DRAGNN: A TRANSITION-BASED FRAMEWORK FOR DYNAMICALLY CONNECTED NEURAL NETWORKS

Lingpeng Kong
Carnegie Mellon University
Pittsburgh, PA
lingpenk@cs.cmu.edu

Chris Alberti Daniel Andor Ivan Bogatyy David Weiss
Google
New York, NY
{chrisalberti,andor,bogatyy,djweiss}@google.com

ABSTRACT

In this work, we present a compact, modular framework for constructing new recurrent neural architectures. Our basic module is a new generic unit, the Transition Based Recurrent Unit (TBRU). In addition to hidden layer activations, TBRUs have discrete state dynamics that allow network connections to be built dynamically as a function of intermediate activations. By connecting multiple TBRUs, we can extend and combine commonly used architectures such as sequence-to-sequence, attention mechanisms, and recursive tree-structured models. A TBRU can also serve as both an *encoder* for downstream tasks and as a *decoder* for its own task simultaneously, resulting in more accurate multi-task learning. We call our approach Dynamic Recurrent Acyclic Graphical Neural Networks, or DRAGNN. We show that DRAGNN 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.

1 INTRODUCTION

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 & Blunsom, 2013; Cho et al., 2014; 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* and predict output tags from those (Ling et al., 2015; Huang et al., 2015; 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). Implictly learned mappings via attention mechanisms can significantly improve the performance of sequence-to-sequence (Bahdanau et al., 2015; Vinyals et al., 2015), but require runtime that's quadratic in the input size.

In this work, we propose a modular neural architecture that generalizes the encoder/decoder concept to include explicit structure. Our framework can represent sequence-to-sequence learning as well as models with explicit structure like bi-directional tagging models and compositional, tree-structured models. Our core idea is to 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.

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);

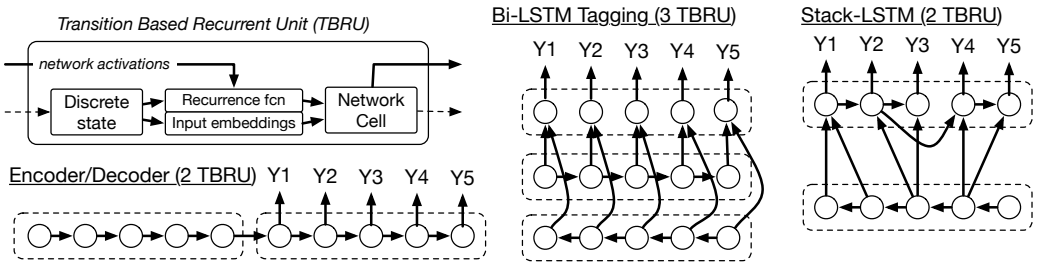


Figure 1: High level schematic of a Transition-Based Recurrent Unit (TBRU), and common network architectures that can be implemented with multiple TBRUs. The discrete state is used to compute recurrences and fixed input embeddings, which are then fed through a network cell. The network predicts an action 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.

Kiperwasser & 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 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 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 linear in the size of the input, in contrast to quadratic-time attention mechanisms. Dynamic connections thus establish a compromise between pure seq2seq and pure attention architectures by providing 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 & 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 utilize 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 re-using representations across NLP tasks that operate on different structures.

We demonstrate the effectiveness of DRAGNN on two NLP tasks that benefit from explicit structure: dependency parsing and extractive sentence summarization (Filippova & 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., 2016) 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.

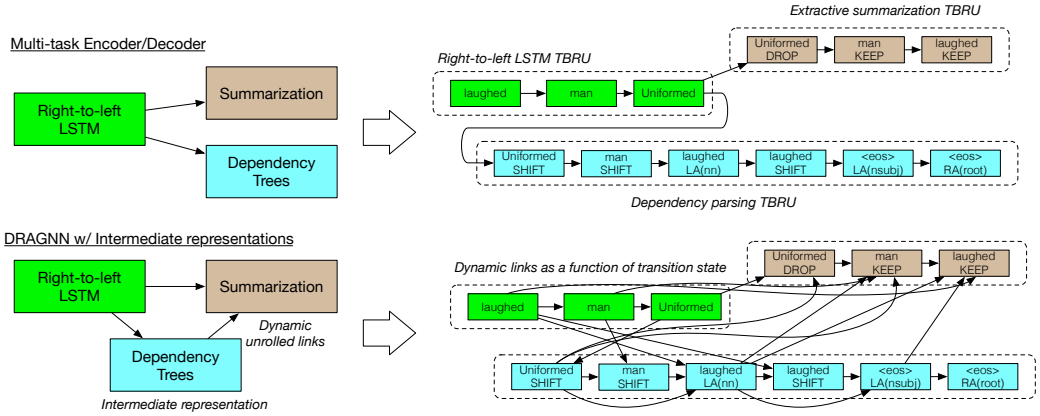


Figure 2: Using TBRUs to share fine-grained, structured representations. Top left: A high level view of multi-task learning with DRAGNN in the style of multi-task seq2seq (Luong et al., 2016). Bottom left: Extending the "stack-propagation" Zhang & Weiss (2016) idea to included dependency parse trees as intermediate representations. Right: Unrolled TBRUs for each setup for a input fragment "Uniformed man laughed", utilizing the transition systems described in Section 4.

2 Transition Systems

We use *transition systems* to map inputs x into a sequence of output symbols, $d_1 \dots d_n$. 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 $\mathcal{S}(x)$.
- A special start state $s^\dagger \in \mathcal{S}(x)$.
- A set of *allowed* decisions $\mathcal{A}(s, x)$ for all $s \in \mathcal{S}$.
- A transition function $t(s, d, x)$ returning a new state s' for any decision $d \in \mathcal{A}(s, x)$.

For brevity, we will drop the dependence of x in the functions given above. Throughout this work we will use transition systems in which all complete structures for the same input x have the same number of decisions $n(x)$ (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.

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 to fixed-size vector representations, for example, an embedding lookup operation for features from the discrete state:

$$\mathbf{m}(s) : \mathcal{S} \mapsto \mathcal{R}^K$$

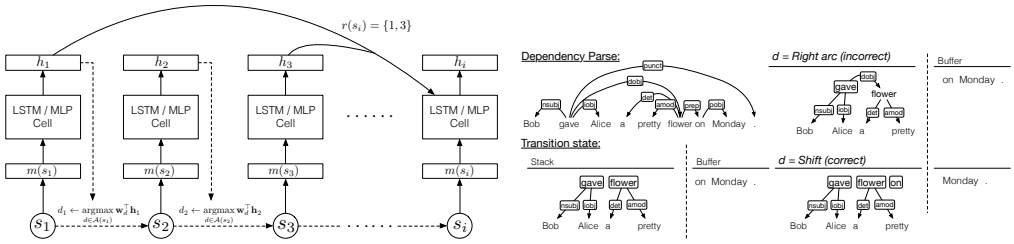


Figure 3: Left: TBRU schematic. Right: Dependency parsing example. For the given gold dependency parse tree and a *arc-standard* transition state with two sub-trees on the stack is shown. From this state, two possible actions are also shown (Shift and Right arc). To reproduce the tree, the Shift action should be taken.

- A recurrrence 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 RNN cell that computes a new hidden representation from the fixed and recurrent inputs:
$$\mathbf{h}_s \leftarrow \mathbf{RNN}(\mathbf{m}(s), \{\mathbf{h}_i \mid i \in \mathbf{r}(s)\}).$$

Example 1. Sequential tagging RNN. Let the input $x = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be a sequence of word embeddings, and the output be a sequence of tags $d_1, \ldots, d_n$. Then we can model a simple LSTM tagger as follows:

- $\mathcal{T}$ sequentially tags each input token, where $s_i = \{1, \ldots, d_{i-1}\}$, and $\mathcal{A}$ is the set of possible tags. We call this the *tagger* transition system.
- $\mathbf{m}(s_i) = \mathbf{x}_i$, the word embedding for the next token to be tagged.
- $\mathbf{r}(s_i) = \{i - 1\}$ to connect the network to the previous state.
- $\mathbf{RNN}$ is a single instance of the LSTM cell.

Example 2. Parsey McParseface. The open-source syntactic parsing model of Andor et al. (2016) can be defined in our framework as follows:

- $\mathcal{T}$ is the *arc-standard* transition system (Figure 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.
- $\mathbf{r}(s_i) = \{\}$ is empty, as this is a feed-forward network.
- $\mathbf{RNN}$ 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: $\mathbf{h}_i \leftarrow \mathbf{RNN}(\mathbf{m}(s_i), \{\mathbf{h}_j \mid j \in \mathbf{r}(s_i)\})$.
   (b) Update the transition state: $d_i \leftarrow \text{argmax}_{d \in \mathcal{A}(s_i)}\, \mathbf{w}_d^\top \mathbf{h}_i, \quad s_{i+1} \leftarrow t(s_i, d_i)$.

A schematic overview of a single TBRU is presented in Figure 3. By adjusting $\mathbf{RNN}$, $\mathbf{r}$, and $\mathcal{T}$, TBRUs can represent a wide variety of neural architectures.

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 compo-

sitional 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 TBRU components, 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 time step $\mathbf{h}_{n+i-1}$.

Example 4. Bi-directional LSTM tagger. With three TBRUs, we can implement a simple 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: $\mathcal{T} =$ *shift-only*, $\mathbf{m}(s_i) = \mathbf{x}_i$, $\mathbf{r}(s_i) = \{i - 1\}$.
- Right to left: $\mathcal{T} =$ *shift-only*, $\mathbf{m}(s_{n+i}) = \mathbf{x}_{n-i}$, $\mathbf{r}(s_{n+i}) = \{n + i - 1\}$.
- Tagger: $\mathcal{T} = tagger$, $\mathbf{m}(s_{2n+i}) = \{\}$, $\mathbf{r}(s_{2n+i}) = \{i, 2n - i\}$.

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 Figure 1 for an unrolled example.

Example 5. Multi-task bi-directional tagging. Here we observe that it's possible to add additional annotation tasks to the bi-directional TBRU stack from Example 4 simply 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 & Weiss, 2016) style of multi-task learning simply by changing $\mathbf{r}$ for the last TBRU:

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

Remark: the raison d'être of DRAGNN. This example highlights the primary advantage of our formulation: *a TBRU can serve as both an encoder for downstream tasks and as a decoder for its own task simultaneously.* 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 $\mathbf{x}_1, \ldots, \mathbf{x}_n$ 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

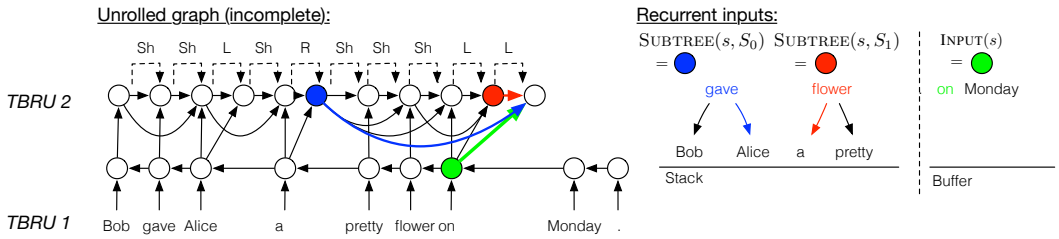


Figure 4: Detailed schematic for the compositional dependency parser used in our experiments. The first TBRU consumes each input word right-to-left; the second uses the arc-standard transition system. Note that each "Shift" action causes the TBRU1→TBRU2 link to advance. The dynamic recurrent inputs to the given state are highlighted; the stack representations are obtained from the last "Reduce" action to modify each sub-tree.

use the *arc-standard* parser directly to produce the parse tree as well as encode sub-phrases into representations.

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 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:

$$\text{SUBTREE}(s, i) = \underset{j}{\text{argmax}}\{d_j \text{ s.t. } d_j \text{ shifts or adds a new child to token } i\}$$

We show an example of the links constructed by these recurrences in Figure 4, and we investigate variants of this model in Section 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.[1]

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 utilize the SUBTREE recurrence function to pull compositional, phrase-based representations of tokens as constructed by the dependency parser. This model is outlined in Figure 2. A full specification is given in the Appendix.

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 x along with gold decision sequences for one of the TBRUs in the DRAGNN.

[1]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 utilize subtree representations in downstream tasks.

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	$O(n)$
$\{n\}$	$\{\textsc{Subtree}(s_i, S_0), \textsc{Subtree}(s_i, S_1)\}$	36.0	75.6	$O(n)$
Attention	$\{n+i-1\}$	76.1	84.8	$O(n^2)$
Attention	$\{\textsc{Subtree}(s_i, S_0), \textsc{Subtree}(s_i, S_1)\}$	89.0	91.9	$O(n^2)$
$\textsc{Input}(s_i)$	$\{n+i-1\}$	87.1	89.7	$O(n)$
$\textsc{Input}(s_i)$	$\{\textsc{Subtree}(s_i, S_0), \textsc{Subtree}(s_i, S_1)\}$	**90.9**	**92.1**	$O(n)$

Table 1: Dynamic links enable much more accurate, efficient linear-time parsing models on the Treebank Union dev set. We vary the recurrences $\mathbf{r}$ to explore utilizing explicit structure in the parsing TBRU. Utilizing the explicit $\textsc{Input}(s_i)$ pointer is more effective and more efficient than a quadratic attention mechanism. Incorporating the explicit stack structure via recurrent links further improves performance.

Note that, at a minimum, we need such data for the final TBRU. Assuming given decisions $d_1 \ldots d_N$ from prior components $1 \ldots T-1$, we define a log-likelihood objective to train the T'th TBRU along its gold decision sequence $d^\star_{N+1}, \ldots, d^\star_{N+n}$, conditioned on prior decisions:

$$L(x, d^\star_{N+1:N+n}; \theta) = \sum_i \log P(d^\star_{N+i} \mid d_{1:N}, d^\star_{N+1:N+i-1}; \theta) \tag{1}$$

where θ are the combined parameters across all TBRUs. We observe that this objective is locally normalized (Andor et al., 2016), since we optimize the probabilities of the individual decisions in the gold sequence.

The remaining question is where do the decisions $d_1 \ldots d_N$ come from. There are two options here: they can either come as part of the gold annotation (e.g. if we have joint tagging and parsing data), or they will be predicted by unrolling the previous components (e.g. when training 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 & Kuchler, 1996) to optimize (1). To train the whole system on a set of C datasets, we use a similar strategy to (Dong et al., 2015; Luong et al., 2016); we sample a target task $c, 1 \leq c \leq C$, from a pre-defined ratio, and take a stochastic optimization step on the objective of that task's TBRU. In practice, task sampling is usually preceded by a deterministic number of pre-training steps, allowing, for example, to schedule a certain number of tagger training steps before running any parser training steps.

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 the Union Treebank setup from Andor 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 & 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 sub-sampled 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).

4.1 Using explicit structure improves encoder/decoder

We explore the impact of different types of recurrences on dependency parsing in Table 1. In this setup, we used relatively small models: single-layer LSTMs with 256 hidden units, taking

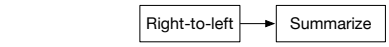
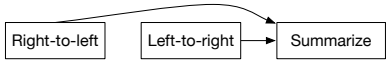
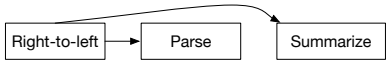
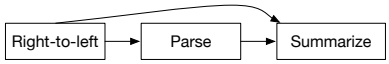
Model Structure	Multi-task?	A (%)	F1 (%)	LAS (%)
	N	28.93	79.75	–
	N	29.51	80.03	–
	Luong et al. (2016)	30.07	80.31	89.42
	Zhang & Weiss (2016)	**30.56**	**80.74**	89.13

Table 2: 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 utilize the parsing data outperform the single-task approach, but the model that uses parses as an intermediate representation in the vein of Zhang & Weiss (2016) (Figure 2) makes better use of the data. Note that the locally normalized model from Andor et al. (2016) obtains 30.50% accuracy and 78.72% F1 on the test set when trained on 100× more data.

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. (2016), where $\mathbf{r}(s_i)$ in the parser takes in all of the *shift-only* TBRU's hidden states and $\mathbf{RNN}$ aggregates over them.

4.2 Utilizing parse representations improves summarization

We evaluate our approach on the summarization task in Table 2. We compare two single-task LSTM tagging baselines against two multi-task approaches: an adaptation of Luong et al. (2016) and the stack-propagation idea of Zhang & 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 utilize 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 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.

4.3 Deep stacked bi-directional parsing

Here we propose a continuous version of the bi-directional parsing model of Attardi & 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 addition 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.

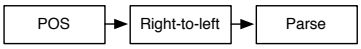
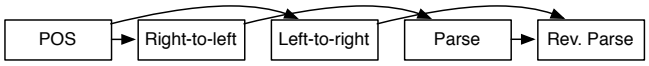
Model	Dev UAS	Dev LAS	Test UAS	Test LAS
	93.08	90.89	92.8	90.8
	94.01	91.93	93.72	91.83
(Above, but with pretrained word2vec)*	**94.07**	**92.06**	**94.09**	**92.12**
Bi-LSTM, graph-based (Kiperwasser & Goldberg, 2016)	–	–	93.10	91.00
Stack LSTM (Dyer et al., 2015)*	–	–	93.10	90.90
20 Stack LSTMs (Kuncoro et al., 2016)*	–	–	94.51	92.57
Globally normalized, transition-based Andor et al. (2016)*	–	–	**94.61**	**92.79**

Table 3: Deep stacked parsing compared to state-of-the-art on PTB. $\star$ indicates that additional resources beyond the Penn Treebank are used. Our model is roughly comparable to an ensemble of multiple Stack-LSTM models, and the most accurate without any additional resources.

Our final model uses 5 TBRU units. Inspired by Zhang & Weiss (2016), a left-to-right POS tagging TBRU provides the first layer of representations. Next, we run two *shift-only* TBRUs, one in each direction, to 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 3) is a state-of-the-art dependency parser, yielding the highest published accuracy for a model trained solely on the Penn Treebank with no additional resources.

5 CONCLUSIONS

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 tasks. There remains low-hanging fruit still 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.

ACKNOWLEDGEMENTS

We thank Kuzman Ganchev, Michael Collins, Dipanjan Das, Slav Petrov, Aliaksei Severyn, Chris Dyer, and Noah Smith for their useful feedback and discussion while preparing this draft.

REFERENCES

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 2442–2452, 2016.

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*, pp. 261–264. Association for Computational Linguistics, 2009.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 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. *ACL*, 2016.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*, 2014.

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*, pp. 1723–1732, 2015.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-based dependency parsing with stack long short-term memory. pp. 334—-343, 2015.

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

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

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *ACL*, 2015.

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

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

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A Smith. Distilling an ensemble of greedy dependency parsers into one mst parser. *EMNLP*, 2016.

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

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

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. *EMNLP*, 2015.

Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *ICLR*, 2016.

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

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inferencfne. *EMNLP*, 2016.

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*, pp. 801–809, 2011.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

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

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*, pp. 2773–2781, 2015.

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

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.

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.

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