Neural Representation Learning in Linguistic Structured Prediction

Lingpeng Kong

September 2017

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Noah A. Smith (co-Chair), Carnegie Mellon University / University of Washington
Chris Dyer (co-Chair), Carnegie Mellon University / Google DeepMind
Alan W. Black, Carnegie Mellon University
Michael Collins, Columbia University / Google Research

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

Advances in neural network architectures and training algorithms have demonstrated the effectiveness of representation learning in natural language processing. This thesis argues for the importance of modeling discrete structure in language, even when learning continuous representations.

We propose that explicit structure representations and learned distributed representations can be efficiently combined for improved performance over (i) traditional approaches to structure and (ii) uninformed neural networks that ignore all but surface sequential structure. We demonstrate, on three distinct problems, how assumptions about structure can be integrated naturally into neural representation learners for NLP problems, without sacrificing computational efficiency.

First, we propose segmental recurrent neural networks (SRNNs) which define, given an input sequence, a joint probability distribution over segmentations of the input and labelings of the segments and show that, compared to models that do not explicitly represent segments such as BIO tagging schemes and connectionist temporal classification (CTC), SRNNs obtain substantially higher accuracies on tasks including phone recognition and handwriting recognition.

Second, we propose dynamic recurrent acyclic graphical neural networks (DRAGNN), a modular neural architecture that generalizes the encoder/decoder concept to include explicit linguistic structures. Linguistic structures guide the building process of the neural networks

by following the transitions and encoding the (partial) structures constructed those transitions explicitly into the hidden layer activations. We show that our framework is significantly more accurate and efficient than sequence-to-sequence with attention for syntactic dependency parsing and yields more accurate multi-task learning for extractive summarization tasks.

Third, we propose to use discrete stochastic attention to model the alignment structures explicitly in the neural sequence-to-sequence translation model. We regularize the posterior distributions of the latent alignment decisions using the posteriors computed from models that make stronger independence assumptions but that have the same latent variables. We show that our posterior regularization scheme leads to substantially improved generalization. Since the posterior regularization objective can be generally expensive to compute, we propose several approximations based on importance sampling and find that they are either as good as or better than the exact objective in terms of held-out generalization.

The techniques proposed in this thesis automatically learn structurally informed representations of the inputs. Linguistically motivated inductive biases help learning better representations in the neural models and these representations and components can be better integrated with other end-to-end deep learning systems within and beyond NLP.

Contents

Chapter 1

Introduction

Computationally modeling the structure in language is crucial for two reasons. First, for the larger goal of automated language understanding, linguists have found that language meaning is derived through composition (Manning, 2016). Understanding novel and complex sentences crucially depends on being able to construct their meaning from smaller parts/atoms (e.g., words in a sentence) compositionally. Structure in language tells what these atoms are and how they fit together (e.g., syntactic or semantic parses) in composition. Second, from the perspective of machine learning, linguistic structures can be understood as a form of inductive bias, which helps learning succeed with less or less ideal data (Mitchell, 1980).

The inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict outputs given inputs that it has not encountered (Mitchell, 1980). Taking a broader view, all the things that aren't estimated from the data directly, we call it inductive bias. It is our understanding about the problem and our assumptions when we design the model, or from a Bayesian perspective, the *prior*.

There are many assumptions you can add or remove when designing the learn-

ing algorithm. Any particular inductive bias can be helpful or harmful, depending on the problem and the way we approach it. Consider the language modeling task as an example, where the goal is to estimate a probability distribution over sequences of words. The classical n-gram model makes an independence assumption that each word depends only on the last $n-1$ words. The assumption helps to alleviate the problem of data sparsity (Chen and Goodman, 1996) when we perform maximum likelihood estimation. The neural language model (Mikolov et al., 2010) removes this Markov assumption and outperforms the n-gram model. However, this isn't saying that we cannot add back certain linguistic driven inductive biases that make the model better. These biases might include syntactic structures (Dyer et al., 2016) or discourse relations (Ji et al., 2016). In this thesis, we specifically look into the question of how to make use of linguistic structures to form right inductive bias in the neural models, when only limited amounts of supervision is available.

Neural models are state-of-the-art for many NLP tasks, including speech recognition (Graves et al., 2013), dependency parsing (Kuncoro et al., 2017; Dozat and Manning, 2017; Kong et al., 2017) and machine translation (Vaswani et al., 2017). For many applications, the model architecture combines dense representations of words (Manning, 2016) with sequential recurrent neural networks (Dyer et al., 2016). Thus, while they generally make use of information about what the words are (rather than operating on sequences of characters), they ignore syntactic and semantic structure or force models to learn it, and thus can be criticized as being structurally naive.

This thesis argues that explicit structure representations and learned distributed representations can be efficiently combined for improved performance over (i) traditional approaches to structure and (ii) uninformed neural networks that ignore all but surface sequential structure. As an example, Dyer et al. (2015) yields better

accuracy than Chen and Manning (2014) by replacing the word representations on the stack with composed representations derived from (partially built) dependency tree structure.

In the first part of this thesis, first published as Kong et al. (2015) and Lu et al. (2016), we propose segmental recurrent neural networks (SRNNs) which define, given an input sequence, a joint probability distribution over segmentations of the input and labelings of the segments. Traditional neural solutions to this problem, e.g., connectionist temporal classification (CTC, Graves et al. (2006a)), reduce the segmental sequence labeling problem to a sequence labeling problem in the same spirit as *BIO* tagging. In our model, representations of the input segments (i.e., contiguous subsequences of the input) are computed by composing their constituent tokens using bi-directional recurrent neural nets, and these *segment embeddings* are used to define compatibility scores with output labels. Our model achieves competitive results in phone recognition, handwriting recognition, and joint word segmentation & POS tagging.

In the second part of the thesis, first published as Kong et al. (2017), we propose dynamic recurrent acyclic graphical neural networks (DRAGNN), a modular neural architecture that generalizes the encoder/decoder concept to include explicit linguistic structures. The core mechanism of DRAGNN is the Transition-Based Recurrent Unit (TBRU). It represents the linguistic structure a sequence of decision/state pairs and guide the building process of the neural networks by following the transitions and encoding the (partial) structures constructed those transitions explicitly into the hidden layer activations. We show that our framework is significantly more accurate and efficient than sequence-to-sequence with attention for syntactic dependency parsing and yields more accurate multi-task

learning for extractive summarization tasks.

In the third part of the thesis, we propose to use stochastic attention to model the alignment structures explicitly in the neural sequence-to-sequence translation model. We regularize the posterior distributions of the latent alignment decisions using the posteriors computed from models that make stronger independence assumptions. We show that our posterior regularization scheme leads to substantially improved generalization and several approximations based on importance sampling is either as good as or better than the exact objective in terms of held-out generalization.

In this thesis, we argue for the importance of explicitly representing structure in neural models. We demonstrate, on three distinct problems, how assumptions about structure can be integrated naturally into neural representation learners for NLP problems, without sacrificing computational efficiency. We demonstrate that, when comparing with structurally naive models, models that reason about the internal linguistic structure of the data demonstrate better generalization performance.

Chapter 2

Notation and Representations

In this thesis, we model the conditional probability in the form of $p(\boldsymbol{y} \mid \boldsymbol{x})$ using neural networks, where the goal is to predict the target outputs $\boldsymbol{y}$ from the input observations $\boldsymbol{x}$.

We introduce the variable $\boldsymbol{z}$, where $\boldsymbol{z}$ is the linguistic structure we would like to explicitly consider. $\boldsymbol{z}$ can take multiple forms. For example, segment structures (chapter 3), parse tree structures (chapter 4), or alignment structures (chapter 5).

During the training phrase, when the structure is observable (chapter 4), we can choose to maximize $p(\boldsymbol{y} \mid \boldsymbol{x})$ directly. In this setting, the loss is defined as

$$\mathcal{L} = -\log p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) \tag{2.1}$$

If the structure not directly observable from the data, we can treat $\boldsymbol{z}$ as a hidden variable, where we follow the marginal likelihood training criterion

$$\mathcal{L} = -\log p(\boldsymbol{y} \mid \boldsymbol{x}) = -\log \sum_{\boldsymbol{z}} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) \tag{2.2}$$

$\boldsymbol{z}$ can also be given as a soft structure (i.e. , $q(\boldsymbol{z})$ models probability of $\boldsymbol{z}$), like in the case of alignment structure, we can add this to the loss of the model in the

framework of posterior regularization (Ganchev et al., 2010):

$$\mathcal{L} = -\log \sum_{\boldsymbol{z}} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) + \gamma \times D_{KL}(p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y}) \mid\mid q(\boldsymbol{z})), \tag{2.3}$$

where D_{KL} denotes the Kullback–Leibler divergence (Kullback and Leibler, 1951). In the decoding phrase, we are interested in the following prediction problem,

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y} \mid \boldsymbol{x}) = \arg\max_{\boldsymbol{y}} \sum_{\boldsymbol{z}} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}). \tag{2.4}$$

We call this exact decoding (chapter 5). However, summing over all the possible structure $\boldsymbol{z}$ can be expensive or even intractable in many cases. Therefore, we jointly maximize over $\boldsymbol{y}$ and $\boldsymbol{z}$ as a computationally tractable approximation for the exact decoding method (chapter 3, chapter 4, chapter 5), which is commonly used in MAP inference problems with latent variables. This is especially convenient when the model can be easily broke down into the following two parts (chapter 4, chapter 5):

$$p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) = p(\boldsymbol{z} \mid \boldsymbol{x}) \times p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{z}) \tag{2.5}$$

In some cases (chapter 4, §3.4), given input $\boldsymbol{x}$, we have a one to many mapping from $\boldsymbol{y}$ to $\boldsymbol{z}$, and a one to one mapping from $\boldsymbol{z}$ to $\boldsymbol{y}$. In these settings, we use $\Phi(\boldsymbol{y})$ to define the set of all the possible $\boldsymbol{z}$ structures that consistent with $\boldsymbol{y}$, and $\phi(\boldsymbol{z})$ to define the corresponding $\boldsymbol{y}$ for $\boldsymbol{z}$.

Chapter 3

Segmental Recurrent Neural Networks

In this chapter, we propose **segmental recurrent neural networks** (SRNNs)[1] which define, given an input sequence, a joint probability distribution over segmentations of the input and labelings of the segments. Representations of the input segments (i.e., contiguous subsequences of the input) are computed by encoding their constituent tokens using bi-directional recurrent neural nets, and these "segment embeddings" are used to define compatibility scores with output labels. These local compatibility scores are integrated using a global semi-Markov conditional random field (Sarawagi and Cohen, 2004). Both fully supervised training—in which segment boundaries and labels are observed—as well as partially supervised training—in which segment boundaries are latent—are straightforward. Applications of fully supervised training include named entity recognition, Chinese word segmentation (Liu et al., 2016) and frame-semantic parsing (Swayamdipta et al., 2017), where boundaries of segments are labeled in the training data. A typical case of partially supervised training is speech

[1]First published as Kong et al. (2015) and Lu et al. (2016).

recognition, where we know the sequence of phonemes during training but the boundaries are not known. Experiments show that, compared to models that do not explicitly represent segments such as BIO tagging schemes and connectionist temporal classification (CTC, Graves et al. (2006a)), SRNNs obtain substantially higher accuracies.

3.1 Model

Given a sequence of input observations $\boldsymbol{x} = \langle \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{|\boldsymbol{x}|} \rangle$ with length $|\boldsymbol{x}|$, a **segmental recurrent neural network** (SRNN) defines a joint distribution $p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$ over a sequence of labeled segments each of which is characterized by a duration ($z_i \in \mathbb{Z}_+$) and label ($y_i \in Y$). The segment durations are constrained such that $\sum_{i=1}^{|\boldsymbol{z}|} z_i = |\boldsymbol{x}|$. The length of the output sequence $|\boldsymbol{y}| = |\boldsymbol{z}|$ is a random variable, and $|\boldsymbol{y}| \leq |\boldsymbol{x}|$ with probability 1. We write the starting time of segment i as $s_i = 1 + \sum_{j<i} z_j$.

To motivate our model form, we state several desiderata. First, we are interested in the following prediction problem,

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y} \mid \boldsymbol{x}) = \arg\max_{\boldsymbol{y}} \sum_{\boldsymbol{z}} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) \approx \arg\max_{\boldsymbol{y}} \max_{\boldsymbol{z}} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}). \quad (3.1)$$

Note the use of joint maximization over $\boldsymbol{y}$ and $\boldsymbol{z}$ as a computationally tractable substitute for marginalizing out $\boldsymbol{z}$; this is commonly done in MAP inference problems with latent variables.

Second, for problems where these explicit durations are unavailable at training time, they can be inferred as a latent variable. To train this model, we use a marginal likelihood training criterion,

$$\mathcal{L} = -\log p(\boldsymbol{y} \mid \boldsymbol{x}) = -\log \sum_{\boldsymbol{z}} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}). \quad (3.2)$$

In Eqs. 3.1 and 3.2, the conditional probability of the labeled segment sequence is (assuming kth order dependencies on $\boldsymbol{y}$):

$$p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{i=1}^{|\boldsymbol{y}|} \exp f(y_{i-k:i}, z_i, \mathbf{x}) \tag{3.3}$$

where $Z(\boldsymbol{x})$ is an appropriate normalization function. To ensure the expressiveness of f and the computational efficiency of the maximization and marginalization problems in Eqs. 3.1 and 3.2, we use the following definition of f,

$$f(y_{i-k:i}, z_i, \mathbf{x}_{s_i:s_i+z_i-1}) = \mathbf{w}^\top \phi(\mathbf{V}[\mathbf{g}_y(y_{i-k}); \ldots; \mathbf{g}_y(y_i); \mathbf{g}_z(z_i); \overrightarrow{\text{RNN}}(\mathbf{c}_{s_i:s_i+z_i-1}); \overleftarrow{\text{RNN}}(\mathbf{c}_{s_i:s_i+z_i-1})] + \mathbf{a}) + b \tag{3.4}$$

where $\overrightarrow{\text{RNN}}(\mathbf{c}_{s_i:s_i+z_i-1})$ is a recurrent neural network that computes the **forward segment embedding** by "encoding" the z_i-length subsequence of $\boldsymbol{x}$ starting at index s_i,[2] and $\overleftarrow{\text{RNN}}$ computes the reverse segment embedding (i.e., traversing the sequence in reverse order), and $\mathbf{g}_y$ and $\mathbf{g}_z$ are functions which map the label candidate $\boldsymbol{y}$ and segmentation duration $\boldsymbol{z}$ into a vector representation. The notation $[\mathbf{a}; \mathbf{b}; \mathbf{c}]$ denotes vector concatenation. Finally, the concatenated segment duration, label candidates and segment embedding are passed through a affine transformation layer parameterized by $\mathbf{V}$ and $\mathbf{a}$ and a nonlinear activation function ϕ (e.g., tanh), and a dot product with a vector $\mathbf{w}$ and addition by scalar b computes the log potential for the clique. Our proposed model is equivalent to a semi-Markov conditional random field (Sarawagi and Cohen, 2004) with local features computed using neural networks. Figure 3.1 shows the model graphically.

We chose bi-directional LSTMs (Graves and Schmidhuber, 2005) as the implementation of the RNNs in Eq. 3.4. LSTMs (Hochreiter and Schmidhuber, 1997a)

[2]Rather than directly reading the $\mathbf{x}_i$'s, each token is represented as the concatenation, $\mathbf{c}_i$, of a forward and backward over the sequence of raw inputs. This permits tokens to be sensitive to the contexts, and this is standardly used with neural net sequence labeling models (Graves et al., 2006b).

are a popular variant of RNNs which have been seen successful in many representation learning problems (Graves and Jaitly, 2014; Karpathy and Fei-Fei, 2015). Bi-directional LSTMs enable effective computation for embeddings in both directions and are known to be good at preserving long distance dependencies, and hence are well-suited for our task.

3.2 Parameter Learning

We consider two different learning objectives. We use the log loss in our models, other losses such as structured hinge loss can also be used (Tang et al., 2017).

Supervised learning In the supervised case, both the segment durations ($\boldsymbol{z}$) and their labels ($\boldsymbol{y}$) are observed.

$$\begin{aligned}\mathcal{L} &= \sum_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\in\mathcal{D}} -\log p(\boldsymbol{y},\boldsymbol{z} \mid \boldsymbol{x}) \\ &= \sum_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\in\mathcal{D}} \log Z(\boldsymbol{x}) - \log Z(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\end{aligned}$$

In this expression, the unnormalized conditional probability of the reference segmentation/labeling, given the input $\boldsymbol{x}$, is written as $Z(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$.

Partially supervised learning In the partially supervised case, only the labels are observed and the segments (the $\boldsymbol{z}$) are unobserved and marginalized.

$$\begin{aligned}\mathcal{L} &= \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} -\log p(\boldsymbol{y} \mid \boldsymbol{x}) \\ &= \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \sum_{\boldsymbol{z}\in\mathcal{Z}(\boldsymbol{x},\boldsymbol{y})} -\log p(\boldsymbol{y},\boldsymbol{z} \mid \boldsymbol{x}) \\ &= \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \log Z(\boldsymbol{x}) - \log Z(\boldsymbol{x},\boldsymbol{y})\end{aligned}$$

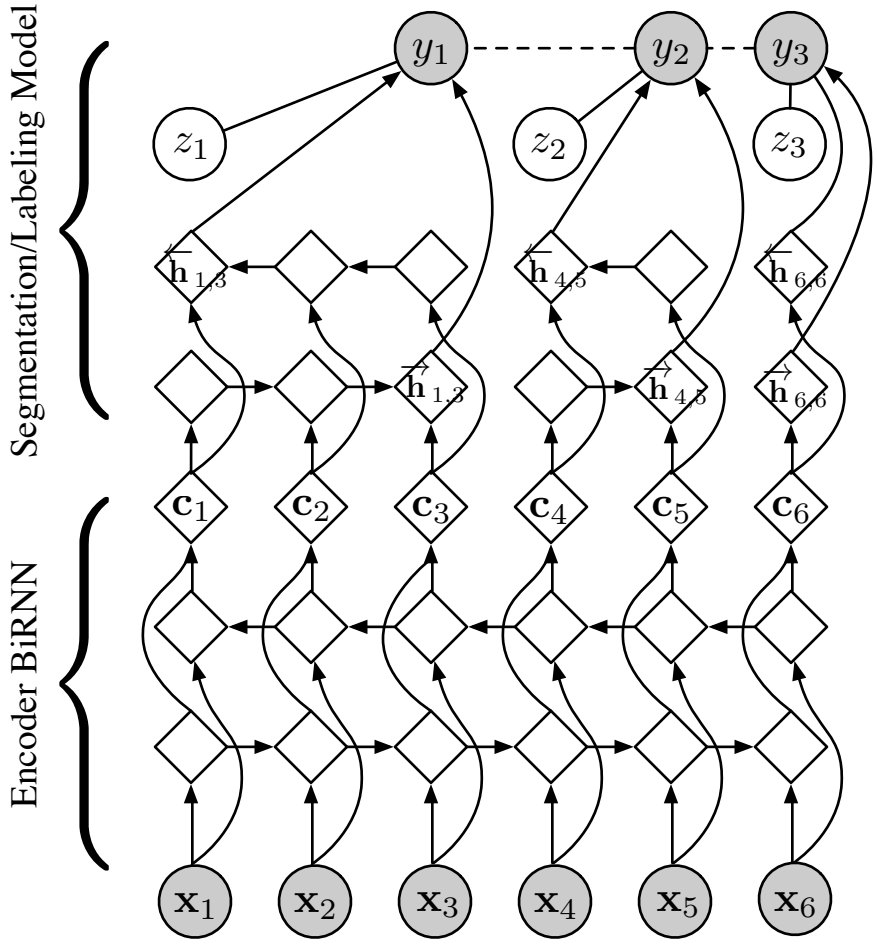

Figure 3.1: Graphical model showing a six-frame input and three output segments with durations $\boldsymbol{z} = \langle 3, 2, 1 \rangle$ (this particular setting of $\boldsymbol{z}$ is shown only to simplify the layout of this figure; the model assigns probabilities to all valid settings of $\boldsymbol{z}$). Circles represent random variables. Shaded nodes are observed in training; open nodes are latent random variables; diamonds are deterministic functions of their parents; dashed lines indicate optional statistical dependencies that can be included at the cost of increased inference complexity. The graphical notation we use here draws on conventions used to illustrate neural networks and graphical models.

For both the fully and partially supervised scenarios, the necessary derivatives can be computed using automatic differentiation or (equivalently) with backward variants of the above dynamic programs (Sarawagi and Cohen, 2004).

3.3 Inference with Dynamic Programming

We are interested in three inference problems: (i) finding the most probable segmentation/labeling for a model given a sequence $\boldsymbol{x}$; (ii) evaluating the partition function $Z(\boldsymbol{x})$; and (iii) computing the posterior marginal $Z(\boldsymbol{x}, \boldsymbol{y})$, which sums over all segmentations compatible with a reference sequence $\boldsymbol{y}$. These can all be solved using dynamic programming. For simplicity, we will assume zeroth order Markov dependencies between the y_is. Extensions to the kth order Markov dependencies are straightforward. Since each of these algorithms relies on the forward and reverse segment embeddings, we first discuss how these can be computed before going on to the inference algorithms.

3.3.1 Computing Segment Embeddings

Let $\overrightarrow{\mathbf{h}}_{i,j}$ designate the $\overrightarrow{\text{RNN}}$ encoding of the input span (i, j), traversing from left to right, and let $\overleftarrow{\mathbf{h}}_{i,j}$ designate the reverse direction encoding using $\overleftarrow{\text{RNN}}$. There are thus $O(|\boldsymbol{x}|^2)$ vectors that must be computed, each of length $O(|\boldsymbol{x}|)$. Naively this can be computed in time $O(|\boldsymbol{x}|^3)$, but the following dynamic program reduces this to $O(|\boldsymbol{x}|^2)$:

$$\begin{aligned}
\overrightarrow{\mathbf{h}}_{i,i} &= \overrightarrow{\text{RNN}}(\overrightarrow{\mathbf{h}}_0, \mathbf{c}_i) \\
\overrightarrow{\mathbf{h}}_{i,j} &= \overrightarrow{\text{RNN}}(\overrightarrow{\mathbf{h}}_{i,j-1}, \mathbf{c}_j) \\
\overleftarrow{\mathbf{h}}_{i,i} &= \overleftarrow{\text{RNN}}(\overleftarrow{\mathbf{h}}_0, \mathbf{c}_i) \\
\overleftarrow{\mathbf{h}}_{i,j} &= \overleftarrow{\text{RNN}}(\overleftarrow{\mathbf{h}}_{i+1,j}, \mathbf{c}_i)
\end{aligned}$$

The algorithm is executed by initializing in the values on the diagonal (representing segments of length 1) and then inductively filling out the rest of the matrix. In practice, we often can put a upper bound for the length of a eligible segment thus reducing the complexity of runtime to $O(|\boldsymbol{x}|)$. This savings can be substantial for

very long sequences (e.g., those encountered in speech recognition).

We define the concatenation of $\overrightarrow{\mathbf{h}}_{i,j}$ and $\overleftarrow{\mathbf{h}}_{i,j}$ as the *segment embedding* of segment i to j. One important advantage of SRNNs over RNNs which simply adopt the *BIO* tagging scheme is that we have these explicit representations for arbitrary given spans.

3.3.2 Computing the most probable segmentation/labeling and $Z(\boldsymbol{x})$

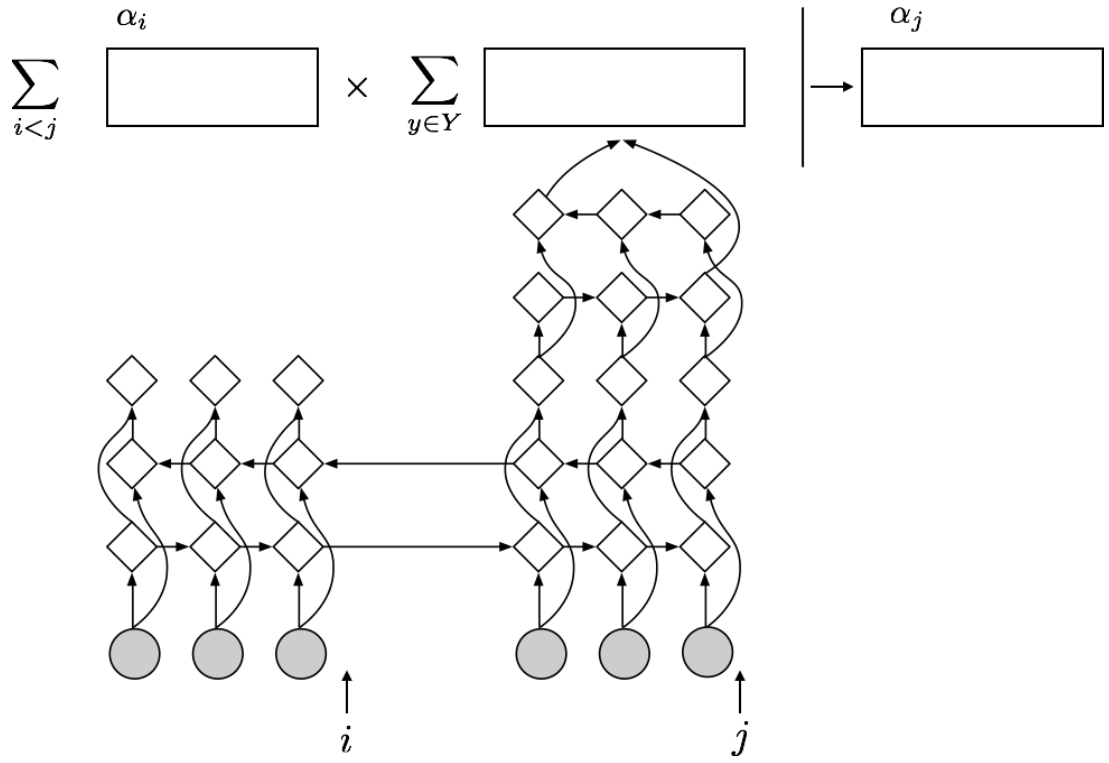

Figure 3.2: Dynamic programming for the computation of $Z(\boldsymbol{x})$. We introduce a auxiliary variable α_i basically sum over all the potentials until the point of i in the sequence. By combining that with the next segment, (we need to sum over all the y labels there), we get α_j. All these partial representations are computed from dynamically constructed neural networks.

For the input sequence $\boldsymbol{x}$, there are $2^{|\boldsymbol{x}|-1}$ possible segmentations and $O(|Y|^{|\boldsymbol{x}|})$

different labelings of these segments, making exhaustive computation of $Z(\boldsymbol{x})$ infeasible. Fortunately, the partition function $Z(\boldsymbol{x})$ may be computed in polynomial time with the following dynamic program (Figure 3.2):

$$\alpha_0 = 1$$
$$\alpha_j = \sum_{i<j} \alpha_i \times \sum_{y \in Y} \left(\exp \mathbf{w}^\top \phi(\mathbf{V}[g_y(y); g_z(z_i); \overrightarrow{\text{RNN}}(\mathbf{c}_{s_i:s_i+z_i-1}); \overleftarrow{\text{RNN}}(\mathbf{c}_{s_i:s_i+z_i-1})] + \mathbf{a}) + b \right).$$

After computing these values, $Z(\boldsymbol{x}) = \alpha_{|\boldsymbol{x}|}$. By changing the summations to a max operator (and storing the corresponding arg max values), the maximum *a posteriori* segmentation/labeling can be computed.

Both the partition function evaluation and the search for the MAP outputs run in time $O(|\boldsymbol{x}|^2 \cdot |Y|)$ with this dynamic program. Adding nth order Markov dependencies between the y_is adds requires additional information in each state and increases the time and space requirements by a factor of $O(|Y|^n)$. However, this may be tractable for small $|Y|$ and n.

Avoiding overflow. Since this dynamic program sums over exponentially many segmentations and labelings, the values in the α_i chart can become very large. Thus, to avoid issues with overflow, computations of the α_i's must be carried out in log space.[3]

3.3.3 Computing $Z(\boldsymbol{x}, \boldsymbol{y})$

To compute the posterior marginal $Z(\boldsymbol{x}, \boldsymbol{y})$, it is necessary to sum over all segmentations that are compatible with a label sequence $\boldsymbol{y}$ given an input sequence $\boldsymbol{x}$.

[3] An alternative strategy for avoiding overflow in similar dynamic programs is to rescale the forward summations at each timestep (Rabiner, 1989; Graves et al., 2006b). Unfortunately, in a semi-Markov architecture each term in α_i sums over different segmentations (e.g., the summation for α_2 will have contain some terms that include α_1 and some terms that include only α_0), which means there are no common factors, making this strategy inapplicable.

To do so requires only a minor modification of the previous dynamic program to track how much of the reference label sequence $\boldsymbol{y}$ has been consumed. We introduce the variable m as the index into $\boldsymbol{y}$ for this purpose. The modified recurrences are:

$$\gamma_0(0) = 1$$
$$\gamma_j(m) = \sum_{i<j} \gamma_i(m-1) \times \left(\exp \mathbf{w}^\top \phi(\mathbf{V}[g_y(y_i); g_z(z_i); \overrightarrow{\text{RNN}}(\mathbf{c}_{s_i:s_i+z_i-1}); \overleftarrow{\text{RNN}}(\mathbf{c}_{s_i:s_i+z_i-1})] + \mathbf{a}) + b\right).$$

The value $Z(\boldsymbol{x}, \boldsymbol{y})$ is $\gamma_{|\boldsymbol{x}|}(|\boldsymbol{y}|)$.

3.4 Connectionist Temporal Classification

We briefly review the connectionist temporal classification (CTC) model here since CTC is used as an important baseline for our experiments. CTC also directly computes the conditional probability $P(\boldsymbol{y} \mid \boldsymbol{x})$, with the key difference from SRNNs in that it normalizes the probabilistic distribution at the frame level [4]. Because of this, CTC doesn't include a segmentation explicitly in its representation. To address the problem of length mismatch between the input and output sequences, CTC allows repetitions of output labels and introduces a special blank token ($-$), which represents the probability of not emitting any label at a particular timestep. The conditional probability is then obtained by summing over all the probabilities of all the paths that corresponding to $\boldsymbol{y}$ after merging the repeated labels and removing the blank tokens, i.e.,

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \sum_{\boldsymbol{z} \in \Psi(\boldsymbol{z})} p(\boldsymbol{z} \mid \boldsymbol{x}), \tag{3.5}$$

[4] In speech recognition, we often segment the audio, which is a time-varying signal, into short snippets of audio. We call these snippets "*frames*".

where $\Psi(\boldsymbol{y})$ denotes the set of all possible paths that correspond to $\boldsymbol{y}$ after repetitions of labels and insertions of the blank token (Figure 3.3). Now the length of $\boldsymbol{z}$ is the same as $\boldsymbol{x}$, the probability $p(z \mid \boldsymbol{x})$ is then approximated by the independence assumption as

$$p(z \mid \boldsymbol{x}) \approx \prod_{t=1}^{T} p(z_t \mid \boldsymbol{x}), \tag{3.6}$$

where z_t ranges over $\mathcal{Y} \cup \{-\}$, and $p(z_t \mid \boldsymbol{x})$ can be computed using the softmax function. As with SRRNs, there are many sequences $\boldsymbol{z}$ that are compatible with $\boldsymbol{y}$. The training criterion for CTC is to maximize the conditional probability of the ground truth labels, which is equivalent to minimizing the negative log likelihood:

$$\mathcal{L} = -\log p(\boldsymbol{y} \mid \boldsymbol{x}), \tag{3.7}$$

which can be reformulated as the cross entropy loss criterion. More details regarding the computation of the loss and the backpropagation algorithm to train CTC models can be found in (Graves et al., 2006a).

3.5 Experiments

3.5.1 Online Handwriting Recognition

Dataset We use the handwriting dataset from (Kassel, 1995). This dataset is an online collection of hand-written words from 150 writers. It is recorded as the coordinates (x, y) at time t plus special pen-down/pen-up notations. We break the coordinates into strokes using the pen-down and pen-up notations. One character typically consists one or more contiguous strokes.[5]

[5]There are infrequent cases where one stroke can go across multiple characters or the strokes which form the character can be not contiguous. We leave those cases for future work.

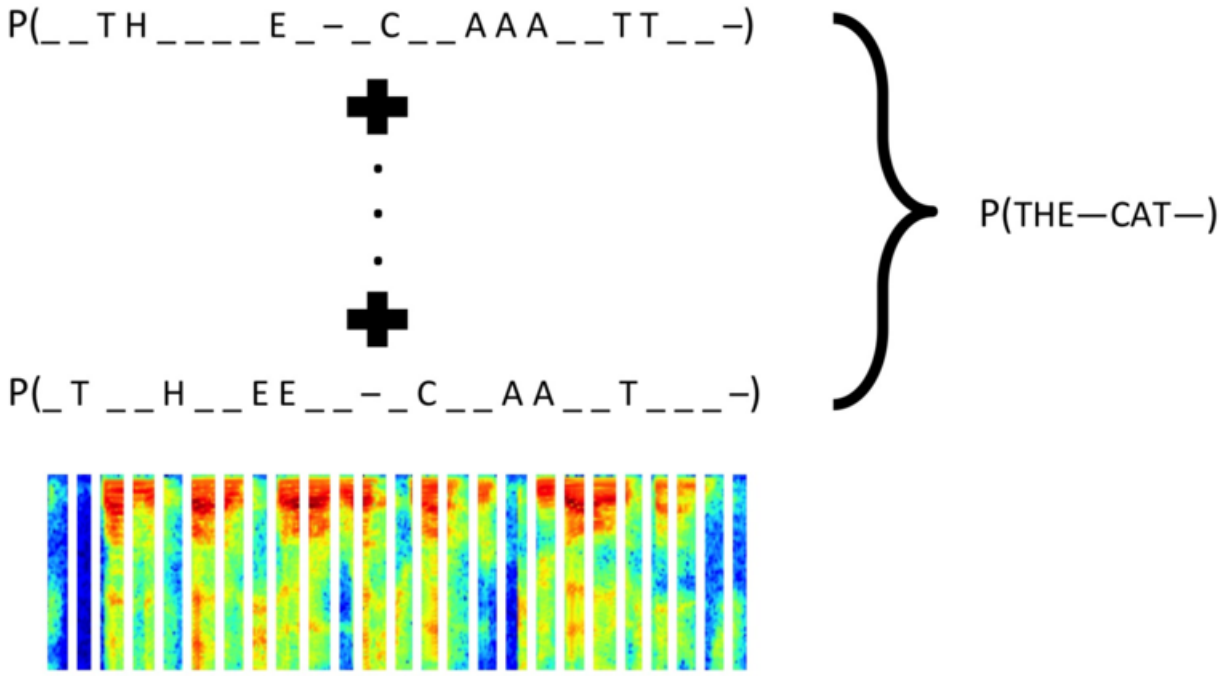

Figure 3.3: The illustration above shows CTC computing the probability of an output sequence ”THE CAT ”, as a sum over all possible alignments of input sequences that could map to ”THE CAT ”, taking into account that labels may be duplicated because they may stretch over several timesteps of the input data (represented by the spectrogram at the bottom of the image) (Amodei et al., 2015).

The dataset is split into train, development and test set following (Kassel, 1995). Table 3.1 presents the statistics for the dataset.

A well-know variant of this dataset was introduced by Taskar et al. (2004). Taskar et al. (2004) selected a “clean” subset of about 6,100 words and rasterized and normalized the images of each letter. Then, the uppercased letters (since they are usually the first character in a word) are removed and only the lowercase letters are used. The main difference between our dataset and theirs is that their dataset is “offline” — Taskar et al. (2004) mapped each character into a bitmap and treated the segmentation of characters as a preprocessing step. We use the richer representation of the sequence of strokes as input.

	#words	#characters
Train	4,368	37,247
Dev	1,269	10,905
Test	637	5,516
Total	6,274	53,668

Table 3.1: Statistics of the Online Handwriting Recognition Dataset

Implementation We trained two versions of our model on this dataset, namely, the fully supervised model (§3.2), which takes advantage of the gold segmentations on training data, and the partially supervised model (§3.2) in which the gold segmentations are only used in the evaluation. A CTC model reimplemented on the top of our Encoder BiRNNs layer (Figure 3.1) is used as a baseline so that we can see the effect of explicitly representing the segmentation.[6] For the decoding of the CTC model, we simply use the best path decoding, where we assume that the most probable path will correspond to the most probable labeling, although it is known that prefix search decoding can slightly improve the results (Graves et al., 2006b).

As a preprocessing step, we first represented each point in the dataset using a 4 dimensional vector, $\mathbf{p} = (p_x, p_y, \Delta p_x, \Delta p_y)$, where p_x and p_y are the normalized coordinates of the point and Δp_x and Δp_y are the corresponding changes in the coordinates with respect to the previous point. Δp_x and Δp_y are meant to capture basic direction information. Then we map the points inside one stroke into a fixed-

[6]The CTC interpretation rules specify that repeated symbols, e.g. , `aa` will be interpreted as a single token of `a`. However since the segments in the handwriting recognition problem are extremely short, we use different rules and interpret this as `aa`. That is, only the blank symbol may be used to represent extended durations. Our experiments indicate this has little effect, and Graves (p.c.) reports that this change does not harm performance in general.

length vector using a bi-direction LSTM. Specifically, we concatenated the last position's hidden states in both directions and use it as the input vector $\mathbf{x}$ for the stroke.

In all the experiments, we use Adam (Kingma and Ba, 2014) with $\lambda = 1 \times 10^{-6}$ to optimize the parameters in the models. We train these models until convergence and picked the best model over the iterations based on development set performance then report performance on the test set.

We used 5 as the hidden state dimension in the bi-directional RNNs, which map the points into fixed-length stroke embeddings (hence the input vector size $5 \times 2 = 10$). We set the hidden dimensions of $\mathbf{c}$ in our model and CTC model to 24 and segment embedding $\mathbf{h}$ in our model as 18. We tried to experiment with larger hidden dimensions and we found the performance did not vary much.

The results of the online handwriting recognition task are presented in Table 3.2. We see that both of our models outperform the baseline CTC model, which does not carry an explicit representation for the segments being labeled, by a significant margin. An interesting finding is that, although the partially supervised model performs slightly worse in the development set, it actually outperforms the fully supervised model in the test set. Because the test set is written by different people from the train and development set, they exhibit different styles in their handwriting; our results suggest that the partially supervised model may generalize better across different writing styles.

3.5.2 Joint Chinese Word Segmentation and POS tagging

In this section, we look into two related tasks. The first task is joint Chinese word segmentation and POS tagging, where the $\boldsymbol{z}$ variables will group the Chinese characters into words and the $\boldsymbol{y}$ variables assign POS tags as labels to these words. We also tested our model on pure Chinese word segmentation task, where the assign-

	Development				Test			
	P_{seg} (%)	R_{seg} (%)	F_{seg} (%)	Error (%)	P_{seg} (%)	R_{seg} (%)	F_{seg} (%)	Error (%)
SRNNs (Partial)	98.7	98.4	98.6	4.2	99.2	99.1	99.2	2.7
SRNNs (Full)	98.9	98.6	98.8	4.3	98.8	98.6	98.6	5.4
CTC	-	-	-	15.2	-	-	-	13.8

Table 3.2: The performance of SRNNs and CTC on the task of hand-writing recognition

ments of $\boldsymbol{z}$ is the only thing we care about (simulated using a single label for all segments).

Dataset We used standard benchmark datasets for these two tasks. For the joint Chinese word segmentation and POS tagging task, we use the Penn Chinese Treebank 5 (Xue et al., 2005), following the standard train/dev/test splits. For the pure Chinese word segmentation task, we used the SIGHAN 2005 dataset[7]. This dataset contains four portions, covering both simplified and traditional Chinese. Since there is no pre-assigned development set in this dataset (only train and test set are provided), we manually split the original train set into two, one of which (roughly the same size as the test set) is used as the development set. For both tasks, we use Wang2Vec (Ling et al., 2015a) to generate the pre-trained character embeddings from the Chinese Gigaword (Graff and Chen, 2005).

Implementation Only the supervised version of SRNNs (§3.2) is tested in these tasks. The baseline model is a bi-directional LSTM tagger (basically the same structure as our Encoder BiRNNs in Figure 3.1). It takes the $\mathbf{c}$ at each timestep and pushes it through an element-wise non-linear transformation (tanh) followed by an affine transformation to map it to the same dimension as the number of labels. The total loss is therefore the sum of negative log probabilities over the sequence. Greedy decoding is applied in the baseline model, making it a zeroth

[7] `http://www.sighan.org/bakeoff2005/`

order model like our SRNNs.

In order to perform segmentation and POS tagging jointly, we composed the POS tags with "B" or "I" to represent the segmentation point. For the segmentation-only task, in the SRNNs we simply used same dummy tag for all y and only care about the z assignments. In the BiRNN case, we used "B" and "I" tags.

For both tasks, the dimension for the input character embedding is 64. For our model, the dimension for $\mathbf{c}$ and the segment embedding $\mathbf{h}$ is set to 24. For the baseline bi-directional LSTM tagger, we set the hidden dimension (the $\mathbf{c}$ equivalent) size to 128. Here we deliberately chose a larger size than in our model hoping to make the number of parameters in the bi-directional LSTM tagger roughly the same as our model. We trained these models until convergence and picked the best model over iterations based on its performance on the development set.

As for speed, the SRNNs run at $\sim$3.7 sentence per second during training on the CTB dataset using a single CPU.

Results Table 3.3 presents the results for the joint Chinese word segmentation task. We can see that in both segmentation and POS tagging, the SRNNs achieve higher F-scores than the BiRNNs.

Table 3.4 presents the results for the Chinese word segmentation task (POS tagging is not required). The SRNNs perform better than the BiRNNs with the exception of the PKU portion of the dataset. The reason for this is probably because the training set in this portion is the smallest among the four. This leads to high variance in the test results.

3.5.3 End-to-end Speech Recognition

Hierarchical Subsampling In speech recognition, since the input $\boldsymbol{x}$ is at the frame level, it is computationally expensive for RNNs to model these long sequences, because the number of possible segmentations is exponential with the

	Development			Test		
	P_{seg} (%)	R_{seg} (%)	F_{seg} (%)	P_{seg} (%)	R_{seg} (%)	F_{seg} (%)
BiRNNs	93.2	92.9	93.0	94.7	95.2	95.0
SRNNs	93.8	93.8	93.8	95.3	95.8	95.5
	P_{tag} (%)	R_{tag} (%)	F_{tag} (%)	P_{tag} (%)	R_{tag} (%)	F_{tag} (%)
BiRNNs	87.1	86.9	87.0	88.1	88.5	88.3
SRNNs	89.0	89.1	89.0	89.8	90.3	90.0

Table 3.3: The performance of BiRNNs and SRNNs on the task of joint Chinese word segmentation and POS tagging. Labeled precision (P), recall (R) and F1-score (F) are measured.

length of the input sequence as mentioned before. The computational cost can be significantly reduced by using the hierarchical subsampling RNN (Graves, 2012a) to shorten the input sequences, where the subsampling layer takes a window of hidden states from the lower layer as input as shown in Figure 3.4. In our speech experiments (§3.5.3), we consider three variants: a) *concatenate* – the hidden states in the subsampling window are concatenated before they are fed into the next layer; b) *add* – the hidden states are added into one vector for the next layer; c) *skip* – only the last hidden state in the window is kept and all the others are skipped. The last two schemes are computationally cheaper as they do not introduce extra model parameters.

We demonstrate the results of the hierarchical subsampling recurrent network, which is the key to speed up our experiments. We set the size of the subsampling window to be 2, therefore each subsampling layer reduced the time resolution by a factor of 2. We set the maximum segment length (§3.3.1) to be 300 milliseconds, which corresponded to 30 frames of FBANKs (sampled at the rate of 10 milliseconds). With two layers of subsampling recurrent networks, the time resolution

	BiRNNs			SRNNs		
	P_{seg} (%)	R_{seg} (%)	F_{seg} (%)	P_{seg} (%)	R_{seg} (%)	F_{seg} (%)
CU	92.7	93.1	92.9	93.3	93.7	93.5
AS	92.8	93.5	93.1	93.2	94.2	93.7
MSR	89.9	90.1	90.0	90.9	90.4	90.7
PKU	91.5	91.2	91.3	90.6	90.6	90.6

Table 3.4: The performance of BiRNNs and SRNNs on the task of Chinese word segmentation on SIGHAN 2005 dataset. There are four portions of the dataset from City University of Hong Kong (CU), Academia Sinica (AS), Microsoft Research (MSR) and Peking University (PKU). The former two are in traditional Chinese and the latter two are in simplified Chinese. Unlabeled precision (P), recall (R) and F1-score (F) are measured.

subsampling	L	speedup
No	30	1
1 layer	15	$\sim$3x
2 layers	8	$\sim$10x

Table 3.5: Speedup by hierarchical subsampling networks.

was reduced by a factor of 4, and the value of L was reduced to be 8, yielding around 10 times speedup as shown in Table 3.5.

Table 3.6 compares the three implementations of the recurrent subsampling network detailed in section 3.5.3. We observed that concatenating all the hidden states in the subsampling window did not yield lower phone error rate (PER) than using the simple *skipping* approach, which may be due to the fact that the TIMIT dataset is small and it prefers a smaller model. On the other hand, adding the hidden states in the subsampling window together worked even worse, possibly due to that the sequential information in the subsampling window was flattened. In the following experiments, we stuck to the *skipping* method, and using two

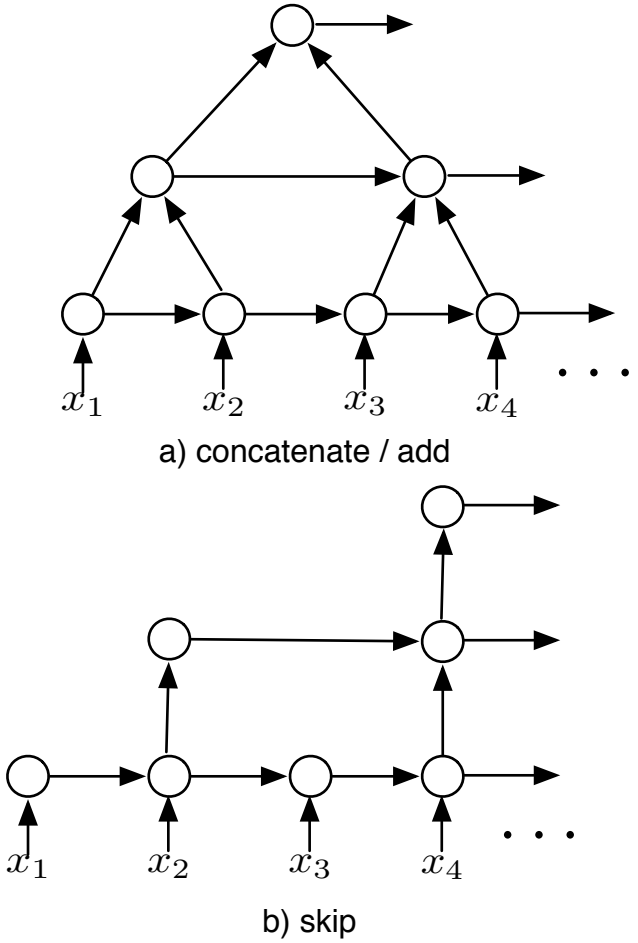

Figure 3.4: Hierarchical subsampling recurrent network (Graves, 2012a) . The size of the subsampling window is two in this example.

subsampling layers.

Hyperparameters We then evaluated the model by tuning the hyperparameters, and the results are given in Table 3.7. We tuned the number of LSTM layers, and the dimension of LSTM cells, as well as the dimensions of **w** and the segment vector **V**. In general, larger models with dropout regularization yielded higher recognition accuracy. Our best result was obtained using 6 layers of 250-dimensional

System	$d(\mathbf{w})$	$d(\mathbf{V})$	layers	hidden	PER(%)
`skip`	64	64	3	128	21.2
`conc`	64	64	3	128	21.3
`add`	64	64	3	128	23.2
`skip`	64	64	3	250	20.1
`conc`	64	64	3	250	20.5
`add`	64	64	3	250	21.5

Table 3.6: Results of hierarchical subsampling networks. $d(\mathbf{w})$ and $d(\mathbf{V})$ denote the dimension of $\mathbf{w}$ and $\mathbf{V}$ in Eqs. (3.4) respectively. `layers` denotes the number of LSTM layers and `hidden` is the dimension of the LSTM cells. `conc` is short for concatenating operation in the subsampling network.

LSTMs. However, without dropout, the model can be easily overfit due to the small size of training set.

Features We then evaluated another two types of features using the same system configuration that achieved the best result in Table 3.7. We increased the number of FBANKs from 24 to 40, which yielded slightly lower PER. We also evaluated the standard Kaldi features (Povey et al., 2011) — 13 dimensional MFCCs spliced by a context window of 7 and then two sets of "delta" functions are added, followed by LDA and MLLT transform and with feature-space speaker-dependent MLLR, which were the same features used in the HMM-DNN baseline in Table 3.9. The well-engineered features improved the accuracy of our system by more than 1% absolute.

Results In Table 3.9, we compare our result to other reported results using segmental CRFs as well as recent end-to-end systems. The previous state-of-the-art result using segmental CRFs on the TIMIT dataset is reported by Tang et al. (2015),

Table 3.7: Results of tuning the hyperparameters on the development set.

Dropout	$d(\mathbf{w})$	$d(\mathbf{V})$	layers	hidden	PER (%)
0.2	64	64	3	128	21.2
	64	32	3	128	21.6
	32	32	3	128	21.4
	64	64	3	250	20.1
	64	32	3	250	20.4
	32	32	3	250	20.6
	64	64	6	250	19.3
	64	32	6	250	20.2
	32	32	6	250	20.2
0.1	64	64	3	128	21.3
	64	64	3	250	20.9
	64	64	6	250	20.4
×	64	64	6	250	21.9

where the first-pass decoding was used to prune the search space, and the second-pass was used to re-score the hypothesis using various features including neural network features. The ground-truth segmentation was used in (Tang et al., 2015). We achieved considerably lower PER with first-pass decoding, despite the fact that our CRF was zeroth-order, and we did not use any language model. Furthermore, our results are also comparable to that from the CTC and attention-based RNN end-to-end systems. The accuracy of segmental RNNs may be further improved by using higher-order CRFs or incorporating a language model into the decode step, using beam search to reduce the search error. However, for the CTC system, Graves et al. (2013) obtained a slightly better result compared to ours (18.4% vs. 18.9% in Table 3.9). Apart from the implementation difference of using different code bases, Graves et al. (2013) applied the prefix decoding with beam

Table 3.8: Results of three types of acoustic features.

Features	Deltas	$d(\mathbf{x})$	PER (%)
24-dim FBANK	$\surd$	72	19.3
40-dim FBANK	$\surd$	120	18.9
Kaldi	$\times$	40	17.3

search, which may have lower search error than our best path decoding algorithm.

3.6 Related Work

Segmental labeling problems have been widely studied. A widely used approach to a segmental labeling problems with neural networks is the connectionist temporal classification (CTC) objective and decoding rule of (Graves et al., 2006b) discussed in § 3.4. CTC reduces the "segmental" sequence label problem to a classical sequence labeling problem in which every position in an input sequence $\boldsymbol{x}$ is explicitly labeled by interpreting repetitions of input labels—or input labels followed by a special "blank" output symbol—as being a single label with a longer duration. During training, the marginal likelihood of the set of labelings compatible (according to the CTC interpretation rules) with the reference label $\boldsymbol{y}$ is maximized. CTC has demonstrated impressive success in various fully discriminative end-to-end speech recognition models (Graves and Jaitly, 2014; Maas et al., 2015; Hannun et al., 2014, *inter alia*).

Although CTC has been used successfully and its reuse of conventional sequence labeling architectures is appealing, it has several potentially serious limitations. First, it is not possible to model interlabel dependencies explicitly—these must instead be captured indirectly by the underlying RNNs. Second, CTC has no explicit segmentation model. Although this is most serious in applications

System	LM	SD	PER
HMM-DNN	√	√	18.5
first-pass SCRF (Zweig, 2012)	√	×	33.1
Boundary-factored SCRF (He and Fosler-Lussier, 2012)	×	×	26.5
Deep Segmental NN (Abdel-Hamid et al., 2013)	√	×	21.9
Discriminative segmental cascade (Tang et al., 2015)	√	×	21.7
+ 2nd pass with various features	√	×	19.9
CTC (Graves et al., 2013)	×	×	18.4
RNN transducer (Graves et al., 2013)	-	×	17.7
Attention-based RNN baseline (Chorowski et al., 2015)	×	×	18.7
+Conv. Features + Smooth Focus (Chorowski et al., 2015)	×	×	17.6
Segmental RNN	×	×	18.9
Segmental RNN	×	√	17.3

Table 3.9: Comparison to Related Works. `LM` denote if the language model is used, and `SD` denotes feature space speaker-dependent transform. The HMM-DNN baseline was trained with cross-entropy using the Kaldi recipe. Sequence training did not improve it due to the small amount of data. RNN transducer can be viewed as a combination of the CTC network with a built-in RNN language model.

where segmentation is a necessary/desired output (e.g., information extraction, protein secondary structure prediction), we argue that explicit segmentation is potentially valuable even when the segmentation is *not* required. To illustrate the value of explicit segments, consider the problem of phone recognition. For this task, segmental duration is strongly correlated with label identity (e.g., while an [o] phone token might last 300ms, it is unlikely that a [t] would) and thus modeling it explicitly may be useful. Finally, making an explicit labeling decision for every position (and introducing a special blank symbol) in an input sequence is conceptually unappealing.

Several alternatives to CTC have been approached, such as using various attention mechanisms in place of marginalization (Chan et al., 2015; Bahdanau et al., 2015a). These have been applied to end-to-end discriminative speech recognition problem. A more direct alternative to our method—indeed it was proposed to solve several of the same problems we identified—is due to (Graves, 2012b). However, a crucial difference is that our model explicitly constructs representations of segments which are used to label the segment while that model relies on a marginalized frame-level labeling with a null symbol.

The work of (Abdel-Hamid, 2013) also seeks to construct embeddings of multi-frame segments. Their approach is quite different than the one taken here. First, they compute representations of variable-sized segments by uniformly sampling a fixed number of frames and using these to construct a representation of the segment with a simple feedforward network. Second, they do not consider them problem of latent segmentation.

Recently, in speech recognition, (Wang et al., 2017a) propose SWAN, which can be regarded as a generalization of CTC to allow segmented outputs. Another related work in machine translation is the online segment to segment neural transduction (Yu et al., 2017), where the model is able to capture unbounded dependencies in both the input and output sequences, while still permitting polynomial inference.

Finally, using neural networks to provide local features in conditional random field models has also been proposed for sequential models (Peng et al., 2009) and tree-structured models (Durrett and Klein, 2015). To our knowledge, this is the first application to semi-Markov structures.

3.7 Conclusion

We have proposed a new model for segment labeling problems that learns explicit representations of segments of an input sequence. Segment structures are represented as *segment embeddings* in our framework and used directly to predict the labels for them. Experiments show that, compared to models that do not explicitly represent segments such as BIO tagging schemes and connectionist temporal classification (CTC), SRNNs obtain substantially higher accuracies on tasks including phone recognition and handwriting recognition..

Chapter 4

A Transition-based Framework for Dynamically Connected Neural Networks

In this chapter, we propose dynamic recurrent acyclic graphical neural networks (DRAGNN)[1], a modular neural architecture that generalizes the encoder/decoder concept to include explicit linguistic structures[2].

Sequence-to-sequence model (Cho et al., 2014a) has been seen successful in many representation learning problems. However, in its simplest form, neither the encoder side nor the decoder side considers more sophisticated linguistic structures that can offer helpful inductive bias for the learning problems. The encoder just consumes the input tokens one by one and the decoder, even when used to predict structured output such as a phrase-structure parse tree, doesn't have the basic constraint that the parentheses should match explicitly (Vinyals et al., 2015).

In DRAGNN, we aim to add these useful linguistic driven inductive biases

[1] `https://github.com/tensorflow/models/tree/master/syntaxnet/dragnn`
[2] First published as Kong et al. (2017).

back to the model. The core mechanism of DRAGNN is the Transition-Based Recurrent Unit (TBRU, §4.3). The linguistic structure $\boldsymbol{z}$ is represented using a sequence of decision/state pairs $(s_1, d_1) \ldots (s_n, d_n)$. Through TBRU, linguistic structures can guide the building process of the neural networks by following the transitions and encoding the partial structure constructed by $(s_1, d_1) \ldots (s_j, d_j)$ explicitly into the hidden layer activations at timestep j. The final target $\boldsymbol{y}$ will be recovered using the mapping function $\boldsymbol{y} = \phi(\boldsymbol{z})$. By following the sequence of decision/state pairs, we can fully recover the final target output.

One advantage of this formulation is that a TBRU can serve as both an encoder for downstream tasks and as a decoder for its own task simultaneously (§4.3.1, example 6). This idea will prove particularly powerful when we consider syntactic parsing, which involves compositional structure over the input. For example, consider a "no-op" TBRU that traverses an input sequence $x_1, \ldots, x_n$ in the order determined by a binary parse tree: this transducer can implement a recursive tree-structured network in the style of (Tai et al., 2015), which computes representations for sub-phrases in the tree. In contrast, with DRAGNN, we can use the *arc-standard* parser directly to produce the parse tree as well as encode sub-phrases into representations.

Our framework can represent sequence-to-sequence learning (Cho et al., 2014b) as well as models with explicit structure like bi-directional tagging models (Wang et al., 2015) and compositional, tree-structured models (Socher et al., 2013). We show that our framework is significantly more accurate and efficient than seq2seq with attention for syntactic dependency parsing and yields more accurate multi-task learning for extractive summarization tasks.

4.1 Background

To apply deep learning models to structured prediction, machine learning practitioners must address two primary issues: (1) how to represent the input, and (2) how to represent the output. The *seq2seq* encoder/decoder framework (Kalchbrenner and Blunsom, 2013; Cho et al., 2014b; Sutskever et al., 2014) proposes solving these generically. In its simplest form, the *encoder* network produces a fixed-length vector representation of an input, while the *decoder* network produces a linearization of the target output structure as a sequence of output symbols. Encoder/decoder is state of the art for several key tasks in natural language processing, such as machine translation (Wu et al., 2016).

However, fixed-size encodings become less competitive when the input structure can be explicitly mapped to the output. In the simple case of predicting tags for individual tokens in a sentence, state-of-the-art taggers learn vector representations for each input *token* in context and predict output tags from those (Ling et al., 2015b; Andor et al., 2016). When the input or output is a syntactic parse tree, networks that explicitly operate over the compositional structure of the network typically outperform generic representations (Dyer et al., 2015; Li et al., 2015; Bowman et al., 2016). Implicitly learned mappings via attention mechanisms can significantly improve the performance of sequence-to-sequence (Bahdanau et al., 2015b; Vinyals et al., 2015), at a computational cost of $O(m \times n)$ where m is the length of the input sequence and n is the length of the output sequence.

In DRAGNN, we define any given architecture as a series of modular units, where connections between modules are unfolded *dynamically* as a function of the intermediate activations produced by the network. These dynamic connections represent the explicit input and output structure produced by the network for a given task.

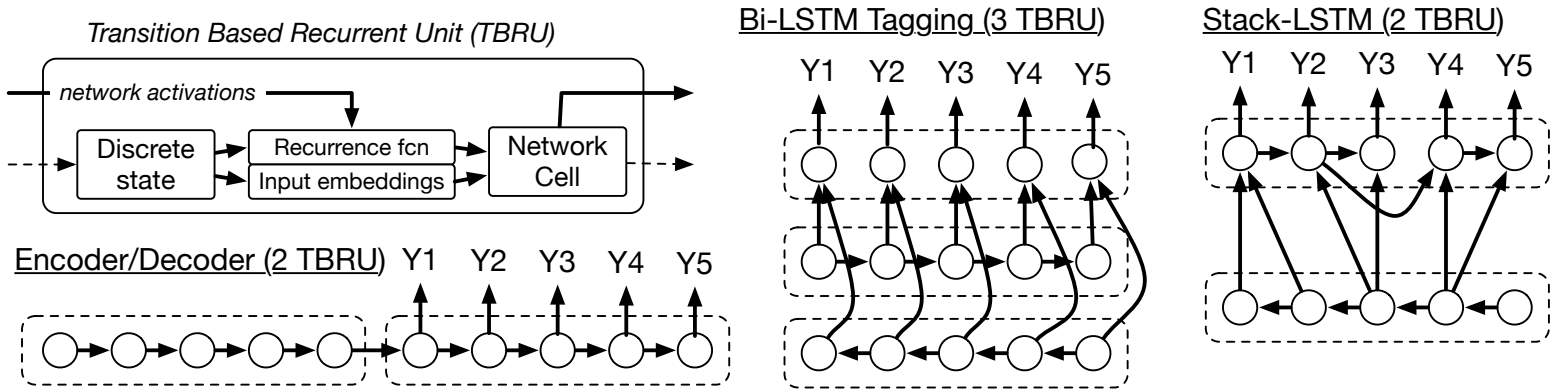

Figure 4.1: High-level schematic of a transition-based recurrent unit (TBRU), and familiar network architectures that can be implemented with multiple TBRUs. The discrete state is used to compute recurrences and fixed input embeddings, which are then fed through a network cell. The network predicts an action which is used to update the discrete state (dashed output) and provides activations that can be consumed through recurrences (solid output). Note that we present a slightly simplified version of Stack-LSTM (Dyer et al., 2015) for clarity.

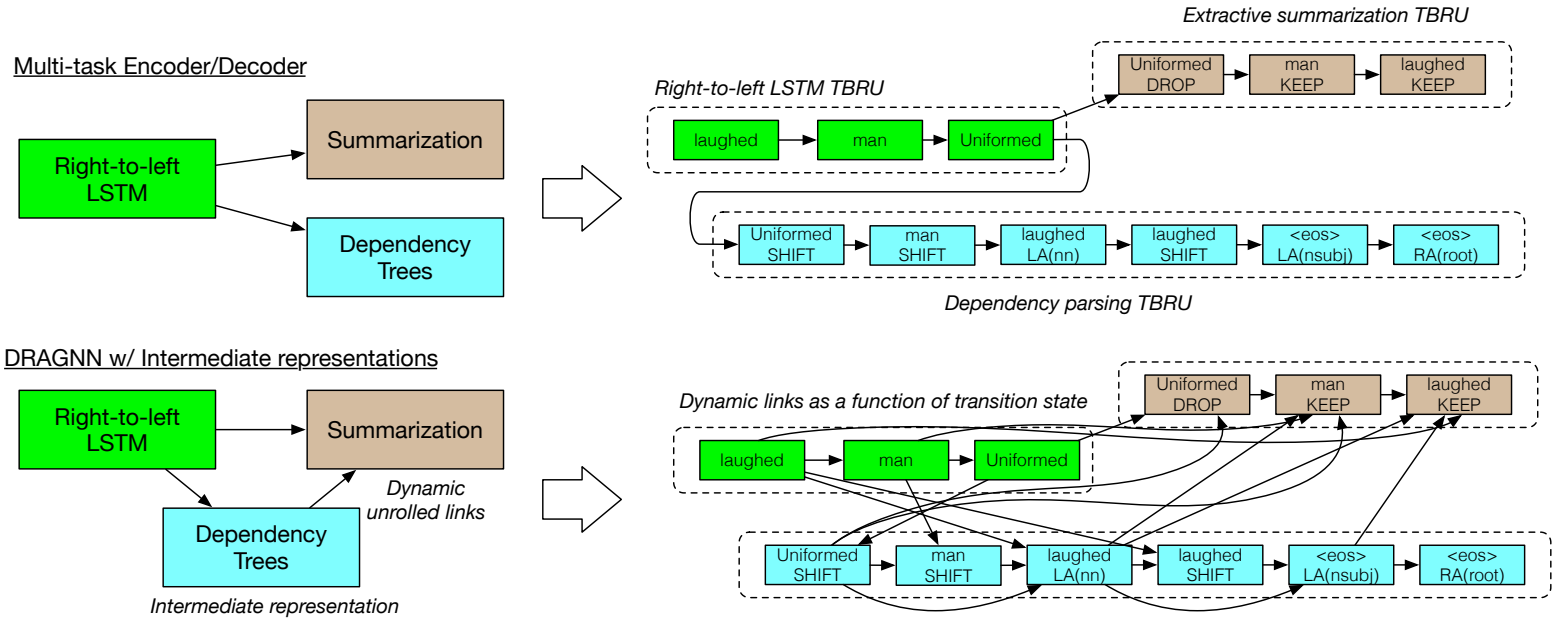

Figure 4.2: Using TBRUs to share fine-grained, structured representations. Top left: A high level view of multi-task learning with DRAGNN in the style of multi-task seq2seq (Luong et al., 2015). Bottom left: Extending the "stack-propagation" (Zhang and Weiss, 2016) idea to included dependency parse trees as intermediate representations. Right: Unrolled TBRUs for each setup for an input fragment "Uniformed man laughed", using the transition systems described in Section 4.4.

We build on the idea of *transition systems* from the parsing literature (Nivre, 2006), which linearize structured outputs as a sequence of (*state*, *decision*) pairs. Transition-based neural networks have recently been applied to a wide variety of NLP problems; (Dyer et al., 2015; Lample et al., 2016; Kiperwasser and Goldberg, 2016; Zhang et al., 2016; Andor et al., 2016), among others. We generalize these approaches with a new basic module, the Transition-Based Recurrent Unit (TBRU), which produces a vector representation for every transition state in the output linearization (Figure 4.1). These representations also serve as the encoding of the explicit structure defined by the states. For example, a TBRU that attaches two sub-trees while building a syntactic parse tree will also produce the hidden layer activations to serve as an encoding for the newly constructed phrase. Multiple TBRUs can be connected and learned jointly to add explicit structure to multi-task learning setups and share representations between tasks with different input or output spaces (Figure 4.2).

This inference procedure will construct an acyclic compute graph representing the network architecture, where recurrent connections are dynamically added as the network unfolds. We therefore call our approach Dynamic Recurrent Acyclic Graphical Neural Networks, or DRAGNN.

DRAGNN has several distinct modeling advantages over traditional fixed neural architectures. Unlike generic seq2seq, DRAGNN supports variable sized input representations that may contain explicit structure. Unlike purely sequential RNNs, the dynamic connections in a DRAGNN can span arbitrary distances in the input space. Crucially, inference remains $O(m + n)$, in contrast $O(m \times n)$ the attention mechanisms, where m is the length of the input sequence and n is the length of the output sequence.

Dynamic connections thus establish a compromise between pure seq2seq (Cho et al., 2014b) and pure attention architectures (Bahdanau et al., 2014) by provid-

ing a finite set of long-range inputs that 'attend' to relevant portions of the input space. Unlike recursive neural networks (Socher et al., 2010, 2011) DRAGNN can both predict intermediate structures (such as parse trees) and utilize those structures in a single deep model, backpropagating downstream task errors through the intermediate structures. Compared to models such as Stack-LSTM (Dyer et al., 2015) and SPINN (Bowman et al., 2016), TBRUs are a more general formulation that allows incorporating dynamically structured multi-task learning (Zhang and Weiss, 2016) and more varied network architectures.

In sum, DRAGNN is not a particular neural architecture, but rather a *formulation for describing neural architectures compactly.* The key to this compact description is a new recurrent unit—the TBRU—which allows connections between nodes in an unrolled compute graph to be specified dynamically in a generic fashion. We use transition systems to provide succinct, discrete representations via linearizations of both the input and the output for structured prediction. We provide a straightforward way of reusing representations across NLP tasks that operate on different structures.

DRAGNN is close related to the idea of "stack-propagation" (Zhang and Weiss, 2016). The key advantage of "stack-propagation" is that the model does not require predicted POS tags for parsing at test time. Instead, the tagger network is run up to the hidden layer over the entire sentence, and then dynamically connect the parser network to the tagger network based upon the discrete parser configurations as parsing unfolds. In this way, they can avoid cascading POS tagging errors to the parser. We generalize "stack-propagation" style pipeline that every task is represented by a TBRU in DRAGNN and can be easily used as intermediate representations.

We demonstrate the effectiveness of DRAGNN on two NLP tasks that benefit from explicit structure: dependency parsing and extractive sentence summariza-

tion (Filippova and Altun, 2013). First, we show how to use TBRUs to incrementally add structure to the input and output of a "vanilla" seq2seq dependency parsing model, dramatically boosting accuracy over seq2seq with no additional computational cost. Second, we demonstrate how the same TBRUs can be used to provide structured intermediate syntactic representations for extractive sentence summarization. This yields better accuracy than is possible with the generic multi-task seq2seq (Dong et al., 2015; Luong et al., 2015) approach. Finally, we show how multiple TBRUs for the same dependency parsing task can be stacked together to produce a single state-of-the-art dependency parsing model.

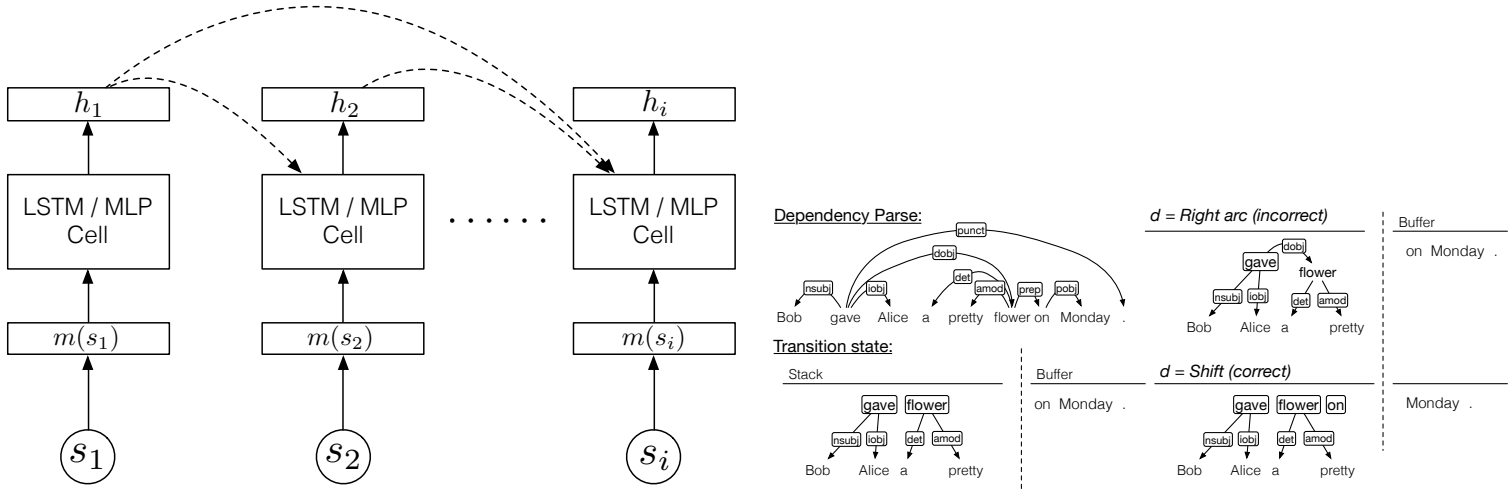

Figure 4.3: Left: TBRU schematic. Right: Dependency parsing example. A gold parse tree and an *arc-standard* transition state with two sub-trees on the stack are shown. From this state, two possible actions are also shown (*Shift* and *Right arc*). To agree with the gold tree, the *Shift* action should be taken.

4.2 Transition Systems

We use *transition systems* to map inputs $\boldsymbol{x}$ to outputs $\boldsymbol{y}$ using a sequence of transitions $d_1 \dots d_n$ which encode the structure $\boldsymbol{z}$. For the purposes of implementing DRAGNN, transition systems make explicit two desirable properties. First, we stipulate that the output symbols represent modifications of a persistent, discrete *state*, which makes book-keeping to construct the dynamic recurrent connections easier to express. Second, transition systems make it easy to enforce arbitrary constraints on the output, e.g. the output should produce a valid tree.

Formally, we use the same setup as (Andor et al., 2016), and define a transition system $\mathcal{T} = \{\mathcal{S}, \mathcal{A}, t\}$ as:

- A set of states $\mathcal{S}$.
- A special start state $s^\dagger \in \mathcal{S}$.
- A set of *allowed* decisions $\mathcal{A}(s, \boldsymbol{x})$ for all $s \in \mathcal{S}$.
- A transition function $t(s, d, \boldsymbol{x})$ returning a new state s' for any decision $d \in \mathcal{A}(s, \boldsymbol{x})$.

For brevity, we will drop the dependency on $\boldsymbol{x}$ in the functions given above. We will use transition systems in which all complete structures for the same input $\boldsymbol{x}$ have the same number of decisions $n(\boldsymbol{x})$ (or n for brevity), although this is not necessary.

A complete structure is then a sequence of decision/state pairs $(s_1, d_1) \dots (s_n, d_n)$ such that $s_1 = s^\dagger$, $d_i \in \mathcal{A}(s_i)$ for $i = 1 \dots n$, and $s_{i+1} = t(s_i, d_i)$. We will now define recurrent network architectures that operate over these linearizations of input and output structure.

4.3 Transition Based Recurrent Networks

We now formally define how to combine transition systems with recurrent networks into what we call a *transition based recurrent unit* (TBRU). A TBRU consists of the following:

- A transition system $\mathcal{T}$,
- An input function $\mathbf{m}(s)$ that maps states[3] to fixed-size vector representations, for example, an embedding lookup operation for features from the discrete state, $\mathbf{m}(s) : \mathcal{S} \mapsto \mathbb{R}^K$
- A recurrence function $\mathbf{r}(s)$ that maps states to a set of previous time steps:

  $$\mathbf{r}(s) : \mathcal{S} \mapsto \mathbb{P}\{1, \ldots, i-1\},$$

  where $\mathbb{P}$ is the power set. Note that in general $|\mathbf{r}(s)|$ is not necessarily fixed and can vary with s. We use $\mathbf{r}$ to specify state-dependent recurrent links in the unrolled computation graph.
- A neural network cell that computes a new hidden representation from the fixed and recurrent inputs:

  $$\mathbf{h}_s \leftarrow \mathbf{NN}(\mathbf{m}(s), \{\mathbf{h}_i \mid i \in \mathbf{r}(s)\}).$$

Example 1. Sequential tagging RNN. Let the input $\boldsymbol{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be a sequence of word embeddings, and the output be a sequence of tags $\boldsymbol{y} = \{y_1, \ldots, y_n\}$. Here the linguistic structure $\boldsymbol{z}$ is simply the same as $\boldsymbol{y}$ and can be easily mapped to transitions $\{d_1, \ldots, d_n\}$ where d_i means to tag token x_i as d_i. We can model a simple LSTM tagger as follows:

- $\mathcal{T}$ sequentially tags each input token, where $s_i = \{1, \ldots, d_{i-1}\}$, i.e., the whole history of past tagging decisions, and $\mathcal{A}$ is the set of possible tags. We call this the *tagger* transition system.

[3]For simplicity, we assume the states contain the information from the input $\boldsymbol{x}$.

Features	Window	Dimensionality
Symbols	1	8
Capitalization	+/- 1	4
Prefixes/Suffixes ($n = 2, 3$)	+/- 1	16
Words	+/- 3	64

Table 4.1: Window-based tagger feature spaces. "Symbols" indicates whether the word contains a hyphen, a digit or a punctuation

- $\mathbf{m}(s_i) = \mathbf{x}_i$, the word embedding for the next token to be tagged.
- $\mathbf{r}(s_i) = \{i - 1\}$ to connect the network to the previous state.
- **NN** is a single instance of the LSTM cell.

Example 2. Parsey McParseface. In the open-source syntactic parsing model (Andor et al., 2016), the input $\boldsymbol{x}$ is a sequence of words. The output $\boldsymbol{y}$, same as the underlying linguistic structure $\boldsymbol{z}$, is the dependency tree. It will be represented by a sequence of transitions $\{d_1, \ldots, d_n\}$ following the *arc-standard* transition system. Parsey McParseface can be defined in our framework as follows:

- $\mathcal{T}$ is the *arc-standard* transition system (Figure 4.3), so the state contains all words and partially built trees on the stack as well as unseen words on the buffer.
- $\mathbf{m}(s_i)$ is the concatenation of 52 feature embeddings extracted from tokens based on their positions in the stack and the buffer. Table 4.1 show the details of these window-based features.
- $\mathbf{r}(s_i) = \{\}$ is empty, as this is a feed-forward network.
- **NN** is a feed-forward multi-layer perceptron (MLP).

Inference with TBRUs. Given the above, inference in the TBRU proceeds as follows:

1. Initialize $s_1 = s^{\dagger}$.
2. For $i = 1, \ldots, n$:
   (a) Update the hidden state:
   $\mathbf{h}_i \leftarrow \mathbf{NN}(\mathbf{m}(s_i), \{\mathbf{h}_j \mid j \in \mathbf{r}(s_i)\})$.
   (b) Update the transition state:
   $d_i \leftarrow \arg\max_{d \in \mathcal{A}(s_i)} \mathbf{w}_d^{\top} \mathbf{h}_i, \quad s_{i+1} \leftarrow t(s_i, d_i)$.

Intuitively, this is just saying, when we don't have the gold transitions $\{d_1, \ldots, d_n\}$, w e make a prediction at every time stamp and update the state and the RNN based on the prediction.

A schematic overview of a single TBRU is presented in Figure 4.3. By adjusting **NN**, **r**, and $\mathcal{T}$, TBRUs can represent a wide variety of neural architectures.

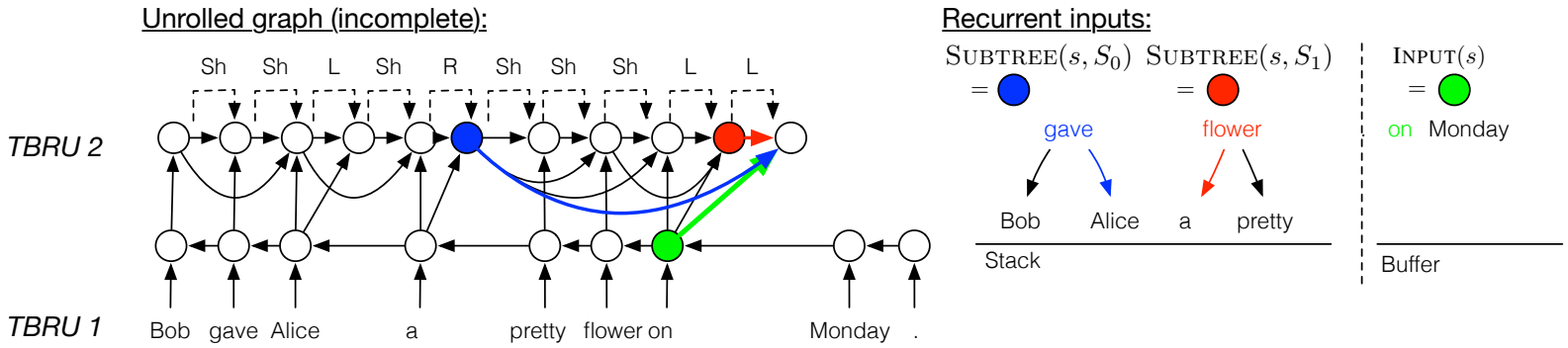

Figure 4.4: Detailed schematic for the compositional dependency parser used in our experiments. *TBRU 1* consumes each input word right-to-left. *TBRU 2* uses the *arc-standard* transition system. Note how each *Shift* action causes the *TBRU 1*→*TBRU 2* link to advance. The dynamic recurrent inputs to each state are highlighted; the stack representations are obtained from the last *Reduce* action to modify each sub-tree.

4.3.1 Connecting multiple TBRUs to learn shared representations

While TBRUs are a useful abstraction for describing recurrent models, the primary motivation for this framework is to allow new architectures by combining representations across tasks and compositional structures. We do this by connecting multiple TBRUs with different transition systems via the recurrence function $\mathbf{r}(s)$. We formally augment the above definition as follows:

1. We execute a list of T TBRUs, one at a time, so that each TBRU advances a global step counter. Note that for simplicity, we assume an earlier TBRU finishes all of its steps before the next one starts execution.
2. Each transition state from the τ'th component s^τ has access to the terminal states from every prior transition system, and the recurrence function $\mathbf{r}(s^\tau)$ for any given component can pull hidden activations from every prior one as well.

Example 3. "Input" transducer TBRUs via no-op decisions. We find it useful to define TBRUs even when the transition system decisions don't correspond to any output. These TBRUs, which we call *no-op* TBRUs, transduce the input according to some linearization. The simplest is the *shift-only* transition system, in which the state is just an input pointer $s_i = \{i\}$, and there is only one transition which advances it: $t(s_i, \cdot) = \{i + 1\}$. Executing this transition system will produce a hidden representation $\mathbf{h}_i$ for every input token.

Example 4. Encoder/decoder networks with TBRUs. We can reproduce the encoder/decoder framework for sequence tagging by using two TBRUs: one using the *shift-only* transition system to encode the input, and the other using the *tagger* transition system. For input $x = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, we connect them as follows:

- For *shift-only* TBRU: $\mathbf{m}(s_i) = \mathbf{x}_i$, $\mathbf{r}(s_i) = \{i - 1\}$.

- For *tagger* TBRU: $\mathbf{m}(s_{n+i}) = \mathbf{y}_{d_{n+i-1}}$, $\mathbf{r}(s_i) = \{n, n+i-1\}$.

We observe that the *tagger* TBRU starts at step n after the *shift-only* TBRU finishes, that $\mathbf{y}_j$ is a fixed embedding vector for the output tag j, and that the *tagger* TBRU has access to both the final encoding vector $\mathbf{h}_n$ as well as its own previous timestep $\mathbf{h}_{n+i-1}$.

Example 4. Bi-directional LSTM tagger. With three TBRUs, we can implement a bi-directional tagger. The first two run the *shift-only* transition system, but in opposite directions. The final TBRU runs the *tagger* transition system and concatenates the two representations:

- Left to right: $\mathcal{T} = \textit{shift-only}$, $\mathbf{m}(s_i) = \mathbf{x}_i$, $\mathbf{r}(s_i) = \{i-1\}$.
- Right to left: $\mathcal{T} = \textit{shift-only}$, $\mathbf{m}(s_{n+i}) = \mathbf{x}_{n-i}$, $\mathbf{r}(s_{n+i}) = \{n+i-1\}$.
- Tagger: $\mathcal{T} = \textit{tagger}$, $\mathbf{m}(s_{2n+i}) = \{\}$, $\mathbf{r}(s_{2n+i}) = \{i, 2n-i\}$.

We observe that the network cell in the tagger TBRU takes recurrences only from the bi-directional representations, and so is not recurrent in the traditional sense. See Fig. 4.1 for an unrolled example.

Example 5. Multi-task bi-directional tagging. Here we observe that it is possible to add additional annotation tasks to the bi-directional TBRU stack from Example 4 by adding more instances of the tagger TBRUs that produce outputs from different tag sets, e.g. parts-of-speech vs. morphological tags. Most important, however, is that any additional TBRUs have access to *all three* earlier TBRUs. This means that we can support the "stack-propagation" (Zhang and Weiss, 2016) style of multi-task learning simply by changing $\mathbf{r}$ for the last TBRU:

- Traditional multi-task (Luong et al., 2015; Søgaard and Goldberg, 2016):

$$\mathbf{r}(s_{3n+i}) = \{i, 2n-i\}$$

Parsing TBRU recurrence, $\mathbf{r}(s_i) \subseteq \{1, \ldots, n+i\}$		Parsing Accuracy (%)		
Input links	Recurrent edges	News	Questions	Runtime
$\{n\}$	$\{n+i-1\}$	27.3	70.1	$O(n)$
$\{n\}$	$\{\text{SUBTREE}(s_i, S_0), \text{SUBTREE}(s_i, S_1)\}$	36.0	75.6	$O(n)$
Attention	$\{n+i-1\}$	76.1	84.8	$O(n^2)$
Attention	$\{\text{SUBTREE}(s_i, S_0), \text{SUBTREE}(s_i, S_1)\}$	89.0	91.9	$O(n^2)$
$\text{INPUT}(s_i)$	$\{n+i-1\}$	87.1	89.7	$O(n)$
$\text{INPUT}(s_i)$	$\{\text{SUBTREE}(s_i, S_0), \text{SUBTREE}(s_i, S_1)\}$	**90.9**	**92.1**	$O(n)$

Table 4.2: Dynamic links enable much more accurate, efficient linear-time parsing models on the Treebank Union development set. We vary the recurrences $\mathbf{r}$ to explore using explicit structure in the parsing TBRU. Using the explicit $\text{INPUT}(s_i)$ pointer is more effective and more efficient than a quadratic attention mechanism. Incorporating the explicit stack structure via recurrent links further improves performance.

- Stack-prop:

$$\mathbf{r}(s_{3n+i}) = \{ \underbrace{i}_{\text{Left-to-right}} , \underbrace{2n-i}_{\text{Right-to-left}} , \underbrace{2n+i}_{\text{Tagger TBRU}} \}$$

Example 6. Compositional representations from *arc-standard* dependency parsing. We use the *arc-standard* transition system (Nivre, 2006) to model dependency trees. The system maintains two data structures as part of the state s: an input pointer and a stack (Figure 4.3). Trees are built bottom up via three possible attachment decisions. Assume that the stack consists of $S = \{A, B\}$, with the next token being C. We use S_0 and S_1 to refer to the top two tokens on the stack. Then the decisions are defined as:

- Shift: Push the next token on to the stack: $S = \{A, B, C\}$, and advance the input pointer.

- Left arc + *label*: Add an arc $A \leftarrow_{label} B$, and remove A from the stack: $S = \{B\}$.
- Right arc + *label*: Add an arc $A \rightarrow_{label} B$, and remove B from the stack: $S = \{A\}$.

For a given parser state s_i, we compute two types of recurrences:

- $\mathbf{r}_{\text{INPUT}}(s_i) = \{\text{INPUT}(s_i)\}$, where INPUT returns the index of the next input token.
- $\mathbf{r}_{\text{STACK}}(s_i) = \{\text{SUBTREE}(s_i, S_0), \text{SUBTREE}(s, S_1)\}$, where SUBTREE(S,I) is a function returning the index of the last decision that modified the i'th token:

$$\text{SUBTREE}(s, i) = \arg\max_j \{d_j \text{ s.t. } d_j \text{ shifts or adds a newchild to token } i\}$$

We show an example of the links constructed by these recurrences in Figure 4.4, and we investigate variants of this model in Section 4.4. This model is recursively compositional according to the decision taken by the network: when the TBRU at step s_i decides to add an arc $A \rightarrow B$ for state, the activations $\mathbf{h}_i$ will be used to represent that new subtree in future decisions.[4] The links we chose here are inspired by standard transitional based parsers (Nivre, 2003). The key advantage of our model is that it uses subtree representations on the stack.

Example 7. Extractive summarization pipeline with parse representations. To model extractive summarization, we follow (Andor et al., 2016) and use a *tagger* transition system with two tags: *Keep* and *Drop*. However, whereas (Andor et al., 2016) use discrete features of the parse tree, we can use the SUBTREE recurrence function to pull compositional, phrase-based representations of tokens as

[4]This composition function is similar to that in the constituent parsing SPINN model (Bowman et al., 2016), but with several key differences. Since we use TBRUs, we compose new representations for "Shift" actions as well as reductions, we take inputs from other recurrent models, and we can use subtree representations in downstream tasks.

Input representation	Multi-task style	A (%)	F1 (%)	LAS (%)
Single LSTM	–	28.93	79.75	–
Bi-LSTM	–	29.51	80.03	–
Multi-task LSTM	(Luong et al., 2015)	30.07	80.31	89.42
Parse sub-trees (Figure 4.2)	(Zhang and Weiss, 2016)	**30.56**	**80.74**	89.13

Table 4.3: Single- vs. multi-task learning with DRAGNN on extractive summarization. "A" is full-sentence accuracy of the extraction model, "F1" is per-token F1 score, and "LAS" is labeled parsing accuracy on the Treebank Union News dev set. Both multi-task models that use the parsing data outperform the single-task approach, but the model that uses parses as an intermediate representation via our extension of (Zhang and Weiss, 2016) (Fig. 4.2) is more effective. The locally normalized model in (Andor et al., 2016) obtains 30.50% accuracy and 78.72% F1.

constructed by the dependency parser. This model is outlined in Fig. 4.2.

4.3.2 How to Train a DRAGNN

Given a list of TBRUs, we propose the following learning procedure. We assume training data consists of examples $\boldsymbol{x}$ along with gold decision sequences $d_1 \dots d_M$ to construct the structure $\boldsymbol{z}$. The function $\psi(\boldsymbol{z})$ will map $\boldsymbol{z}$ to the target output $\boldsymbol{y}$.

$$\mathcal{L} = -\log p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) \tag{4.1}$$

$$= -\sum_{i=1}^{M} \log p(\psi(\boldsymbol{z}), \boldsymbol{z} \mid \boldsymbol{x}), \tag{4.2}$$

$$= -\sum_{i=1}^{M} \log p(d_i \mid \boldsymbol{x}) \tag{4.3}$$

Eq. (4.1) is locally normalized (Andor et al., 2016), since we optimize the probabilities of the individual decisions in the gold sequence.

A slight variant of this objective is assuming we only have such data for the final TBRU. Given decisions $d_1 \dots d_N$ from prior components $1 \dots T-1$, we define

a log-likelihood objective to train the Tth TBRU along its gold decision sequence to construct z_N, i.e., $d_{N+1}, \ldots, d_{N+n}$:

$$\mathcal{L} = -\log p(\boldsymbol{y}_N, z_N \mid z_{1:N}, \boldsymbol{x}) \tag{4.4}$$

$$= -\sum_{i=1}^{n} \log p(\psi(z_N), z \mid z_{1:N}, \boldsymbol{x}), \tag{4.5}$$

$$= -\sum_{i=1}^{n} \log p(d_{N+i} \mid d_{1:N}, d_{N+1:N+i-1}, \boldsymbol{x}) \tag{4.6}$$

The remaining question is where the decisions $d_1 \ldots d_N$ come from. There are two options here: either 1) they come as part of the gold annotation (e.g., if we have joint tagging and parsing data), or 2) they are predicted by unrolling the previous components. When training the stacked extractive summarization model, the parse trees will be predicted by the previously trained parser TBRU.

When training a given TBRU, we unroll an entire input sequence and then use backpropagation through structure (Goller and Kuchler, 1996) to optimize Equation (4.1). To train the whole system on a set of C datasets, we use a strategy similar to Dong et al. (2015) and Luong et al. (2015). At each iteration, we sample a target task $c, 1 \leq c \leq C$, based on a pre-defined distribution, and take a stochastic optimization step on the objective of task c's TBRU. In practice, task sampling is usually preceded by a deterministic number of pre-training steps, allowing, for example, to run a certain number of tagger training steps before running any parser training steps.

4.4 Experiments

In this section, we evaluate three aspects of our approach on two NLP tasks: English dependency parsing and extractive sentence summarization. For English dependency parsing, we primarily use the Union Treebank setup from (Andor

Model	Union-News			Union-Web			Union-QTB		
	UAS	LAS	POS	UAS	LAS	POS	UAS	LAS	POS
Andor et al. (2016)	94.44	92.93	97.77	90.17	87.54	94.80	95.40	93.64	96.86
Left-to-right parsing	94.60	93.17	97.88	90.09	87.50	94.75	95.62	94.06	96.76
Deep stacked parsing	**94.66**	**93.23**	**98.09**	**90.22**	**87.67**	**95.06**	**96.05**	**94.51**	**97.25**

Table 4.4: Deep stacked parsing compared to state-of-the-art on Treebank Union for parsing and POS.

et al., 2016). By evaluating on both news and questions domains, we can separately evaluate how the model handles naturally longer and shorter form text. On the Union Treebank setup there are 93 possible actions considering all arc-label combinations. For extractive sentence summarization, we use the dataset of (Filippova and Altun, 2013), where a large news collection is used to heuristically generate compression instances. The final corpus contains about 2.3M compression instances, but since we evaluated multiple tasks using this data, we subsampled the training set to be comparably sized to the parsing data (≈60K training sentences). The test set contains 160K examples. We implement our method in TensorFlow, using mini-batches of size 4 and following the averaged momentum training and hyperparameter tuning procedure of (Weiss et al., 2015).

Using explicit structure improves encoder/decoder We explore the impact of different types of recurrences on dependency parsing in Table 4.2. In this setup, we used relatively small models: single-layer LSTMs with 256 hidden units, taking 32-dimensional word or output symbol embeddings as input to each cell. In each case, the parsing TBRU takes input from a right-to-left *shift-only* TBRU. Under these settings, the pure encoder/decoder seq2seq model simply does not have the capacity to parse newswire text with any degree of accuracy, but the TBRU-

based approach is nearly state-of-the-art *at the same exact computational cost*. As a point of comparison and an alternative to using input pointers, we also implemented an attention mechanism within DRAGNN. We used the dot-product formulation from (Parikh et al., 2016a), where $\mathbf{r}(s_i)$ in the parser takes in all of the *shift-only* TBRU's hidden states and **NN** aggregates over them. Using the $\text{INPUT}(s_i)$ pointer is more effective and more efficient than a quadratic attention mechanism. It also outperforms the vanilla seq2seq solution, which is equivalent of fixing the input links at n. We also vary the recurrences $\mathbf{r}$ to explore using explicit structure in the parsing TBRU. Incorporating the explicit stack structure via recurrent links improves performance. This shows that the information encoded in the hidden states representing the subtrees is helpful for resolving ambiguities in parsing.

Utilizing parse representations improves summarization We evaluate our approach on the summarization task in Table 4.3. We compare two single-task LSTM tagging baselines against two multi-task approaches: an adaptation of (Luong et al., 2015) and the stack-propagation idea of (Zhang and Weiss, 2016). In both multi-task setups, we use a right-to-left *shift-only* TBRU to encode the input, and connect it to both our compositional *arc-standard* dependency parser and the *Keep/Drop* summarization tagging model.

In both setups we do not follow seq2seq, but use the INPUT function to connect output decisions directly to input token representations. However, in the stack-prop case, we use the SUBTREE function to connect the tagging TBRU to the parser TBRU's phrase representations directly (Figure 4.2). We find that allowing the compressor to directly use the parser's phrase representations significantly improves the outcome of the multi-task learning setup. In both setups, we pretrained the parsing model for 400K steps and tuned the subsequent ratio of parser/tagger update steps using a development set.

Deep stacked bi-directional parsing Here we propose a continuous version of the bi-directional parsing model of (Attardi and Dell'Orletta, 2009): first, the sentence is parsed in the left-to-right order as usual; then a right-to-left transition system analyzes the sentence in reverse order using additional features extracted from the left-to-right parser. In our version, we connect the right-to-left parsing TBRU directly to the phrase representations of the left-to-right parsing TBRU, again using the SUBTREE function. Our parser has the significant advantage that the two directions of parsing can affect each other during training. During each training step the right-to-left parser uses representations obtained using the *predictions* of the left-to-right parser. Thus, the right-to-left parser can backpropagate error signals through the left-to-right parser and reduce cascading errors caused by the pipeline.

Our final model uses 5 TBRU units. Inspired by (Zhang and Weiss, 2016), a left-to-right POS tagging TBRU provides the first layer of representations. Next, two *shift-only* TBRUs, one in each direction, provide representations to the parsers. Finally, we connect the left-to-right parser to the right-to-left parser using links defined via the SUBTREE function. The result (Table 4.4) is a state-of-the-art dependency parser, yielding the highest published accuracy on the Treebank Union setup for both part of speech tagging and parsing.

4.5 Conclusion

We presented a compact, modular framework for describing recurrent neural architectures. We evaluated our dynamically structured model and found it to be significantly more efficient and accurate than attention mechanisms for dependency parsing and extractive sentence summarization in both single- and multi-task setups. While we focused primarily on syntactic parsing, the framework

provides a general means of sharing representations between structured prediction tasks. There remains space to be explored: in particular, our approach can be globally normalized with multiple hypotheses in the intermediate structure. We also plan to push the limits of multi-task learning by combining many different NLP tasks, such as translation, summarization, tagging problems, and reasoning tasks, into a single model.

Chapter 5

Stochastic Attention and Posterior Regularization for Neural Machine Translation

Given the existence of abundant parallel data, neural machine translation (NMT) has been established as state-of-the-art approaches in machine translation, when compared to rule-based and statistical machine translation (SMT) systems, particularly in the case of human evaluation (Wu et al., 2016; Klein et al., 2017; Vaswani et al., 2017). The best performing models are built on the sequence-to-sequence models (Cho et al., 2014a) and use the attention mechanism to connect the decoder with the encoder side (Bahdanau et al., 2014). Recent work (Vaswani et al., 2017) argues that the attention mechanism is a crucial part in neural machine translation systems.

Attention mechanism is built on the intuition of adding the linguistically plausible (soft-)alignment structures between a source sentence and the corresponding target sentence, as a form of inductive bias into the original pure sequence-to-sequence models (Bahdanau et al., 2014). These alignment structures also play

an essential part in the statistical machine translation (SMT) systems. The main difference between the NMT and the SMT approach, is that the SMT system models these alignments information using random variables, while the NMT system models them using a deterministic function of the input.

In this chapter, we further strengthen the inductive biases introduced by the alignment structures in two ways. First, we explicitly model the alignment decisions in the probabilistic framework by replacing the soft weighting scheme with a hard (but stochastic) decision. Second, we regularize the posterior distributions of the latent alignment decisions using the posteriors computed from IBM model 2 (Brown et al., 1993; Dyer et al., 2013). This alleviates the problem that neural networks can fit noisy data or explain away stochastic latent variables, because they are general-purpose function approximators (Chen et al., 2017; Zhang et al., 2017).

5.1 Background

In sequence transduction tasks (e.g. machine translation), a conditional distribution over sequences $\boldsymbol{y}$ given a sequence $\boldsymbol{x}$ is modeled, and parameters are learned by maximizing $\log p(\boldsymbol{y} \mid \boldsymbol{x})$ from a set of $(\boldsymbol{x}, \boldsymbol{y})$ pairs. A key innovation that has made neural autoregressive models effective for this task is *attention*, due to Bahdanau et al. (2015a). Attention enables different parts of the input to be attended selectively as the output sequence is generated. In most models, attention is a weighting of all input positions, that is computed deterministically as a function of the decoder RNN's hidden state. The attended view of the input is a weighted combination of the input representations.

In this chapter, we use a hard stochastic decision in place of the standard soft weighting scheme, focusing on the problem of translation (§5.2). In contrast to

soft weighting over all positions of the input, we sample a single position from a distribution over positions, and then, conditional on the chosen input (as well as the RNN's summary of the generation history), generate the next output symbol. Our model is thus a joint model $p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$, where each $\boldsymbol{z}$ is a sequence of alignment decisions, one for each output symbol, similar to traditional statistical translation models introduced by Brown et al. (1993). This captures our *a priori* expectation that translation is mostly a word-to-word translation process. However, since the representations attended to are computed via a bi-directional RNN encoding, contextual information still can inform the lexical translation decisions.

By marginalizing the latent alignments, the log (marginal) likelihood is differentiable, making end-to-end learning feasible. However, learning with stochastic latent variables in neural networks is nontrivial, since the required posterior distributions are generally intractable to compute. Usually, any of a variety of approximate inference techniques are used (Kingma and Welling, 2014; Bornschein and Bengio, 2015; Burda et al., 2016; Miao and Blunsom, 2016; Maddison et al., 2017). We first compare several strategies for marginalizing the latent alignments. Furthermore, to understand the effect of different inference algorithms, we also make a conditional independence assumption that allows us to tractably marginalize the alignments analytically. Thus, we can compare the effects of different approximation algorithms on the quality of the model learned against a gold standard inference technique and evaluate them objectively in terms of marginal likelihood. Our experiments find that stochastic attention, by itself, offers little benefit over classical deterministic soft attention, and worse, that several standard approximation algorithms can harm performance.

We then improve the generalization of our model by regularizing the posterior distributions over the latent variables during learning. We are motivated by the observation that neural networks are general-purpose function approxima-

tors that can fit noisy data or, when stochastic units are present, explain away stochastic latent variables (Chen et al., 2017; Zhang et al., 2017). Indeed, we find evidence that this is happening in our model. Our regularization strategy is inspired by posterior regularization (Ganchev et al., 2010), but rather than imposing declarative moment constraints on the posteriors, we regularize the posterior distributions over the latent variables to be similar to posterior distributions obtained from models that make stronger independence assumptions and therefore use their latent variables (rather than the deterministic autoregressive component) to explain the outputs. In those models with stronger independence assumptions, it is easier to learn sensible alignment structures with less data. Our approach combines this strength with expensiveness of the neural networks to offer a better performance.

We propose both an exact training objective based on analytical marginals, and several approximations based on importance sampling that are suitable when exact marginals cannot be computed. Experimentally, we show (1) that our posterior regularization scheme leads to substantially improved generalization, (2) that a similar penalty imposed on the deterministic model underperforms the stochastic model, and (3) that an approximation based on importance sampling (which is therefore suitable for problems with intractable marginals) is either as good as or better than the exact objective in terms of held-out generalization.

5.2 Model

Our model, illustrated in Figure 5.2, is based on a standard attention-based sequence to sequence model (Bahdanau et al., 2015a), illustrated in Figure 5.1. Given a source sentence $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)$, the goal is to predict a target sentence $\boldsymbol{y} = (y_1, y_2, \ldots, y_M)$. We are using a form of attention termed "late fusion" by Wang

and Cho (2015), wherein the decoder attends to the source representation based on the current RNN state, and makes a joint prediction of the next word based on the vector returned by attention, and the current RNN state.

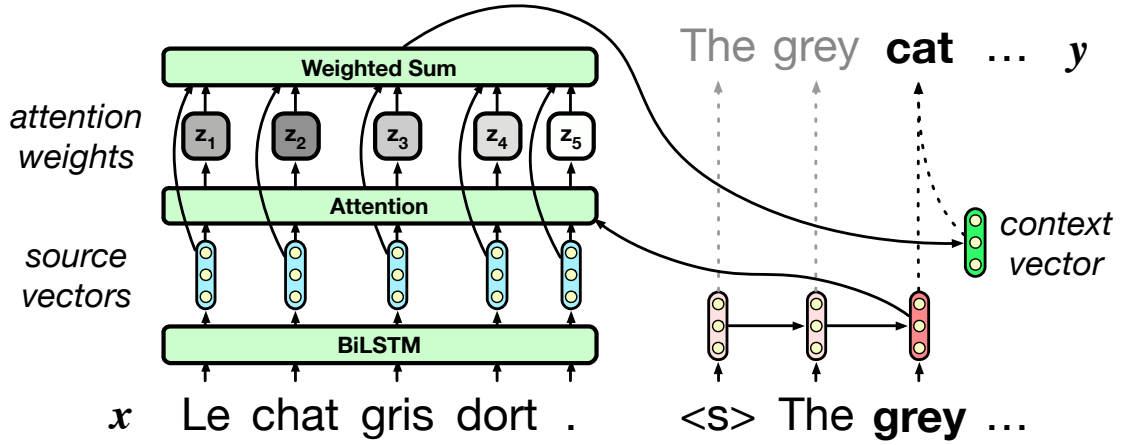

Figure 5.1: Sequence to sequence model with deterministic soft attention

Encoder. The encoder embeds each source word into a vector $\mathbf{x}_i \in \mathbb{R}^D$ and encode the source sentence with a bi-directional long short-term memory network (Hochreiter and Schmidhuber, 1997b). We take the representation of the i-th source word as the concatenation of forward and reverse LSTM states ($\overrightarrow{\mathbf{h}}_i$ and $\overleftarrow{\mathbf{h}}_i$ respectively), and designate this as $\mathbf{h}_i \in \mathbb{R}^{2H}$.

Decoder. The decoder is a uni-directional (forward) LSTM that takes the final state of the encoder's forward LSTM $\overrightarrow{\mathbf{h}}_N$ as its initial state. The hidden state at output timestep t is designated with $\mathbf{g}_t$. $\mathbf{g}_t$ is computed from the previous hidden state $\mathbf{g}_{t-1}$ and the input vector $\mathbf{x}_t$.

Deterministic Soft Attention. For the attention mechanism, we take an inner product between $\mathbf{g}_t$ and each source word encoding, exponentiate and normalize

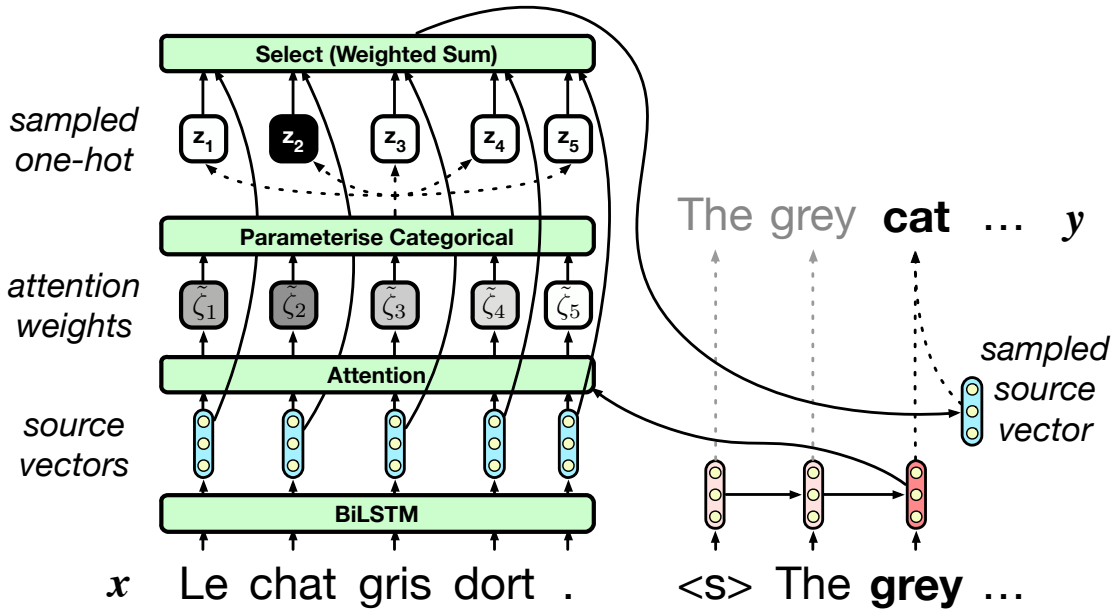

Figure 5.2: Sequence to sequence model with hard stochastic attention

it to obtain $\tilde{\boldsymbol{\zeta}}_t$ (the attention weighting over the source positions at time t (Parikh et al., 2016b)). We then compute the *context vector* as follows:

$$\mathbf{c}_t = \sum_{i=1}^{N} \tilde{\zeta}_{t,i} \mathbf{h}_i.$$

We can see here that if $\tilde{\boldsymbol{\zeta}}_t$ is treated as the parameters of a categorical distribution $\text{Categorical}(z_t; \tilde{\boldsymbol{\zeta}}_t)$, soft alignment is in fact computing the expectation of the representation according to the distribution:

$$\mathbf{c}_t = \mathbb{E}_{\text{Categorical}(z_t;\tilde{\boldsymbol{\zeta}}_t)} \left[\mathbf{h}_{a_t} \right].$$

What makes this form of attention "late fusion" is that our definition of $\mathbf{c}_t$ is defined as a function of $\mathbf{g}_t$ rather than in terms of $\mathbf{g}_{t-1}$, as done in "early fusion".

Stochastic Hard Attention. As an alternative to the deterministic computation of $\mathbf{c}_t$, we can also define this stochastically. To do so we compute the weighting

vector $\tilde{\boldsymbol{\zeta}}_t$ as above, but we instead treat it as the parameter of a categorical distribution and sample a position in the source sentence represented as a one-hot encoding; $\mathbf{c}_t$ is the defined as the source encoding indexed by this sampled position:

$$z_t \sim \text{Categorical}(z_t; \tilde{\boldsymbol{\zeta}}_t) \tag{5.1}$$

$$\mathbf{c}_t = \mathbf{h}_{z_t}. \tag{5.2}$$

Word generator. Given our context vector $\mathbf{c}_t$, which contains the result of the attention (computed either stochastically or deterministically), and $\mathbf{g}_t$, the token y_t is generated as follows:

$$\mathbf{p}_t = \text{softmax}(\mathbf{W}[\mathbf{g}_t; \mathbf{c}_t] + \mathbf{b})$$
$$y_t \sim \text{Categorical}(\mathbf{p}_t).$$

5.2.1 Marginal likelihood and training objective

To train the model with stochastic attention, it is necessary to optimize the marginal likelihood of the training data. Because we use "late fusion", z_t and z_{t+1} are independent given $\boldsymbol{x}$. Therefore, we can be computed in time $O(MN)$ as follows:

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) \tag{5.3}$$

$$= \prod_{t=1}^{M} \sum_{z_t=1}^{N} p(z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) p(y_t \mid z_t, \boldsymbol{x}, \boldsymbol{y}_{<t}), \tag{5.4}$$

where the probabilities are defined as above. M is the length of the source sentence and N is the length of the target sentence.

The training objective is the negative log-likelihood,

$$\mathcal{L} = -\log p(\boldsymbol{y} \mid \boldsymbol{x}).$$

5.2.2 Decoding

Most simply, one can jointly sample (or greedily maximize) sequences of $\boldsymbol{z}, \boldsymbol{y}$ given an input $\boldsymbol{x}$. However, the same independence assumption means that generating samples from the marginal distribution is possible. For the translation results reported in this paper, we perform a simple greedy search, alternating between choosing the next best alignment and choosing the next best output symbol.

5.3 Approximating the Marginal Likelihood

Exact calculation of the marginal likelihood for models with latent variables is, in general, intractable. Furthermore, although strictly tractable, Eq. 5.4 is costly to compute relative to deterministic attention. Although the time complexity of the them are both $O(MN)$, the stochastic version requires to compute M times softmax over the full vocabulary at every timestep on the decoder side, while deterministic attention mechanism only requires to compute the softmax once every timestep on the decoder side. We therefore consider a variety of strategies for approximating this quantity.

Variational lower bound. Variational inference (Jordan et al., 1999; Blei et al.) is a widely method for approximating probability densities which are difficult to compute. The main idea behind variational inference is to find a approximate distribution q that is easy to deal with and use that q to construct a lower bound for the original posterior distribution to optimize. We use variational inference to

approximate the marginal likelihood in Equation 5.4 as follows:

$$\log p(\boldsymbol{y} \mid \boldsymbol{x}) = \log \sum_{z} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) \tag{5.5}$$

$$= \sum_{t=1}^{M} \log \sum_{z_t=1}^{N} p(z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) p(y_t \mid z_t, \boldsymbol{x}, \boldsymbol{y}_{<t}) \tag{5.6}$$

$$= \sum_{t=1}^{M} \log \sum_{z_t=1}^{N} p(y_t, z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) \tag{5.7}$$

$$\geq \sum_{t=1}^{M} \mathop{\mathbb{E}}_{q} \log \frac{p(y_t, z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t})}{q(z_t)} \tag{5.8}$$

$$= \sum_{t=1}^{M} \mathop{\mathbb{E}}_{q} \log p(y_t, z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) + H(q) \tag{5.9}$$

Here q is the variational approximation, an approximate distribution created to make inference easier. $H(q)$ defines the *entropy* of the q distribution.

In the literature of neural variational inference, the approximate distribution q is often referred as a *recognition model* (Salakhutdinov and Larochelle, 2010; Mnih and Gregor, 2014; Rezende et al., 2014). The parameters in the q distribution come from the neural networks. We parameterize this *recognition model* basically the same way as we parameterize the categorical distribution in Equation 5.2, except the hidden representations on the target side of the recognition model are computed from a bi-directional LSTM conditioning on the whole $\boldsymbol{y}$ structure instead of the uni-directional (forward) LSTM conditioning on the history $\boldsymbol{y}_{<t}$. The parameters are not shared between the recognition model and the encoder/decoder. We use the Monte Carlo method to approximate the expectation under the q distribution. Equation 5.9 can be approximated using:

$$\log p(\boldsymbol{y} \mid \boldsymbol{x}) \leq \sum_{t=1}^{M} \mathop{\mathbb{E}}_{q} \log \frac{p(y_t, z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t})}{q(z_t)} \tag{5.10}$$

$$\approx \sum_{t=1}^{M} \frac{1}{K} \left(\sum_{i=1}^{K} \frac{\log p(y_t, \tilde{z_t}^{(i)} \mid \boldsymbol{x}, \boldsymbol{y}_{<t})}{q(\tilde{z_t}^{(i)})}\right) \tag{5.11}$$

where $\tilde{z_t}^{(i)}$ is ith sample from the q distribution (i.e., the recognition model), we take K samples in total.

REINFORCE. Another way to approximate the marginal likelihood is to use the REINFORCE algorithm (Williams, 1987, 1992). It can be essentially seen as a one-sample approximation of the original marginal likelihood objective in Equation 5.4:

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{M} \sum_{z_t=1}^{N} p(z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) p(y_t \mid z_t, \boldsymbol{x}, \boldsymbol{y}_{<t}) \tag{5.12}$$

$$= \prod_{t=1}^{M} p(\tilde{z_t} \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) p(y_t \mid \tilde{z_t}, \boldsymbol{x}, \boldsymbol{y}_{<t}) \tag{5.13}$$

where $\tilde{z_t}$ is sampled from the Categorical distribution in Equation 5.2.

5.4 Experiment: Deterministic vs. Stochastic Attention

We conducted our experiments for translating from Chinese to English. The experiments focus on a (simulated) low resource setting, where only a limited amount of training data is available.

We used the BTEC corpus, where the number of training sentence pairs is 44,016. We followed the standard split in this data set, using 'devset1 2' as the development and 'devset 3' as test set and in both cases, we used the first reference for evaluation.

For all the models, we used word embedding dimensions D of 128 and search over hidden dimensions H of {64, 128} in 2-layer LSTMs. We tuned the dropout rate in the grid of {0.2, 0.3, 0.4, 0.5} to maximize validation set likelihood. In all the experiments, we use the stochastic gradient decent algorithm and set the step size

Model	Inference	BLEU	Perplexity
Deterministic	–	*31.87*	5.25
Stochastic	exact	**31.91**	**4.65**
Stochastic	variational	30.10	5.40
Stochastic	REINFORCE	29.85	5.31

Table 5.1: Deterministic vs. Stochastic Attention.

to be 0.1 to optimize the parameters in the models. All parameters were initialized according to recommendations given by Glorot and Bengio (2010). We give each setting 3 times of random initialization and pick the best among these.

We measure the BLEU score and perplexity (PPL). Log probabilities of the target side words are normalized by the number of target side words, then inverted and exponentiated to yield the perplexity (lower perplexity suggests better performance). When computing the perplexity, we always use the exact marginal likelihood so that it is fair to compare perplexity of approximate inference methods because we do not approximate perplexity during the evaluation. For the BLEU score, we use greedy decoding (§5.2.2) and use the single best prediction for evaluation.

Table 5.1 shows the comparison between deterministic and stochastic attention mechanisms. We use our own implementation of the deterministic attention sequence-to-sequence model as the baseline and test the performance of three different variants (i.e., the exact marginal objective (exact; Equation 5.4, the variational lower bound objective (variational; Equation 5.9) and the REINFORCE algorithm (REINFORCE; Equation 5.13)) of our stochastic attention sequence-to-sequence model. The later two variants are approximating methods.

We find that introducing the stochastic variable for the alignments, although the exact version achieves the best BLEU score and perplexity, does not improve

the performance of the model significantly. The reason behind this might be that, as a general-purpose function approximators, the current neural networks has no strong inductive bias to bring the power of those stochastic attention fully in to play. In fact, there is a chance that the model will ignore these stochastic variables and degenerate into a pure sequence-to-sequence model, especially in the low resource setting where the training set is not so large and thus easy to fit. For the approximate methods, because the variance of the samples can be large, they harm the performance of the system. These motivate us to regularize the posterior distributions of the latent alignment decisions using the posteriors computed from IBM model 2 (Brown et al., 1993; Dyer et al., 2013) in the next section (§5.5).

5.5 Posterior Regularization

Neural networks are powerful approximators that can fit noisy data or, when stochastic units are present, explain away stochastic latent variables (Chen et al., 2017; Zhang et al., 2017). They tend not to use more sophisticated stochastic mechanisms, because with the expressiveness offered by the nonlinearity, just observing input patterns the model can explain the data quite well. We find evidence that this is happening in our model. It is easy for the neural models to achieve high performance on the training set, but on the testing set. It may not generalize well. That is the reason people introduces many regularization strategies, for example the dropout method which almost become a essential in the neural network training procedure.

For these reasons, we should not let the neural networks arbitrarily fit the stochastic units. Because these units represent the alignment structures, they should be linguistically sound. The aligned parts should share the same/similar semantic meanings in different languages. Our regularization strategy is inspired

by posterior regularization (Ganchev et al., 2010), but rather than imposing declarative moment constraints on the posteriors, we regularize the posterior distributions over the latent variables to be similar to posterior distributions obtained from models that make stronger independence assumptions and therefore use their latent variables (rather than the deterministic autoregressive component) to explain the outputs.

Exact regularized objective. In the framework of posterior regularization (Ganchev et al., 2010), we can compute the loss as follows:

$$\mathcal{L} = -\log \sum_{z} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) + \gamma \times D_{KL}(p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y}) \mid\mid \tilde{q}(\boldsymbol{z})), \tag{5.14}$$

where D_{KL} denotes the Kullback–Leibler divergence (Kullback and Leibler, 1951) and $\gamma \geq 0$ is the hyperparameter which controls the regularization strength. We use $\tilde{q}$ instead of q here to emphasize that the $\tilde{q}$ here is a fixed distribution. There are no parameters in the $\tilde{q}$ distribution for the model to optimize, different from the q in Equation 5.9. We follow this notation convention when we discuss important sampling methods in the next sections.

To compute posterior $p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})$, we follow the definition of conditional probability:

$$p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y}) = \frac{p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})}{\sum_{z} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})}, \tag{5.15}$$

where $p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$ can be computed same way as in Equation 5.4.

Now the only question remains is how to choose the distribution q. For that, we use a model which makes stronger independence assumptions. As an essential part in the statistical machine translation (SMT) systems, IBM model 2 (Brown et al., 1993; Dyer et al., 2013) is a natural choice for the $\tilde{q}$ distribution. For the importance sampling and Jensen important sampling, we also use this as the $\tilde{q}$ distribution.

Indirect regularization via importance sampling Importance sampling (IS) (Doucet and Johansen, 2009) is a general approximation technique and it has a theoretical guarantee that if the true posterior is used as the approximate distribution, the importance sampling estimator would have zero variance (Mnih and Rezende, 2016). IBM model 2 in this case, makes a sensible $\tilde{q}$ distribution here because it is a good approximation for the true alignment structure information.

$$\log p(\boldsymbol{y} \mid \boldsymbol{x}) = \log \sum_{z} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) \tag{5.16}$$

$$= \sum_{t=1}^{M} \log \sum_{z_t=1}^{N} p(z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) p(y_t \mid z_t, \boldsymbol{x}, \boldsymbol{y}_{<t}) \tag{5.17}$$

$$= \sum_{t=1}^{M} \log \sum_{z_t=1}^{N} p(y_t, z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) \tag{5.18}$$

$$= \sum_{t=1}^{M} \log \sum_{z_t=1}^{N} \tilde{q}(z_t) w(z_t, \boldsymbol{x}, \boldsymbol{y}) \tag{5.19}$$

$$= \sum_{t=1}^{M} \mathop{\mathbb{E}}_{\tilde{q}} w(z_t, \boldsymbol{x}, \boldsymbol{y}), \tag{5.20}$$

where,

$$w(z_t, \boldsymbol{x}, \boldsymbol{y}) = \frac{p(y_t, z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t})}{\tilde{q}(z_t)} \tag{5.21}$$

defines the *importance weights*.

A variant of importance sampling we refer it as Jensen importance sampling (Jensen IS). Inspired by Mnih and Rezende (2016), when we consider the variational lower bound objective in Equation 5.9, one reason it does not work so well might be that the samples from the q distribution has too high variance. Therefore, if we replace that with a fixed distribution $\tilde{q}$ from IBM model 2, we will achieve

variance reduction here and may achieve better performance.

$$\log p(\boldsymbol{y} \mid \boldsymbol{x}) = \log \sum_{z} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) \tag{5.22}$$

$$= \sum_{t=1}^{M} \log \sum_{z_t=1}^{N} p(z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) p(y_t \mid z_t, \boldsymbol{x}, \boldsymbol{y}_{<t}) \tag{5.23}$$

$$= \sum_{t=1}^{M} \log \sum_{z_t=1}^{N} p(y_t, z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) \tag{5.24}$$

$$\geq \sum_{t=1}^{M} \mathop{\mathbb{E}}_{\tilde{q}} \log \frac{p(y_t, z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t})}{\tilde{q}(z_t)} \tag{5.25}$$

$$= \sum_{t=1}^{M} \mathop{\mathbb{E}}_{\tilde{q}} \log p(y_t, z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) - H(\tilde{q}) \tag{5.26}$$

$$\tag{5.27}$$

The derivation of Jensen importance sampling (Jensen IS) is almost the same as in the derivation of the variational lower bound in Equation 5.9. The key difference is that, here $\tilde{q}$ is a fixed distribution rather than a parameterized recognition model. Because of this, when we optimize this objective, we are effectively optimizing the importance sampling objective expect that we do not divide the $p(y_t, z_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t})$ term by the approximate distribution since $H(\tilde{q})$ is a fixed value for a given $\tilde{q}$ distribution.

5.6 Experiment: The Effect of Posterior Regularization

In these experiments, we follow the same setting as in §5.4. We tuned the γ in the exact posterior regularization objective in the grid of $0.1, 1, 10$ by picking the best value using the separate development data. We report the BLEU score and the perplexity on the test set.

Model	Inference	PR	BLEU	Perplexity
Deterministic	–	none	31.87	5.25
Stochastic	exact	none	31.91	4.65
Deterministic	–	full	32.48	5.20
Stochastic	exact	full	35.17	4.03
Stochastic	IS	approximate	34.68	4.04
Stochastic	Jensen IS	approximate	**35.40**	**3.94**

Table 5.2: The effects of posterior regularization.

Table 5.2 shows that adding the information from alignment structures in the framework of posterior regularization generally improves the performance of the model. In the deterministic attention version, we add the KL distance between $\tilde{q}(z_t)$ andCategorical$(z_t; \tilde{\boldsymbol{\zeta}}_t)$ as a term to simulate the PR penalty. Chen et al. (2016) used this approach to bias the attention mechanism of a sequence-to-sequence neural machine translation (NMT) model towards the well-studied statistical word alignment models. It is not in the framework of posterior regularization but the closest we think to use the information from the alignment structure information in the $\tilde{q}$ distribution. We find this underperforms our stochastic model.

Approximations based on importance sampling is either as good as or better than the exact objective in terms of held-out generalization, suggesting a computationally effective way to add these inductive bias introduced by the alignment structure information into the model.

5.7 Related Work

Neural sequence to sequence models with stochastic attention have been explored twice, and both used a kind of regularization of the posterior values. Wang et al.

(2017b) used an HMM dependency on the z_t's, and trained with an EM algorithm whose initial "E-step" values were set to the posteriors computed from IBM Model 1 (similar in structure to the $\tilde{q}$ we used). Yu et al. (2016) assumed a latent variable parameterized as a series of independent Bernoulli trials initialized to favor a diagonal alignment.

Initializing complex models using parameters and/or posteriors derived from latent variables from simpler models that make more independence assumptions has an enduring history, and has been particularly effective for alignment models (Brown et al., 1993; Och and Ney, 2003; Gimpel and Smith, 2012).

Our training objective is related to posterior regularization (Ganchev et al., 2010). However, PR was proposed as a strategy for correcting the biases associated with overly naïve statistical models. In other words, while PR was designed to deal with errors arrising from overly biased models, we have used it to deal with models that make errors due to high variance.

Our posterior regularization approach is also related to (Smith and Eisner, 2006), where their task is unsupervised dependency parsing, and they regularize their posteriors to be close to a simple length-based model.

5.8 Conclusion

In this chapter, we further strengthen the inductive biases introduced by the alignment structures in two ways. First, we explicitly model the alignment decisions in the probabilistic framework by replacing the soft weighting scheme with a hard (but stochastic) decision. Second, we regularize the posterior distributions of the latent alignment decisions using the posteriors computed from IBM model 2 (Brown et al., 1993; Dyer et al., 2013). This alleviates the problem that neural networks can fit noisy data or explain away stochastic latent variables, because

they are general-purpose function approximators (Chen et al., 2017; Zhang et al., 2017). We propose both an exact training objective based on analytical marginals, and several approximations based on importance sampling that are suitable when exact marginals cannot be computed. We show that by adding the helpful inductive biases leads to substantially improved generalization. The approximations based on importance sampling (which is therefore suitable for problems with intractable marginals) is either as good as or better than the exact objective in terms of held-out generalization.

Chapter 6

Conclusion and Future work

The work presented in this thesis describes several instances of how adding linguistic driven inductive biases can improve the performance of the neural representation learners for NLP problems, without sacrificing computational efficiency.

Neural nets can enrich the expressive power of probability distributions in a similar manner to latent variables and marginalization. We stress the importance of explicitly representing structure in neural models. We show that the linguistic driven inductive biases can regularize the model while keep the expressiveness of the neural networks. When comparing with structurally naive models, models that reason about the internal linguistic structure of the data demonstrate better generalization performance.

The techniques proposed in thesis automatically learn the structurally informed representations of the inputs. These representations and components in the models can be better integrated with other end-to-end deep learning systems within and beyond NLP (Cho et al., 2014b; Graves et al., 2013, *inter alia*).

There are many interesting directions that can be explored further in the future. We discuss three of them in the following.

Segment Structures in Neural Machine Translation Segment structures are proved to be very useful in machine translation (Koehn, 2009). However, in most sequence-to-sequence based neural machine translation systems, segment structures are mostly ignored. Recent work by Huang et al. (2017) showed promising results by introducing segment information into their model. In chapter 3 we introduce to model the segments explicitly using SRNNs. Future work can explore how to effectively use these segment embeddings and combine them with effective phrase-based decoding algorithms such as (Chang and Collins, 2017) on the decoder side.

Automatic Linguistic Structure Discovery Linguists have found that language meaning is derived through composition (Manning, 2016). In this thesis, we show that explicitly modeling of these structures improves the performance of neural representation learners. In chapter 4, these structures are learned in a fully supervised fashion. One promising direction is to use reinforcement learning to automatically discover these structures from the distance supervision of the end task. Yogatama et al. (2016) show promising results of learning tree structures this way. Because DRAGNN can encode many different types of linguistic structures, it can be interesting to explore how to automatically construct those structures and how different kinds of linguist structures work together to affect the performance of the end task.

Hard Constraints in Attention Mechanism A key challenge in phrase-based machine translation is to solve the inference problem of finding the best translation given the constrain that each source-language word must be exactly once

(Koehn et al., 2003; Chang and Collins, 2011; Rush, 2012). In Chapter 5 we show that regularize the posterior distributions of the latent alignment decisions in the discrete stochastic attention model leads to substantially improved generalization. Future work can explore if we can effectively hard constraints, such as attending to each source-language word exactly once, into the framework and to see if these constraints improve or harm the performance of the model.

Bibliography

Christopher D Manning. Computational linguistics and deep learning. *Computational Linguistics*, 2016.

Tom M Mitchell. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ. New Jersey, 1980.

Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. Recurrent neural network grammars. *Proc. NAACL*, 2016.

Yangfeng Ji, Gholamreza Haffari, and Jacob Eisenstein. A latent variable recurrent neural network for discourse relation language models. *arXiv preprint arXiv:1603.01913*, 2016.

Alex Graves, A-R Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Proc. ICASSP*, pages 6645–6649. IEEE, 2013.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. What do recurrent neural network grammars learn about syntax? *arXiv preprint arXiv:1611.05774*, 2017.

Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*, 2017.

Lingpeng Kong, Chris Alberti, Daniel Andor, Ivan Bogatyy, and David Weiss. Dragnn: A transition-based framework for dynamically connected neural networks. *arXiv preprint arXiv:1703.04474*, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

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

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

Lingpeng Kong, Chris Dyer, and Noah A Smith. Segmental recurrent neural networks. *Proc. ICLR*, 2015.

Liang Lu, Lingpeng Kong, Chris Dyer, Noah A Smith, and Steve Renals. Segmental recurrent neural networks for end-to-end speech recognition. *Proc. Interspeech*, 2016.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proc. ICML*, 2006a.

Kuzman Ganchev, João Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. 11:2001–2049, 2010.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Sunita Sarawagi and William W Cohen. Semi-markov conditional random fields for information extraction. In *Advances in neural information processing systems*, pages 1185–1192, 2004.

Yijia Liu, Wanxiang Che, Jiang Guo, Bing Qin, and Ting Liu. Exploring segment representations for neural segmentation models. *arXiv preprint arXiv:1604.05499*, 2016.

Swabha Swayamdipta, Sam Thomson, Chris Dyer, and Noah A. Smith. Frame-semantic parsing with softmax-margin segmental rnns and a syntactic scaffold. *arXiv preprint arXiv:1706.09528*, 2017.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proc. ICML*, 2006b.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18 (5):602–610, 2005.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997a.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proc. ICML*, 2014.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proc. CVPR*, 2015.

Hao Tang, Liang Lu, Lingpeng Kong, Kevin Gimpel, Karen Livescu, Chris Dyer, Noah A. Smith, and Steve Renals. End-to-end neural segmental models for speech recognition. *arXiv preprint arXiv:1708.00531*, 2017.

Lawrence R. Rabiner. A tutorion on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2), 1989.

Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

Robert H. Kassel. *A comparison of approaches to on-line handwritten character recognition*. PhD thesis, Massachusetts Institute of Technology, 1995.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov networks. *NIPS*, 16:25, 2004.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Martha Palmer. The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238, 2005.

Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. Two/too simple adaptations of word2vec for syntax problems. In *Proc. NAACL*, 2015a.

David Graff and Ke Chen. Chinese gigaword. *LDC Catalog No.: LDC2003T09*, 1, 2005.

Alex Graves. Hierarchical subsampling networks. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 109–131. Springer, 2012a.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek,

Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011. IEEE Catalog No.: CFP11SRW-USB.

Geoffrey Zweig. Classification and recognition with direct segment models. In *Proc. ICASSP*, pages 4161–4164. IEEE, 2012.

Yanzhang He and Eric Fosler-Lussier. Efficient segmental conditional random fields for phone recognition. In *Proc. INTERSPEECH*, pages 1898–1901, 2012.

Ossama Abdel-Hamid, Li Deng, Dong Yu, and Hui Jiang. Deep segmental neural networks for speech recognition. In *Proc. INTERSPEECH*, pages 1849–1853, 2013.

Hao Tang, Weiran Wang, Kevin Gimpel, and Karen Livescu. Discriminative segmental cascades for feature-rich phone recognition. In *Proc. ASRU*, 2015.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585, 2015.

Andrew L. Maas, Ziang Xie, Dan Jurafsky, and Andrew Y. Ng. Lexicon-free conversational speech recognition with neural networks. In *Proc. NAACL*, 2015.

Awni Y. Hannun, Carl Case, Jared Casper, Bryan C. Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. Listen, attend, and spell. *CoRR*, abs/1508.01211, 2015.

Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philémon Brakel, and

Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. *CoRR*, abs/1508.04395, 2015a.

Alex Graves. Sequence transduction with recurrent neural networks. In *Proc. ICML*, 2012b.

Huda Abdel-Hamid. *Structural-Functional Analysis of Plant Cyclic Nucleotide Gated Ion Channels*. PhD thesis, University of Toronto, 2013.

Chong Wang, Yining Wang, Po-Sen Huang, Abdelrahman Mohamed, Dengyong Zhou, and Li Deng. Sequence modeling via segmentations. *arXiv preprint arXiv:1702.07463*, 2017a.

Lei Yu, Jan Buys, and Phil Blunsom. Online segment to segment neural transduction. *arXiv preprint arXiv:1609.08194*, 2017.

Jian Peng, Liefeng Bo, and Jinbo Xu. Conditional neural fields. In *Proc. NIPS*, 2009.

Greg Durrett and Dan Klein. Neural CRF parsing. In *Proc. ACL*, 2015.

Kyunghyun Cho et al. Foundations and advances in deep learning. 2014a.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781, 2015.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *Pro. EMNLP*, 2014b.

Peilu Wang, Yao Qian, Frank K Soong, Lei He, and Hai Zhao. A unified tagging solution: Bidirectional lstm recurrent neural network with word embedding.

arXiv preprint arXiv:1511.00215, 2015.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *Proc. ACL*, pages 455–465, 2013.

Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. *EMNLP*, 2013.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*, 2015b.

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

Jiwei Li, Minh-Thang Luong, Dan Jurafsky, and Eudard Hovy. When are tree structures necessary for deep learning of representations? *arXiv preprint arXiv:1503.00185*, 2015.

Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. *arXiv preprint arXiv:1603.06021*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine trans-

lation by jointly learning to align and translate. *ICLR*, 2015b.

Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *CoRR*, abs/1511.06114, 2015. URL `http://arxiv.org/abs/1511.06114`.

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

Joakim Nivre. *Inductive dependency parsing*. Springer, 2006.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional lstm feature representations. *ACL*, 2016.

Meishan Zhang, Yue Zhang, and Guohong Fu. Transition-based neural word segmentation. In *Proceedings of the 54nd Annual Meeting of the Association for Computational Linguistics*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Richard Socher, Christopher D. Manning, and Andrew Y. Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, 2010.

Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, pages 801–809, 2011.

Katja Filippova and Yasemin Altun. Overcoming the lack of parallel data in sentence compression. In *EMNLP*, pages 1481–1491. Citeseer, 2013.

Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the ACL and the 7th International Joint Conference on Natural Language Processing*, pages 1723–1732, 2015.

Anders Søgaard and Yoav Goldberg. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 231–235, 2016.

Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Citeseer, 2003.

Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE, 1996.

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

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016a.

Giuseppe Attardi and Felice Dell'Orletta. Reverse revision and linear tree combination for dependency parsing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 261–264. Association for Computational Linguistics, 2009.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush.

Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*, 2017.

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

Chris Dyer, Victor Chahuneau, and Noah A Smith. A simple, fast, and effective reparameterization of ibm model 2. Association for Computational Linguistics, 2013.

Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. In *Proc. ICLR*, 2017.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *Proc. ICLR*, 2017.

Dirk P. Kingma and Max Welling. Autoencoding variational Bayes. In *Proc. ICLR*, 2014.

Jörg Bornschein and Yoshua Bengio. Reweighted wake-sleep. In *Proc. ICLR*, 2015.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In *Proc. ICLR*, 2016.

Yishu Miao and Phil Blunsom. Language as a latent variable: Discrete generative models for sentence compression. In *Proc. EMNLP*, 2016.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *Proc. ICLR*, 2017.

Tian Wang and Kyunghyun Cho. Larger-context language modelling. *arXiv preprint arXiv:1511.03729*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997b.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In *Proc. EMNLP*, 2016b.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

DM Blei, A Kucukelbir, and JD McAuliffe. Variational inference: A review for statisticians. 2016. *arXiv preprint arXiv:1601.00670*.

Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 693–700, 2010.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Ronald J Williams. A class of gradient-estimating algorithms for reinforcement learning in neural networks. In *Proceedings of the IEEE First International Conference on Neural Networks San Diego, CA*, 1987.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and

smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3, 2009.

Andriy Mnih and Danilo Rezende. Variational inference for monte carlo objectives. In *International Conference on Machine Learning*, pages 2188–2196, 2016.

Wenhu Chen, Evgeny Matusov, Shahram Khadivi, and Jan-Thorsten Peter. Guided alignment training for Topic-Aware neural machine translation. 6 July 2016.

Weiyue Wang, Derui Zhu, and Hermann Ney. Hybrid neural network alignment and lexicon model in direct HMM for statistical machine translation. In *Proc. ACL*, 2017b.

Lei Yu, Jan Buys, and Phil Blunsom. Online segment to segment neural machine translation. In *Proc. EMNLP*, 2016.

Franz Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.

Kevin Gimpel and Noah A. Smith. Concavity and initialization for unsupervised dependency parsing. In *Proc. NAACL*, 2012.

Noah A Smith and Jason Eisner. Annealing structural bias in multilingual weighted grammar induction. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 569–576. Association for Computational Linguistics, 2006.

Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 17 December 2009.

Po-Sen Huang, Chong Wang, Dengyong Zhou, and Li Deng. Toward neural phrase-based machine translation. 17 June 2017.

Yin-Wen Chang and Michael Collins. A Polynomial-Time dynamic programming algorithm for Phrase-Based decoding with a fixed distortion limit. *Transactions*

of the Association for Computational Linguistics, 5(0):59–71, 3 February 2017.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. 28 November 2016.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

Yin-Wen Chang and Michael Collins. Exact decoding of phrase-based translation models through lagrangian relaxation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 26–37, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

Alexander M Rush. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *J. Artif. Intell. Res.*, 45:305–362, 2012.