# Tree-structured neural networks revisited

**Samuel R. Bowman**[*†] sbowman@stanford.edu **Jon Gauthier**[†‡] jgauthie@stanford.edu **Abhinav Rastogi**[†§] arastogi@stanford.edu

**Raghav Gupta**[†¶] rgupta93@stanford.edu **Christopher D. Manning**[*†¶] manning@stanford.edu **Christopher Potts**[*] cgpotts@stanford.edu

[*]Stanford Linguistics [†]Stanford NLP Group [‡]Stanford Symbolic Systems
[§]Stanford Electrical Engineering [¶]Stanford Computer Science

**TODO:** Anonymize.

## Abstract

- TreeRNNs are great, don't work for big data.
- We present the transition-based TreeRNN implementation – makes efficient training possible, show that it helps relative to seq models.
- Further, attention has proven powerful for extending sequence models. We introduce constit-by-constit attention, and find it to be effective here, too.

## 1 Introduction

- Brief history of TreeRNNs
- Speed issues
- Attention issues

## 2 The transition-based sentence model

### 2.1 Intuition: Transition-based parsing

### 2.2 The model

**TODO:** [SB] Reframe in terms of Model 0 alone.

### 2.3 Model 0

Model 0, depicted in Figure 1a, is the simplest instantiation of our design, using only a conventional stack and a learned composition function to incrementally build a sentence representation over a series of timesteps. For a sentence of $N$ words its input is a sequence of $2N-3$ inputs. These inputs can take two types. At some timesteps, the input will be a word embedding vector. This triggers the SHIFT operation, in which the vector is pushed onto the top of a stack. At other timesteps, the input will be the special REDUCE token, which triggers the reduction operation. In that operation, the top two word vectors are popped from the stack, fed into a learned composition function that maps them to a single vector (in the simplest case, this is a single neural network layer), and then pushed back onto the stack.

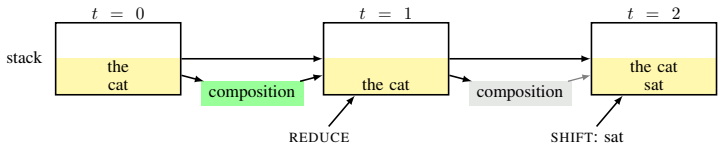
If we add no additional features, this model computes the same function as a plain TreeRNN. However, we expect it to be substantially faster than conventional TreeRNN implementations. Unlike a TreeRNN, the Model 0 computation graph is essentially static across examples, so examples of varying structures and lengths can be batched together and run on the same graph in a single step. This simply requires ensuring that the graph is run for enough timesteps to finish all of the sentences. This involves some wasted computation, since the composition function will be run $2N-3$ times (with the output of composition at non-REDUCE steps discarded), rather than $N-1$ times in a TreeRNN. However, this loss can be dramatically offset by the gains of batching, which stem from the ability to exploit highly optimized CPU and GPU libraries for batched matrix multiiplication.

### 2.4 Model 1

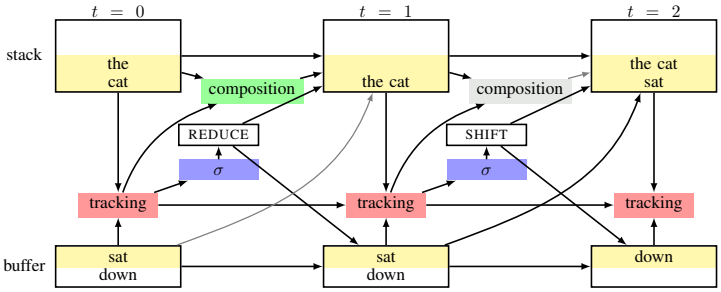
Model 1, depicted in Figures 1b and 1c, adapts Model 0 to use a stack and a buffer, making it more closely resemble a shift–reduce parser, and laying the groundwork for a model which can parse novel sentences at test time.

The model runs for a fixed number of transition steps: $2N-3$. In its starting configuration, it contains a stack that is prepopulated with the first two words of the sentence (since SHIFT SHIFT is the only legal operation sequence for the first two timesteps of a true shift-reduce parser), as well as a buffer (a queue) prepopulated with all of the remaining words in the sentence. Both the stack and buffer represent words using their embeddings.

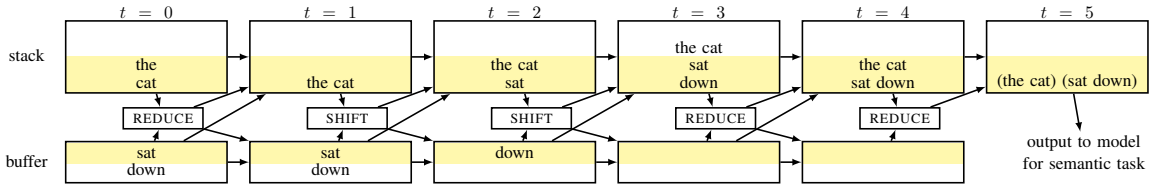
At each timestep at test time, the model combines views of the stack and buffer (the top ele-




(a) The Model 0 network unrolled for two transitions on the input *the cat sat down*. The operations at the bottom of the figure are given as inputs to the models, as is the start state of the stack at $t = 0$.




(b) The Model 1/2 network unrolled for two transitions on the input *the cat sat down*. The start state of the stack and buffer at the initial step $t = 0$ are the sole inputs to the model.




(c) The fully unrolled Model 1/2 network for *the cat sat down* with some layers omitted for clarity.

Figure 1: Two views of Models 1 and 2 (which use equivalent model graphs). In both views, the lower boxes represent the input buffer, and the upper boxes represent the stack. Yellow highlighting indicates which portions of these data structures are visible to the tracking LSTM and to the composition function. Thin gray arrows indicate connections which are blocked by a gating function, and so contribute no information.

ment of the buffer and the top two elements of the stack, highlighted in yellow) as the input to a tracking LSTM (red). This LSTM's output is fed into a sigmoid operation classifier (blue) which chooses between the SHIFT and REDUCE operations. If SHIFT is chosen, one word embedding is popped from the buffer and pushed onto the stack. If REDUCE is chosen, the buffer is left as is, and the top two elements of the stack are popped and composed using a learned composition function (green), with the result placed back on top of the stack.

**Supervision** The model is trained using two objective functions simultaneously. The semantic objective function is computed by feeding the value from the top of the stack at the final timestep—the full sentence encoding—into a downstream neural network model for some semantic task, like a sentiment classifier or an entailment classifier. The gradients from that classifier propagate to every part of the model except the operation classifier (blue). The syntactic objective function takes the form of direct supervision on the operation classifier (blue) which encourages that classifier to produce the same sequence of operations that an existing parser would produce for that sentence. The gradients from the syntactic objective function propagate to every part of the model but the downstream semantic model.

At training time, following the strategy used in LSTM text decoders, the decisions made by the operation classifier (blue) is discarded, and the model instead uses the correct operation as specified in the (already parsed) training corpus. At test time, this signal is not available, and the model uses its own predicted operations.

### 2.5 Model 2

Model 2 makes a small change to Model 1 that is likely to substantially change the dynamics of learning: It uses the operation sequence predicted by the operation classifier (blue) at training time as well as at test time. It may be possible to accelerate Model 2 training by initializing it with parameters learned by Model 1.

By exposing Model 2 to the results of its own decisions during training, we encourage it to become more robust to its own prediction errors. Bengio et al. 2015 applied a similar strategy[1] to an image captioning model. They suggest that the resulting model can avoid propagating prediction errors through long sequences due to this training regime.

### 2.6 Tracking left context with an auxiliary LSTM

Includes both the tracking LSTM and context-sensitive composition.

### 2.7 Building a model

**The size of the stack** The size of the stack should be $N$ for sentences of $N$ words, in case the first reduce merges the final two words. The size of the buffer should be $N$.

**Handling overly long sentences**

### 2.8 Data preparation

For Models 0–3, all training data must be parsed in advance into an unlabeled binary constituency tree. In addition, Model 0 requires that parses be available at test time as well. For both SST and SNLI we use the parses included with the corpus distributions whenever parses are needed.

For model 0, training data can be prepared by linearizing the provided parse, then deleting left brackets and replacing right brackets with REDUCE instructions. That is demonstrated here with the example sentence *the cat sat down*:

( ( the cat ) ( sat down ) )⇒
the cat REDUCE sat down REDUCE REDUCE

The input for models 1–4 is simply the word sequence from the parse, with the first two words moved into the stack. The syntactic supervision labels for models 1–3 are simply a binarized version

[1]The authors experiment with several strategies which interpolate between oracle-driven training and oracle-free training (Models 1 and 2 in our presentation, respectively). It may be useful to adopt a similar interpolating approach.

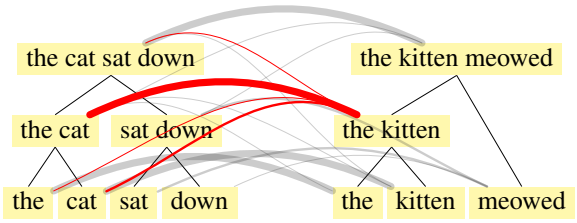



Figure 2: Soft alignments between the nodes of two trees, with alignments to "the kitten" highlighted in red.

of the Model 0 inputs, with the first two tokens (which are necessarily SHIFT SHIFT) omitted:

( ( the cat ) ( sat down ) )⇒
stack: ⟨the, cat⟩
buffer: ⟨sat, down⟩
ops: REDUCE SHIFT SHIFT REDUCE REDUCE

**Representing the stack in memory** Or, thin stack. **TODO:** [JG] Explain.

**Optimization and hyperparameters**

**Software infrastructure** We will make Theano code available.

### 2.9 Step time

**TODO:** Does anyone know of a better baseline TreeRNN implementation that we can use on Jagupard? We can use the CoreNLP SST model, but using a Java model as a baseline seems worrying unless we're guaranteed that it's competitively fast.

Comparing model step time to the plain RNN of Li et al. 2015. We use the small Pang and Lee 2004 sentiment corpus that they use, and train with identical hyperparameters: ...

Evaluation metrics: Time per minibatch on a jag machine, with and without GPU access.

## 3 Attention over trees

**Constituent-by-constituent attention** We aim to build on the results of Rocktäschel et al. 2015 and Wang and Jiang 2015, who find that neural soft attention models is an extremely effective technique for learning natural language inference. In particular, both papers use versions of word-by-word entailment, in which a latent alignment is learned from every word in the hypothesis to one or more words of the premise. We propose to borrow this basic idea, but to adapt it to a tree-structured setting, proposing *constituent-by-constituent* attention. While these models do

attention over a matrix $\mathbf{Y}$ of word-in-context representations from the premise encoder, we will perform attention instead over our own primary data structure, $\mathbf{Y}^{st}$, the matrix of vectors that have appeared at the top of the stack during premise encoding, which correspond one-to-one to the constituents in the tree structure representing the premise. Similarly, while the previous models perform one instance of soft attention conditioned on each word in the hypothesis, we perform one instance of soft attention conditioned on each stack top in the hypothesis encoder, representing the constituents of the hypothesis tree.

In our model, attention is performed at each step $t$ of the premise encoder. At step $t$, the query vector that drives attention will be $S_0^t$, the top of the stack.

**TODO:** [AR, RG] Write up the attention equations that you use.

### 3.1 Accumulating the results of attention

**Accumulating results sequentially** (After Rocktschel, Wang and Jiang)

**Accumulating results using a second tree layer**

## 4 Experiments

**The Stanford Natural Language Inference task** (Bowman et al., 2015)

**Results**

## 5 Discussion

**Patterns in the examples that the tree model got right that the LSTM didn't**

**Induced alignments**

## 6 Conclusions and future work

- Future work: Incorporate the parser
- Future work: Latent parses

**Acknowledgments**

(some of) the Tesla K40(s) used for this research was/were donated by the NVIDIA Corporation.

## References

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

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

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

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proc. ACL*, page 271. Association for Computational Linguistics.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. 2015. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*.

Shuohang Wang and Jing Jiang. 2015. Learning natural language inference with lstm. *arXiv preprint arXiv:1512.08849*.