Expected F-Measure Training for Shift-Reduce Parsing with Recurrent Neural Networks

Wenduan Xu[†] **Michael Auli**[‡] **Stephen Clark**[†]
[†]Computer Laboratory, Cambridge University
[‡]Facebook AI Research
wx217@cam.ac.uk, michaelauli@fb.com, sc609@cam.ac.uk

Abstract

We present expected F-measure training for shift-reduce parsing with RNNs, which enables the learning of a global parsing model optimized for sentence-level F1. We apply the model to CCG parsing, where it improves over a strong greedy RNN baseline, by 1.47% F1, yielding state-of-the-art results for shift-reduce CCG parsing.

1 Introduction

Shift-reduce parsing is a popular parsing paradigm, one reason being the potential for fast parsers based on the linear number of parsing actions needed to analyze a sentence (Nivre and Scholz, 2004; Sagae and Lavie, 2006; Zhang and Clark, 2011; Goldberg et al., 2013; Zhu et al., 2013; Xu et al., 2014). Recent work has shown that by combining distributed representations and neural network models (Chen and Manning, 2014), accurate and efficient shift-reduce parsing models can be obtained with little feature engineering, largely alleviating the feature sparsity problem of linear models.

In practice, the most common objective for optimizing neural network shift-reduce parsing models is maximum likelihood. In the greedy search setting, the log-likelihood of each target action is maximized during training, and the most likely action is committed to at each step of the parsing process during inference (Chen and Manning, 2014; Dyer et al., 2015). In the beam search setting, Zhou et al. (2015) show that sentence-level likelihood, together with contrastive learning (Hinton, 2002), can be used to derive a global model which incorporates beam search at both training and inference time (Zhang and Clark, 2008), giving significant accuracy gains over a fully greedy model. However, despite the effectiveness of optimizing likelihood, it is often desirable to directly optimize for task-specific metrics, which often leads to higher accuracies for a variety of models and applications (Goodman, 1996; Och, 2003; Smith and Eisner, 2006; Rosti et al., 2010; Auli and Lopez, 2011; He and Deng, 2012; Auli et al., 2014; Auli and Gao, 2014; Gao et al., 2014).

In this paper, we present a global neural network parsing model, optimized for a task-specific loss based on expected F-measure. The model naturally incorporates beam search during training, and is globally optimized, to learn shift-reduce action *sequences* that lead to parses with high expected F-scores. In contrast to Auli and Lopez (2011), who optimize a CCG parser for F-measure via softmax-margin (Gimpel and Smith, 2010), we directly optimize an expected F-measure objective, derivable from only a set of shift-reduce action sequences and sentence-level F-scores. More generally, our method can be seen as an alternative approach for training a neural beam search parsing model (Watanabe and Sumita, 2015; Weiss et al., 2015; Zhou et al., 2015), combining the benefits of global learning and task-specific optimization.

We also introduce a simple recurrent neural network (RNN) model to shift-reduce parsing on which the greedy baseline and the global model is based. Compared with feed-forward networks, RNNs have the potential to capture and use an *unbounded* history, and they have been used to learn explicit representations for parser states as well as actions

Proceedings of NAACL-HLT 2016, pages 210–220,
San Diego, California, June 12-17, 2016.

performed on the stack and queue in shift-reduce parsers (Dyer et al., 2015; Watanabe and Sumita, 2015), following Miikkulainen (1996) and Mayberry and Miikkulainen (1999). In comparison, our model is a natural extension of the feed-forward architecture in Chen and Manning (2014) using Elman RNNs (Elman, 1990).

We apply our models to CCG, and evaluate the resulting parsers on standard CCGBank data (Hockenmaier and Steedman, 2007). More specifically, by combining the global RNN parsing model with a bidirectional RNN CCG supertagger that we have developed (§4) — building on the supertagger of Xu et al. (2015), we obtain accuracies higher than the shift-reduce CCG parsers of Zhang and Clark (2011) and Xu et al. (2014). Finally, although we choose to focus on shift-reduce parsing for CCG, we expect the methods to generalize to other shift-reduce parsers.

2 RNN Models

In this section, we start by describing the baseline model, which is also taken as the pretrained model to train the global model (§2.4). We abstract away from the details of CCG and present the models in a canonical shift-reduce parsing framework (Aho and Ullman, 1972), which is henceforth assumed: partially constructed derivations are maintained on a *stack*, and a *queue* stores remaining words from the input string; the initial parse item has an empty stack and no input has been consumed on the queue. Parsing proceeds by applying a sequence of shift-reduce actions to transform the input until the queue has been exhausted and no more actions can be applied.

2.1 Model

Our recurrent neural network model is a standard Elman network (Elman, 1990) which is factored into an input layer, a hidden layer with recurrent connections, and an output layer. Similar to Chen and Manning (2014), the input layer x_t encodes stack and queue contexts of a parse item through concatenation of feature embeddings. The output layer y_t represents a probability distribution over possible parser actions for the current item.

The current state of the hidden layer is determined by the current input and the previous hidden layer state. The weights between the layers are represented by a number of matrices: matrix $\mathbf{U}$ contains weights between the input and hidden layers, $\mathbf{V}$ contains weights between the hidden and output layers, and $\mathbf{W}$ contains weights between the previous hidden layer and the current hidden layer.

The hidden and output layers at time step t are computed via a series of vector-matrix products and non-linearities:

$$h_t = f(x_t\mathbf{U} + h_{t-1}\mathbf{W}),$$
$$y_t = g(h_t\mathbf{V}),$$

where

$$f(z) = \frac{1}{1+e^{-z}}, \quad g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

are sigmoid[1] and softmax functions, respectively.

2.2 Feature Embeddings

Given a parse item, we first extract features using a set of predefined feature templates; each template belongs to a feature type f (such as word or POS tag), which has an associated look-up table, denoted as L_f, to project a feature to its distributed representation; and $L_f \in \mathbb{R}^{n_f \times d_f}$, where n_f is the vocabulary size of feature type f and d_f is its embedding dimension. The embedding for a concrete feature is obtained by retrieving the corresponding row from L_f. At time step t, the input layer x_t is:

$$x_t = [e_{f_{1,1}}; \ldots; e_{f_{1,|f_1|}}; \ldots; e_{f_{k,1}}; \ldots; e_{f_{k,|f_k|}}],$$

where ";" denotes concatenation, $|f_k|$ is the number of feature templates for the k^{th} feature type and $x_t \in \mathbb{R}^{1\times(d_{f_1}|f_1|+\ldots+d_{f_k}|f_k|)}$. For each feature type, a special embedding is used for unknown features.

2.3 Greedy Training

To train a greedy model, we extract gold-standard actions from the training data and minimize cross-entropy loss with stochastic gradient descent (SGD) using backpropagation through time (BPTT; Rumelhart et al., 1988). Similar to Chen and Manning (2014), we compute the softmax over only feasible actions at each step.

Unfortunately, although we use an RNN, which keeps a representation of previous parse items in its

[1] We also experimented with $\tanh$, and found no difference in resulting performance.

hidden state and has the potential to capture long-term dependencies, the resulting model is still fully greedy: a locally optimal action is taken at each step given the current input x_t and the previous hidden state h_{t-1}. Therefore, once a sub-optimal action has been committed to by the parser at any step, it has no means to recover and has to continue from that mistake. Such mistakes accumulate until the goal is reached, and they are referred to as *search errors*.

In order to enlarge the search space of the greedy model thereby alleviating some search errors, we experiment with applying beam search decoding during inference; and we observe some accuracy improvements by taking the highest scored action sequence as the output (Table 3). However, since the greedy model itself is only optimized locally, as expected, the improvements diminish after a certain beam size. Instead, we show below that by using the greedy model weights as a starting point, we can train a global model optimized for an expected F-measure loss, which gives further significant accuracy improvements (§5).

2.4 Expected F1 Training

The RNN we use to train the global model has the same Elman architecture as the greedy model. Given the greedy model, we summarize its weights as $\theta = \{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$ and initialize the weights of the global model to θ, and training proceeds as follows:

1. We use a beam-search decoder to parse a sentence x_n in the training data and let the decoder generate a *k*-best list[2] of output parses using the *current* θ, denoted as $\Lambda(x_n)$. Similar to other structured training approaches that use inexact beam search (Zhang and Clark, 2008; Weiss et al., 2015; Watanabe and Sumita, 2015; Zhou et al., 2015), $\Lambda(x_n)$ is as an approximation to the set of all possible parses of an input sentence.

2. Let y_i be the shift-reduce action sequence of a parse in the *k*-best list $\Lambda(x_n)$, and let $|y_i|$ be its total number of actions and y_{ij} be the j^{th} action in y_i, for $1 \leq j \leq |y_i|$. We compute the log-linear action sequence score of y_i, $\rho(y_i)$, as a sum of individual action scores in that sequence: $\rho(y_i) = \sum_{j=1}^{|y_i|} \log s_\theta(y_{ij})$, where $s_\theta(y_{ij})$ is the softmax action score of y_{ij} given by the RNN model. For each y_i, we also compute its sentence-level F1 using the set of labeled, directed dependencies, denoted as Δ, associated with its parse item. (We assume F1 over labeled, directed dependencies is also the parser evaluation metric.)

3. We compute the negative expected F1 objective (-xF1, defined below) for x_n using the scores obtained in the above step and minimize this objective using SGD (maximizing the expected F1 for x_n). These three steps repeat for other sentences in the training data, updating θ after processing each sentence, and training iterates in epochs until convergence.

We note that the above process is different from parse reranking (Collins, 2000; Charniak and Johnson, 2005), in which $\Lambda(x_n)$ would stay the same for each x_n in the training data across all epochs, and a reranker is trained on all fixed $\Lambda(x_n)$; whereas the xF1 training procedure is on-line learning with parameters updated after processing each sentence and each $\Lambda(x_n)$ is generated with a new θ.

More formally, we define the loss $J(\theta)$, which incorporates all action scores in each action sequence, and all action sequences in $\Lambda(x_n)$, for each x_n as

$$\begin{aligned} J(\theta) &= -\text{xF1}(\theta) \\ &= -\sum_{y_i \in \Lambda(x_n)} p(y_i|\theta)\text{F1}(\Delta_{y_i}, \Delta^G_{x_n}), \end{aligned} \quad (1)$$

where $\text{F1}(\Delta_{y_i}, \Delta^G_{x_n})$ is the sentence level F1 of the parse derived by y_i, with respect to the gold-standard dependency structure $\Delta^G_{x_n}$ of x_n; $p(y_i|\theta)$ is the normalized probability score of the action sequence y_i, computed as

$$p(y_i|\theta) = \frac{\exp\{\rho(y_i)\}}{\sum_{y \in \Lambda(x_n)} \exp\{\rho(y)\}}. \quad (2)$$

To apply SGD, we derive the error gradients used for backpropagation. First, by applying the chain

[2]We do not put a limit on k, and whenever an item is finished, it is appended to the *k*-best list. We found the size of the *k*-best lists were on average twice the size of a given beam size.

rule to $J(\theta)$, we have

$$\frac{\partial J(\theta)}{\partial \theta} = -\sum_{y_i \in \Lambda(x_n)} \sum_{y_{ij} \in y_i} \frac{\partial J(\theta)}{\partial s_\theta(y_{ij})} \frac{\partial s_\theta(y_{ij})}{\partial \theta}$$
$$= -\sum_{y_i \in \Lambda(x_n)} \sum_{y_{ij} \in y_i} \delta_{y_{ij}} \frac{\partial s_\theta(y_{ij})}{\partial \theta},$$

where $\frac{\partial s_\theta(y_{ij})}{\partial \theta}$ is the standard softmax gradients. Next, to compute $\delta_{y_{ij}}$, which are the error gradients propagated from the loss to the softmax layer, we rewrite the loss in (1) as

$$J(\theta) = -\text{xF1} = -\frac{G(\theta)}{Z(\theta)} \tag{3}$$
$$= -\frac{\sum_{y_i \in \Lambda(x_n)} \exp\{\rho(y_i)\} \text{F1}(\Delta_{y_i}, \Delta^G_{x_n})}{\sum_{y_i \in \Lambda(x_n)} \exp\{\rho(y_i)\}},$$

and by simplifying:

$$\frac{\partial G(\theta)}{\partial s_\theta(y_{ij})} = \frac{1}{s_\theta(y_{ij})} \exp\{\rho(y_i)\} \text{F1}(\Delta_{y_i}, \Delta^G_{x_n}),$$

$$\frac{\partial Z(\theta)}{\partial s_\theta(y_{ij})} = \frac{1}{s_\theta(y_{ij})} \exp\{\rho(y_i)\},$$

since

$$\frac{\partial \rho(y_i)}{\partial s_\theta(y_{ij})} = \frac{1}{s_\theta(y_{ij})}.$$

Finally, using (2) and (3) plus the above simplifications, the error term $\delta_{y_{ij}}$ can be derived using the quotient rule:

$$\begin{aligned}
\delta_{y_{ij}} &= -\frac{\partial \text{xF1}(\theta)}{\partial s_\theta(y_{ij})} \\
&= -\frac{\partial (G(\theta)/Z(\theta))}{\partial s_\theta(y_{ij})} \\
&= \frac{G(\theta)Z'(\theta) - G'(\theta)Z(\theta)}{Z^2(\theta)} \\
&= \frac{\exp\{\rho(y_i)\}}{Z(\theta)} (\text{xF1}(\theta) - \text{F1}(\Delta_{y_i}, \Delta^G_{x_n})) \frac{1}{s_\theta(y_{ij})} \\
&= p(y_i|\theta)(\text{xF1}(\theta) - \text{F1}(\Delta_{y_i}, \Delta^G_{x_n})) \frac{1}{s_\theta(y_{ij})},
\end{aligned} \tag{4}$$

which has a simple closed form.

A naive implementation of the xF1 training procedure would backpropagate the error gradients individually for each y_i in $\Lambda(x_n)$. To make it efficient, we observe that the unfolded network in the beam containing all y_i becomes a DAG (with one hidden state leading to one or more resulting hidden states) and apply backpropagation through structure (Goller and Kuchler, 1996) to obtain the gradients.

3 Shift-Reduce CCG Parsing

We explain the application of the RNN models to CCG by first describing the CCG mechanisms used in our parser, followed by details of the shift-reduce transition system.

3.1 Combinatory Categorial Grammar

A *lexicon*, together with a set of CCG *rules*, formally constitute a CCG. The former defines a mapping from words to sets of lexical categories representing syntactic types, and the latter gives schemas which dictate whether two categories can be combined. Given the lexicon and the rules, the syntactic types of complete constituents can be obtained by recursive combination of categories using the rules.

More generally, both lexical and non-lexical CCG categories can be either *atomic* or *complex*: atomic categories are categories without any slashes, and complex categories are constructed recursively from atomic ones using forward ($/$) and backward slashes ($\backslash$) as two binary operators. As such, all categories can be represented as follows (Vijay-Shanker and Weir, 1993; Kuhlmann and Satta, 2014):

$$x \coloneqq \alpha|_1 z_1 |_2 z_2 \ldots |_m z_m,$$

where $m \geq 0$, α is an atomic category, $|_1, \ldots, |_m \in \{\backslash, /\}$ and z_i are meta-variables for categories.

CCG *rules* have the following two schematic forms, each a generalized version of *functional composition* (Vijay-Shanker and Weir, 1993):

$$x/y \quad y|_1 z_1 \ldots |_m z_m \rightarrow x|_1 z_1 \ldots |_m z_m,$$
$$y|_1 z_1 \ldots |_m z_m \quad x\backslash y \rightarrow x|_1 z_1 \ldots |_m z_m.$$

The first schematic form above instantiates into a forward application rule ($>$) for $m = 0$, and forward composition rules ($>_\mathbf{B}$) for $m > 0$. Similarly, the second schematic form, which is symmetric to the first, instantiates into backward application ($<$) and composition ($<_\mathbf{B}$) rules.

Fig.1 shows an example CCG derivation. All the *rule instances* in this derivation are instantiated from

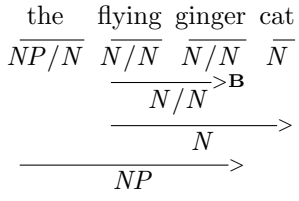


Figure 1: An example CCG derivation.

forward rules; for example, $N/N\ N/N \rightarrow N/N$ is an instance of forward composition and $N/N\ N \rightarrow N$ is an instance of forward application.

Given CCGBank (Hockenmaier and Steedman, 2007), there are two approaches to extract a grammar from this data. The first is to treat all CCG derivations as phrase-structure trees, and a binary, context-free "cover" grammar, consisting of all CCG rule instances in the treebank, is extracted from local trees in all the derivations (Fowler and Penn, 2010; Zhang and Clark, 2011). In contrast, one can extract the lexicon from the treebank and define only the rule schemas, without explicitly enumerating any rule instances (Hockenmaier, 2003). This is the approach taken in the C&C parser (Clark and Curran, 2007) and the one we use here. Moreover, following Zhang and Clark (2011), our CCG parsing model is also a normal-form model, which models action sequences of normal-form derivations in CCGBank.

3.2 The Transition System

The transition system we use in this work is based on the CCG transition system of Zhang and Clark (2011). We denote parse items as $(j, \delta, \beta, \Delta)$[3], where δ is the stack (with top element $\delta|s_0$), β is the queue (with top element $x_{w_j}|\beta$), j is the positional index of the word at the front of the queue, and Δ is the set of CCG dependencies realized for the input consumed so far (needed to calculate the expected F-score). We also assume a set of lexical categories has been assigned to each word using a supertagger (Bangalore and Joshi, 1999; Clark and Curran, 2004). The transition system is specified using three action types:

- SHIFT (sh) removes one of the lexical categories x_{w_j} of the front word w_j in the queue, and pushes it onto the stack; and removes w_j from the queue.

[3]We partly adopt standard notations from dependency parsing (Nivre, 2008).

input: $w_0 \ldots w_{n-1}$

axiom: $0 : (0, \epsilon, \beta, \phi)$

goal: $2n - 1 + \mu : (n, \delta, \epsilon, \Delta)$

$$\frac{\omega : (j, \delta, x_{w_j}|\beta, \Delta)}{\omega + 1 : (j + 1, \delta|x_{w_j}, \beta, \Delta)} \quad (\mathsf{sh}; 0 \leq j < n)$$

$$\frac{\omega : (j, \delta|s_1|s_0, \beta, \Delta)}{\omega + 1 : (j, \delta|x, \beta, \Delta \cup \langle x \rangle))} \quad (\mathsf{re}; s_1 s_0 \rightarrow x)$$

$$\frac{\omega : (j, \delta|s_0, \beta, \Delta)}{\omega + 1 : (j, \delta|x, \beta, \Delta)} \quad (\mathsf{un}; s_0 \rightarrow x)$$

Figure 2: The shift-reduce deduction system.

- REDUCE (re) combines the top two subtrees s_0 and s_1 on the stack using a CCG rule ($s_1 s_0 \rightarrow x$) and replaces them with a subtree rooted in x. It also appends the set of newly created dependencies on x, denoted as $\langle x \rangle$, to Δ.

- UNARY (un) applies either a type-raising or type-changing rule ($s_0 \rightarrow x$) to the stack-top element and replaces it with a unary subtree rooted in x.

The deduction system (Fig. 2) of our shift-reduce parser follows from the transition system.[4] Each parse item is associated with a step indicator ω, which denotes the number of actions used to build it. Given a sentence of length n, a full derivation requires $2n - 1 + \mu$ steps to terminate, where μ is the total number of un actions applied. In Zhang and Clark (2011), a *finish* action is used to indicate termination, which we do not use in our parser: an item finishes when no further action can be taken. Another difference between the transition systems is that Zhang and Clark (2011) omit the Δ field in each parse item, due to their use of a context-free, phrase-structure cover, and dependencies are recovered at a post-processing step; in our system, we build dependencies as parsing proceeds.

[4]We abuse notation slightly for the sh deduction, using $x_{w_j}|\beta$ to denote that the lexical category x_{w_j} is available for the front word on the queue.

s_0.w	s_1.w	s_2.w	s_3.w
s.w$_0$	s.w$_1$	s.w$_2$	s.w$_3$
s_0.l.w	s_1.l.w	s_o.r.w	s_1.r.w
q_0.w	q_1.w	q_2.w	q_3.w
s_0.c	s_0.l.c	s_0.r.c	
s_1.c	s_1.l.c	s_1.r.c	
s_2.c	s_3.c		

Table 1: Atomic feature templates.

3.3 RNN CCG Parsing

We use the same set of CCG rules as in Clark and Curran (2007) and the total number of output units in our RNN model is equal to the number of lexical categories (i.e., all possible sh actions), plus 10 units for re[5] and 18 units for un actions.

All features in our model fall into three types: word, POS tag and CCG category. Table 1 shows the atomic feature templates and we have $|f_w| = 16$, $|f_p| = 16$ and $|f_c| = 8$ (all word-based features are generalized to POS features). Each template has two parts: the first part denotes parse item context and the second part denotes the feature type. s denotes stack contexts and q denotes queue contexts; e.g., s_0 is the top subtree on the stack, and s_o.l is its left child. w represents head words of constituents and w$_0$ is the right-most word of the input string that has been shifted onto the stack.

4 Bidirectional Supertagging

We extend the RNN supertagging model of Xu et al. (2015) by using a bidirectional RNN (BRNN). The BRNN processes an input in both directions with two separate hidden layers, which are then fed to one output layer to make predictions. At each time step t, we compute the *forward* hidden state h_t for $t = (0, 1, \ldots, n-1)$; the *backward* hidden state h'_t is computed similarly but from the reverse direction for $t = (n-1, n-2, \ldots, 0)$ as

$$h'_t = f(x_t \mathbf{U}' + h_{t+1} \mathbf{W}'), \quad (5)$$

and the output layer, for $t = (0, 1, \ldots, n-1)$, is computed as

$$y_t = f([h_t; h'_t]\mathbf{V}'). \quad (6)$$

The BRNN introduces two new parameter matrices $\mathbf{U}'$ and $\mathbf{W}'$ and replaces the old hidden-to-output matrix $\mathbf{V}$ with $\mathbf{V}'$ to take two hidden layers as input. We use the same three feature embedding types as Xu et al. (2015), namely word, suffix and capitalization, and all features are extracted from a context window size of 7 surrounding the current word.

5 Experiments

Setup. All experiments were performed on CCGBank (Hockenmaier and Steedman, 2007) with the standard split.[6] We used the C&C supertagger (Clark and Curran, 2007) and the RNN supertagger model of Xu et al. (2015) as two supertagger baselines. For the parsing experiments, the baselines were the shift-reduce CCG parsers of Zhang and Clark (2011) and Xu et al. (2014) and the C&C parser of (Clark and Curran, 2007).

To train the RNN parser, we used 10-fold cross validation for both POS tagging and supertagging. For both development and test parsing experiments, we used the C&C POS tagger and automatically assigned POS tags. The BRNN supertagging model was used as the supertagger by all RNN parsing models for both training and testing. F-score over directed, labeled CCG predicate-argument dependencies was used as the parser evaluation metric, obtained using the script from C&C.

Hyperparameters. For the BRNN supertagging model, we used identical hyperparameter settings as in Xu et al. (2015). For all RNN parsing models, the weights were uniformly initialized using the interval $[-2.0, 2.0]$, and scaled by their fan-in (Bengio, 2012); the hidden layer size was 220, and 50-dimensional embeddings were used for all feature types and scaled Turian embeddings were used (Turian et al., 2010) for word embeddings. We also pretrained CCG lexcial category and POS embeddings by using the GENSIM word2vec implementation.[7] The data used for this was obtained by parsing a Wikipedia dump using the C&C parser and concatenating the output with CCGBank Sections 02-21. Embeddings for unknown words and CCG categories outside of the lexical category set were uniformly initialized ($[-2.0, 2.0]$) without scaling.

[5] In principle, only 1 re unit is needed, but we use 9 additional units to handle non-standard CCG rules in the treebank.

[6] Training: Sections 02-21; development: Section 00; test Section 23.

[7] https://radimrehurek.com/gensim/

Supertagger	Dev	Test
C&C (gold POS)	92.60	93.32
C&C (auto POS)	91.50	92.02
RNN	93.07	93.00
BRNN	**93.49**	**93.52**

Table 2: 1-best supertagging accuracy comparison.

To train all the models, we used a fixed learning rate of 0.0025 and did not truncate the gradients for BPTT, except for training the greedy RNN parsing model where we used a BPTT step size of 9. We applied dropout at the input layer (Legrand and Collobert, 2015), with a dropout rate of 0.25 for the supertagger and 0.30 for the parser.

5.1 Supertagging Results

Table 2 shows 1-best supertagging results. The MaxEnt C&C supertagger uses POS tag features and a tag dictionary, neither of which are used by the RNN supertaggers. For all supertaggers, the same set of 425 lexical categories is used (Clark and Curran, 2007). On the test set, our BRNN supertagger achieves a 1-best accuracy of 93.52%, an absolute improvement of 0.52% over the RNN model, demonstrating the usefulness of contextual information from both input directions.

Fig. 3a shows multi-tagging accuracy comparison for the three supertaggers by varying the variable-width beam probability cut-off value β for each supertagger. The β value determines the average number of supertags (ambiguity) assigned to each word by pruning supertags whose probabilities are not within β times the probability of the 1-best supertag; for this experiment we used β values ranging from 0.09 to 2×10^{-4} and it can be seen that the BRNN supertagger consistently achieves better accuracies at similar ambiguity levels.

Finally, all shift-reduce CCG parsers mentioned in this paper take multi-tagging output obtained with a *fixed* β for training and testing; and in general, a smaller β value can be used by a shift-reduce CCG parser than by the C&C parser. This is because a β value too small may explode the dynamic program of the C&C parser, and it thus relies on an adaptive supertagging strategy (Clark and Curran, 2007), by starting from a large β value and backing off to smaller values if no spanning analysis can found with the current β.

	Supertagger β			
	0.09	0.08	0.07	0.06
$b = 1$	84.61	84.58	84.55	84.50
$b = 2$	84.94	84.86	84.86	84.81
$b = 4$	85.01	84.95	84.92	84.92
$b = 6$	**85.02**	84.96	84.94	84.93
$b = 8$	**85.02**	84.99	84.96	84.95
$b = 16$	85.01	84.95	84.97	84.98

Table 3: The effect on dev F1 by varying the beam size and supertagger β value for the greedy RNN model.

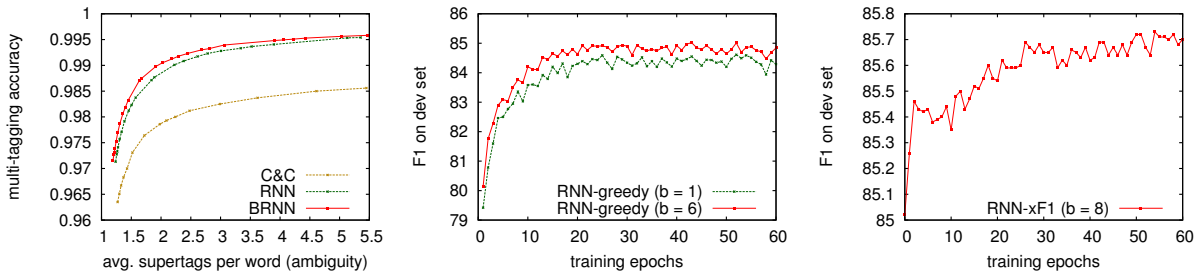
5.2 Parsing Results

To pretrain the greedy model, we trained 10 cross-validated BRNN supertagging models to supply supertags for the parsing model, and used a supertagger β value of 0.00025 which gave on average 5.02 supertags per word. We ran SGD training for 60 epochs, observing no accuracy gains after that, and the best greedy model was obtained after the 52$^{\text{nd}}$ epoch (Fig. 3b).

Furthermore, we found that using a relatively smaller supertagger β value (higher ambiguity) for training, and a larger β value (lower ambiguity) for testing, resulted in more accurate models; and we chose the final β value used for the greedy model to be 0.09 using the dev set (Table 3). This observation was different from Zhang and Clark (2011) and Xu et al. (2014), which are two shift-reduce CCG parsers using the averaged perceptron and beam search (Collins, 2002; Collins and Roark, 2004; Zhang and Clark, 2008): they used the same β values for training and testing, which resulted in lower accuracy for our greedy model.

Table 3 also shows the effect on dev F1 by using different beam sizes at test time for the greedy model: with $b = 6$, we obtained an accuracy of 85.02%, an improvement of 0.41% over $b = 1$ (with a β value of 0.09); we saw accuracy gains up to $b = 8$ (with very minimal gains with $b = 16$ for β values 0.06 and 0.07), after which the accuracy started to drop. F1 on dev with $b = 6$ across all training epochs are shown in Fig. 3b as well, and the best model was obtained after the 43$^{\text{rd}}$ epoch.

For the xF1 model, we used $b = 8$ and a supertagger β value of 0.09 for both training and testing. Fig.3c shows dev F1 versus the number of training epochs. The best dev F1 was obtained after the 54$^{\text{th}}$ epoch with an accuracy of 85.73%, 1.12%



(a) multi-tagging accuracy on dev (b) F1 on dev with RNN-greedy (c) F1 on dev with RNN-xF1

Figure 3: Experiment results on the dev set. (a) shows multi-tagging accuracy using the best tagging model. (b) shows F1 scores for the greedy RNN parsing models with beam size $b \in \{1, 6\}$. (c) shows F1 scores for the xF1 models with $b = 8$.

	Section 00				Section 23				
Model	LP	LR	LF	CAT	LP	LR	LF	CAT	Speed
C&C (normal)	85.18	82.53	83.83	92.39	85.45	83.97	84.70	92.83	97.90
C&C (hybrid)	86.07	82.77	84.39	92.57	86.24	84.17	85.19	93.00	95.25
Zhang and Clark (2011) ($b = 16$)	87.15	82.95	85.00	92.77	87.43	83.61	85.48	93.12	-
Zhang and Clark (2011)* ($b = 16$)	86.76	83.15	84.92	92.64	87.04	84.14	85.56	92.95	49.54
Xu et al. (2014) ($b = 128$)	86.29	**84.09**	85.18	92.75	87.03	**85.08**	86.04	93.10	12.85
RNN-greedy ($b = 1$)	88.12	81.38	84.61	93.42	88.53	81.65	84.95	93.57	337.45
RNN-greedy ($b = 6$)	87.96	82.27	85.02	93.47	88.54	82.77	85.56	93.68	96.04
RNN-xF1 ($b = 8$)	**88.20**	83.40	**85.73**	**93.56**	**88.74**	84.22	**86.42**	**93.87**	67.65

Table 4: Final parsing results on Section 00 and Section 23 (100% coverage). Zhang and Clark (2011)* is a reimplementation of the original. All speed results (sents/sec) are obtained using Section 23 and precomputation is used for all RNN parsers. LP (labeled precision); LR (labeled recall); LF (labeled F-score over CCG dependencies); CAT (lexical category assignment accuracy). All experiments using auto POS.

higher than that of the greedy model with $b = 1$ and 0.71% higher than the greedy model with $b \in \{6, 8\}$. This result improves over shift-reduce CCG models of Zhang and Clark (2011) and Xu et al. (2014) by 0.73% and 0.55%, respectively (Table 4).

Table 4 summarizes final results.[8] RNN-xF1, the xF1 trained beam-search model, is currently the most accurate shift-reduce CCG parser, achieving a final F-score of 86.42%, and gives an F-score improvement of 1.47% over the greedy RNN baseline. We show the results for the model of Xu et al. (2014) for reference only, since it uses a more sophisticated dependency, rather than normal-form derivation, model.

At test time, we also used the precomputation trick of Devlin et al. (2014) to speed up the RNN models by caching the top 20K word embeddings and all POS embeddings,[9] and this made the greedy RNN parser more than 3 times faster than the C&C parser (all speed experiments were measured on a workstation with an Intel Core i7 4.0GHz CPU).[10]

6 Related Work

Optimizing for Task-specific Metrics. Our training objective is largely inspired by task-specific optimization for parsing and MT. Goodman (1996) proposed algorithms for optimizing a parser for various constituent matching criteria, and it was one of the earliest work that we are aware of on optimizing a parser for evaluation metrics. Smith and Eisner (2006) proposed a framework for minimizing expected loss for log-linear models and applied it to dependency parsing by optimizing for labeled attachment scores, although they obtained little per-

[8] The C&C parser fails to produce spanning analyses for a very small number of sentences (Clark and Curran, 2007) on both dev and test sets, which is not the case for any of the shift-reduce parsers; and for brevity, we omit C&C coverage results.

[9] We used $b = 8$ to do the precomputation.

[10] The speed results for the C&C parser were obtained using the per-compiled C&C binary for Linux available from `http://svn.ask.it.usyd.edu.au/trac/candc/wiki/Download`.

formance improvements. Auli and Lopez (2011) optimized the C&C parser for F-measure. However, they used the softmax-margin (Gimpel and Smith, 2010) objective, which required decomposing precision and recall statistics over parse forests. Instead, we directly optimize for an F-measure loss. In MT, task-specific optimization has also received much attention (e.g., see Och (2003)). Closely related to our work, Gao and He (2013) proposed training a Markov random field translation model as an additional component in a log-linear phrase-based translation system using a *k*-best list based expected BLEU objective; using the same objective, Auli et al. (2014) and Auli and Gao (2014) trained a large scale phrase-based reordering model and a RNN language model respectively, all as additional components within a log-linear translation model. In contrast, our RNN parsing model is trained in an end-to-end fashion with an expected F-measure loss and all parameters of the model are optimized using backpropagation and SGD.

Parsing with RNNs. A line of work is devoted to parsing with RNN models, including using RNNs (Miikkulainen, 1996; Mayberry and Miikkulainen, 1999; Legrand and Collobert, 2015; Watanabe and Sumita, 2015) and LSTM (Hochreiter and Schmidhuber, 1997) RNNs (Vinyals et al., 2015; Ballesteros et al., 2015; Dyer et al., 2015; Kiperwasser and Goldberg, 2016). Legrand and Collobert (2015) used RNNs to learn conditional distributions over syntactic rules; Vinyals et al. (2015) explored sequence-to-sequence learning (Sutskever et al., 2014) for parsing; Ballesteros et al. (2015) utilized character-level representations and Kiperwasser and Goldberg (2016) built an easy-first dependency parser using tree-structured compositional LSTMs. However, all these parsers use greedy search and are trained using the maximum likelihood criterion (except Kiperwasser and Goldberg (2016), who used a margin-based objective). For learning global models, Watanabe and Sumita (2015) used a margin-based objective, which was not optimized for the evaluation metric; although not using RNNs, Weiss et al. (2015) proposed a method using the averaged perceptron with beam search (Collins, 2002; Collins and Roark, 2004; Zhang and Clark, 2008), which required fixing the neural network representations, and thus their model parameters were not learned using end-to-end backpropagation.

Finally, a number of recent work (Bengio et al., 2015; Vaswani and Sagae, 2016) explored training neural network models for parsing and other tasks such that the network learns from the oracle as well as its own predictions, and are hence more robust to search errors during inference. In principle, these techniques are largely orthogonal to both global learning and task-based optimization, and we would expect further accuracy gains are possible by combining these techniques in a single model.

7 Conclusion

Neural network shift-reduce parsers are often trained by maximizing likelihood, which does not optimize towards the final evaluation metric. In this paper, we addressed this problem by developing expected F-measure training for an RNN shift-reduce parsing model. We have demonstrated the effectiveness of our method on shift-reduce parsing for CCG, achieving higher accuracies than all shift-reduce CCG parsers to date and the de facto C&C parser.[11] We expect the general framework will be applicable to models using other types of neural networks such as feed-forward or LSTM nets, and to shift-reduce parsers for constituent and dependency parsing.

Acknowledgments

We thank the anonymous reviewers for their detailed comments. Xu acknowledges the Carnegie Trust for the Universities of Scotland and the Cambridge Trusts for funding. Clark is supported by ERC Starting Grant DisCoTex (306920) and EPSRC grant EP/I037512/1.

References

Alfred V Aho and Jeffrey D Ullman. 1972. *The theory of parsing, translation, and compiling*. Prentice-Hall.

Michael Auli and Jianfeng Gao. 2014. Decoder integration and expected BLEU training for recurrent neural network language models. In *Proc. of ACL (Volume 2)*.

[11] Auli and Lopez (2011) present higher accuracies but on a different coverage to enable a comparison to Fowler and Penn (2010). Their results are thus not directly comparable to ours.

Michael Auli and Adam Lopez. 2011. Training a log-linear parser with loss functions via softmax-margin. In *Proc. of EMNLP*.

Michael Auli, Michel Galley, and Jianfeng Gao. 2014. Large-scale expected BLEU training of phrase-based reordering models. In *Proc. of EMNLP*.

Miguel Ballesteros, Chris Dyer, and Noah A Smith. 2015. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proc. of EMNLP*.

Srinivas Bangalore and Aravind K Joshi. 1999. Supertagging: An approach to almost parsing. In *Computational linguistics*. MIT Press.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam M. Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proc. of NIPS*.

Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*. Springer.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proc. of ACL*.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proc. of EMNLP*.

Stephen Clark and James R Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proc. of COLING*.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. In *Computational Linguistics*. MIT Press.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proc. of ACL*.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proc. of ICML*.

Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP*.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *Proc. of ACL*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proc. of ACL*.

Jeffrey L Elman. 1990. Finding structure in time. In *Cognitive science*. Elsevier.

Timothy A.D. Fowler and Gerald Penn. 2010. Accurate context-free parsing with Combinatory Categorial Grammar. In *Proc. of ACL*.

Jianfeng Gao and Xiaodong He. 2013. Training MRF-based phrase translation models using gradient ascent. In *Proc. of NAACL*.

Jianfeng Gao, Xiaodong He, Wen-tau Yih, and Li Deng. 2014. Learning continuous phrase representations for translation modeling. In *Proc. of ACL*.

Kevin Gimpel and Noah Smith. 2010. Softmax-margin CRFs: training log-linear models with cost functions. In *Proc. of NAACL*.

Yoav Goldberg, Kai Zhao, and Liang Huang. 2013. Efficient implementation for beam search incremental parsers. In *Proc. of ACL (Volume 2)*.

Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by back-propagation through structure. In *Proc. of IEEE International Conference on Neural Networks*.

Joshua Goodman. 1996. Parsing algorithms and metrics. In *Proc. of ACL*.

Xiaodong He and Li Deng. 2012. Maximum expected BLEU training of phrase and lexicon translation models. In *Proc. of ACL*.

Geoffrey E Hinton. 2002. Training products of experts by minimizing contrastive divergence. In *Neural computation*. MIT Press.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. In *Neural computation*. MIT Press.

Julia Hockenmaier and Mark Steedman. 2007. CCGBank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. In *Computational Linguistics*. MIT Press.

Julia Hockenmaier. 2003. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Easy-first dependency parsing with hierarchical tree LSTMs. *arXiv:1603.00375*.

Marco Kuhlmann and Giorgio Satta. 2014. A new parsing algorithm for combinatory categorial grammar. In *Transactions of the Association for Computational Linguistics*. ACL.

Joël Legrand and Ronan Collobert. 2015. Joint RNN-based greedy parsing and word composition. In *Proc. of ICLR*.

Marshall R. Mayberry and Risto Miikkulainen. 1999. Sardsrn: A neural network shift-reduce parser. In *Proc. of IJCAI*.

Risto Miikkulainen. 1996. Subsymbolic case-role analysis of sentences with embedded clauses. In *Cognitive Science*. Elsevier.

Tomáš Mikolov. 2012. *Statistical Language Models Based on Neural Networks*. Ph.D. thesis, Brno University of Technology.

J. Nivre and M Scholz. 2004. Deterministic dependency parsing of English text. In *Proc. of COLING*.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. In *Computational Linguistics*. MIT Press.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proc. of ACL*.

Antti-Veikko I Rosti, Bing Zhang, Spyros Matsoukas, and Richard Schwartz. 2010. BBN system description for WMT10 system combination task. In *Proc. of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1988. Learning representations by back-propagating errors. In *Nature*.

Kenji Sagae and Alon Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proc. of COLING/ACL*.

David A. Smith and Jason Eisner. 2006. Minimum-risk annealing for training log-linear models. In *Proc. of COLING-ACL*.

Ilya Sutskever, Oriol Vinyals, and Quoc Le. 2014. Sequence to sequence learning with neural networks. In *Proc. of NIPS*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proc. of ACL*.

Ashish Vaswani and Kenji Sagae. 2016. Efficient structured inference for transition-based parsing with neural networks and error states. In *Transactions of the Association for Computational Linguistics*. ACL.

Krishnamurti Vijay-Shanker and David J Weir. 1993. Parsing some constrained grammar formalisms. In *Computational Linguistics*. MIT Press.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Proc. of NIPS*.

Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. In *Proc. of ACL*.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proc. of ACL*.

Wenduan Xu, Stephen Clark, and Yue Zhang. 2014. Shift-Reduce CCG parsing with a dependency model. In *Proc. of ACL*.

Wenduan Xu, Michael Auli, and Stephen Clark. 2015. CCG supertagging with a recurrent neural network. In *Proc. of ACL (Volume 2)*.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proc. of EMNLP*.

Yue Zhang and Stephen Clark. 2011. Shift-Reduce CCG parsing. In *Proc. of ACL*.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proc. of ACL*.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proc. of ACL*.