STRUCTURED ATTENTION NETWORKS

Yoon Kim∗ Carl Denton∗ Luong Hoang Alexander M. Rush
{yoonkim@seas,carldenton@college,lhoang@g,srush@seas}.harvard.edu
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138, USA

ABSTRACT

Attention networks have proven to be an effective approach for embedding categorical inference within a deep neural network. However, for many tasks we may want to model richer structural dependencies without abandoning end-to-end training. In this work, we experiment with incorporating richer structural distributions, encoded using graphical models, within deep networks. We show that these structured attention networks are simple extensions of the basic attention procedure, and that they allow for extending attention beyond the standard soft-selection approach, such as attending to partial segmentations or to subtrees. We experiment with two different classes of structured attention networks: a linear-chain conditional random field and a graph-based parsing model, and describe how these models can be practically implemented as neural network layers. Experiments show that this approach is effective for incorporating structural biases, and structured attention networks outperform baseline attention models on a variety of synthetic and real tasks: tree transduction, neural machine translation, and natural language inference. We further find that models trained in this way learn interesting unsupervised hidden representations that generalize simple attention.

1 INTRODUCTION

Attention networks are now a standard part of the deep learning toolkit, contributing to impressive results in neural machine translation (Bahdanau et al., 2015; Luong et al., 2015), image captioning (Xu et al., 2015), speech recognition (Chorowski et al., 2015; Chan et al., 2015), question answering (Hermann et al., 2015; Sukhbaatar et al., 2015), and algorithm-learning (Graves et al., 2014; Vinyals et al., 2015), among many other applications (see Cho et al. (2015) for a comprehensive review). This approach alleviates the bottleneck of compressing a source into a fixed-dimensional vector by equipping a model with variable-length memory (Weston et al., 2014; Graves et al., 2014; 2016), thereby providing random access into the source as needed. Attention is implemented as a hidden layer which computes a categorical distribution (or hierarchy of categorical distributions) to make a soft-selection over source elements.

Noting the empirical effectiveness of attention networks, we also observe that the standard attention-based architecture does not directly model any structural dependencies that may exist among the source elements, and instead relies completely on the hidden layers of the network. While one might argue that these structural dependencies can be learned implicitly by a deep model with enough data, in practice, it may be useful to provide a structural bias. Modeling structural dependencies at the final, output layer has been shown to be important in many deep learning applications, most notably in seminal work on graph transformers (LeCun et al., 1998), key work on NLP (Collobert et al., 2011), and in many other areas (Peng et al., 2009; Do & Artières, 2010; Jaderberg et al., 2014; Chen et al., 2015; Durrett & Klein, 2015; Lample et al., 2016; Kipperwasser & Goldberg, 2016, inter alia).

In this work, we consider applications which may require structural dependencies at the attention layer, and develop internal structured layers for modeling these directly. This approach generalizes categorical soft-selection attention layers by specifying possible structural dependencies in a soft

∗Equal contribution.
manner. Key applications will be the development of an attention function that segments the source input into subsequences and one that takes into account the latent recursive structure (i.e. parse tree) of a source sentence.

Our approach views the attention mechanism as a graphical model over a set of latent variables. The standard attention network can be seen as an expectation of an annotation function with respect to a single latent variable whose categorical distribution is parameterized to be a function of the source. In the general case we can specify a graphical model over multiple latent variables whose edges encode the desired structure. Computing forward attention requires performing inference to obtain the expectation of the annotation function, i.e. the context vector. This expectation is computed over an exponentially-sized set of structures (through the machinery of graphical models/structured prediction), hence the name structured attention network. Notably each step of this process (including inference) is differentiable, so the model can be trained end-to-end without having to resort to deep policy gradient methods (Schulman et al., 2015).

The differentiability of inference algorithms over graphical models has previously been noted by various researchers, (Li & Eisner, 2009; Domke, 2011; Stoyanov et al., 2011; Stoyanov & Eisner, 2012; Gormley et al., 2015), primarily outside the area of deep learning. For example, Gormley et al. (2015) treat an entire graphical model as a differentiable circuit and backpropagate risk through variational inference (loopy belief propagation) for minimum risk training of dependency parsers. Our contribution is to combine these ideas to produce structured internal attention layers within deep networks, noting that these approaches allow us to use the resulting marginals to create new features, as long as we do so in a differentiable way.

We focus on two concrete instantiations of structured attention: linear-chain conditional random fields (CRFs) (Lafferty et al., 2001) and first-order graph-based dependency parsers (Eisner, 1996). The initial work of Bahdanau et al. (2015) was particularly interesting in the context of machine translation, as the model was able to implicitly learn an alignment model as a hidden layer, effectively embedding inference into a neural network. In similar vein, under our framework the model has the capacity to learn a segmenter as a hidden layer or a parser as a hidden layer, without ever having to see a segmented sentence or a parse tree. Our experiments apply this approach to a difficult synthetic reordering task, as well as to machine translation and sentence inference. We find that models trained with structured attention outperform standard attention attention models. Analysis of learned representations further reveal that interesting structures emerge as an internal layer of the model. All code, models, and experimental setup will be available at http://lstm.seas.harvard.edu/structured-attention/

2 Background: Attention Networks

A standard neural network consist of a series of non-linear transformation layers, where each layer produces a fixed-dimensional hidden representation. For tasks with large input spaces, this paradigm makes it hard to control the interaction between components. For example in machine translation, the source consists of an entire sentence, and the output is a prediction for each word in the translated sentence. Utilizing a standard network leads to an information bottleneck, where one hidden layer must encode the entire source sentence. Attention provides an alternative approach.1 An attention network maintains a set of hidden representations that scale with the size of the source. The model uses an internal inference step to perform a soft-selection over these representations. This method allows the model to maintain a variable-length memory and has shown to be crucially important for scaling systems for many tasks.

Formally, let $x = [x_1, \ldots, x_n]$ represent a sequence of inputs, let $q$ be a query, and let $z$ be a categorical latent variable with sample space $\{1, \ldots, n\}$ that encodes the desired selection among these inputs. Our aim is to produce a context $c$ based on the sequence and the query. To do so, we assume access to an attention distribution $z \sim p(z | x, q)$, where we condition $p$ on the inputs $x$ and a query $q$. The context over a sequence is defined as expectation, $c = E_{z \sim p(z | x, q)}[f(x, z)]$ where $f(x, z)$ is an annotation function. Attention of this form can be applied over any type of input, however, we will primarily be concerned with “deep” networks, where both the annotation function

\footnote{Another line of work involves marginalizing over latent variables (e.g. latent alignments) with dynamic programming for sequence-to-sequence transduction (Kong et al., 2016; Lu et al., 2016; Yu et al., 2016).}
and attention distribution are parameterized with neural networks, and the context produced is a vector fed to a downstream network.

For example, consider the case of attention-based neural machine translation (Bahdanau et al., 2015). Here the sequence of inputs \([x_1, \ldots, x_n]\) are the hidden states of a recurrent neural network (RNN), running over the words in the source sentence, \(q\) is the RNN hidden state of the target decoder (i.e. vector representation of the query \(q\)), and \(z\) represents the source position to be attended to for translation. The attention distribution \(p\) is simply \(p(z = i \mid x, q) = \text{softmax}(\theta_i)\) where \(\theta \in \mathbb{R}^n\) is a parameterized potential typically based on a neural network, e.g. \(\theta_i = \text{MLP}(x_i; q)\). The annotation function is defined to simply return the selected hidden state, \(f(x, z) = x_i\). The context vector can then be computed using a simple sum,

\[
    c = \mathbb{E}_{z \sim p(z \mid x, q)}[f(x, z)] = \sum_{i=1}^n p(z = i \mid x, q)x_i
\]

Other tasks such as question answering use attention in a similar manner, for instance by replacing source \([x_1, \ldots, x_n]\) with a set of potential facts and \(q\) with a representation of the question.

In summary we interpret the attention mechanism as taking the expectation of an annotation function \(f(x, z)\) with respect to a latent variable \(z \sim p\), where \(p\) is parameterized to be function of \(x\) and \(q\).

## 3 Structured Attention

Attention networks simulate selection from a set using a soft model. In this work we consider generalizing selection to types of attention, such as selecting chunks, segmenting inputs, or even attending to latent subtrees. One interpretation of this attention is as using soft-selection that considers all possible structures over the input, of which there may be exponentially many possibilities. Of course, this expectation can no longer be computed using a simple sum, and we need to incorporate the machinery of inference directly into our neural network.

Define a structured attention model as being an attention model where \(z\) is now a vector of discrete latent variables \([z_1, \ldots, z_m]\) and the attention distribution is \(p(z \mid x, q)\) is defined as a conditional random field (CRF), specifying the independence structure of the \(z\) variables. Formally, we assume an undirected graph structure with \(m\) vertices. The CRF is parameterized with clique (log-)potentials \(\theta_C(z_C) \in \mathbb{R}\), where the \(z_C\) indicates the subset of \(z\) given by clique \(C\). Under this definition, the attention probability is defined as, \(p(z \mid x, q; \theta) = \text{softmax}(\sum_C \theta_C(z_C))\), where for symmetry we use softmax in a general sense, i.e. \(\text{softmax}(g(z)) = \frac{1}{Z} \exp(g(z))\) where \(Z = \sum_{z'} \exp(g(z'))\) is the implied partition function. In practice we use a neural CRF, where \(\theta\) comes from a deep model over \(x, q\).

In structured attention, we also assume that the annotation function \(f\) factors (at least) into clique annotation functions \(f(x, z) = \sum_C f_C(x, z_C)\). Under standard conditions on the conditional independence structure, inference techniques from graphical models can be used to compute the forward-pass expectations and the context:

\[
    c = \mathbb{E}_{z \sim p(z \mid x, q)}[f(x, z)] = \sum_C \mathbb{E}_{z \sim p(z_C \mid x, q)}[f_C(x, z_C)]
\]

### 3.1 Example 1: Subsequence Selection

Suppose instead of soft-selecting a single word, we wanted to explicitly model the selection of contiguous subsequences. We could naively apply categorical attention over all subsequences, or hope the model learns a multi-modal distribution to combine neighboring words. Structured attention provides an alternate approach.

Concretely, let \(m = n\), define \(z\) to be a random vector \(z = [z_1, \ldots, z_n]\) with \(z_i \in \{0, 1\}\), and define our annotation function to be, \(f(x, z) = \sum_{i=1}^n f_i(x, z_i)\) where \(f_i(x, z_i) = I\{z_i = 1\}x_i\). The explicit expectation is then,

\[
    \mathbb{E}_{z_1, \ldots, z_n}[f(x, z)] = \sum_{i=1}^n p(z_i = 1 \mid x, q)x_i
\]
Equation (2) is similar to equation (1)—both are a linear combination of the input representations where the scalar is between $[0, 1]$ and represents how much attention should be focused on each input. However, (2) is fundamentally different in two ways: (i) it allows for multiple inputs (or no inputs) to be selected for a given query; (ii) we can incorporate structural dependencies across the $z_i$’s. For instance, we can model the distribution over $z$ with a linear-chain CRF with pairwise edges,

$$p(z_1, \ldots, z_n | x, q) = \text{softmax} \left( \sum_{i=1}^{n-1} \theta_{i,i+1}(z_i, z_{i+1}) \right)$$

where $\theta_{k,l}$ is the pairwise potential for $z_i = k$ and $z_{i+1} = l$. This model is shown in Figure 1c. Compare this model to the standard attention in Figure 1a, or to a simple Bernoulli (sigmoid) selection method, $p(z_i = 1 | x, q) = \sigma(\theta_i)$, shown in Figure 1b. All three of these methods can use potentials from the same neural network or RNN that takes $x$ and $q$ as inputs.

Under this setup, the marginal distribution $p(z_i = 1 | x)$ can be calculated efficiently in linear-time for all $i$ using message-passing, i.e. the forward-backward algorithm. These marginals allow us to calculate (2), and in doing so we implicitly sum over an exponentially-sized set of structures (i.e. all binary sequences of length $n$) through dynamic programming. We refer to this type of attention layer as a segmentation attention layer.

Note that the forward-backward algorithm is being used as parameterized pooling (as opposed to output computation), and can be thought of as generalizing the standard attention softmax. Crucially this generalization from vector softmax to forward-backward is just a series of differentiable steps, and we can compute gradients of its output (marginals) with respect to its input (potentials). This will allow the structured attention model to be trained end-to-end as part of a deep model.

### 3.2 Example 2: Syntactic Tree Selection

This same approach can be used for more involved structural dependencies. One popular structure for natural language tasks is a dependency tree, which enforces a structural bias on the recursive dependencies common in many languages. In particular a dependency tree enforces that each word in a source sentence is assigned exactly one parent word (head word), and that these assignments do not cross (projective structure). Employing this bias encourages the system to make a soft-selection based on learned syntactic dependencies, without requiring linguistic annotations or a pipelined decision.

A dependency parser can be partially formalized as a graphical model with the following cliques (Smith & Eisner, 2008): latent variables $z_{ij} \in \{0, 1\}$ for all $i \neq j$, which indicates that the $i$-th word is the parent of the $j$-th word (i.e. $x_i \rightarrow x_j$); and a special global constraint that rules out configurations of $z_{ij}$’s that violate parsing constraints (e.g. one head, projectivity).

The parameters to the graph-based CRF dependency parser are the log-potentials $\theta_{ij}$, which reflect the score of selecting $x_i$ as the parent of $x_j$. The probability of a parse tree $z$ given the

\[\text{As are other dynamic programming algorithms for inference in graphical models, such as (loopy and non-loopy) belief propagation.}\]
Under review as a conference paper at ICLR 2017

\[ \text{procedure} \ \text{ForwardBackward}(\theta) \]
\[
\alpha[0, (t)] \leftarrow 0 \\
\beta[n + 1, (t)] \leftarrow 0 \\
\text{for } i \in [1, n]; z \in C \text{ do} \\
\quad \alpha[i, z] \leftarrow \bigoplus_y \alpha[i - 1, y] \odot \theta_i(y, z) \\
\text{for } i \in [n]; y \in C \text{ do} \\
\quad \beta[i, y] \leftarrow \bigoplus_z \beta[i + 1, z] \odot \theta_i(y, z) \\
\quad A \leftarrow \alpha[n + 1, (t)] \\
\text{for } i \in [1, n]; z \in C \text{ do} \\
\quad p(z_i = z | x) \leftarrow \exp(\alpha[i, z] \odot \beta[i, z]) \quad \odot -A \\
\text{return } p
\]

\[ \text{procedure} \ \text{BackpropForwardBackward}(\theta, p, \nabla^\theta \mathcal{L}) \]
\[
\nabla^\alpha \leftarrow \log p \odot \log \nabla^\alpha \mathcal{L} \odot \beta[i, z] \odot -A \\
\nabla^\beta \leftarrow \log p \odot \log \nabla^\beta \mathcal{L} \odot \alpha[i, z] \odot -A \\
\hat{\alpha}[0, (t)] \leftarrow 0 \\
\hat{\beta}[n + 1, (t)] \leftarrow 0 \\
\text{for } i \in [1, n]; y \in C \text{ do} \\
\quad \hat{\beta}[i, y] \leftarrow \nabla^\beta [i, y] \odot \bigoplus_z \theta_i(y, z) \odot \hat{\beta}[i + 1, z] \\
\text{for } i \in [1, n]; z \in C \text{ do} \\
\quad \hat{\alpha}[i, z] \leftarrow \nabla^\alpha [i, z] \odot \bigoplus_y \theta_i(y, z) \odot \hat{\alpha}[i - 1, y] \\
\text{for } i \in [1, n]; y, z \in C \text{ do} \\
\quad \nabla^\theta_{\theta_i(y, z)} \leftarrow \exp(\hat{\alpha}[i, y] \odot \hat{\beta}[i + 1, z]) \\
\quad \quad \quad \quad \quad \quad \quad \odot \alpha[i, y] \odot \bigoplus \hat{\beta}[i + 1, z] \\
\quad \quad \quad \quad \quad \quad \quad \quad \odot \alpha[i, y] \odot \bigoplus \hat{\beta}[i + 1, z] \odot -A \\
\text{return } \nabla^\theta \mathcal{L}
\]

Figure 2: Algorithms for linear-chain CRF: (left) computation of forward/backward tables \( \alpha, \beta \) and marginal probabilities from potentials \( \theta \) (forward-backward algorithm); (right) backpropagation of loss gradients with respect to the marginals \( \nabla^\theta \mathcal{L} \) (right). \( C \) denotes the state space. Backpropagation uses the identity \( \nabla \log p = p \nabla^\theta \mathcal{L} \) to calculate \( \nabla^\theta = \nabla \log p \nabla^\theta \mathcal{L} \). Typically the forward-backward with marginals is performed in log-space semifield \( \mathbb{R} \cup \{-\infty\} \) with \( \odot = + \) and \( \oplus = \log \exp \) for numerical precision. However, backpropagation requires an additive inverse, so we extend to a field \( \mathbb{R} \cup \{-\infty\} \times \{+, -\} \) with special \(+/-\) log-space operations. See Section 3.3 and Table 1 for more details.

sentence \( x = [x_1, \ldots, x_n] \) is,
\[
p(z | x, q) = \text{softmax} \left( \mathbb{I} \{ z \text{ is valid} \} \sum_{i \neq j} \mathbb{I} \{ z_{ij} = 1 \} \theta_{ij} \right)
\]
where \( z \) is represented as a vector of \( z_{ij} \)'s for all \( i \neq j \). It is possible to calculate the marginal probability of each edge \( p(z_{ij} = 1 | x) \) for all \( i, j \) in \( O(n^3) \) time using the inside-outside algorithm (Baker, 1979) on the data structures of Eisner (1996).

The parsing constraints ensure that each word has exactly one head (i.e. \( \sum_{i=1}^{n} z_{ij} = 1 \)). Therefore if we want to utilize the soft-head selection of a position \( j \), the context vector is defined as:
\[
f(x, z, j) = \sum_{i=1}^{n} \mathbb{I} \{ z_{ij} = 1 \} x_i \\
c_j = \mathbb{E}_z [f(x, z, j)] = \sum_{i=1}^{n} p(z_{ij} = 1 | x, q) x_i
\]
Note that the annotation function takes an additional argument \( j \) to produce a context vector for each word. Similar types of attention can be applied for other tree properties such as soft-children, or query-specific trees. We refer to this type of attention layer as a syntactic attention layer.

3.3 END-TO-END TRAINING

Graphical models of this form have been widely used as the final layer of deep models. Our contribution is to argue that these networks can be added within deep networks in place of simple attention layers. The whole model can then be trained end-to-end.

The main complication in utilizing this approach within the network itself is the need to backpropagate through an inference algorithm as part of the structured attention network. Past work has demonstrated that the techniques necessary for this approach (see Stoyanov et al. (2011)), but to our knowledge it is very rarely employed.

Consider the case of the simple linear-chain CRF layer from Section 3.1. Figure 2 (left) shows the standard forward-backward algorithm for computing \( p(z_i = 1 | x, q; \theta) \). If we treat this step as a neural network layer, its input is the log-potentials \( \theta \), and its output after the forward pass is these marginals.\(^3\) To backpropagate a loss through this layer we need to compute the gradient of the loss \( \mathcal{L} \)

\(^3\)Confusingly “forward” in this layer is different than in the forward-backward algorithm, as the marginals themselves are our output. However the two uses of the term are actually quite related. The forward-backward
with respect to \( \theta \), \( \nabla_{\theta} L \), as a function of the gradient of the loss with respect to the marginals, \( \nabla_{\theta} L \). As the forward-backward algorithm consists of differentiable steps, this function can be derived using reverse-mode automatic differentiation of the forward-backward algorithm itself. Note that this reverse-mode algorithm conveniently has a parallel structure to the forward version, and can also be implemented using dynamic programming. For dependency parsing, the forward pass can be computed using the analogous inside-outside implementation of Eisner’s algorithm (Eisner, 1996). Similarly, the reverse-mode parallels the inside-outside structure. Forward/backward pass through the inside-outside algorithm is described in Appendix B.

However, in practice, one cannot simply use current off-the-shelf tools for this task. For one, efficiency is quite important for these models and so the benefits of hand-optimizing the reverse-mode implementation still outweighs simplicity of automatic differentiation. Secondly, numerical precision becomes a major issue for structured attention networks. For computing the forward-pass and the marginals, it is important to use the standard log-space semifield \((\oplus, \otimes)\) to avoid underflow of probabilities. For computing the reverse-mode pass, we need to remain in log-space, but also handle true negative values and also include an additive inverse. This issue requires the non-standard signed log-space field. Table 1, based on (Li & Eisner, 2009), demonstrates how to handle this issue, and Figure 2 (right) describes backpropagation through the forward-backward algorithm.

### 4 Experiments

We experiment with these two types of structured attention networks on three different tasks: (a) a simple, synthetic tree manipulation task using the syntactic attention layer, (b) a machine translation task with segmenting attention, (c) an extension of a sentence inference model with syntactic tree attention. These experiments are not intended to boost the state-of-the-art for these tasks but to test whether these methods can be trained effectively in an end-to-end fashion, can yield improvements over standard selection-based attention, and can learn plausible latent structures. The full model architectures, hyperparameters, and training details are further described in Appendix A.

#### 4.1 Tree Transduction

The first set of experiments look at a tree-transduction task. These experiments use synthetic data to explore a failure case of soft-selection attention models. The task is to learn to convert a random formula given in prefix notation to one in infix notation, e.g.,

\[
( \ast ( ( + ( + 15 7 ) 1 8 ) ( + 19 0 11 ) ) ) 
\Rightarrow ( ( 15 + 7 ) + 1 + 8 ) \ast ( 19 + 0 + 11 )
\]

The alphabet consists of symbols \(\{,),+,\ast\}\), numbers between 0 and 20, and a special root symbol \(\ast\). This task is used as a preliminary task to see if the model is able to learn the implicit tree structure on the source side. The model itself is an encoder-decoder model, where the encoder is defined below and the decoder is an LSTM. See Appendix A.2 for the full model.

Training uses 15K prefix-infix pairs where the maximum nesting depth is set to be between 2-4 (the above example has depth 3), with 5K pairs in each depth bucket. The number of expressions in each parenthesis is limited to be at most 4. Test uses 1K unseen sequences with depth between 2-6 (note specifically deeper than train), with 200 sequences for each depth. The performance is measured as the average proportion of correct target tokens produced until the first failure (as in Grefenstette et al. (2015)).

Algorithm can be interpreted as a forward and backpropagation pass on the partition function \(Z\). See Eisner (2016) for further details (appropriately titled “Inside-Outside and Forward-Backward Algorithms Are Just Backprop”). As such this approach can be seen as computing second-order information. This interpretation is central to Li & Eisner (2009).

In general we use \(\nabla_{\theta} L\) to denote the Jacobian of \(a\) with respect to \(b\).
Figure 3: Visualization of the source attention distribution for the simple (left) and structured (right) attention models on the tree transduction task. $\$ \$ is the special root symbol. Each row delineates the distribution over the parents (i.e. each row sums to one). The attention distribution obtained from the parsing marginals are more able to capture the tree structure—e.g. the attention weights of closing parentheses are generally placed on the opening parentheses.

For experiments we try using different forms of attention over embedding-only encoders. Let $x_j$ be an embedding for each source symbol; our three variants of the source representation $\hat{x}_j$ are:

- (a) no atten, just symbol embeddings by themselves, i.e. $\hat{x}_j = x_j$;
- (b) simple attention, symbol embeddings and soft-pairing for each symbol, i.e. $\hat{x}_j = [x_j; c'_j]$ where $c'_j = \sum_{i=1}^n \text{softmax}(\theta_{ij})x_i$ is calculated using soft-selection;
- (c) structured attention, symbol embeddings and soft-parent, i.e. $\hat{x}_j = [x_j; c_j]$ where $c_j = \sum_{i=1}^n p(z_{ij} = 1 | x)x_i$ is calculated using parsing marginals, obtained from the syntactic attention layer. None of these models use an explicit query value—the log-potentials come from running a bidirectional LSTM over the source, producing hidden vectors $h_i$, and then computing

$$\theta_{ij} = \tanh(s^T \tanh(W_1 h_i + W_2 h_j + b))$$

where $s, b, W_1, W_2$ are parameters (see Appendix A.1).

The source representation $[\hat{x}_1, \ldots, \hat{x}_n]$ are attended over using the standard attention mechanism at each decoding step by an LSTM decoder. Additionally, symbol embedding parameters are shared between the parsing LSTM and the source encoder.

Results Table 2 has the results for the task. Note that this task is quite hard as the encoder gives a single symbol and attention over the bag. The baseline model (unsurprisingly) performs poorly as it has no information about the source ordering. The simple attention model performs better, but is significantly outperformed by the structured model with a tree structure bias. We hypothesize that the model is partially reconstructing the arithmetic tree. Figure 3 shows the attention distribution for the simple/structured models on the same source sequence, which indicates that the structured model is able to learn boundaries (i.e. parentheses).

### 4.2 Neural Machine Translation

Our second set of experiments use a full neural machine translation model utilizing attention over subsequences in a character-to-word task. Here both the encoder/decoder are LSTMs, and we re-

---

Table 2: Performance (average length to failure %) of models on the tree-transduction task.

<table>
<thead>
<tr>
<th>Depth</th>
<th>No Atten</th>
<th>Simple</th>
<th>Structured</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7.6</td>
<td>87.4</td>
<td>99.2</td>
</tr>
<tr>
<td>3</td>
<td>4.1</td>
<td>49.6</td>
<td>87.0</td>
</tr>
<tr>
<td>4</td>
<td>2.8</td>
<td>23.3</td>
<td>64.5</td>
</tr>
<tr>
<td>5</td>
<td>2.1</td>
<td>15.0</td>
<td>30.8</td>
</tr>
<tr>
<td>6</td>
<td>1.5</td>
<td>8.5</td>
<td>18.2</td>
</tr>
</tbody>
</table>

---

Thus there are two attention mechanisms at work under this setup. First, structured attention over the source only to obtain soft-parents for each symbol. Second, standard softmax alignment attention over the source representations during decoding.
place standard simple attention with a segmentation attention layer. We experiment with translating directly from unsegmented Japanese characters to English words. This differs from prior NMT-based approaches to Japanese-English translation, which typically run Japanese word segmentation as a preprocessing step (Neubig, 2016).

The data comes from the Workshop on Asian Translation (WAT) (Nakazawa et al., 2016). We randomly pick 300K sentences from the original training set (of 3M sentences) and additionally discard sentence pairs where either the Japanese or English sentence had length greater than 50. The vocabulary consists of all tokens that occurred at least 10 times in the training corpus.

The segmentation attention layer is a two-state CRF whose unary potentials at the $j$-th decoder step are obtained with two bilinear maps over the encoder hidden states $[h_1, \ldots, h_n]$ and decoder/query $q = h'_j$ hidden states, i.e. $\theta_i(k) = h_i \cdot W_k q$, where $k \in \{0, 1\}$. Note that this is slightly different from the usual parameterization in the standard attention mechanism which only requires a single score $\theta_i$ (usually obtained with a single bilinear map or an MLP). The pairwise potentials are parameterized linearly with a vector $b$, i.e. all together $\theta_i, j+1(z_i, z_{i+1}) = \theta_i(z_i) + \theta_{i+1}(z_{i+1}) + b_{z_i, z_{i+1}}$.

Appendix A.3 describes the full model architecture.

We experiment with three attention configurations: (a) standard simple attention, i.e. $c_j = \sum_{i=1}^n \text{softmax}(\theta_i) h_i$; (b) sigmoid attention: multiple selection with Bernoulli random variables, i.e. $c_j = \sum_{i=1}^n \text{sigmoid}(\theta_i) h_i$; (c) structured attention, encoded with an CRF, i.e. $c_j = \sum_{i=1}^n p(z_i = 1|x) h_i$.

**Results**

<table>
<thead>
<tr>
<th>Simple</th>
<th>Sigmoid</th>
<th>Structured</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLEU</td>
<td>6.18</td>
<td>8.19</td>
</tr>
</tbody>
</table>

Table 3: Translation performance on character-to-word Japanese-English translation.
<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handcrafted features (Bowman et al., 2015)</td>
<td>78.2</td>
</tr>
<tr>
<td>LSTM encoders (Bowman et al., 2015)</td>
<td>80.6</td>
</tr>
<tr>
<td>Tree-Based CNN (Mou et al., 2016)</td>
<td>82.1</td>
</tr>
<tr>
<td>Stack-Augmented Parser-Interpreter Neural Net (Bowman et al., 2016)</td>
<td>83.2</td>
</tr>
<tr>
<td>LSTM with word-by-word attention (Rocktäschel et al., 2016)</td>
<td>83.5</td>
</tr>
<tr>
<td>Matching LSTMs (Wang &amp; Jiang, 2016)</td>
<td>86.1</td>
</tr>
<tr>
<td>Decomposable attention over word embeddings (Parikh et al., 2016)</td>
<td>86.3</td>
</tr>
<tr>
<td>Decomposable attention + intra-sentence attention (Parikh et al., 2016)</td>
<td>86.8</td>
</tr>
<tr>
<td>Attention over constituency tree nodes (Zhao et al., 2016)</td>
<td>87.2</td>
</tr>
<tr>
<td>Neural Tree Indexers (Munkhdalai &amp; Yu, 2016)</td>
<td>87.3</td>
</tr>
<tr>
<td>Enhanced BiLSTM Inference Model (Zhou et al., 2016)</td>
<td>87.7</td>
</tr>
<tr>
<td>Enhanced BiLSTM Inference Model + ensemble (Zhou et al., 2016)</td>
<td>88.3</td>
</tr>
<tr>
<td>No Attention</td>
<td>85.7</td>
</tr>
<tr>
<td>No Attention + Hard parent</td>
<td>86.2</td>
</tr>
<tr>
<td>Simple Attention</td>
<td>86.3</td>
</tr>
<tr>
<td>Structured Attention</td>
<td>87.0</td>
</tr>
<tr>
<td>Pretrained Structured Attention</td>
<td>86.6</td>
</tr>
</tbody>
</table>

**Figure 5:** Results of our models (bottom) and others (top) on the Stanford NLI test set. Our baseline model has the same architecture as Parikh et al. (2016) but the performance is slightly different due to different training settings (e.g. we train for 80 epochs with a batch size of 32 while Parikh et al. (2016) train for 400 epochs with a batch size of 4.).

### 4.3 Natural Language Inference

The final experiment looks at the task of natural language inference (NLI). In NLI, the model is given two sentences (hypothesis/premise) and has to predict their relationship: entailment, contradiction, neutral. It is thought that syntactic information is useful for this task, and a state-of-the-art system uses attention over pre-constructed parse trees (Zhou et al., 2016).

For this task, we use the Stanford NLI dataset (Bowman et al., 2015) and model our approach off of the decomposable attention model of Parikh et al. (2016) (which does not use syntactic trees). This model also takes in the matrix of word embeddings as the input. Appendix A.4 describes the full model.

As in the transduction task, we focus on modifying the input representation to take into account soft-parents. In addition to the three baselines described for tree transduction (No Attention, Simple, Structured), we also explore two additional settings: (d) hard pipeline parent selection, i.e. $x_j = [x_j; x_{\text{head}(j)}]$, where $\text{head}(j)$ is the index of $x_j$'s parent; (e) pretrained structured attention: structured attention but where the parsing layer is pretrained for one epoch on a parsed dataset (which was enough for convergence).

**Results** Results of our models are shown in Table 5. Simple attention improves upon the no attention model, and this is consistent with Parikh et al. (2016). Surprisingly, pretraining the syntactic attention layer on the parse tree performs worse than training it from scratch. We can also obtain the hard parse for an example sentence by running the Viterbi algorithm on the syntactic attention layer with the non-pretrained model:

$$\text{The men are fighting outside a deli.}$$

Despite being trained without ever being exposed to an explicit parse tree, the syntactic attention layer learns an almost plausible dependency structure. In the above example it is able to correctly identify the main verb fighting, but makes mistakes on determiners (e.g. head of The should be

---

6 The parents are obtained from running the dependency parser of Andor et al. (2016), available at https://github.com/tensorflow/models/tree/master/syntaxnet
We generally observed this pattern across sentences, possibly because the verb structure is more important for the inference task.

5 CONCLUSION

This work outlines structured attention networks, which incorporate graphical models to generalize simple attention, and describes the technical machinery and computational techniques for backpropagating through models of this form. We implement two types of structured attention, a general-purpose linear chain CRF, which we use for subsequence selection, and a more complicated first-order dependency parser. Experiments show that this method can learn interesting structural properties and improve on top of standard models.

The combination of graphical models and deep networks is an exciting area for future work. Similar technique can be used with embedded approximate inference such as mean-field (Domke, 2012). Additionally, this method could be used to learn latent labelers or parsers by attention on other tasks. Finally there is much to be explored in the parameterization of the CRF, such as using deep higher-order potentials, as opposed to simple transitions.

ACKNOWLEDGMENTS

We thank Tao Lei, Ankur Parikh, Tim Vieira, Matt Gormley, Andre Martins, Jason Eisner, and Yoav Goldberg for helpful discussion, notes, and code. We additionally thank Yasumasa Miyamoto for verifying Japanese-English translations.

REFERENCES


A Model Details

A.1 Syntactic Attention

The syntactic attention layer is similar to the first-order graph-based dependency parser of Kipperwasser & Goldberg (2016). Given an input sentence $[x_1, \ldots, x_n]$ and the corresponding word vectors $[x_1, \ldots, x_n]$, we use a bidirectional LSTM to get the hidden states for each time step $i \in [1, \ldots, n]$,
\[
\hat{h}_{i}^{\text{fwd}} = \text{LSTM}(x_i, \hat{h}_{i-1}^{\text{fwd}}) \quad \hat{h}_{i}^{\text{bwd}} = \text{LSTM}(x_i, \hat{h}_{i+1}^{\text{bwd}}) \quad h_{i} = [\hat{h}_{i}^{\text{fwd}}, \hat{h}_{i}^{\text{bwd}}]
\]
where the forward and backward LSTMs have their own parameters. The score for $x_i \rightarrow x_j$ (i.e. $x_i$ is the parent of $x_j$), is given by an MLP
\[
\theta_{ij} = \tanh(s^\top \tanh(W_i h_i + W_2 h_j + b))
\]
These scores are used as input to the inside-outside algorithm (see Appendix B) to obtain the probability of each word’s parent $p(z_{ij} = 1 \mid x)$, which is used to obtain the soft-parent $(e_j)$ for each word $x_j$.

A.2 Tree Transduction

Let $[x_1, \ldots, x_n], [y_1, \ldots, y_m]$ be the sequence of source/target symbols, with the associated embeddings $[x_1, \ldots, x_n], [y_1, \ldots, y_m]$ with $x_i, y_j \in \mathbb{R}^l$. In the simplest baseline model we take the source representation to be the matrix of the symbol embeddings. The decoder is a one-layer LSTM which produces the hidden states $h_{j}' = \text{LSTM}(y_j, h_{j-1}')$, with $h_{j}' \in \mathbb{R}^l$. The hidden states are combined with the input representation via a bilinear map $W \in \mathbb{R}^{l \times l}$ to produce the attention distribution used to obtain the vector $m_{i}$, which is combined with the decoder hidden state as follows,
\[
\alpha_i = \frac{\exp x_i W h_{j}'}{\sum_{k=1}^{n} \exp x_k W h_{j}'} \quad m_{i} = \sum_{i=1}^{n} \alpha_i x_i \quad \hat{h}_{j} = \tanh(U[m_{i}; h_{j}'])
\]
Here we have $W \in \mathbb{R}^{l \times l}$ and $U \in \mathbb{R}^{2l \times l}$. Finally, $\hat{h}_{j}$ is used to to obtain a distribution over the next symbol $y_{j+1}$,
\[
p(y_{j+1} \mid x_1, \ldots, x_n, y_1, \ldots, y_j) = \text{softmax}(\hat{V} h_{j} + b)
\]
For structured/simple models, the $j$-th source representation are respectively
\[
\tilde{x}_i = \left[ x_i; \sum_{k=1}^{n} p(z_{ki} = 1 \mid x) x_k \right] \quad \tilde{x}_i = \left[ x_i; \sum_{k=1}^{n} \text{softmax}(\theta_{ki}) x_k \right]
\]
with $\alpha_i$ and $m_i$ changed accordingly,
\[
\alpha_i = \frac{\exp \tilde{x}_i W h_{j}'}{\sum_{k=1}^{n} \exp \tilde{x}_k W h_{j}'} \quad m_{i} = \sum_{i=1}^{n} \alpha_i \tilde{x}_i
\]
Note that in this case we have $W \in \mathbb{R}^{2l \times l}$ and $U \in \mathbb{R}^{3l \times l}$. We use $l = 50$ in all our experiments. The forward/backward LSTMs for the parsing LSTM are also 50-dimensional. Symbol embeddings are shared between the encoder and the parsing LSTMs.

Additional training details include: batch size of 20; training for 13 epochs with a learning rate of 1.0, which starts decaying by half after epoch 9 (or the epoch at which performance does not improve on validation, whichever comes first); parameter initialization over a uniform distribution $U[-0.1, 0.1]$; gradient normalization at 1.

A.3 Neural Machine Translation

The baseline NMT system is from Luong et al. (2015). Let $[x_1, \ldots, x_n], [y_1, \ldots, y_m]$ be the source/target sentence, with the associated word embeddings $[x_1, \ldots, x_n], [y_1, \ldots, y_m]$. The encoder is an LSTM over the source sentence to obtain the hidden states $[h_1, \ldots, h_n]$, where
\[
h_i = \text{LSTM}(x_i, h_{i-1})
\]
Additional training details include: batch size of 64. The encoder/decoder LSTMs have 2 time steps $W_{th}$. Via a bilinear map $W \in \mathbb{R}^{l \times l}$ and this distribution is used to obtain the context vector at the $j$-th time step

$$\theta_i = h_i W h_j^i$$

$$c_j = \sum_{i=1}^{n} \text{softmax}(\theta_i) h_i$$

The Bernoulli attention network has the same $\theta_i$ but instead uses a sigmoid to obtain the weights of the linear combination, i.e.,

$$c_j = \sum_{i=1}^{n} \text{sigmoid}(\theta_i) h_i$$

And finally, the structured attention model uses two bilinear maps to obtain the unary (log) potentials

$$\theta_i(k) = h_i W_k h_j^i, k \in \{0, 1\}$$

$$\theta_i(z_i, z_{i+1}) = \theta_i(z_i) + \theta_{i+1}(z_{i+1}) + b_{z_i, z_{i+1}}$$

where $b$ are the pairwise potentials. These potentials are used as inputs to the forward-backward algorithm to get the marginals $p(z_i = 1 | x, q)$, and these are used to obtain the context vector

$$c_j = \sum_{i=1}^{n} p(z_i = 1 | x, q) h_i$$

The marginals are globally normalized for additional stability. The context vector is combined with the decoder hidden state

$$\hat{h}_j = \text{tanh}(U[c_j; h_j^i])$$

and $\hat{h}_j$ is used to obtain the distribution over the next target word $y_{j+1}$

$$p(y_{j+1} | x_1, \ldots, x_n, y_1, \ldots, y_j) = \text{softmax}(V \hat{h}_j + b)$$

Following Luong et al. (2015), we feed $c_j$ as additional input to the next decoder step, so the decoder LSTM equation is

$$h_j^i = \text{LSTM}(y_{j+1}; c_j, h_j^i)$$

The encoder/decoder LSTMs have 2 layers and 500 hidden units (i.e. $l = 500$).

Additional training details include: batch size of 64; training for 20 epochs with a learning rate of 1.0, which starts decaying by half after the first epoch at which performance does not improve on validation; dropout with probability 0.3; parameter initialization over a uniform distribution $U[-0.1, 0.1]$; gradient normalization at 1.

### A.4 Natural Language Inference

Our baseline model/setup is essentially the same as that of Parikh et al. (2016). Let $[x_1, \ldots, x_n], [y_1, \ldots, y_m]$ be the premise/hypothesis, with the corresponding input representations $[x_1, \ldots, x_n], [y_1, \ldots, y_m]$. The input representations are obtained by a linear transformation of the 300-dimensional pretrained GloVe embeddings (Pennington et al., 2014) after normalizing the GloVe embeddings to have unit norm. The pretrained embeddings remain fixed but the linear layer (which is also 300-dimensional) is trained. Words not in the pretrained vocabulary are hashed to one of 100 Gaussian embeddings with mean 0 and standard deviation 1.

We concatenate each input representation with a convex combination of the other sentence’s input representations, where the weights are determined through a dot product followed by a softmax,

$$e_{ij} = f(x_i)^T f(y_j) \quad \bar{x}_i = \left[ \begin{array}{c} x_i; \sum_{j=1}^{m} \frac{\exp e_{ij}}{\sum_{k=1}^{m} \exp e_{ik}} y_j \end{array} \right] \quad \bar{y}_j = \left[ \begin{array}{c} y_j; \sum_{i=1}^{n} \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{kj}} x_i \end{array} \right]$$

We use the GloVe embeddings pretrained over the 840 billion word Common Crawl, publicly available at [http://nlp.stanford.edu/projects/glove/](http://nlp.stanford.edu/projects/glove/)
Here \( f(\cdot) \) is an MLP. The new representations are fed through another MLP \( g(\cdot) \), summed, combined with the final MLP \( h(\cdot) \) and fed through a softmax layer to obtain a distribution over the labels \( l \),

\[
\hat{x} = \sum_{i=1}^{n} g(\bar{x}_i) \quad \hat{y} = \sum_{j=1}^{m} g(\bar{y}_j)
\]

\[
p(l \mid x_1, \ldots, x_n, y_1, \ldots, y_m) = \text{softmax}(Vh([\hat{x}; \hat{y}]) + b)
\]

All the MLPs have 2-layers, 300 ReLU units, and dropout probability of 0.2. For structured/simple models, we first employ the bidirectional parsing LSTM (A.1) to obtain the scores \( \theta_{ij} \). In the structured case each word representation is simply concatenated with its soft-parent

\[
\hat{x}_i = [x_i; \sum_{k=1}^{n} p(z_{ki} = 1 \mid x_k) x_k]
\]

and \( \hat{x}_i \) (and analogously \( \hat{y}_j \)) is used as the input to the above model. In the simple case (which closely corresponds to the intra-sentence attention model of Parikh et al. (2016)), we have

\[
\hat{x}_i = [x_i; \sum_{k=1}^{n} \frac{\exp \theta_{ki}}{\sum_{l=1}^{n} \exp \theta_{li}} x_k]
\]

The word embeddings for the parsing LSTMs are also initialized with GloVe, and the parsing layer is shared between the two sentences. The forward/backward LSTMs for the parsing layer are 100-dimensional.

Additional training details include: batch size of 32; training for 80 epochs with Adagrad (Duchi et al., 2011) where the global learning rate is 0.05 and sum of gradient squared is initialized to 0.1; parameter initialization over a Gaussian distribution with mean 0 and standard deviation 0.01; gradient normalization at 5. In the pretrained scenario, pretraining is done with Adam (Kingma & Ba, 2015) with learning rate equal to 0.01, \( \beta_1 = 0.9, \beta_2 = 0.999 \).

\section{Forward/Backward through the Inside-Outside Algorithm}

Figure 6 shows the procedure for obtaining the parsing marginals from the input potentials. This corresponds to running the inside-outside version of Eisner’s algorithm (Eisner, 1996). The intermediate data structures used during the dynamic programming algorithm are the (log) inside tables \( \alpha \), and the (log) outside tables \( \beta \). Both \( \alpha, \beta \) are of size \( n \times n \times 2 \times 2 \), where \( n \) is the sentence length. First two dimensions encode the start/end index of the span (i.e. subtree). The third dimension encodes whether the root of the subtree is the left (L) or right (R) index of the span. The fourth dimension indicates if the span is complete (1) or incomplete (0). We can calculate the marginal distribution of each word’s parent (for all words) in \( O(n^3) \) using this algorithm.

Backward pass through the inside-outside algorithm is slightly more involved, but still takes \( O(n^3) \) time. Figure 7 illustrates the backward procedure, which receives the gradient of the loss \( L \) with respect to the marginals, \( \nabla_{\alpha} L \), and computes the gradient of the loss with respect to the potentials \( \nabla_{\theta} L \). The computations must be performed in the double log-space semifield to handle log of negative values. See section 3.3 and Table 1 for more details.
Under review as a conference paper at ICLR 2017

Figure 6: Forward step of the syntactic attention layer to compute the marginals, using the inside-outside algorithm (Baker, 1979) on the data structures of Eisner (1996). We assume the special root symbol is the first element of the sequence, and that the sentence length is \( n \). Calculations are performed in log-space semifield with \( \oplus = + \) and \( \otimes = \text{logadd} \) for numerical precision. \( a, b \leftarrow c \) means \( a \leftarrow c \) and \( b \leftarrow c \). \( a \leftarrow \oplus b \) means \( a \leftarrow a \oplus b \).
procedure BACKPROPINSIDEOUTSIDE(θ, p, ν)

for s ∈ {1, n}, t ∈ {1, n}, s ≠ t do  ν Backpropagation uses the identity ∇^c_θ = (p ⊗ ∇^c_p)∇^log_p
    δ[s, t] ← log p[s, t] ⊗ log \nabla^p_p[s, t]  ν δ = (log(p ⊗ ∇^p_p))
    \nabla^c_0, \nabla^s, log ∇^c_0 ← −∞  ν Initialize inside (∇^c_0), outside (∇^s) gradients, and log of ∇^c_0

for s ∈ {1, n − 1} do  ν Backpropagate δ to ∇^c_0 and ∇^s
    for t ∈ {s + 1, n} do
        \nabla^c_0[s, t, R, 0], \nabla^s_0[s, t, R, 0] ← δ[s, t]
        \nabla^c_0[1, n, R, 1] ← −δ[s, t]
        if s > 1 then
            \nabla^c_0[s, t, L, 0], \nabla^s_0[s, t, L, 0] ← δ[t, s]
            \nabla^c_0[1, n, R, 1] ← −δ[s, t]
        else
            \nabla^c_0[s, t, R, 0], \nabla^s_0[s, t, R, 0] ← δ[s, t]
    \nabla^c_0[s, u, R, 1], \nabla^s_0[t, u, R, 1] ← (ν ⊗ β)[s, u, R, 1] ⊗ α[t, u, R, 1]

for k ∈ {1, n} do  ν Backpropagate through outside step
    for s ∈ {1, n − k} do
        t ← s + k
        ν ← \nabla^c_0[s, t, R, 0] ⊗ β[s, t, R, 0]
        ν, γ are temporary values

for u ∈ {1, n} do
    \nabla^c_0[u, s, R, 1], \nabla^s_0[u, s, R, 1] ← (ν ⊗ β)[u, s, R, 1] ⊗ α[u, s, R, 1]

if s > 1 then
    ν ← \nabla^c_0[s, t, L, 0] ⊗ β[s, t, L, 0]

for u ∈ [0, s] do
    \nabla^c_0[u, s, R, 1], \nabla^s_0[u, s, R, 1] ← (ν ⊗ β)[u, s, R, 1] ⊗ α[u, s, R, 1]

if s > 1 then
    ν ← \nabla^c_0[s, t, L, 0] ⊗ β[s, t, L, 1]

for u ∈ [0, s] do
    \nabla^c_0[u, s, R, 1], \nabla^s_0[u, s, R, 1] ← (ν ⊗ β)[u, s, R, 1] ⊗ α[u, s, R, 0]

for u ∈ [s + 1, n] do
    \nabla^c_0[u, t, R, 1], \nabla^s_0[u, t, R, 1] ← (ν ⊗ β)[u, t, R, 1] ⊗ α[u, s, R, 0]

for u ∈ [1, s − 1] do
    γ ← β[u, t, R, 0] ⊗ α[u, s − 1, R, 1] ⊗ \theta_{ut}
    \nabla^c_0[u, t, R, 0], \nabla^s_0[u, s − 1, R, 1], log ∇^p_0[u, t] ← (ν ⊗ γ)

for u ∈ [1, s − 1] do
    γ ← β[u, t, R, 0] ⊗ α[u, s − 1, R, 1] ⊗ \theta_{ut}
    \nabla^c_0[u, t, R, 0], \nabla^s_0[u, s − 1, R, 1], log ∇^p_0[u, t] ← (ν ⊗ γ)

for k ∈ {1, n} do  ν Backpropagate through inside step
    for s ∈ {1, n − k} do
        t ← s + k
        \nabla^c_0[s, t, R, 1] ↔ \nabla^s_0[s, t, R, 1] ⊗ α[s, t, R, 1]
        ν ← \nabla^c_0[s, t, L, 1] ⊗ α[s, t, L, 1]

for u ∈ [0, s] do
    \nabla^c_0[u, t, R, 1], \nabla^s_0[u, t, R, 1] ← (ν ⊗ β)[u, t, R, 1] ⊗ α[u, s, R, 0]

for u ∈ [0, s] do
    \nabla^c_0[u, t, R, 1], \nabla^s_0[u, t, R, 1] ← (ν ⊗ β)[u, t, R, 1] ⊗ α[u, s, R, 0]

for u ∈ [s − 1, n] do
    \nabla^c_0[u, t, R, 1], \nabla^s_0[u, t, R, 1] ← (ν ⊗ β)[u, t, R, 1] ⊗ α[u, s, R, 0]

for u ∈ [s, t − 1] do
    γ ← α[u, s, R, 1] ⊗ α[u + 1, t, L, 1] ⊗ \theta_{st}
    \nabla^c_0[u, s, R, 1], \nabla^s_0[u + 1, t, L, 1], log ∇^p_0[t, s] ← (ν ⊗ γ)

for u ∈ [s, t − 1] do
    γ ← α[u, s, R, 1] ⊗ α[u + 1, t, L, 1] ⊗ \theta_{st}
    \nabla^c_0[u, s, R, 1], \nabla^s_0[u + 1, t, L, 1], log ∇^p_0[t, s] ← (ν ⊗ γ)

return \exp\log ∇^c_0
  ν Exponentiate the log table and return ∇^c_0

Figure 7: Backpropagation through the inside-outside algorithm to calculate the gradient with respect to the input potentials. ∇^c_0 denotes the Jacobian of θ with respect to b (so ∇^c_0 is the gradient with respect to θ). a, b ← c means a ← a ⊕ c and b ← b ⊕ c.