For the i-th word w_i in a sentence, the model selects one transition t_i^τ from the set of possible transitions T_i^τ for each transition time step τ. Generally, the possible transitions T_i for the i-th word are expressed as follows:

$$\{\text{NULL}, \text{ARC}_{i,\text{ROOT}}, \text{ARC}_{i,1}, \cdots, \text{ARC}_{i,n}\}$$

where $\text{ARC}_{i,j}$ is a transition to create an arc from the j-th word to the i-th word, encoding that the semantic predicate w_j takes w_i as an semantic argument. NULL is a special transition that does not create an arc. The set of possible transitions T_i^τ for the i-th word at time step τ is a subset of possible transitions T_i that satisfy two constraints: (i) no arcs can be reflexive, i.e., w_i cannot be an argument of itself, and (ii) the new arc must not be a member of the set of arcs A^τ comprising the partial parse graph $\mathbf{y}^\tau$ constructed at time step τ. Therefore, we obtain: $T_i^\tau = T_i / (\text{ARC}_{i,i} \cup A^\tau)$. The model then creates semantic dependency arcs by iterating over the sentence as follows:[1]

1 For each word w_i, select a head arc from T_i^τ.

2 Update the partial semantic dependency graph.

3 If all words select NULL, the parser halts. Otherwise, go to **1**.

Fig. 2 shows the transitions of the IPS algorithm during the DM parsing of the sentence “The man went back and spoke to the desk clerk.” In this case, there are several paths from the initial state to the final parsing state, depending on the orders of creating the arcs. This is known as the non-deterministic oracle problem (Goldberg and Nivre, 2013). In IPS parsing, some arcs are easy to predict; others are very hard to predict. Long-distance arcs are generally difficult to predict, but they are very important for down-stream applications, including reordering for machine translation (Xu et al., 2009). Since long-distance arcs are harder to predict, and transition-based parsers are prone to error propagation, several easy-first strategies have been introduced, both in supervised (Goldberg and Elhadad, 2010; Ma et al., 2013) and unsupervised dependency parsing (Spitkovsky et al., 2011), to prefer some paths over others in the face of the non-deterministic oracle problem. Easy-first principles have also proven effective with sequence taggers (Tsuruoka and Tsujii, 2005; Martins and Kreutzer, 2017). In this paper, we take an arguably more principled approach, *learning* a strategy for choosing transition paths over others using reinforcement learning. We observe, however, that the learned strategies exhibit a clear easy-first preference.

[1] This algorithm can introduce circles. However, circles were extremely rare in our experiments, and can be avoided by simple heuristics during decoding. We discuss this issue in the Supplementary Material, §A.1.

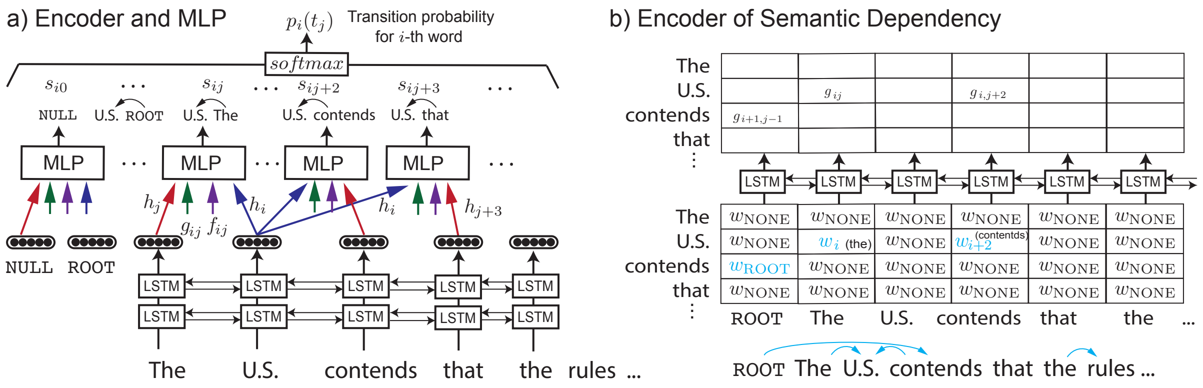

Figure 3: Our network architecture: (a) The encoder of the sentence into the hidden representations h_i and h_j, and the MLP for the transition probabilities. (b) The encoder of the semantic dependency matrix for the representation of h^d_{ij}. The MLP also takes the arc flag representation f_{ij} (see text for explanation).

3.2 Neural Model

Fig. 3 shows the overall neural network. It consists of an encoder for input sentences and partial SDP graphs, as well as a multi-layered perceptron (MLP) for the semantic head-selection of each word.

Sentence encoder We employ bidirectional long short-term memory (BiLSTM) layers for encoding words in sentences. A BiLSTM consists of two LSTMs that reads the sentence forward and backward, and concatenates their output before passing it on. For a sequence of tokens $[w_1, \cdots, w_n]$, the inputs for the encoder are words, POS tags and lemmas.[2] They are mapped to the same p-dimensional embedding vectors in a look-up table. Then they are concatenated to form $3p$-dimensional vectors and used as the input of BiLSTMs. We denote the mapping function of tokens into $3p$-dimensional vectors as $u(w_*)$ for later usages. Finally, we obtain the hidden representations of all words $[h(w_1), \cdots, h(w_n)]$ from the three-layer BiLSTMs. We use three-layer stacked BiLSTMs. We also use special embeddings h_{NULL} for the NULL transition and h_{ROOT} for the ROOT of the sentence.

Encoder of partial SDP graphs The model updates the partial SDP graph at each time step of the parsing procedure. The SDP graph $\mathbf{y}^\tau$ at time step τ is stored in a semantic dependency matrix $G^\tau \in \{0,1\}^{n\times(n+1)}$ for a sentence of n words.[3] The rows of the matrix G represent arguments and the columns represent head-candidates, including the ROOT of the sentence, which is represented by the first column of the matrix. For each transition for a word, the model fills in one cell in a row, if the transition is not NULL. In the initial state, all cells in G are 0. A cell $G[i,j]$ is updated to 1, when the model predicts that the $(i-1)$-th word is an argument of the j-th word or ROOT when $j=0$.

We convert the semantic dependency matrix G into a rank three tensor $G' \in \mathbb{R}^{n\times(n+1)\times p}$, by replacing elements with embeddings of tokens $u(w_*)$ by

$$g'_{ij} = \begin{cases} u(w_{j-1}) & (g_{ij} = 1) \\ u(w_{\text{NONE}}) & (g_{ij} = 0) \end{cases} \tag{1}$$

where $g_{ij} \in G$ and $g'_{ij} \in G'$. g'_{i*} contains the representations of the semantic predicates for the i-th word in the partial SDP graph. We use a single layer Bi-LSTM to encode the semantic predicates g'_{i*} of each word; see Fig. 3 (b). Finally, we concatenate the hidden representation of the NULL transition and obtain the partial SDP graph representation G^τ of the time step τ:

$$G^\tau = [g^\tau_{\text{NULL}}, g^\tau_{*,1}, \cdots, g^\tau_{*,n+1}] \tag{2}$$

We also employ dependency flags that directly encode the semantic dependency matrix and indicate whether the corresponding arcs are already created or not. Flag representations F' are also three-rank tensors, consisting of two hidden representations: f_{ARC} for $g_{i,j} = 1$ and f_{NOARC} for $g_{i,j} = 0$ depending on G. f_{ARC} and f_{NOARC} is q-dimensional vectors. Then we concatenate the hidden representation of the NULL transition and

[2] In the analysis of our experiments, we include an ablation test, where we leave out lemma information for a more direct comparison with one of our baselines.

[3] In this subsection, we omit the time step subscription τ of the partial SDP graph from some equations for simplicity.

obtain the flag representation F^τ:

$$F^\tau = [f^\tau_{\text{NULL}}, f^\tau_{*,1}, \cdots, f^\tau_{*,n+1}] \qquad (3)$$

. We do not use BiLSTMs to encode these flags. These flags also reflect the current state of the semantic dependency matrix.

Predicate selection model The semantic predicate selection model comprises an MLP with inputs from the encoder of the sentence and the partial semantic dependency graph: the sentence representation H, the SDP representation G^τ, and the dependency flag F^τ. They are rank three tensors and concatenated at the third axis. Formally, the score s_{ij} of the i-th word and the j-th transition is expressed as follows.

$$s^\tau_{ij} = \text{MLP}([h_i, h_j, g^\tau_{ij}, f^\tau_{ij}]) \qquad (4)$$

For the MLP, we use a concatenation of outputs from three different networks: a three-layer MLP, a two-layer MLP and a matrix multiplication with bias terms as follows.

$$\begin{aligned} \text{MLP}(\mathbf{x}) = {} & W^3_3 a\big(W^3_2 a(W^3_1\mathbf{x} + b^3_1) + b^3_2\big) \\ & + W^2_2 a(W^2_1\mathbf{x} + b^2_2) + W^1_1\mathbf{x} + b^1_1 \end{aligned}$$

$W^*_{*'}$ are matrices or vectors used in this MLP and $W^*_{*'}$ are bias terms. Here, we use this MLP for predicting a scalar score s_{ij}; therefore, W^3_3, W^2_2, W^1_1 are vectors. The model computes the probability of the transition t_j for each word i by applying a softmax function over the candidates of the semantic head words w_j.

$$p_i(t^\tau_j) = \text{softmax}_j(s^\tau_{ij}) \qquad (5)$$

These transition probabilities $p_i(t_j)$ of selecting a semantic head word w_j, are defined for each word w_i in a sentence.

For supervised learning, we employ a cross entropy loss

$$L^\tau(\theta) = -\sum_{i,j} l_i \log p_i(t^\tau_j | G^\tau) \qquad (6)$$

for the partial SDP graph G^τ at time step τ. Here l_i is a gold transition label for the i-th word and θ represents all trainable parameters. Note that this supervised training regime, as mentioned above, does not have a principled answer to the non-deterministic oracle problem (Goldberg and Nivre, 2013), and samples transition paths randomly from those consistent with the gold anntoations to create transition labels.

Algorithm 1 Policy gradient learning for IPS Algorithm

```
Input: Sentence $\mathbf{x}$ with an empty parsing tree $\mathbf{y}^0$.
  Let a time step $\tau = 0$ and finish flags $f_* = 0$.
  for $0 \leq \tau <$ the number of maximum iterations do
    Compute $\pi^\tau$ and argmax transitions $\hat{t}_i = \arg\max \pi^\tau_i$.
    if $\forall i$ ; $\hat{t}^\tau_i = \text{NULL}$ then
      break
    end if
    for $i$-th word in a sentence do
      if check a finish flag $f_i = 1$ then
        continue
      end if
      if all arcs to word $i$ are correctly created in $\mathbf{y}^\tau$ and
      $\hat{t}_i = \text{NULL}$ then
        Let a flag $f = 1$
        continue
      end if
      Sample $t^\tau_i$ from $\pi^\tau_i$.
      Update the parsing tree $\mathbf{y}^\tau$ to $\mathbf{y}^{\tau+1}$.
      Compute a new reward $r^\tau_i$ from $\mathbf{y}^\tau$, $\mathbf{y}^{\tau+1}$ and $\mathbf{y}^g$.
    end for
    Store a tuple of the state, transitions and rewards for
    words $\{\mathbf{y}^\tau, t^\tau_*, r^\tau_*\}$.
  end for
  Shuffle tuples of $\{\mathbf{y}^\tau, t^\tau_*, r^\tau_*\}$ for a time step $\tau$.
  for a tuple $\{\mathbf{y}^{\tau'}, t^\tau_*, r^{\tau'}_*\}$ of time step $\tau'$ do
    Compute gradient and update parameters.
  end for
```

Labeling model We also develop a semantic dependency labeling neural network. This neural network consists of three-layer stacked BiLSTMs and a MLP for predicting a semantic dependency label between words and their predicates. We use a MLP that is a sum of the outputs from a three-layer MLP, a two-layer MLP and a matrix multiplication. Note that the output dimension of this MLP is the number of semantic dependency labels. The input of this MLP is the hidden representations of a word i and its predicates j: $[h_i, h_j]$ extracted from the stacked BiLSTMs. The score $s'_{ij}(l)$ of the label l for the arc from predicate j to word i is predicted as follows.

$$s'_{ij}(l) = \text{MLP}'([h_i, h_j]) \qquad (7)$$

We minimize the softmax cross entropy loss using supervised learning.

3.3 Reinforcement Learning

Policy gradient Reinforcement learning is a method for learning to iteratively act according to a dynamic environment in order to optimize future rewards. In our context, the agent corresponds to the neural network model predicting the transition probabilities $p_i(t^\tau_j)$ that are used in the parsing algorithm. The environment includes the partial SDP graph $\mathbf{y}^\tau$, and the rewards r^τ are computed by comparing the predicted parse graph to the gold parse graph $\mathbf{y}^g$.

We adapt a variation of the policy gradient method (Williams, 1992) for IPS parsing. Our objective function is to maximize the rewards

$$J(\theta) = E_\pi \left[r_i^\tau \right] \tag{8}$$

and the transition policy for the i-th word is given by the probability of the transitions $\pi \sim p_i(t_j^\tau | \mathbf{y}^\tau)$. The gradient of Eq.8 is given as follows:

$$\nabla J(\theta) = E_\pi \left[r_i^\tau \nabla \log p_i(t_j^\tau | \mathbf{y}^\tau) \right] \tag{9}$$

When we compute this gradient, given a policy π, we approximate the expectation E_π for any transition sequence with a single transition path $\mathbf{t}$ that is sampled from policy π:

$$\nabla J(\theta) \approx \sum_{t_j^\tau \in \mathbf{t}} [r_i^\tau \nabla \log p_i(t_j^\tau | \mathbf{y}^\tau)] \tag{10}$$

We summarize our policy gradient learning algorithm for SDP in Algorithm 1. For time step τ, the model samples one transition t_j^τ selecting the j-th word as a semantic head word of the i-th word, from the set of possible transitions T_i, following the transition probability of π. After sampling t_j^τ, the model updates the SDP graph to $\mathbf{y}^{\tau+1}$ and computes the reward r_i^τ. When NULL becomes the most likely transition for all words, or the time step exceeds the maximum number of time steps allowed, we stop.[4] For each time step, we then update the parameters of our model with the gradients computed from the sampled transitions and their rewards.[5]

Note how the cross entropy loss and the policy gradient loss are similar, if we do not sample from the policy π, and rewards are non-negative. However, these are the important differences between supervised learning and reinforcement learning: (1) Reinforcement learning uses sampling of transitions. This allows our model to explore transition paths that supervised models would never follow. (2) In supervised learning, decisions are independent of the current time step τ, while in reinforcement learning, decisions depend on τ. This means that the θ parameters are updated *after* the parser finishes parsing the input sentence. (3) Loss must be non-negative in supervised learning, while rewards can be negative in reinforcement learning.

In general, the cross entropy loss is able to optimize for choosing good transitions given a parser configuration, while the policy gradient objective function is able to optimize the entire sequence of transitions drawn according to the current policy. We demonstrate the usefulness of reinforcement learning in our experiments below.

Reward	Transitions
$r_i^\tau = 1$	(1) The model creates a new correct arc from a semantic predicate to the i-th word. (2) The first time the model chooses the NULL transition after all gold arcs to the i-th word have been created, and no wrong arcs to the i words have not been created.
$r_i^\tau = -1$	(3) The model creates a wrong arc from a semantic predicate candidate to the i-th word.
$r_i^\tau = 0$	(4) All other transitions.

Table 1: Rewards in SDP policy gradient.

Rewards for SDP We also introduce intermediate rewards, given during parsing, at different time steps. The reward r_i^τ of the i-th word is determined as shown in Table 1. The model gets a positive reward for creating a new correct arc to the i-th word, or if the model for the first time chooses a NULL transition after all arcs to the i-th word are correctly created. The model gets a negative reward when the model creates wrong arcs. When our model chooses NULL transitions for the i-th word before all gold arcs are created, the reward r_i^τ becomes 0.

3.4 Implementation Details

This section includes details of our implementation.[6] We use 100-dimensional, pre-trained Glove (Pennington et al., 2014) word vectors. Words or lemmas in the training corpora that do not appear in pre-trained embeddings are associated with randomly initialized vector representations. Embeddings of POS tags and other special symbol are also randomly initialized. We apply Adam as our optimizer. Preliminary experiments show that mini-batching led to a degradation in performance. When we apply policy gradient, we pre-train our model using supervised learning. We then use policy gradient for task-specific fine-tuning of our model. We find that updating parameters of BiLSTM and word embeddings during policy gradient

[4] We limit the number of transitions during training, but not at test time.

[5] We update the parameters for each time step to reduce memory requirements.

[6] The code is available at https://github.com/shuheikurita/semrl

Name	Value
Encoder BiLSTM hidden layer size	600
Dependency LSTM hidden layer size	200
The dimensions of embeddings p,q	100, 128
MLPs hidden layer size	4000
Dropout rate in MLPs	0.5
Max transitions during reinforcement learning	10

Table 2: Hyper-parameters in our experiments.

Model	DM	PAS	PSD	Avg.
Peng+ 17 Freda3	90.4	92.7	78.5	88.0
Wang+ 18 Ens.	90.3	91.7	78.6	86.9
Peng+ 18	91.6	-	78.9	-
IPS	91.1	92.4	78.6	88.2
IPS +ML	91.2	92.5	78.8	88.3
IPS +RL	91.6‡	**92.8**‡	79.2‡	88.7‡
IPS +ML +RL	**92.0**‡	**92.8**‡	**79.3**‡	**88.8**‡

Table 3: Labeled parsing performance on in-domain test data. Avg. is the micro-averaged score of three formalisms. ‡ of the *+RL* models represents that the scores are statistically significant at $p < 10^{-3}$ with their non-*RL* counterparts.

makes training quite unstable. Therefore we fix the BiLSTM parameters during policy gradient. In our multi-task learning set-up, we apply multi-task learning of the shared stacked BiLSTMs (Søgaard and Goldberg, 2016; Hashimoto et al., 2017) in supervised learning. We use task-specific MLPs for the three different linguistic formalisms: DM, PAS and PSD. We train the shared BiLSTM using multi-task learning beforehand, and then we fine-tune the task-specific MLPs with policy gradient. We summarize the rest of our hyper-parameters in Table 2.

4 Experiments

We use the SemEval 2015 Task18 (Oepen et al., 2015) SDP dataset for evaluating our model. The training corpus contains 33,964 sentences from the WSJ corpus; the development and in-domain test were taken from the same corpus and consist of 1,692 and 1,410 sentences, respectively. The out-of-domain test set of 1,849 sentences is drawn from Brown corpus. All sentences are annotated with three semantic formalisms: DM, PAS and PSD. We use the standard splits of the datasets (Almeida and Martins, 2015; Du et al., 2015). Following standard evaluation practice in semantic dependency parsing, all scores are *micro-averaged* F-measures (Peng et al., 2017; Wang et al., 2018) with labeled attachment scores (LAS).

Model	DM	PAS	PSD	Avg.
Peng+ 17 Freda3	85.3	**89.0**	76.4	84.4
Peng+ 18	86.7	-	77.1	-
IPS +ML	86.0	88.2	77.2	84.6
IPS +ML +RL	**87.2**‡	88.8‡	**77.7**‡	**85.3**‡

Table 4: Labeled parsing performance on out-of-domain test data. Avg. is the micro-averaged score of three formalisms. ‡ of the *+RL* models represents that the scores are statistically significant at $p < 10^{-3}$ with their non-*RL* counterparts.

The system we propose is the IPS parser trained with a multi-task objective and fine-tuned using reinforcement learning. This is referred to as *IPS+ML+RL* in the results tables. To highlight the contributions of the various components of our architecture, we also report ablation scores for the IPS parser without multi-task training nor reinforcement learning (*IPS*), with multi-task training (*IPS+ML*) and with reinforcement learning (*IPS+RL*). At inference time, we apply heuristics to avoid predicting circles during decoding (Camerini et al., 1980); see Supplementary Material, §A.1. This improves scores by 0.1 % or less, since predicted circles are extremely rare. We compare our proposed system with three state-of-the-art SDP parsers: Freda3 of Peng et al. (2017), the ensemble model in Wang et al. (2018) and Peng et al. (2018). In Peng et al. (2018), they use syntactic dependency trees, while we do not use them in our models.[7]

The results of our experiments on in-domain dataset are also shown in Table 3. We observe that our basic *IPS* model achieves competitive scores in DM and PAS parsing. Multi-task learning of the shared BiLSTM (*IPS+ML*) leads to small improvements across the board, which is consistent with the results of Peng et al. (2017). The model trained with reinforcement learning (*IPS+RL*) performs better than the model trained by supervised learning (*IPS*). These differences are significant ($p < 10^{-3}$). Most importantly, the combination of multi-task learning and policy gradient-based reinforcement learning (*IPS+ML+RL*) achieves the best results among all IPS models and the previous state of the art models, by some margin. We also obtain similar results for the out-of-domain

[7] Dozat and Manning (2018) report *macro-averaged* scores instead, as mentioned in their ACL 2018 talk, and their results are therefore not comparable to ours. For details, see the video of their talk on ACL2018 that is available on Vimeo.

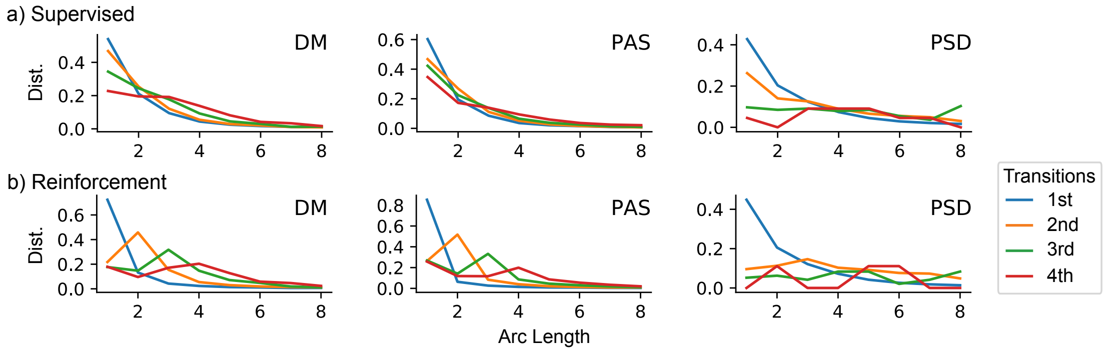

Figure 4: Arc length distributions: (a) Supervised learning (*IPS+ML*). (b) Reinforcement learning (*IPS+ML+RL*). The four lines correspond to the first to fourth transitions in the derivations.

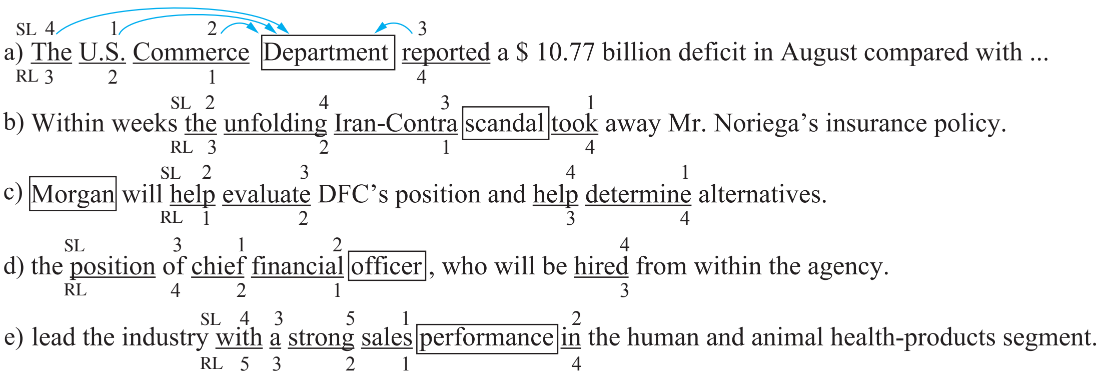
SL 4 1 2 3
a) The U.S. Commerce Department reported a $ 10.77 billion deficit in August compared with ...
RL 3 2 1 4
SL 2 4 3 1
b) Within weeks the unfolding Iran-Contra scandal took away Mr. Noriega's insurance policy.
RL 3 2 1 4
SL 2 3 4 1
c) Morgan will help evaluate DFC's position and help determine alternatives.
RL 1 2 3 4
SL 3 1 2 4
d) the position of chief financial officer, who will be hired from within the agency.
RL 4 2 1 3
SL 4 3 5 1 2
e) lead the industry with a strong sales performance in the human and animal health-products segment.
RL 5 3 2 1 4

Figure 5: Examples of clauses parsed with DM formalism. The underlined words are the semantic predicates of the argument words in rectangles in the annotation. The superscript numbers (SL) are the orders of creating arcs by *IPS+ML* and the subscript numbers (RL) are the orders by *IPS+ML+RL*. In the clause (a), we show a partial SDP graph to visualize the SDP arcs.

Model	DM	PAS	PSD	Avg.
Peng+ 17 Freda3	90.4	92.5	78.5	88.0
IPS +ML -Lemma	90.7	92.3	78.3	88.0
IPS +ML +RL -Lemma	**91.2**‡	**92.9**‡	**78.8**‡	**88.5**‡

Table 5: Evaluation of our parser when *not* using lemma embeddings (for a more direct comparison with Freda3), on in-domain test datasets. ‡ of *+RL* models represents that the scores are statistically significant at $p < 10^{-3}$ with their non-*RL* counterparts.

datasets, as shown in Table 4. All improvements with reinforcement learning are also statistically significant ($p < 10^{-3}$).

Evaluating Our Parser without Lemma Since our baseline (Peng et al., 2017) does not rely on neither lemma or any syntactic information, we also make a comparison of *IPS+ML* and *IPS+ML+RL* trained with word and POS embeddings, but without lemma embeddings. The results are given in Table 5. We see that our model is still better on average and achieves better performance on all three formalisms. We also notice that the lemma information does not improve the performance in the PAS formalism.

Effect of Reinforcement Learning Fig. 4 shows the distributions of the length of the created arcs in the first, second, third and fourth transitions for all words, in the various IPS models in the development corpus. These distributions show the length of the arcs the models tend to create in the first and later transitions. Since long arcs are harder to predict, an easy-first strategy would typically amount to creating short arcs first.

In supervised learning (*IPS+ML*), there is a slight tendency to create shorter arcs first, but while the ordering is relatively consistent, the differences are small. This is in sharp contrast with the distributions we see for our policy gradient parser (*IPS+ML+RL*). Here, across the board, it is very likely that the first transition connects neighboring words; and very unlikely that neighboring words are connected at later stages. This suggests that reinforcement learning learns an easy-first strategy of predicting short arcs first. Note that unlike easy-first algorithms in syntactic parsing (Goldberg and Nivre, 2013), we do not hardwire an easy-first strategy into our parser; but rather, we learn it from the data, because it optimizes our long-term rewards. We present further analyses and analyses on WSJ syntactic dependency trees in Appendix A.2.

Fig. 5 shows four sentence excerpts from the development corpus, and the order in which arcs are created. We again compare the model trained with supervised learning (*IPS+ML* notated as `SL` here) to the model with reinforcement learning (*IPS+ML+RL* notated as `RL` here). In examples (a) and (b), the `RL` model creates arcs inside noun phrases first and then creates arcs to the verb. The `SL` model, in contrast, creates arcs with inconsistent orders. There are lots of similar examples in the development data. In clause (c), for example, it seems that the `RL` model follows a grammatical ordering, while the `SL` model does not. In the clause (d), it seems that the `RL` model first resolves arcs from modifiers, in "*chief financial officer*", then creates an arc from the adjective phrase "*, who will be hired*", and finally creates an arc from the external phrase "*the position of*". Note that both the `SL` and `RL` models make an arc from "*of*" in stead of the annotated label of the word "*position*" in the phrase "*the position of*". In the clause (e), the `RL` model resolve the arcs in the noun phrase "*a strong sales performance*" and then resolve arcs from the following prepositional phrase. Finally, the `RL` model resolve the arc from the word "*with*" that is the headword in the syntactic dependency tree. In the example (d) and (e), the `RL` model elaborately follows the syntactic order that are not given in any stages of training and parsing.

5 Conclusion

We propose a novel iterative predicate selection (IPS) parsing model for semantic dependency parsing. We apply multi-task learning to learn general representations of parser configurations, and use reinforcement learning for task-specific fine-tuning. In our experiments, our multi-task reinforcement IPS model achieves a new state of the art for three SDP formalisms. Moreover, we show that fine-tuning with reinforcement learning learns an easy-first strategy and some syntactic features.

Acknowledgements

This work was done when Shuhei Kurita visited the University of Copenhagen. Shuhei Kurita was supported by JST ACT-I Grant Number JPMJPR17U8, Japan and partly supported by JST CREST Grant Number JPMJCR1301, Japan. Anders Søgaard was supported by a Google Focused Research Award.

References

M. Almeida and A. Martins. 2015. Lisbon: Evaluating turbosemanticparser on multiple languages and out-of-domain data.

Leemon C. Baird III. 1999. Reinforcement learning through gradient descent. School of Computer Science Carnegie Mellon University.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of the EMNLP*, pages 349–359.

P. M. Camerini, L. Fratta, and F. Maffioli. 1980. The k best spanning arborescences of a network. *Networks*, 10:91–110.

Ann Copestake, Dan Flickinger, Ivan A. Sag, and Carl Pollard. 2005. Minimal recursion semantics: An introduction. In *Research on Language & Computation*, pages 3(4):281–332.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the ACL (Short Papers)*, pages 484–490.

Yantao Du, Fan Zhang, Xun Zhang, Weiwei Sun, and XiaojunWan. 2015. Peking: Building semantic dependency graphs with a hybrid parser.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the NAACL: HLT*, pages 199–209, San Diego, California.

J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *COLING*.

Daniel Flickinger, Yi Zhang, and Valia Kordoni. 2012. Deepbank: Adynamically annotated treebank of the wall street journal. In *In Proc. of TLT*.

Daniel Fried and Dan Klein. 2018. Policy gradient as a proxy for dynamic oracles in constituency parsing. In *Proceedings of the ACL*, pages 469–476.

Michel Galley and Christopher D. Manning. 2009. Quadratic-time dependency parsing for machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the*

4th International Joint Conference on Natural Language Processing of the AFNLP, pages 773–781. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: NAACL*, pages 742–750, Los Angeles, California.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. pages 403–414.

Jan Hajič, Eva Hajičová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fučíková, Marie Mikulová, Petr Pajas, Jan Popelka, Jiří Semecký, Jana Šindlerová, Jan Štěpánek, Josef Toman, Zdeňka Urešová, and Zdeněk Žabokrtský. 2012. Announcing prague czech-english dependency treebank 2.0. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 3153–3160.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2017. A joint many-task model: Growing a neural network for multiple nlp tasks. In *Proceedings of the EMNLP*, pages 1923–1933.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for ucca. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1127–1138. Association for Computational Linguistics.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *TACL*, 4:313–327.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182. Association for Computational Linguistics.

Ji Ma, Jingbo Zhu, Tong Xiao, and Nan Yang. 2013. Easy-first POS tagging and dependency parsing with beam search. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 110–114, Sofia, Bulgaria. Association for Computational Linguistics.

Andre Martins, Noah Smith, Mario Figueiredo, and Pedro Aguiar. 2011. Dual decomposition with many overlapping components. In *Proceedings of the 2011 Conference on EMNLP*, pages 238–249, Edinburgh, Scotland, UK.

André F. T. Martins and Julia Kreutzer. 2017. Learning what's easy: Fully differentiable neural easy-first taggers. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 349–362, Copenhagen, Denmark. Association for Computational Linguistics.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on EMNLP-CoNLL*, pages 122–131.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *11th Conference of the European Chapter of the Association for Computational Linguistics*.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi. Tsujii. 2004. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the penn treebank. In *In Proceedings of IJCNLP-04*.

Joakim Nivre and Mario Scholz. 2004b. Deterministic dependency parsing of english text. In *Proceedings of Coling 2004*, pages 64–70. COLING.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, and Zdenka Uresova. 2015. Semeval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 915–926, Denver, Colorado. Association for Computational Linguistics.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. 2014. Semeval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 63–72, Dublin, Ireland.

Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep multitask learning for semantic dependency parsing. In *Proceedings of the ACL*, pages 2037–2048, Vancouver, Canada.

Hao Peng, Sam Thomson, and Noah A. Smith. 2018a. Backpropagating through structured argmax using a spigot. In *Proceedings of the 56th Annual Meeting of the ACL*, pages 1863–1873.

Hao Peng, Sam Thomson, Swabha Swayamdipta, and Noah A. Smith. 2018b. Learning joint semantic parsers from disjoint data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1492–1502, New Orleans, Louisiana. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Corentin Ribeyre, Eric Villemonte de la Clergerie, and Djamé Seddah. 2014. Alpage: Transition-based semantic graph parsing with syntactic features. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 97–103, Dublin, Ireland. Association for Computational Linguistics and Dublin City University.

Kenji Sagae and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 753–760. Coling 2008 Organizing Committee.

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the ACL (Short Papers)*, pages 231–235.

Valentin I. Spitkovsky, Hiyan Alshawi, Angel X. Chang, and Daniel Jurafsky. 2011. Unsupervised dependency parsing without gold part-of-speech tags. In *Proceedings of the 2011 Conference on EMNLP*, pages 1281–1290.

Alper Tokgöz and Eryigit Gülsen. 2015. Transition-based dependency dag parsing using dynamic oracles. In *Proceedings of the ACL Student Research Workshop.*, pages 22–27.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 467–474, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Yuxuan Wang, Wanxiang Che, Jiang Guo, and Ting Liu. 2018. A neural transition-based approach for semantic dependency graph parsing. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. pages 5–32. Springer.

Peng Xu, Jaeho Kang, Michael Ringgaard, and Franz Och. 2009. Using a dependency parser to improve smt for subject-object-verb languages. In *Proceedings of HLT:NAACL*, pages 245–253, Boulder, Colorado.

Lidan Zhang and Kwok Ping Chan. 2009. Dependency parsing with energy-based reinforcement learning. In *Proceedings of the IWPT*, pages 234–237, Paris, France.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017. Dependency parsing as head selection. In *Proceedings of the ACL*, pages 665–676, Valencia, Spain.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the EMNLP*, pages 562–571.