LSTM Shift-Reduce CCG Parsing

Wenduan Xu
Computer Laboratory
University of Cambridge
wx217@cam.ac.uk

Abstract

We describe a neural shift-reduce parsing model for CCG, factored into four unidirectional LSTMs and one bidirectional LSTM. This factorization allows the linearization of the complete parsing history, and results in a highly accurate greedy parser that outperforms all previous beam-search shift-reduce parsers for CCG. By further deriving a globally optimized model using a task-based loss, we improve over the state of the art by up to 2.67% labeled F1.

1 Introduction

Combinatory Categorial Grammar (CCG; Steedman, 2000) parsing is challenging due to its so-called "spurious" ambiguity that permits a large number of non-standard derivations (Vijay-Shanker and Weir, 1993; Kuhlmann and Satta, 2014). To address this, the *de facto* models resort to chart-based CKY (Hockenmaier, 2003; Clark and Curran, 2007), despite CCG being naturally compatible with shift-reduce parsing (Ades and Steedman, 1982). More recently, Zhang and Clark (2011) introduced the first shift-reduce model for CCG, which also showed substantial improvements over the long-established state of the art (Clark and Curran, 2007).

The success of the shift-reduce model (Zhang and Clark, 2011) can be tied to two main contributing factors. First, without any feature locality restrictions, it is able to use a much richer feature set; while intensive feature engineering is inevitable, it has nevertheless delivered an effective and conceptually simpler alternative for both parameter estimation and inference. Second, it couples beam search with global optimization (Collins, 2002; Collins and Roark, 2004; Zhang and Clark, 2008), which makes it less prone to search errors than fully greedy models (Huang et al., 2012).

In this paper, we present a neural architecture for shift-reduce CCG parsing based on long short-term memories (LSTMs; Hochreiter and Schmidhuber, 1997). Our model is inspired by Dyer et al. (2015), in which we explicitly linearize the complete history of parser states in an incremental fashion by requiring no feature engineering (Zhang and Clark, 2011; Xu et al., 2014), and no atomic feature sets (Chen and Manning, 2014). However, a key difference is that we achieve this linearization without relying on any additional control operations or compositional tree structures (Socher et al., 2010; Socher et al., 2011; Socher et al., 2013), both of which are vital in the architecture of Dyer et al. (2015). Crucially, unlike the sequence-to-sequence transduction model of Vinyals et al. (2015), which primarily conditions on the input words, our model is sensitive to all aspects of the parsing history, including arbitrary positions in the input.

As another contribution, we present a global LSTM parsing model by adapting an expected F-measure loss (Xu et al., 2016). As well as naturally incorporating beam search during training, this loss optimizes the model towards the final evaluation metric (Goodman, 1996; Smith and Eisner, 2006; Auli and Lopez, 2011b), allowing it to learn shift-reduce action sequences that lead to parses with high expected F-scores. We further show the globally optimized model can be leveraged with greedy inference, resulting in a deterministic parser as accurate

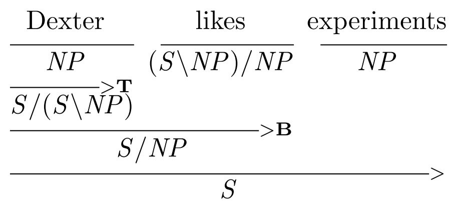


Figure 1: A CCG derivation, in which each point corresponds to the result of a shift-reduce action. In this example, composition (**B**) and application ($>$) are re actions, and type-raising (**T**) is a un action.

as its beam-search counterpart.

On standard CCGBank tests, we clearly outperform all previous shift-reduce CCG parsers; and by combining the parser with an attention-based LSTM supertagger (§4), we obtain further significant improvements (§5).

2 Shift-Reduce CCG Parsing

CCG is strongly lexicalized by definition. A CCG grammar extracted from CCGBank (Hockenmaier and Steedman, 2007) contains over 400 lexical types and over 1,500 non-terminals (Clark and Curran, 2007), which is an order of magnitude more than those of a typical CFG parser. This lexicalized nature raises another unique challenge for parsing—any parsing model for CCG needs to perform lexical disambiguation. This is true even in the approach of Fowler and Penn (2010), in which a context-free cover grammar extracted from CCGBank is used to parse CCG. Indeed, as noted by Auli and Lopez (2011a), the search problem for CCG parsing is equivalent to finding an optimal derivation in the weighted intersection of a regular language (generated by the supertagger) and a mildly context-sensitive language (generated by the parser), which can quickly become expensive.

The shift-reduce paradigm (Aho and Ullman, 1972; Yamada and Matsumoto, 2003; Nivre and Scholz, 2004) applied to CCG (Zhang and Clark, 2011) presents a more elegant solution to this problem by allowing the parser to conduct lexical assignment "incrementally" as a complete parse is being built by the decoder. This is not possible with a chart-based parser, in which complete derivations must be built first. Therefore, a shift-reduce parser is able to consider a much larger set of categories per word for a given input, achieving higher lexical assignment accuracy than the C&C parser (Clark and Curran, 2007), even with the same supertagging model (Zhang and Clark, 2011; Xu et al., 2014).

In our parser, we follow this strategy and adopt the Zhang and Clark (2011) style shift-reduce transition system, which assumes a set of lexical categories has been assigned to each word using a supertagger (Bangalore and Joshi, 1999; Clark and Curran, 2004). Parsing then proceeds by applying a sequence of actions to transform the input maintained on a *queue*, into partially constructed derivations, kept on a *stack*, until the queue and available actions on the stack are both exhausted. At each time step, the parser can choose to *shift* (sh) one of the lexical categories of the front word onto the stack, and remove that word from the queue; *reduce* (re) the top two subtrees on the stack using a CCG rule, replacing them with the resulting category; or take a *unary* (un) action to apply a CCG type-raising or type-changing rule to the stack-top element. For example, the deterministic sequence of shift-reduce actions that builds the derivation in Fig.1 is: sh $\Rightarrow NP$, un $\Rightarrow S/(S\backslash NP)$, sh $\Rightarrow (S\backslash NP)/NP$, re $\Rightarrow S/NP$, sh $\Rightarrow NP$ and re $\Rightarrow S$, where we use $\Rightarrow$ to indicate the CCG category produced by an action.[1]

3 LSTM Shift-Reduce Parsing

3.1 LSTM

Recurrent neural networks (RNNs; e.g., see Elman, 1990) are factored into an input layer $\mathbf{x}_t$ and a hidden state (layer) $\mathbf{h}_t$ with recurrent connections, and they can be represented by the following recurrence:

$$\mathbf{h}_t = \Phi_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}), \tag{1}$$

where $\mathbf{x_t}$ is the current input, $\mathbf{h}_{t-1}$ is the previous hidden state and Φ is a set of affine transformations parametrized by θ. Here, we use a variant of RNN referred to as LSTMs, which augment Eq. 1 with a cell state, $\mathbf{c}_t$, s.t.

$$\mathbf{h}_t, \mathbf{c}_t = \Phi_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}). \tag{2}$$

Compared with conventional RNNs, this extra facility gives LSTMs more persistent memories over

[1]Our parser models normal-form derivations (Eisner, 1996) in CCGBank. However, unlike Zhang and Clark (2011), derivations are not restricted to be normal-form during inference.

longer time delays and makes them less susceptible to the vanishing gradient problem (Bengio et al., 1994). Hence, they are better at modeling temporal events that are arbitrarily far in a sequence.

Several extensions to the vanilla LSTM have been proposed over time, each with a modified instantiation of Φ_θ that exerts refined control over e.g., whether the cell state could be reset (Gers et al., 2000) or whether extra connections are added to the cell state (Gers and Schmidhuber, 2000). Our instantiation is as follows for all LSTMs:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fc}\mathbf{c}_{t-1} + \mathbf{b}_f) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \\
&\qquad \mathbf{i}_t \odot \tanh(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_t + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}$$

where σ is the sigmoid activation and $\odot$ is the element-wise product.

In addition to unidirectional LSTMs that model an input sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$ in a strict left-to-right order, we also use bidirectional LSTMs (BLSTMs; Graves and Schmidhuber, 2005), which read the input from both directions with two independent LSTMs. At each step, the forward hidden state $\mathbf{h}_t$ is computed using Eq. 2 for $t = (0, 1, \ldots, n-1)$; and the backward hidden state $\hat{\mathbf{h}}_t$ is computed similarly but from the reverse direction for $t = (n-1, n-2, \ldots, 0)$. Together, the two hidden states at each step t capture both past and future contexts, and the representation for each $\mathbf{x}_t$ is obtained as the concatenation $[\mathbf{h}_t; \hat{\mathbf{h}}_t]$.

3.2 Embeddings

The neural network model employed by Chen and Manning (2014), and followed by a number of other parsers (Weiss et al., 2015; Zhou et al., 2015; Ambati et al., 2016; Andor et al., 2016; Xu et al., 2016) allows higher-order feature conjunctions to be automatically discovered from a set of dense feature embeddings. However, a set of atomic feature templates, which are only sensitive to contexts from the top few elements on the stack and queue are still needed to dictate the choice of these embeddings. Instead, we dispense with such templates and seek to design a model that is sensitive to both local and non-local contexts, on both the stack and queue.

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

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

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

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

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

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

Figure 2: The shift-reduce deduction system. For the sh deduction, x_{w_j} denotes an available lexical category for w_j; for re, $\langle x \rangle$ denotes the set of dependencies on x.

Consequently, embeddings represent atomic input units that are added to our parser and are preserved throughout parsing. In total we use four types of embeddings, namely, word, CCG category, POS and action, where each has an associated look-up table that maps a string of that type to its embedding. The look-up table for words is $\mathcal{L}_w \in \mathbb{R}^{k \times |w|}$, where k is the embedding dimension and $|w|$ is the size of the vocabulary. Similarly, we have look-up tables for CCG categories, $\mathcal{L}_c \in \mathbb{R}^{l \times |c|}$, for the three types of actions, $\mathcal{L}_a \in \mathbb{R}^{m \times 3}$, and for POS tags, $\mathcal{L}_p \in \mathbb{R}^{n \times |p|}$.

3.3 Model

Parser. Fig. 2 shows the deduction system of our parser.[2] We denote each parse item as $(j, \delta, \beta, \Delta)$, where j is the positional index of the word at the front of the queue, δ is the stack (with its top element s_0 to the right), and β is the queue (with its top element w_j to the left) and Δ is the set of CCG dependencies realized for the input consumed so far. Each item is also associated with a step indicator t, signifying the number of actions applied to it and the goal is reached in $2n - 1 + \mu$ steps, where μ is the total number of un actions. We also define each action in our parser as a 4-tuple $(\tau_t, c_t, w_{c_t}, p_{w_{c_t}})$, where $\tau_t \in \{\mathsf{sh}, \mathsf{re}, \mathsf{un}\}$ for $t \ge 1$, c_t is the resulting category of τ_t, and w_{c_t} is the head word attached to

[2]We assume greedy inference unless otherwise stated.

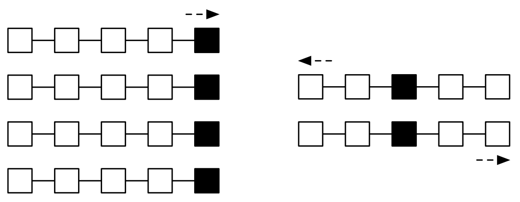


Figure 3: An example representation for a parse item at time step t, with the 4 unidirectional LSTMs (left) and the bidirectional LSTM (right). The shaded cells on the left represent $\boldsymbol{\delta}_t = [\mathbf{h}^{\mathrm{U}}_t; \mathbf{h}^{\mathrm{V}}_t; \mathbf{h}^{\mathrm{X}}_t; \mathbf{h}^{\mathrm{Y}}_t]$ (Eq. 3), and the shaded cells on the right represent $\mathbf{w}_j = [\mathbf{h}^{\mathrm{W}}_j; \hat{\mathbf{h}}^{\mathrm{W}}_j]$.

c_t with $p_{w_{c_t}}$ being its POS tag.[3]

LSTM model. LSTMs are designed to handle time-series data, in a purely sequential fashion, and we try to exploit this fact by completely linearizing all aspects of the parsing history. Concretely, we factor the model as five LSTMs, comprising four unidirectional ones, denoted as U, V, X and Y, and an additional BLSTM, denoted as W (Fig. 3). Before parsing each sentence, we feed W with the complete input (padded with a special embedding $\perp$ as the end of sentence token); and we use $\mathbf{w}_j = [\mathbf{h}^{\mathrm{W}}_j; \hat{\mathbf{h}}^{\mathrm{W}}_j]$ to represent w_j in subsequent steps.[4] We also add $\perp$ to the other 4 unidirectional LSTMs as initialization.

Given this factorization, the stack representation for a parse item $(j, \delta, \beta, \Delta)$ at step t, for $t \geq 1$, is obtained as

$$\boldsymbol{\delta}_t = [\mathbf{h}^{\mathrm{U}}_t; \mathbf{h}^{\mathrm{V}}_t; \mathbf{h}^{\mathrm{X}}_t; \mathbf{h}^{\mathrm{Y}}_t], \tag{3}$$

and together with $\mathbf{w}_j$, $[\boldsymbol{\delta}_t; \mathbf{w}_j]$ gives a representation for the parse item. For the axiom item, we represent it as $[\boldsymbol{\delta}_0; \mathbf{w}_0]$, where $\boldsymbol{\delta}_0 = [\mathbf{h}^{\mathrm{U}}_\perp; \mathbf{h}^{\mathrm{V}}_\perp; \mathbf{h}^{\mathrm{X}}_\perp; \mathbf{h}^{\mathrm{Y}}_\perp]$.

Each time the parser applies an action $(\tau_t, c_t, w_{c_t}, p_{w_{c_t}})$, we update the model by adding the embedding of τ_t, denoted as $\mathcal{L}_a(\tau_t)$, onto U, and adding the other three embeddings of the action 4-tuple, that is, $\mathcal{L}_c(c_t)$, $\mathcal{L}_w(w_{c_t})$ and $\mathcal{L}_p(p_{w_{c_t}})$, onto V, X and Y respectively.

To predict the next action, we first derive an action hidden layer $\mathbf{b}_t$, by passing the parse item representation $[\boldsymbol{\delta}_t; \mathbf{w}_j]$ through an affine transformation, s.t.

$$\mathbf{b}_t = f(\mathbf{B}[\boldsymbol{\delta}_t; \mathbf{w}_j] + \mathbf{r}), \tag{4}$$

where $\mathbf{B}$ is a parameter matrix of the model, $\mathbf{r}$ is a bias vector and f is a ReLU non-linearity (Nair and Hinton, 2010). Then we apply another affine transformation (with $\mathbf{A}$ as the weights and $\mathbf{s}$ as the bias) to $\mathbf{b}_t$:

$$\mathbf{a}_t = f(\mathbf{A}\mathbf{b}_t + \mathbf{s}),$$

and obtain the probability of the i^{th} action in $\mathbf{a}_t$ as

$$p(\tau^i_t | \mathbf{b}_t) = \frac{\exp\{\mathbf{a}^i_t\}}{\sum_{\tau^k_t \in \mathcal{T}(\delta_t, \beta_t)} \exp\{\mathbf{a}^k_t\}},$$

where $\mathcal{T}(\delta_t, \beta_t)$ is the set of feasible actions for the current parse item, and $\tau^i_t \in \mathcal{T}(\delta_t, \beta_t)$.

3.4 Derivations and Dependency Structures

Our model naturally linearizes CCG derivations "incrementally" following their post-order traversals. As such, the four unidirectional LSTMs always have the same number of steps; and at each step, the concatenation of their hidden states (Eq. 3) represents a point in a CCG derivation (i.e., an action 4-tuple). Due to the large amount of flexibility in how dependencies are realized in CCG (Hockenmaier, 2003; Clark and Curran, 2007), and in line with most existing CCG parsing models, including dependency models, we have chosen to model CCG derivations, rather than dependency structures.[5] We also hypothesize that tree structures are not necessary for the current model, since they are already implicit in the linearized derivations; similarly, we have found the action embeddings to be nonessential (§5.2).

3.5 Training

As a baseline, we first train a greedy model, in which we maximize the log-likelihood of each target action in the training data. More specifically, let $(\tau^g_1, \ldots, \tau^g_{T_n})$ be the gold-standard action sequence for a training sentence n, a cross-entropy criterion is used to obtain the error gradients, and for each sentence, training involves minimizing

$$L(\theta) = -\log \prod_{t=1}^{T_n} p(\tau^g_t | \mathbf{b}_t) = -\sum_{t=1}^{T_n} \log p(\tau^g_t | \mathbf{b}_t),$$

where θ is the set of all parameters in the model.

[3] In case of multiple heads, we always choose the first one.

[4] Word and POS embeddings are concatenated at each input position j, for $0 \leq j < n$; and $\mathbf{w}_n = [\mathbf{h}^{\mathrm{W}}_\perp; \hat{\mathbf{h}}^{\mathrm{W}}_\perp]$.

[5] Most CCG dependency models (e.g., see Clark and Curran (2007) and Xu et al. (2014)) model CCG derivations with dependency features.

As other greedy models (e.g., see Chen and Manning (2014) and Dyer et al. (2015)), our greedy model is *locally* optimized, and suffers from the label bias problem (Andor et al., 2016). A partial solution to this is to use beam search at test time, thereby recovering higher scoring action sequences that would otherwise be unreachable with fully greedy inference. In practice, this has limited effect (Table 2), and a number of more principled solutions have been recently proposed to derive *globally* optimized models during training (Watanabe and Sumita, 2015; Weiss et al., 2015; Zhou et al., 2015; Andor et al., 2016). Here, we extend our greedy model into a global one by adapting the expected F-measure loss of Xu et al. (2016). To our best knowledge, this is the first attempt to train a globally optimized LSTM shift-reduce parser.

Let $\theta = \{\mathbf{U}, \mathbf{V}, \mathbf{X}, \mathbf{Y}, \mathbf{W}, \mathbf{B}, \mathbf{A}\}$ be the weights of the baseline greedy model,[6] we initialize the weights of the global model, which has the same architecture as the baseline, to θ, and we reoptimize θ in multiple training epochs as follows:

1. Pick a sentence x_n from the training set, decode it with beam search, and generate a k-best list of output parses with the *current* θ, denoted as $\Lambda(x_n)$.[7]

2. For each parse y_i in $\Lambda(x_n)$, compute its sentence-level F1 using the set of dependencies in the Δ field of its parse item. In addition, let $|y_i|$ be the total number of actions that derived y_i and $s_\theta(y_i^j)$ be the softmax action score of the j^{th} action, given by the LSTM model. Compute the log-linear score of its action sequence as $\rho(y_i) = \sum_{j=1}^{|y_i|} \log s_\theta(y_i^j)$.

3. Compute the negative expected F1 objective (defined below) for x_n and minimize it using stochastic gradient descent (maximizing expected F1). Repeat these three steps for the remaining sentences.

[6]We use boldface letters to designate the weights of the corresponding LSTMs, and omit bias terms for brevity.

[7]As in Xu et al. (2016), we did not preset k, and found $k = 11.06$ on average with a beam size of 8 that we used for this training.

More formally, the loss $J(\theta)$, is defined as

$$\begin{aligned} J(\theta) &= -\text{XF1}(\theta) \\ &= -\sum_{y_i \in \Lambda(x_n)} p(y_i|\theta)\text{F1}(\Delta_{y_i}, \Delta_{x_n}^G), \end{aligned}$$

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

$$p(y_i|\theta) = \frac{\exp\{\gamma\rho(y_i)\}}{\sum_{y \in \Lambda(x_n)} \exp\{\gamma\rho(y)\}},$$

where γ is a parameter that sharpens or flattens the distribution (Tromble et al., 2008).[8] Different from the maximum-likelihood objective, XF1 optimizes the model on a sequence level and towards the final evaluation metric, by taking into account all action sequences in $\Lambda(x_n)$.

4 Attention-Based LSTM Supertagging

In addition to the size of the label space, supertagging is difficult because CCG categories can encode long-range dependencies and tagging decisions frequently depend on non-local contexts. For example, in *He went to the zoo with a cat*, a possible category for *with*, $(S\backslash NP)\backslash(S\backslash NP)/NP$, depends on the word *went* further back in the sentence.

Recently a number of RNN models have been proposed for CCG supertagging (Xu et al., 2015; Lewis et al., 2016; Vaswani et al., 2016; Xu et al., 2016), and such models show dramatic improvements over non-recurrent models (Lewis and Steedman, 2014b). Although the underlying models differ in their exact architectures, all of them make each tagging decision using only the hidden states at the current input position, and this imposes a potential bottleneck in the model. To mitigate this, we generalize the attention mechanisms of Bahdanau et al. (2015) and Luong et al. (2015), and adapt them to supertagging, by allowing the model to explicitly use hidden states from more than one input positions for tagging each word. Similar to Bahdanau et al. (2015) and Luong et al. (2015), a key feature in

[8]We found $\gamma = 1$ to be a good choice during development.

our model is a soft alignment vector that weights the relative importance of the considered hidden states.

For an input sentence $w_0, w_1, \ldots, w_{n-1}$, we consider $\mathbf{w}_t = [\mathbf{h}_t; \hat{\mathbf{h}}_t]$ (§3.1) to be the representation of the t^{th} word ($0 \leq t < n$, $\mathbf{w}_t \in \mathbb{R}^{2d \times 1}$), given by a BLSTM with a hidden state size d for both its forward and backward layers.[9] Let k be a context window size hyperparameter, we define $\mathbf{H}_t \in \mathbb{R}^{2d \times (k-1)}$ as

$$\mathbf{H}_t = [\mathbf{w}_{t-\lfloor k/2 \rfloor}, \ldots, \mathbf{w}_{t-1}, \mathbf{w}_{t+1}, \ldots, \mathbf{w}_{t+\lfloor k/2 \rfloor}],$$

which contains representations for all words in the size k window except $\mathbf{w}_t$. At each position t, the attention model derives a context vector $\mathbf{c}_t \in \mathbb{R}^{2d \times 1}$ (defined below) from $\mathbf{H}_t$, which is used in conjunction with $\mathbf{w}_t$ to produce an attentional hidden layer:

$$\mathbf{x}_t = f(\mathbf{M}[\mathbf{c}_t; \mathbf{w}_t] + \mathbf{m}), \tag{5}$$

where f is a ReLU non-linearity, $\mathbf{M} \in \mathbb{R}^{g \times 4d}$ is a learned weight matrix, $\mathbf{m}$ is a bias term, and g is the size of $\mathbf{x}_t$. Then $\mathbf{x}_t$ is used to produce another hidden layer (with $\mathbf{N}$ as the weights and $\mathbf{n}$ as the bias):

$$\mathbf{z}_t = \mathbf{N}\mathbf{x}_t + \mathbf{n},$$

and the predictive distribution over categories is obtained by feeding $\mathbf{z}_t$ through a softmax activation.

In order to derive the context vector $\mathbf{c}_t$, we first compute $\mathbf{b}_t \in \mathbb{R}^{(k-1) \times 1}$ from $\mathbf{H}_t$ and $\mathbf{w}_t$ using $\boldsymbol{\alpha} \in \mathbb{R}^{1 \times 4d}$, s.t. the i^{th} entry in $\mathbf{b}_t$ is

$$\mathbf{b}_t^i = \boldsymbol{\alpha}[\mathbf{w}_{T[i]}; \mathbf{w}_t],$$

for $i \in [0, k-1)$, $T = [t - \lfloor k/2 \rfloor, \ldots, t-1, t+1, \ldots, t + \lfloor k/2 \rfloor]$; and $\mathbf{c}_t$ is derived as follows:

$$\mathbf{a}_t = \text{softmax}(\mathbf{b}_t),$$
$$\mathbf{c}_t = \mathbf{H}_t \mathbf{a}_t,$$

where $\mathbf{a}_t$ is the alignment vector. We also experiment with two types of attention reminiscent of the *global* and *local* models in Luong et al. (2015), where the first attends over all input words ($k = n$) and the second over a local window.

It is worth noting that two other works have concurrently tackled supertagging with BLSTM models. In Vaswani et al. (2016), a language model layer is added on top of a BLSTM, which allows embeddings of previously predicted tags to propagate through and influence the pending tagging decision. However, the language model layer is only effective when both scheduled sampling for training (Bengio et al., 2015) and beam search for inference are used. We show our attention-based models can match their performance, with only standard training and greedy decoding. Additionally, Lewis et al. (2016) presented a BLSTM model with two layers of stacking in each direction; and as an internal baseline, we show a non-stacking BLSTM without attention can achieve the same accuracy.

[9]Unlike in the parsing model, POS tags are excluded.

5 Experiments

Dataset and baselines. We conducted all experiments on CCGBank (Hockenmaier and Steedman, 2007) with the standard splits.[10] We assigned POS tags with the C&C POS tagger, and used 10-fold jackknifing for both POS tagging and supertagging. All parsers were evaluated using F1 over labeled CCG dependencies.

For supertagging, the baseline models are the RNN model of Xu et al. (2015), the bidirectional RNN (BRNN) model of Xu et al. (2016), and the BLSTM supertagging models in Vaswani et al. (2016) and Lewis et al. (2016). For parsing experiments, we compared with the global beam-search shift-reduce parsers of Zhang and Clark (2011) and Xu et al. (2014). One neural shift-reduce CCG parser baseline is Ambati et al. (2016), which is a beam-search shift-reduce parser based on Chen and Manning (2014) and Weiss et al. (2015); and the others are the RNN shift-reduce models in Xu et al. (2016). Additionally, the chart-based C&C parser was included by default.

Model and training parameters.[11] All our LSTM models are non-stacking with a single layer.[12] For the supertagging models, the LSTM

[10]Training: Sections 02-21 (39,604 sentences). Development: Section 00 (1,913 sentences). Test: Section 23 (2,407 sentences).

[11]We implemented all models using the CNN toolkit: `https://github.com/clab/cnn`.

[12]The BLSTMs have a single layer in each direction. We experimented with 2 layers in all models during development and found negligible improvements.

Model	Dev	Test
C&C	91.50	92.02
Xu et al. (2015)	93.07	93.00
Xu et al. (2016)	93.49	93.52
Lewis et al. (2016)	94.1	94.3
Vaswani et al. (2016)	94.08	-
Vaswani et al. (2016) +LM +beam	94.24	94.50
BLSTM	94.11	94.29
BLSTM-local	**94.31**	**94.46**
BLSTM-global	94.22	94.42

Table 1: 1-best supertagging results on both the dev and test sets. BLSTM is the baseline model without attention; BLSTM-local and -global are the two attention-based models.

	Supertagger β				
Beam	0.09	0.07	0.06	0.01	0.001
1	86.49	86.52	86.56	86.26	85.80
2	86.55	86.58	86.63	86.39	86.01
8	86.61	86.64	**86.67**	86.40	86.07

Table 2: Tuning beam size and supertagger β on the dev set.

Model	LP	LR	LF	CAT
LSTM-w	**90.13**	76.99	83.05	94.24
LSTM-w+c	89.37	83.25	86.20	94.34
LSTM-w+c+a	89.31	83.39	86.25	94.38
LSTM-w+c+a+p	89.43	**83.86**	**86.56**	**94.47**

Table 3: F1 on dev for all the greedy models.

hidden state size is 256, and the size of the attentional hidden layer ($\mathbf{x}_t$, Eq. 5) is 200. All parsing model LSTMs have a hidden state size of 128, and the size of the action hidden layer ($\mathbf{b}_t$, Eq. 4) is 80.

Pretrained word embeddings for all models are 100-dimensional (Turian et al., 2010), and all other embeddings are 50-dimensional. We also pretrained CCG lexical category and POS embeddings on the concatenation of the training data and a Wikipedia dump parsed with C&C.[13] All other parameters were uniformly initialized in $\pm\sqrt{6/(r+c)}$, where r and c are the number of rows and columns of a matrix (Glorot and Bengio, 2010).

For training, we used plain non-minibatched stochastic gradient descent with an initial learning rate $\eta_0 = 0.1$ and we kept iterating in epochs until accuracy no longer increases on the dev set. For all models, a learning rate schedule $\eta_e = \eta_0/(1+\lambda e)$ with $\lambda = 0.08$ was used for $e \geq 11$. Gradients were clipped whenever their norm exceeds 5. Dropout training as suggested by Zaremba et al. (2014), with a dropout rate of 0.3, and an ℓ_2 penalty of 1×10^{-5}, were applied to all models.

5.1 Supertagging Results

Table 1 summarizes 1-best supertagging results. Our baseline BLSTM model without attention achieves the same level of accuracy as Lewis et al. (2016) and the baseline BLSTM model of Vaswani et al. (2016). Compared with the latter, our hidden state size is 50% smaller (256 vs. 512).

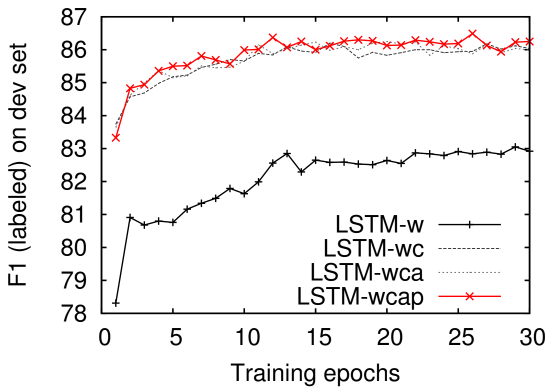
For training and testing the local attention model (BLSTM-local), we used an attention window size of 5 (tuned on the dev set), and it gives an improvement of 0.94% over the BRNN supertagger (Xu et al., 2016), achieving an accuracy on par with the beam-search (size 12) model of Vaswani et al. (2016) that is enhanced with a language model. Despite being able to consider wider contexts than the local model, the global attention model (BLSTM-global) did not show further gains, hence we used BLSTM-local for all parsing experiments below.

5.2 Parsing Results

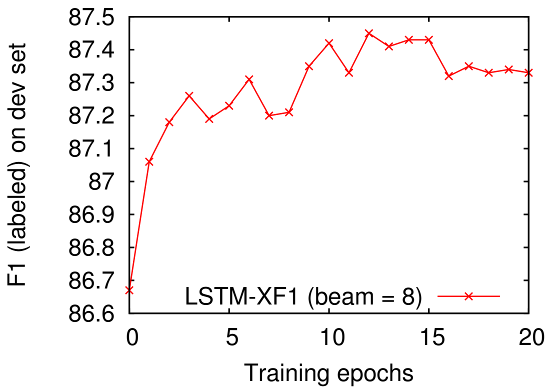
All parsers we consider use a supertagger probability cutoff β to prune categories less likely than β times the probability of the best category in a distribution: for the C&C parser, it uses an *adaptive* strategy to backoff to smaller β values if no spanning analysis is found given an initial β setting; for all the shift-reduce parsers, fixed β values are used without backing off. Since β determines the derivation space of a parser, it has a large impact on the final parsing accuracy.

For the maximum-likelihood greedy model, we found using a small β value (bigger ambiguity) for training significantly improved accuracy, and we chose $\beta = 1 \times 10^{-5}$ (5.22 categories per word with jackknifing) via development experiments. This reinforces the findings in a number of other CCG parsers (Clark and Curran, 2007; Auli and Lopez, 2011a; Lewis and Steedman, 2014a): even though a smaller β increases ambiguity, it leads to more accurate models at test time. On the other hand, we found using larger β values at test time led to significantly better results (Table 2). And this differs from the beam-search models that use the same β value for both training and testing (Zhang and Clark, 2011; Xu et al., 2014).

[13]We used the gensim word2vec toolkit: `https://radimrehurek.com/gensim/`.



(a) Dev F1 of the greedy models



(b) Dev F1 of the XF1 model

Figure 4: Learning curves with dev F-scores for all models.

		Section 00				Section 23			
Model	Beam	LP	LR	LF	CAT	LP	LR	LF	CAT
C&C (normal-form)	-	85.18	82.53	83.83	92.39	85.45	83.97	84.70	92.83
C&C (dependency hybrid)	-	86.07	82.77	84.39	92.57	86.24	84.17	85.19	93.00
Zhang and Clark (2011)	16	87.15	82.95	85.00	92.77	87.43	83.61	85.48	93.12
Xu et al. (2014)	128	86.29	84.09	85.18	92.75	87.03	85.08	86.04	93.10
Ambati et al. (2016)	16	-	-	85.69	93.02	-	-	85.57	92.86
Xu et al. (2016)-greedy	1	88.12	81.38	84.61	93.42	88.53	81.65	84.95	93.57
Xu et al. (2016)-XF1	8	88.20	83.40	85.73	93.56	88.74	84.22	86.42	93.87
LSTM-greedy	1	89.43	83.86	86.56	**94.47**	89.75	84.10	86.83	**94.63**
LSTM-XF1	1	**89.68**	85.29	87.43	94.41	**89.85**	85.51	87.62	94.53
LSTM-XF1	8	89.54	**85.46**	**87.45**	94.39	89.81	**85.81**	**87.76**	94.57

Table 4: Parsing results on the dev (Section 00) and test (Section 23) sets with 100% coverage, with all LSTM models using the BLSTM-local supertagging model. All experiments using auto POS. CAT (lexical category assignment accuracy). LSTM-greedy is the full greedy parser.

The greedy model. Table 3 shows the dev set results for all greedy models, where the four types of embeddings, that is, word (w), CCG category (c), action (a) and POS (p), are gradually introduced. The full model LSTM-w+c+a+p surpasses all previous shift-reduce models (Table 4), achieving a dev set accuracy of 86.56%. Category embeddings (LSTM-w+c) yielded a large gain over using word embeddings alone (LSTM-w); action embeddings (LSTM-w+c+a) provided little improvement, but further adding POS embeddings (LSTM-w+c+a+p) gave noticeable recall (+0.61%) and F1 improvements (+0.36%) over LSTM-w+c. Fig. 4a shows the learning curves, where all models converged in under 30 epochs.

The XF1 model. Table 4 also shows the results for the XF1 models (LSTM-XF1), which use all four types of embeddings. We used a beam size of 8, and a β value of 0.06 for both training and testing (tuned on the dev set); and training took 12 epochs to converge (Fig. 4b), with an F1 of 87.45% on the dev set. Decoding the XF1 model with greedy inference only slightly decreased recall and F1, and this resulted in a highly accurate deterministic parser. On the test set, our XF1 greedy model gives 2.67% F1 improvement over the greedy model in Xu et al. (2016); and the beam-search XF1 model achieves an F1 improvement of 1.34% compared with the XF1 model of Xu et al. (2016).

Model	Dev	Test
LSTM-BRNN	85.86	86.37
LSTM-BLSTM	86.26	86.64
LSTM-greedy	**86.56**	**86.83**

Table 5: Effect of different supertaggers on the full greedy parser. LSTM-greedy is the same parser as in Table 4, which uses the BLSTM-local supertagger.

Model	LP	LR	LF
Xu et al. (2015)	87.68	86.41	87.04
Lewis et al. (2016)	87.7	86.7	87.2
Lewis et al. (2016)*	88.6	87.5	88.1
Vaswani et al. (2016)*	-	-	88.32
Lee et al. (2016)	-	-	**88.7**
LSTM-XF1 (beam = 1)	89.85	85.51	87.62
LSTM-XF1 (beam = 8)	89.81	85.81	87.76

Table 6: Comparison of our XF1 models with chart-based parsers on the test set. $\star$ denotes a tri-trained model and $*$ indicates a different POS tagger.

Effect of the supertagger. To isolate the parsing model from the supertagging model, we first experimented with the BRNN supertagging model as in Xu et al. (2016) for both training and testing the full greedy LSTM parser. Using this supertagger, we still achieved the highest F1 (85.86%) on the dev set (LSTM-BRNN, Table 5) in comparison with all previous shift-reduce models; and an improvement of 1.42% F1 over the greedy model of Xu et al. (2016) was obtained on the test set (Table 4). We then experimented with using the baseline BLSTM supertagging model for parsing (LSTM-BLSTM), and observed the attention-based setup (LSTM-greedy) outperformed it, despite the attention-based supertagger (BLSTM-local) did not give better multi-tagging accuracy. We owe this to the fact that large β cutoff values—resulting in almost deterministic supertagging decisions on average—are required by the parser during inference; for instance, BLSTM-local has an average ambiguity of 1.09 on the dev set with $\beta = 0.06$.[14]

Comparison with chart-based models. For completeness and to put our results in perspective, we compare our XF1 models with other CCG parsers in the literature (Table 6): Xu et al. (2015) is the log-linear C&C dependency hybrid model with an RNN supertagger front-end; Lewis et al. (2016) is an LSTM supertagger-factored parser using the A* CCG parsing algorithm of Lewis and Steedman (2014a); Vaswani et al. (2016) combine a BLSTM supertagger with a new version of the C&C parser (Clark et al., 2015) that uses a max-violation perceptron, which significantly improves over the original C&C models; and finally, a global recursive neural network model with A* decoding (Lee et al., 2016). We note that all these alternative models—with the exception of Xu et al. (2015) and Lewis et al. (2016)—use structured training that accounts for violations of the gold-standard, and we conjecture further improvements for our model are possible by incorporating such mechanisms.[15]

6 Conclusion

We have presented an LSTM parsing model for CCG, with a factorization allowing the linearization of the complete parsing history. We have shown that this simple model is highly effective, with results outperforming all previous shift-reduce CCG parsers. We have also shown global optimization benefits an LSTM shift-reduce model; and contrary to previous findings with the averaged perceptron (Zhang and Clark, 2008), we empirically demonstrated beam-search inference is not necessary for our globally optimized model. For future work, a natural direction is to explore integrated supertagging and parsing in a single neural model (Zhang and Weiss, 2016).

Acknowledgment

I acknowledge the support from the Carnegie Trust for the Universities of Scotland through a Carnegie Scholarship.

References

Anthony Ades and Mark Steedman. 1982. On the order of words. In *Linguistics and philosophy*. Springer.

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

Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. 2016. Shift-reduce CCG parsing using neural network models. In *Proc. of NAACL (Volume 2)*.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proc. of ACL*.

Michael Auli and Adam Lopez. 2011a. A comparison of loopy belief propagation and dual decomposition for

[14]All β cutoffs were tuned on the dev set; for BRNN, we found the same β settings as in Xu et al. (2016) to be optimal; for BLSTM, $\beta = 4 \times 10^{-5}$ for training (with an ambiguity of 5.27) and $\beta = 0.02$ for testing (with an ambiguity of 1.17).

[15]Our XF1 training considers shift-reduce action sequences, but not violations of the gold-standard (e.g., see Huang et al. (2012), Watanabe and Sumita (2015), Zhou et al. (2015) and Andor et al. (2016)).

integrated CCG supertagging and parsing. In *Proc. of ACL*.

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*.

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

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. In *Neural Networks, IEEE Transactions on*. IEEE.

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

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

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

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

Stephen Clark, Darren Foong, Luana Bulat, and Wenduan Xu. 2015. The Java version of the C&C parser. Technical report, University of Cambridge Computer Laboratory.

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

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

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*.

Jason Eisner. 1996. Efficient normal-form parsing for Combinatory Categorial Grammar. In *Proc. of ACL*.

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

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

Felix Gers and Jürgen Schmidhuber. 2000. Recurrent nets that time and count. In *Neural Networks*. IEEE.

Felix Gers, Jürgen Schmidhuber, and Fred Cummins. 2000. Learning to forget: Continual prediction with LSTM. In *Neural computation*. MIT Press.

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

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

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. In *Neural Networks*. Elsevier.

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

Julia Hockenmaier and Mark Steedman. 2007. CCG-Bank: 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.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proc. of NAACL*.

Marco Kuhlmann and Giorgio Satta. 2014. A new parsing algorithm for Combinatory Categorial Grammar. In *Transactions of the Association for Computational Linguistics*. ACL.

Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2016. Global neural CCG parsing with optimality guarantees. In *Proc. of EMNLP*.

Mike Lewis and Mark Steedman. 2014a. A* CCG parsing with a supertag-factored model. In *Proc. of EMNLP*.

Mike Lewis and Mark Steedman. 2014b. Improved CCG parsing with semi-supervised supertagging. In *Transactions of the Association for Computational Linguistics*. ACL.

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG parsing. In *Proc. of NAACL*.

Minh-Thang Luong, Hieu Pham, and Christopher Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proc. of EMNLP*.

Vinod Nair and Geoffrey Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proc. of ICML*.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *Proc. of COLING*.

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

Richard Socher, Christopher Manning, and Andrew Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In

Proc. of the NIPS Deep Learning and Unsupervised Feature Learning Workshop.

Richard Socher, Cliff Lin, Christopher Manning, and Andrew Ng. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proc. of ICML*.

Richard Socher, John Bauer, Christopher Manning, and Andrew Ng. 2013. Parsing with compositional vector grammars. In *Proc. of ACL*.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press.

Roy Tromble, Shankar Kumar, Franz Och, and Wolfgang Macherey. 2008. Lattice minimum bayes-risk decoding for statistical machine translation. In *Proc. of EMNLP*.

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, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with LSTMs. In *Proc. of NAACL (Volume 2)*.

Krishnamurti Vijay-Shanker and David 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)*.

Wenduan Xu, Michael Auli, and Stephen Clark. 2016. Expected F-measure training for shift-reduce parsing with recurrent neural networks. In *Proc. of NAACL*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis using support vector machines. In *Proc. of IWPT*.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. In *Proc. of ICLR*.

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*.

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. 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*.