Neural Transition Based Parsing of Web Queries: An Entity Based Approach

Rivka Malca and **Roi Reichart**
Technion, Israel Institute of Technology
srikim@st.technion.ac.il, roiri@technion.ac.il

Abstract

Web queries with question intent manifest a complex syntactic structure and the processing of this structure is important for their interpretation. Pinter et al. (2016) has formalized the grammar of these queries and proposed semi-supervised algorithms for the adaptation of parsers originally designed to parse according to the standard dependency grammar, so that they can account for the unique forest grammar of queries. However, their algorithms rely on resources typically not available outside of big web corporates. We propose a new BiLSTM query parser that: (1) Explicitly accounts for the unique grammar of web queries; and (2) Utilizes named entity (NE) information from a BiLSTM NE tagger, that can be jointly trained with the parser. In order to train our model we annotate the query treebank of Pinter et al. (2016) with NEs. When trained on 2500 annotated queries our parser achieves UAS of 83.5% and segmentation F1-score of 84.5, substantially outperforming existing state-of-the-art parsers.[1]

1 Introduction

Web queries, authored by users in order to search for information, form a major gate to the Web, and their correct interpretation is hence invaluable. While earlier research (Bergsma and Wang, 2007; Barr et al., 2008) suggested that many queries are trivial in structure, Pinter et al. (2016) (henceforth PRS16) demonstrated that this is often not the case. Particularly, they demonstrated that queries related to questions that are answered in Community Question Answering (CQA) sites (social QA forums such as Yahoo Answers), follow a complex dependency grammar. As such queries are quite frequent (e.g. an early study (White et al., 2015) showed that they constitute $\sim$10% of all queries issued to search engines) and their interpretation can benefit from structural analysis (e.g. (Tsur et al., 2016)), effective query parsing is of importance.[2]

In order to properly describe the syntactic structure of queries, PRS16 extended the standard dependency grammar so that it accounts for dependency forests that consist of syntactically independent segments, each of which has its own internal dependency tree. Additionally, they constructed a query treebank consisting of 4,000 CQA queries, manually annotated according to the query grammar. Examples of annotated queries from the treebank are given in Figures 2 and 4.

PRS16 presented two algorithms that can adapt off-the-shelf dependency parsers trained on standard edited text, so that they can produce syntactic structures that conform to the query grammar. Importantly, their methods do not contain any change to the states and transitions of the parser. Instead, they require millions of unannotated queries each paired with the title (usually a grammatical question) of the Yahoo Answers question page that was clicked by the user who initiated the query; it is the alignment between the query and the title that provides the training signal for their algorithms. Unfortunately, millions of (query, title) pairs are typically not available outside of big web corporates, and the practical value of the PRS16 algorithms is hence limited. Moreover, despite the unique supervision signal, their parsers achieve a segmentation F1-score of up to 70.4, which leaves a large room for improvement.

In this paper we present a transition-based BiLSTM query parser that requires no distant supervision. Our parser is based on two ideas: (a) We change the standard transition system so that the parser explicitly supports the PRS16 query gram-

[1]Our code and data are available at: `https://bitbucket.org/riki_malca/ptsparser/src/master`.

[2]In the rest of the paper we will refer to queries with question intent, like those addressed in PRS16, simply as queries.

mar (§ 3.2); and (b) Observing that entities are very frequent in CQA queries and provide a strong structural signal (§ 4), we extend our parser to consider information from a named entity (NE) tagger. We explore both sequential and joint training of the NE tagger and the parser and demonstrate the superiority of the joint approach.

As another contribution of this paper, we annotate the dataset of PRS16 with NEs (§ 4). We use this data to establish our observation about the importance of NEs for query parsing, and in order to train and test our entity-aware parser.

We split the PRS16 corpus to train (2500 queries), development (750) and test (750) sections (§ 6). In this training setup our segmentation and entity-aware parser achieves a segmentation F1-score of 84.5 (100 on single-segment queries, 60.7 on multi-segment queries) and a dependency parsing UAS of 83.5. Our model outperforms its simpler variants that do not utilize segmentation and/or NE information. For example, the BiLSTM parser of Kiperwasser and Goldberg (2016), which forms the basis for our model, scores 67.7 in segmentation F1 and 77.0 in UAS.

We note that our training setup is very different than that of PRS16. They trained their parser on edited text from the OntoNotes 5 corpus (Weischedel et al., 2013) augmented with millions of (query, title) pairs, and their test set consists of the 4000 queries of the query treebank, as they do not train on queries.[3] While our work is not directly comparable to theirs, it is worth mentioning that their best model scores 70.4 in segmentation F1 and 76.4 in UAS, much lower than the numbers we report here for our best models.

2 Previous Work

We divide this section to two: We start with works that analyze the structure of queries and the role of NEs in their processing, and then discuss work on parsing of user generated content on the web.

Query structure and entity analysis As noted in PRS16, web queries differ from standard sentences, as they tend to be shorter and have a unique dependency structure. Hence, prior to PRS16 several works have addressed the syntactic structure of web queries. However, all these works were restricted to tasks that are much simpler than full dependency parsing, including POS tagging (Bendersky et al., 2010; Ganchev et al., 2012), phrase chunking (Bendersky et al., 2011), semantic tagging (Manshadi and Li, 2009; Li, 2010) and classification of queries into syntactic classes (Allan and Raghavan, 2002; Barr et al., 2008).

NER has been recognized as a fundamental problem in query processing by Guo et al. (2009), and many works since (e.g. (Alasiry et al., 2012; Eiselt and Figueroa, 2013; Zhai et al., 2016)) explored various models and features for the task. Differently from those works, our goal is to design a BiLSTM model that can be easily integrated with modern BiLSTM parsers. We hence use simple input features and a simple NE scheme (e.g. see (Guo et al., 2009) for more fine-grained distinctions). More sophisticated features, entity schemes and deep learning architectures (e.g. (Lample et al., 2016)) are left for the future.

Syntactic parsing of Web data Only a handful of papers aimed to parse web data. One important example is the shared task of Petrov and McDonald (2012) on parsing web data from the Google Web Treebank, consisting of texts from the email, weblog, CQA, newsgroup, and review domains. Other relevant works are the tweet parsers of Foster et al. (2011), Kong et al. (2014) and Liu et al. (2018). However, all these works did not address the unique properties of web queries with question intent that express information needs in a concise manner (e.g. with one or more phrases or sentence fragments) and follow a forest-based grammar.

PRS16 were the first, and to the best of our knowledge the only work to address the parsing of web queries with question intent. However, as noted in § 1 their algorithms rely on millions of (query, title) pairs, which deems their algorithm impractical for most users. In practice, they started with a query log of 60M Yahoo Answers pages and ended up using 7.5M queries as distant supervision. In this paper we aim to overcome this limitation by introducing a high quality query parser that can train on several thousands annotated queries to provide higher UAS and segmentation F1 figures compared to those reported in PRS16 (see footnote 3 for their training protocol and data).

We finally note that joint parsing and NER was explored in past (Reichart et al., 2008; Finkel and Manning, 2009, 2010), but for edited text, standard grammar and different modeling techniques. Our work re-emphasizes the strong ties between

[3]In some setups they used the segmentation signal from the queries and experimented with a five fold cross-validation over the 4000 queries

NER and parsing, in the context of query analysis.

3 Segmentation-Aware Parsing

In this section we present a parser that explicitly accounts for the query dependency grammar of PRS16. We start (§ 3.1) with a brief description of the BiLSTM parser of Kiperwasser and Goldberg (2016) (henceforth KG16), that forms the basis for our parser, and then describe our query parser.

3.1 The KG16 BiLSTM Parser

KG16 presented a BiLSTM model for transition based dependency parsing (Figure 1). Given a sentence s with words $w_1, ..., w_n$ and corresponding POS tags $p_1, ..., p_n$, the word w_i is represented as:

$$x_i = e(w_i) \circ e(p_i) \tag{1}$$

where $e(w_i)$ and $e(p_i)$ are the embeddings of w_i and p_i, respectively, and $\circ$ is the vector concatenation operator. [4]

The BiLSTM consists of two LSTMs: $LSTM_{forward}$ and $LSTM_{backward}$. Given an input vector x_i, $LSTM_{forward}(x_i)$ captures the past context, and is calculated using the information in the input vectors $x_1, \ldots, x_{i-1}$. Similarly, $LSTM_{backward}(x_i)$ captures the future context and is calculated using the information in the input vectors $x_n, \ldots, x_{i+1}$. v_i, the resulting representation of x_i, is given by:

$$\begin{aligned} v_i &= BiLSTM(x_{1..n}, i) \qquad (2) \\ &= LSTM_{forward}(x_i) \circ LSTM_{backward}(x_i) \end{aligned}$$

The parser implements the arc-hybrid system (Kuhlmann et al., 2011) which uses a configuration $c = (\sigma, \beta, A)$ where $\sigma = [s_0, s_1, ..]$ is a stack, $\beta = [b_0, b_1..]$ is a buffer and A is a set of dependency arcs. The arc-hybrid system allows three transitions: $SHIFT$, $LEFT_{arc}$ and $RIGHT_{arc}$. At each step the parser scores the possible transitions and selects the highest scoring one.

The parser represents each configuration by the concatenation of the BiLSTM embeddings of the first word in the buffer (b_0) and the three words at the top of the stack (s_0, s_1 and s_2). Then, a multi-layer perceptron (MLP) with one hidden layer scores the possible transitions given the current configuration:

$$MLP_\theta(c) = W^2 \cdot tanh(W^1 \cdot c + b^1) + b^2 \tag{3}$$

[4]The embedding vectors are initialized using the Xavier initialization (Glorot and Bengio, 2010) and trained as part of the BiLSTM.

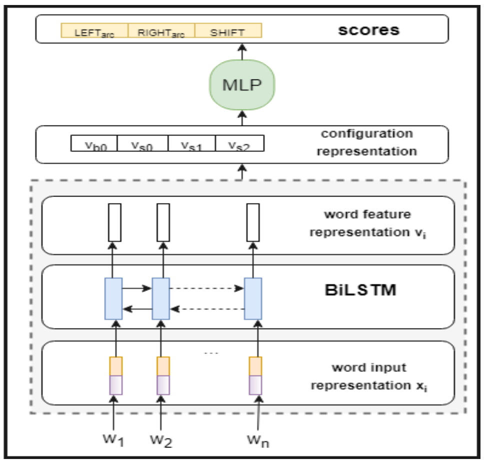

LEFT_arc
RIGHT_arc
SHIFT
scores
MLP
v_b0
v_s0
v_s1
v_s2
configuration representation
word feature representation v_i
BiLSTM
word input representation x_i
w1
w2
wn

Figure 1: A sketch of the KG16 arc-hybrid parser.

where $\theta = W^1, W^2, b^1, b^2$ are the MLP parameters and $c = v_{s_0} \circ v_{s_1} \circ v_{s_2} \circ v_{b_0}$.

The parser employs a margin-based loss (MBL) function at each step:

$$\begin{aligned} MBL = max(0, 1 - \max_{t_o \in G} MLP_\theta(c)[t_o] \\ + \max_{t_p \in T \setminus G} MLP_\theta(c)[t_p]) \end{aligned} \tag{4}$$

where T is the set of possible transitions and G is the set of correct transitions. The losses are summed throughout the parsing of a sentence and the parameters are updated accordingly.

The parser employs a *dynamic oracle* (Goldberg and Nivre, 2013), which enables exploration in training. We next describe our modification of the KG16 parser. Specifically, we change the transition logic so that it can directly account for the query grammar defined in PRS16.

3.2 A Segmentation-Aware BiLSTM Parser

In order for our parser to directly account for the forest-based query grammar of PRS16, we follow previous work (e.g. Nivre (2009)) and modify its set of actions and transition logic. Before we do that, we start with a more standard modification.

An arc-eager KG16 parser The first step in the design of our parser is changing the arc-hybrid system of the KG16 parser to an arc-eager system (Nivre, 2008). To do that we change the definitions of the $RIGHT_{arc}$ and $LEFT_{arc}$ transitions and add a *REDUCE* transition. The (original) arc-hybrid and the (modified) arc-eager KG16 parsers are denoted with P^H and P^E, respectively.

The motivation for this change is the addition of the $REDUCE$ transition that explicitly facilitates segmentation. After the words in the stack σ are reduced, they cannot be connected to the unprocessed words in the buffer β. This state constitutes a *segmentation point*. We use this connection between the $REDUCE$ transition and the segmentation operation as an integral part of our new segmentation-aware parser, and hence all the following parsers extend this arc-eager parser.

A segmentation-aware parser In our parser, denoted as P^B (for BASIC), a configuration $c = (\sigma, \beta_1, \beta_2, A)$ consists of a stack $\sigma = [s_0, s_1, ..]$, two buffers: $\beta_1 = [b_{10}, .., b_{1last}]$ and $\beta_2 = [b_{20}, .., b_{2last}]$, and a dependency arcs set A. The buffers β_1, β_2 contain the unprocessed tokens, and the words within β_2 form the current segment. We expand the configuration representation to include not only the representations of the first token in the buffer β_1 (b_{10}) and the three tokens at the top of the stack σ (s_0, s_1, s_2) but also the representation of the last token in the buffer β_2 (b_{2last}).

Given a sentence $s = w_1, \ldots, w_n$, the initial configuration is $([ROOT], [w_i \mid i > 1], [w_1], \emptyset)$. In the final configuration the stack σ contains only the $ROOT$ token (we refer to this as an empty stack) and both buffers, β_1 and β_2 are empty. The new transition set, described in Table 1, includes a new transition: *PushToSeg*. This transition adds a new token to the current segment by pushing the top token of β_1 to the end of β_2. The $REDUCE$ transition preconditions have also been modified: this transition is only allowed if β_2 is empty or there is more then one word in the stack.

This new parser performs a two-step process that repeats until convergence, to induce a parse forest. The first step is segment allocation, consisting of a sequence of PushToSeg transitions. When the parser reaches a configuration in which β_2 and σ are empty, only this transition is allowed. There can be one or more consecutive PushToSeg transitions, each pushes a new token from β_1 to β_2. This step ends once the parser selects any other transition, to form a segmentation point. In the second step the allocated segment is parsed. In this step the parser acts as an arc-eager parser with β_2 as the main buffer, and the PushToSeg transition and the β_1 buffer being ignored until the segment is completely parsed (the PushToSeg transition is forbidden while the stack is not empty).

An example of the parsing process is provided in Figure 2. Appendix A provides a proof that the parser is complete and sound as required in Nivre (2008) from any dependency parser.

We next describe two auxiliary segmentation models that can be integrated with our parser.

3.3 Auxiliary Segmentation Models

We consider two models: one is independent of the parser while the other is added as a component to the parser.

Independent segmentation model Similarly to our parser, this model, denoted as SEG, is a BiLSTM that feeds an MLP classifier which predicts for every input word whether it is a segmentation point or not. The loss is a sum of word level MBL functions (equation 4). The input word representation, x_i, and the definition of the hidden word vector, v_i, are as in equations 1 and 2, respectively; the output scores vector $o^{seg}(w_i)$ is derived from the hidden vector, h^{seg}, as in equation 3:

$$\begin{aligned} h^{seg}(v_i) &= tanh(W_s^1 \cdot v_i + b_s^1) \\ o^{seg}(v_i) &= W_s^2 \cdot h^{seg}(v_i) + b_s^2 \end{aligned} \tag{5}$$

where $W_s^1, W_s^2, b_s^1, b_s^2$ are parameters.

We consider two ways through which our P^B parser uses the information from the SEG model. The parser we denote with P^S concatenates the SEG hidden vector of the top word of the stack to the configuration representation:

$$c_{seg} = v_{s_0} \circ v_{s_1} \circ v_{s_2} \circ v_{b_{10}} \circ v_{b_{2last}} \circ h^{seg}(s_0)$$

Alternatively, the parser we denote with P^{FS} (for $FULL\_SEG$) concatenates the hidden vectors of all the configuration elements to the configuration representation:

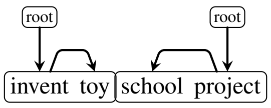
$$\begin{aligned} c_{full\_seg} = & v_{s_0} \circ v_{s_1} \circ v_{s_2} \circ v_{b_{10}} \circ v_{b_{2last}} \\ & \circ h^{seg}(s_0) \circ h^{seg}(s_1) \circ h^{seg}(s_2) \\ & \circ h^{seg}(b_{10}) \circ h^{seg}(b_{2last}) \end{aligned}$$

This segmentation model can be trained independently of the parser or jointly with it. In development data experiments independent training was superior so we report results with this option.

Configuration-based segmentation model This model, denoted as *FLAG*, predicts whether a given parser configuration is a segmentation point. The answer is positive if the processed words (words in the stack σ) are in the same segment and the unprocessed words (words in the buffer β_2 or β_1) are not in this segment.

Transition		Precondition
$LEFT_{arc}$	$(\sigma\|i, \beta_1, j\|\beta_2, A) \Rightarrow (\sigma, \beta_1, j\|\beta_2, A \cup (j, l, i))$	$\neg[i = 0] \wedge \neg\exists k \exists l'[(k, l', i) \in A]$
$RIGHT_{arc}$	$(\sigma\|i, \beta_1, j\|\beta_2, A) \Rightarrow (\sigma\|i\|j, \beta_1, \beta_2, A \cup (i, l, j))$	
$REDUCE$	$(\sigma\|i, \beta_1, \beta_2, A) \Rightarrow (\sigma, \beta_1, \beta_2, A)$	$[\exists k \exists l[(k, l, i) \in A]]$ $\wedge\neg[\boldsymbol{size(\beta_2)! = 0 \wedge (size(\sigma) == 1)}]$
$SHIFT$	$(\sigma, \beta_1, i\|\beta_2, A) \Rightarrow (\sigma\|i, \beta_1, \beta_2, A)$	
$\boldsymbol{PushToSeg}$	$(\sigma, i\|\beta_1, \beta_2, A) \Rightarrow (\sigma, \beta_1, \beta_2\|i, A)$	$size(\sigma) == 1$

Table 1: The transition logic of the segmentation-aware parser. The bold items are the new transitions and conditions of our parser, compared to a standard arc-eager parser.



	Action	$\boldsymbol{\beta_2}$	$\boldsymbol{\beta_1}$	$\boldsymbol{\sigma}$	Arcs
1	$PushToSeg$	[invent]	[toy, school, project]	[ROOT]	$\emptyset$
2	$PushToSeg$	[invent, toy]	[school, project]	[ROOT]	$\emptyset$
3	$RIGHT_{arc}$	[toy]	[school, project]	[ROOT, invent]	A = (ROOT, invent)
4	$RIGHT_{arc}$	[]	[school, project]	[ROOT, invent, toy]	$A = A \cup (invent, toy)$
5	$REDUCE$	[]	[school, project]	[ROOT, invent]	A
6	$REDUCE$	[]	[school, project]	[ROOT]	A
7	$PushToSeg$	[school]	[project]	[ROOT]	A
8	$PushToSeg$	[school, project]	[]	[ROOT]	A
9	$SHIFT$	[project]	[]	[ROOT, school]	A
10	$LEFT_{arc}$	[project]	[]	[ROOT]	$A = A \cup (project, school)$
11	$RIGHT_{arc}$	[]	[]	[ROOT, project]	$A = A \cup (ROOT, project)$
12	$REDUCE$	[]	[]	[ROOT]	A

Figure 2: Example of the application of our segmentation-aware parser to the multi-segment query *invent toy school project* (borrowed from PRS16). First, a sequence of PushToSeg transitions is performed in order to insert the first segment *invent toy* to the buffer β_2 (transitions 1-2). Then, the segment is parsed until the buffer β_2 and the stack σ are empty (3-6). Similarly, the second segment *school project* is pushed to the buffer β_2 through a sequence of PushToSeg transitions (7-8) and then parsed (9-12).

The model is a simple MLP that receives a parser configuration as input and produces a hidden vector (denoted with $h^{flag}(c)$) and a scores vector (denoted with $o^{flag}(c)$). The equations for $h^{flag}(c)$ and $o^{flag}(c)$ are similar to equation 5, and the loss is an MBL loss as in equation 4.

Information from this model is integrated into the parser configuration representation through:

$$c_{flag}(c) = c \circ h^{flag}(c)$$

We refer to this parser as P^{Fl} (for FLAG). The $FLAG$ model must be trained jointly with the parser as its input is a parser configuration.

As shown in § 7, adapting the KG16 parser to explicitly account for multiple segments improves over the KG16 parser in the task of query parsing. We next show how additional gains can be achieved when recognizing the role of NEs.

4 Entities in Query Parsing

In this section we explore the role of NEs in the syntactic structure of queries. We first describe our NE annotation process, and then qualitatively demonstrate the valuable structural cues they provide. In § 5 we will describe extensions of the segmentation-aware BiLSTM parser (§ 3.2) that integrate information from a BiLSTM NE tagger.

Data We consider five entity types: Location (e.g ”London”), Person (e.g ”Marilyn Monroe”), Organization (e.g ”Google”), Product (e.g ”Iphone 4”) and Other (e.g. see Figure 4 for NEs such as ”song name” and ”computer game”). Two human annotators annotated the dataset. Of the 4000 queries, 400 were randomly selected for initial tagging by both annotators, so that they could discuss ambiguous cases and resolve conflicts (the labeled micro-F1 score between the annotators at this stage was 85.6). Then, the remaining 3600 queries were equally split between the two annotators, who again consulted each other in ambiguous cases (the inter-annotator micro-F1, measured on a randomly sampled set of 100 queries, was 92.0).

NEs as a dependency parsing signal The dataset consists of 3010 single-segment (henceforth *SSG*) and 990 multi-segment (henceforth *MSG*) queries. 62.5% of the queries (59% of the SSG and 72% of the MSG) contain at least one NE. Figure 3 (top) provides segment and query level NE statistics. The middle part of the figure shows the proportion of segments and queries that start with an NE. Finally, the bottom part of the table provides word level statistics. The figures clearly demonstrate the prominence of NEs

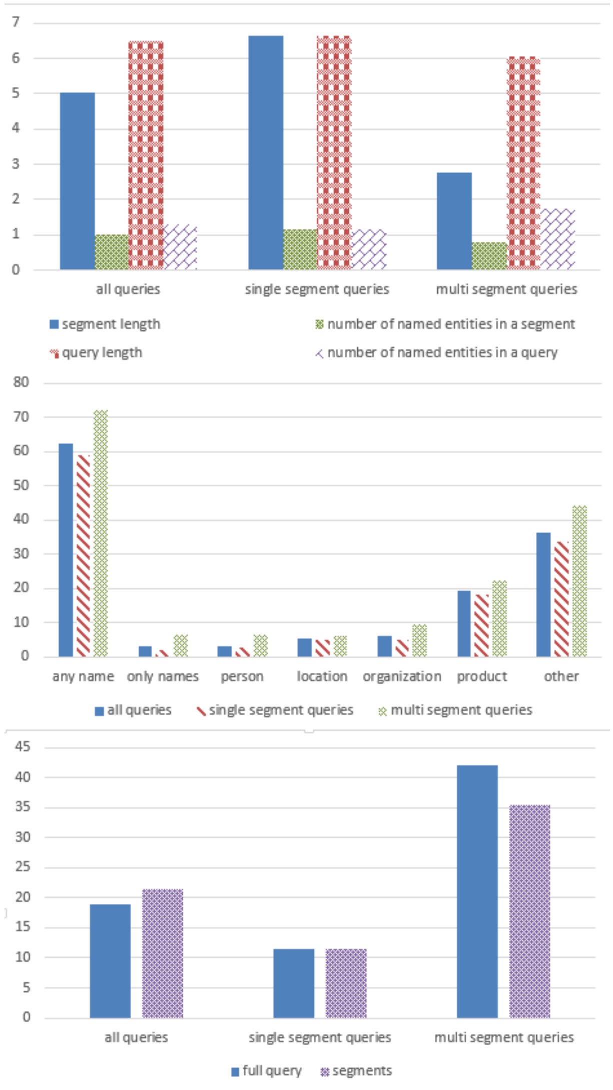


Figure 3: Top: Query and segment level NE distribution (numbers are averages across the queries in the relevant subset). Middle: NE type distribution (the "only names" column refers to queries that consist of named entities only, and the "any name" column refers to queries that contain at least one named entity). Bottom: The percentage of queries and segments that start with an NE.

in queries and the segmentation signal they provide. For example, as many as 35.4% of segments within the MSG queries start with an NE (42% of the first segments and 30% of the other segments). Finally, Figure 4 presents three example queries where NEs provide invaluable cues about the syntactic structure.

Now that we have established the importance of NEs for query parsing, we are ready to describe our entity-aware query parser.

5 A Segmentation and Entity-Aware BiLSTM Parser

Here we describe the integration of NE signals in our segmentation-aware query parser (§ 3.2).

A BiLSTM NE tagger Our NE tagger is a BiLSTM with an MLP classification layer, very similar to our independent segmentation model (SEG, § 3.3). We denote the MLP's hidden state with $h^{ne}(w_i)$ and its output scores vector with $o^{ne}(w_i)$. The model equations are:

$$\begin{aligned} h^{ne}(w_i) &= tanh(W_n^1 \cdot v_i + b_n^1) \\ o^{ne}(w_i) &= W_n^2 \cdot h^{ne}(w_i) + b_n^2 \end{aligned} \tag{6}$$

The margin-based loss (MBL) function we use in this model is:

$$\begin{aligned} MBL = max(0, 1 - MLP_\theta(w_i)[ne_{correct}] + \\ MLP_\theta(w_i)[ne_{predicted}]) \end{aligned} \tag{7}$$

where $\theta = (W_n^1, b_n^1, W_n^2, b_n^2)$ are the model parameters, $ne_{predicted}$ is the named entity type predicted by model and $ne_{correct}$ is the gold named entity type. $MLP_\theta(w_i)[ne_i]$ is the score given by the MLP to the ne_i named entity type.

NE-aware parsing We consider two methods for integrating information from the NE tagger into the parser. In both methods, we construct a new feature representation for each input word, denoted with v_i^M, where M stands for the integration method. The new word representations are then used in the configuration representation (§ 3).

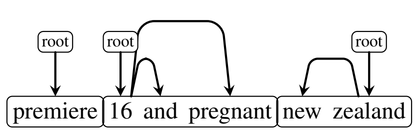
The first method, denoted with *Hi* (for Hidden), uses the hidden vector $h^{ne}(w_i)$:

$$v_i^{Hi} = v_i \circ h^{ne}(w_i)$$

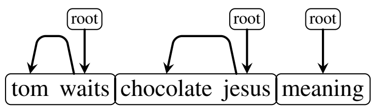
The second method, denoted with *Fi* (for Final), uses the NE embeddings:

$$v_i^{Fi} = v_i \circ e(ne_{predicted}(w_i))$$

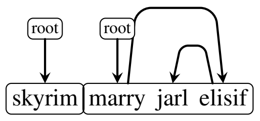
For both methods v_i is the word representation generated by the parser (equation 2). For v_i^{Fi}, $ne_{predicted}(w_i)$ is the named entity type predicted by the tagger for the word w_i and $e(ne_{predicted}(w_i))$ is its embedding (part of the parser parametrization).



(a) ”16 and pregnant” is a name of a television series; ”new zealand” is a name of a place.



(b) ”tom waits” is a name of a person; ”chocolate jesus” is a name of a song.



(c) ”skyrim” is a name of a computer game; ”jarl elisif” is a name of a character in the game.

Figure 4: Three example queries from the PRS16 dataset, along with their parse trees. NEs provide an important signal about the structure of the queries.

Tagger and parser training We consider two approaches:

(a) Independent training: First, the tagger is trained with the gold NEs of the training set (§ 4), then the tagger is applied to the training set, and finally the parser is trained with the gold parse trees and the tagger's NE tagging of the training set.

(b) Joint training: The parameters of both models are updated together, each update is taking place after observing a single input sentence. In this joint model, the parser and the tagger are both using the same BiLSTM to learn the word representation v_i. The loss function of this model is the sum of the losses of the parser (sum of the stepwise losses of Eq. 4) and the tagger (Eq. 7):

$$MBL_{joint} = MBL_{parser} + MBL_{ner} \qquad (8)$$

6 Experiments

Task and data Our task is the query parsing task of PRS16, but unlike them we do not use millions of unannotated queries. Instead, we experiment with a supervised setup were the parser is trained on parsed web queries and no unannotated queries are used. For our experiments we randomly split the PRS16 dataset of 4000 queries annotated with dependency structures and POS tags,[5] into train, development and test sections. This split is done so that: (a) The train set consists of 2500 queries while the dev and the test sets consist of 750 queries each; (b) For each $k \geq 1$, k-segment queries are split between the three sets so that to keep the 2500:750:750 proportion. As a result, the train, dev and test sets contain 618, 185 and 186 MSG queries, respectively, for the total of 989 MSG queries.

We consider the evaluation measures of PRS16: (a) The standard dependency parsing Unlabeled Attachment Score (UAS); and (b) Segmentation F1-score, where a segment is considered correct if both its start and end point are correctly identified.

Hyper-parameter	Value
Word embedding dim.	100, 200, 300
POS tag embedding dim.	25, 50, 100
Named entity embedding dim.	6
Hidden units in MLP	100
BiLSTM hidden dim.	50,125,200
BiLSTM output dim.	125
α (for word dropout)	0.25
p_{agg} (for exploration training)	1

Table 2: The hyper-parameters considered in our development data experiments.

Models and baselines We experiment with three model families: (a) The baseline KG16 parser and our arc-eager variant of the parser (§ 3.1, § 3.2); (b) Our segmentation-aware parsers (§ 3.2, § 3.3); and (c) Our segmentation and entity-aware parsers (§ 5) where the parser and the NE tagger are trained either jointly or independently (we also consider the integration of NE information into the original (arc-hybrid) KG16 parser).[6]

Hyper-parameter tuning Following (Kiperwasser and Goldberg, 2016) and due to the large number of models we experiment with, we consider a relatively small grid of hyper-parameter values, focused around the values chosen by these authors, as described in Table 2.

To avoid a very large number of experiments, we tune the parameters for the original KG16 arc-hybrid model (P^H) and for our arc-eager version of the parser (P^E). We then report test-set results for the P^H model with its tuned hyper-parameters, and for all the other models with the hyper-parameters that were estimated for the P^E model. While this setup gives an unfair advantage

[5]webscope.sandbox.yahoo.com (dataset L-28)

[6]A comment about naming conventions: unless H (for the arc-hybrid KG16 parser) or E (for the arc-eager KG16 parser) is part of the model name, a model is an extension of our P^B parser.

for the baseline P^H model, it helps us avoid an expensive model-specific tuning process. The auxiliary segmentation models (§ 3.3) and the NE tagger (§ 5) use the hyper-parameters of Table 2, but for these models we do not perform any tuning – for each hyper-parameter with more than one option, we use the leftmost number from the table.

7 Results

Our results are presented in Table 3. We focus on selected members of each model family, and within each model family we focus on the simplest models (P^H, P^E and P^B, with segmentation and entity information when appropriate), and on the most complex ones. We make sure to include the best performing model of each category, which happens to be one of the most complex models for all families, emphasizing the quality of our modeling choices. The results for the full list of models are in the spp. material.

The *Baseline parsers* section of the table demonstrates the impact of moving from the arc-hybrid variant to an arc-eager variant of KG16, to better support segmentation. While the P^H model performs slightly better than P^E in terms of UAS (78.3 vs. 77.0), the segmentation F1-score of P^E is 4.3 points better (72.0 vs. 67.7). Interestingly, this improvement is not achieved through better segmentation of MSG queries, but by avoiding unnecessary segmentation decisions on SSG queries.

The *segmentation-aware parsers* section of the table shows that further extending the arc-eager KG16 parser to explicitly account for segmentation results in substantial segmentation improvements. Particularly, the overall F1-score of P^B – our segmentation-aware parser that does not use information from any auxiliary segmentation models – is as high as 77.4. This amounts to 5.4 and 9.7 additional F1 points compared to the arc-eager and the original arc-hybrid KG16 parsers, respectively. Information from auxiliary segmentation models (P^{S+Fl} and P^{FS+Fl}) does not substantially increase performance in this family.

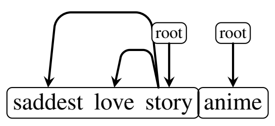
When considering entity information, the performance of our models and of the P^E baseline substantially improves. However, while the UAS of the P^E baseline increases to 80.2 (P^{E+Fi} with independent training), its segmentation F1-score does not cross the 66.9 bound (P^{E+Hi} with joint training).[7] The gain of the segmentation-aware parser from entity information is much more substantial. First, regardless of how the entity information is integrated into the model (Hi vs. Fi) and of whether the auxiliary segmentation models (S, FS and Fl) are used or not, the model perfectly segments the SSG queries. Moreover, it demonstrates substantial performance boosts with respect to all measures. Our best performing model, $P^{S+Fl+Fi}$ with joint training (bold result in the bottom model section of the table) improves the original KG16 parser (P^H, top row of the table) by 6.5 UAS points (83.5 vs. 77.0) and by 16.8 segmentation F1 scores (84.5 vs. 67.7).

Overall, joint training of the parsing and NER

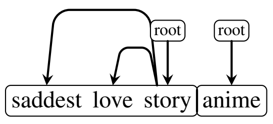
Model	UAS			Seg. F1-score		
	all	msg	ssg	all	msg	ssg
Baseline parsers						
P^H	**77.0**	**74.9**	**78.3**	67.7	**51.5**	77.9
P^E	75.4	70.5	77.0	**72.0**	47.6	**87.6**
Seg. aware parsers - no entity information						
P^B	76.7	73.4	77.8	**77.4**	48.7	**95.6**
P^{S+Fl}	**77.1**	**74.1**	**78.1**	76.0	**53.3**	90.8
P^{FS+Fl}	76.6	72.1	78.0	**77.4**	52.1	93.8
Seg. and entity parsers - independent training						
P^{E+Hi}	80.2	70.3	83.3	63.3	34.3	80.3
P^{E+Fi}	80.2	70.8	83.2	61.4	42.0	73.2
P^{Hi}	79.3	68.3	82.8	63.3	40.5	**100**
P^{Fi}	80.9	69.5	84.5	79.1	45.2	**100**
$P^{S+Fl+Hi}$	80.6	70.5	83.8	77.0	48.3	**100**
$P^{S+Fl+Fi}$	81.2	71.5	84.2	80.6	49.0	**100**
$P^{FS+Fl+Hi}$	**81.6**	**72.3**	**84.6**	80.1	**54.1**	**100**
$P^{FS+Fl+Fi}$	80.6	70.4	83.9	**81.7**	52.8	**100**
Seg. and entity parsers - joint training						
P^{E+Hi}	79.3	66.7	83.4	66.9	26.2	90.2
P^{E+Fi}	79.8	67.7	83.6	66.1	36.9	83.6
P^{Hi}	79.7	66.5	83.4	75.1	33.1	**100**
P^{Fi}	81.6	70.5	85.1	78.6	43.7	**100**
$P^{S+Fl+Hi}$	81.6	71.9	84.6	82.0	53.4	**100**
$P^{S+Fl+Fi}$	**83.5**	**73.2**	**86.7**	**84.5**	**60.7**	**100**
$P^{FS+Fl+Hi}$	79.6	67.3	83.5	76.0	36.1	**100**
$P^{FS+Fl+Fi}$	82.4	72.1	85.7	81.2	51.4	**100**

Table 3: Results. Best numbers within each model section are highlighted in bold. H and E stand for the arc-hybrid and arc-eager KG16 parser, while B is our segmentation aware parser without any auxiliary segmentation model or NE information. All models where these letters do not appear refer to extensions of our segmentation-aware parser (B). S: independent segmentation model. FS: full independent segmentation model. Fl: configuration based segmentation model. Hi: NE aware parser (Hidden). Fi: NE aware parser (Final).

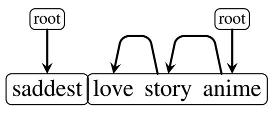
[7]The UAS numbers of the p^H models with entity information are similar to those of the p^E models, but their segmentation quality is lower. Due to space limitations, we do not provide these numbers.



(a) Gold tree.

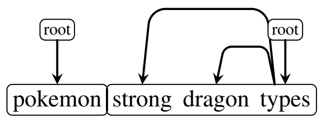


(b) The P^B parser (perfect) tree.

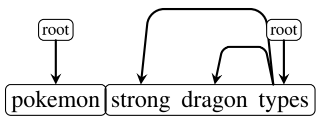


(c) The P^H parse tree.

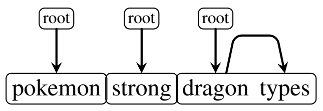
Figure 5: Example multi-segment query where P^B succeeds and P^H fails.



(a) Gold tree.

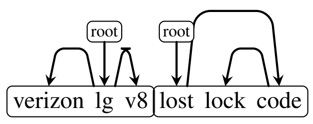


(b) The (perfect) tree of the $P^{S+Fl+Fi}$ parser when trained jointly with the named entity tagger.

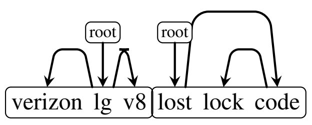


(c) The P^{S+Fl} parse tree.

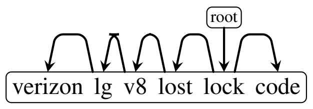
Figure 6: Example multi-segment query where the $P^{S+Fl+Fi}$ parser succeeds when trained jointly with the named entity tagger, and P^{S+Fl} fails.



(a) Gold tree.



(b) The (perfect) tree of the $P^{S+Fl+Fi}$ parser when trained jointly with the named entity tagger.



(c) The erroneous tree of the $P^{S+Fl+Fi}$ parser when trained independently of the named entity tagger.

Figure 7: Example multi-segment query where the $P^{S+Fl+Fi}$ parser succeeds when trained jointly with the named entity tagger, and fails when using a pre-trained named entity tagger.

models improves over independent training for both segmentation F1 and parsing UAS (in 5 of 8 cases for each measure, two bottom model sections of the table). This improvement comes mostly from SSG queries (7 of 8 UAS cases and the 2 cases where segmentation F1 could improve), but at the cost of some degradation on MSG queries. Moreover, our best model is a jointly trained one.

While the goal of this paper is mostly to improve the syntactic analysis of web queries, our simple NE tagger provides decent results. When trained independently of the parser its test-set (labeled) micro-F1 score is 85.6. When trained jointly with the parser it achieves similar scores in some cases: e.g. when jointly trained with the best performing parsing model, $P^{S+Fl+Fi}$, it achieves a micro-F1 of 85.2. Yet, in other cases such as joint training with P^{S+H} and P^{S+Fi} its micro-F1 drops to 78.0 and 78.5, respectively.

Finally, figure 5-7 provide some qualitative analysis of our models and baselines.

8 Conclusions

We presented a new BiLSTM transition-based parser for web queries. Our parser is the first that explicitly accounts for the forest-based query grammar of PRS16. Moreover, we demonstrated the importance of NEs for understanding the syntactic structure of web queries, annotated the Query Treebank of PRS16 with NEs, and demonstrated how to effectively use NE information in the syntactic parsing of web queries.

In future work we intend to explore methods for closing the performance gap our algorithms still have for MSG queries (both UAS and segmentation F1) and for SSG queries (UAS only). Relevant directions include improving the transition logic of our parser, the BiLSTM NE model and the interactions between the two models.

Acknowledgements

We would like to thank Yuval Pinter, Daniel Hershcovich and the Technion NLP group members for their valuable feedback.

References

Areej Alasiry, Mark Levene, and Alexandra Poulovassilis. 2012. Detecting candidate named entities in search queries. In *Proceedings of SIGIR*.

James Allan and Hema Raghavan. 2002. Using part-of-speech patterns to reduce query ambiguity. In *Proceedings of SIGIR*.

Cory Barr, Rosie Jones, and Moira Regelson. 2008. The linguistic structure of english web-search queries. In *Proceedings of EMNLP*.

Michael Bendersky, W Bruce Croft, and David A Smith. 2010. Structural annotation of search queries using pseudo-relevance feedback. In *Proceedings of CIKM*.

Michael Bendersky, W Bruce Croft, and David A Smith. 2011. Joint annotation of search queries. In *Proceedings of ACL*.

Shane Bergsma and Qin Iris Wang. 2007. Learning noun phrase query segmentation. In *Proceedings of EMNLP-CoNLL*.

Andreas Eiselt and Alejandro Figueroa. 2013. A two-step named entity recognizer for open-domain search queries. In *Proceedings of IJCNLP*.

Jenny Rose Finkel and Christopher D Manning. 2009. Joint parsing and named entity recognition. In *Proceedings of NAACL-HLT*.

Jenny Rose Finkel and Christopher D Manning. 2010. Hierarchical joint learning: Improving joint parsing and named entity recognition with non-jointly labeled data. In *Proceedings of ACL*.

Jennifer Foster, Özlem Çetinoglu, Joachim Wagner, Joseph Le Roux, Stephen Hogan, Joakim Nivre, Deirdre Hogan, Josef Van Genabith, et al. 2011. # hardtoparse: Pos tagging and parsing the twitterverse. In *proceedings of the Workshop On Analyzing Microtext (AAAI 2011)*.

Kuzman Ganchev, Keith Hall, Ryan McDonald, and Slav Petrov. 2012. Using search-logs to improve query tagging. In *Proceedings of ACL (Volume 2: Short Papers)*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTAT*.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association of Computational Linguistics* 1:403–414.

Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. 2009. Named entity recognition in query. In *Proceedings of SIGIR*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics* 4:313–327.

Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archna Bhatia, Chris Dyer, and Noah A. Smith. 2014. A dependency parser for tweets. In *Proceedings of EMNLP*.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of ACL*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of NAACL*.

Xiao Li. 2010. Understanding the semantic structure of noun phrase queries. In *Proceedings of ACL*.

Yijia Liu, Yi Zhu, Wanxiang Che, Bing Qin, Nathan Schneider, and Noah A. Smith. 2018. Parsing tweets into universal dependencies. In *Proceedings of NAACL*.

Mehdi Manshadi and Xiao Li. 2009. Semantic tagging of web search queries. In *Proceedings of the ACL-IJCNLP*.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* 34(4):513–553.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of ACL*.

Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on parsing the web. In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*.

Yuval Pinter, Roi Reichart, and Idan Szpektor. 2016. Syntactic parsing of web queries with question intent. In *Proceedings of NAACL*.

Roi Reichart, Katrin Tomanek, Udo Hahn, and Ari Rappoport. 2008. Multi-task active learning for linguistic annotations. In *Proceedings of ACL-08: HLT*.

Gilad Tsur, Yuval Pinter, Idan Szpektor, and David Carmel. 2016. Identifying web queries with question intent. In *Proceedings of WWW*.

Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Robert Belvin, and Ann Houston. 2013. *OntoNotes Release 5.0*. https://catalog.ldc.upenn.edu/LDC2013T19.

Ryen W. White, Matthew Richardson, and Wen-tau Yih. 2015. Questions vs. queries in informational search tasks. In *Proceedings of WWW*.

Ke Zhai, Zornitsa Kozareva, Yuening Hu, Qi Li, and Weiwei Guo. 2016. Query to knowledge: Unsupervised entity extraction from shopping queries using adaptor grammars. In *Proceedings SIGIR*.

A Parser correctness

In this appendix we discuss the correctness of our segmentation-aware dependency parser, as described in §3.2 of the main paper.

According to Nivre (2008), a parsing system is correct for a class $\mathbb{G}$ of dependency graphs if and only if it is sound and complete for $\mathbb{G}$. We prove correctness for the set $\mathbb{G}$ of all possible dependency trees. The system is *sound* if and only if for every sentence x and every transition sequence $C_{0,m}$ for x, it holds that $G_{C_{0,m}} \in \mathbb{G}$. The system is *complete* for the class $\mathbb{G}$ of all possible dependency trees if and only if for every sentence x and every dependency graph $G_x \in \mathbb{G}$, there is a transition sequence $C_{0,m}$ for x such that $G_{C_{0,m}} = G_x$.

We prove that our segmentation-aware parser is sound by showing that its resulting dependency graph is necessarily a valid parse tree. In order to do that we show that in every possible output graph of the parser each word has exactly one head.

We start by showing that a word must have at least one head. This stems from the fact that a word processing is completed once it is reduced. This can be done using the $REDUCE$ transition or the $LEFT_{arc}$ transition. The preconditions of the $REDUCE$ transition prevent a node from being reduced without a head. The $LEFT_{arc}$ transition, by definition, reduces the node after assigning it with a head. There can be no more than one head to each node as the arcs are created using the $RIGHT_{arc}$ and $LEFT_{arc}$ transitions. If the $RIGHT_{arc}$ transition is applied to a word, it inserts the node to the stack. Once a node is in the stack, $RIGHT_{arc}$ cannot be applied to it and the $LEFT_{arc}$ preconditions prevent the generation of a second head. If the $LEFT_{arc}$ transition applies to the word, the word is automatically reduced and cannot have additional heads, as discussed above.

To prove completeness we develop a valid transition sequence that produces any given dependency tree G_x for a sentence x. We denote the transition sequence that produces a dependency graph G_x for a sentence x in the arc-eager system as $C^{eager}_{0,m}(x, G_x)$. $C^{eager}_{0,m}(x, G_x)$ exists according to the completeness of the arc-eager system. We also denote the sub-graph of a segment s within a dependency tree G_x as G^s_x. An input sentence contains one or more segments. We construct the transition sequence by performing $|s|$ $PushToSeg$ transitions for every segment s in x followed by performing $C^{eager}_{0,m'}(s, G^s_x)$ with the stack σ and the buffer β_2. This transition sequence produces a valid tree G_x for sentence x.