NSHRDLU:*
Work in progress on the joint learning of parsing and semantic encoding

Samuel R. Bowman[*†]
sbowman@stanford.edu

Jon Gauthier[†‡]
angeli@stanford.edu

Christopher D. Manning[*†§]
manning@stanford.edu

Christopher Potts[*]
cgpotts@stanford.edu

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

1 Introduction

This project aims to use ideas from greedy transition-based parsing to build neural network models that can jointly learn to parse sentences and to use those parses to guide semantic composition.

Table 1 shows the sequence of model designs that we plan to build, and Figure 1 depicts some representative models.

We see three reasons to pursue this approach:

- Simply adapting a greedy transition-based approach to sentence encoding makes it possible to exploit semantic compositionality in the same manner as in a TreeRNN, but using a static graph structure that can take advantage of existing neural network libraries like Theano for both automatic differentiation and for highly optimized matrix computations on both CPUs and GPUs. Model 0 pursues this property directly, and all subsequent models share it.

- When a parsing system and an interpretation system are trained jointly and forced to share representations, it is likely that the performance of both models will benefit: the semantic composition function will have better access to syntactic type information that could provide additional evidence on the functional behavior of rare words, and the syntactic parser will have access to information about the semantic interpretability of constituents, making semantically-conditioned parsing decisions like PP attachment easier to learn. Models 1–3 have this property.

- When a system that uses constituent structure information as a latent variable is initialized from scratch and trained solely to perform some semantic task, it will likely learn some coherent syntax for the genre of language on which it is trained. If this syntax is reasonably stable across similar training sets and across random initializations of training, it can reasonably be said to be latent in the text itself, offering a new kind of evidence about natural language syntax. Model 4 aims to collect this kind of evidence.

All five models function as sentence encoding models: their inputs are sentences (input as sequences of words, with the help of a learned embedding matrix), and their outputs are single sentence encoding vectors which can be used as the inputs to downstream models for tasks like sentiment analysis, translation, or inference. Provided that these downstream models are differentiable—as is the case for neural network models and simpler regression models—their gradient signals can be used to train the sentence encoding models.

2 Models

2.1 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

*Name is provisional, and stolen from Geoff Hinton.

Name	Stack Representation	Input Representation	Ops Classifier	Op Predictions Used in Training
Model 0	Discrete	Op. sequence	N	–
Model 1	Discrete	**Discrete Buffer**	**Y: Directly supervised**	**N**
Model 2	Discrete	Discrete Buffer	Y: Directly supervised	**Y**
Model 3	**Soft**	**Soft Buffer**	Y: Directly supervised	Y
Model 4	Soft	Soft Buffer	**Y: Indirectly supervised**	Y

Table 1: Model variants, ordered by increasing reliance on learning. Bolding indicates the differences between each model and its parent model.

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.

2.2 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.

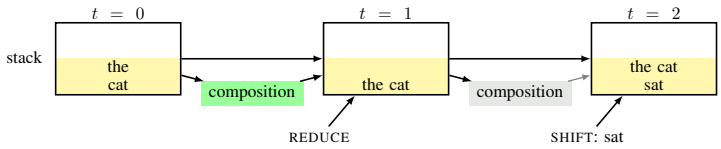
At each timestep at test time, the model combines views of the stack and buffer (the top element of the buffer and the top two elements of the stack, highlighted in yellow) as the input to a tracking LSTM (red). This LSTM's output is fed into a sigmoid operation classifier (blue) which chooses between 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.

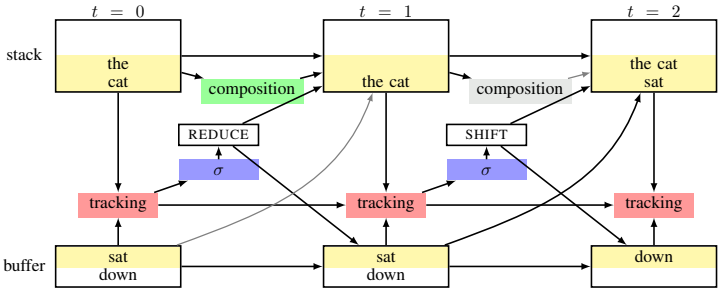
At training time, following the strategy used in LSTM text decoders, the decisions made by the operation classifier (blue) is discarded, and the model instead uses the correct operation as specified in the (already parsed) training corpus. At test time, this signal is not available, and the model uses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

2.3 Model 2

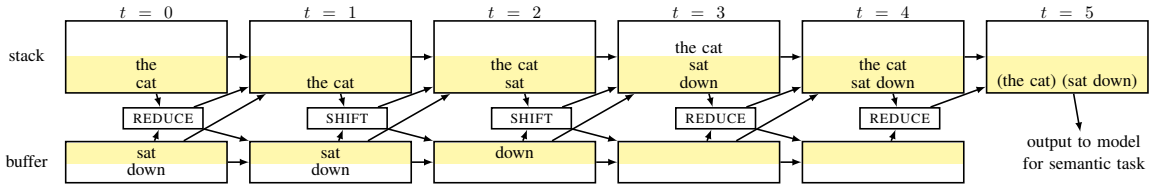
Model 2 makes a small change to Model 1 that is likely to substantially change the dynamics of learning: It uses the operation sequence predicted by the operation classifier (blue) at training time as well as at test time. It may be possible to accelerate Model 2 training by initializing it with parameters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



(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.

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. **?** 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.

2.4 Model 3

Model 3 modifies Model 2 by introducing the soft stack/soft queue from (**?**) in place of the hard, conventional stack and buffer. The soft stack makes it possible to for the model to predict smooth distributions over operations of the form (0.93 SHIFT, 0.07 REDUCE), instead of making hard decisions. These soft decisions allow for gradient information to flow from the stack and the buffer back into the operation classifier (blue). This is crucial to our ultimate goal, as it makes it possible for semantic considerations to influence the model's parsing decisions.

Model 3 still receives a direct supervision signal from some existing parser. In order to train the soft stack, we must represent the hard supervision signal from the parser by a soft prediction which matches the soft stack operation output. The supervision signal simply assigns 100% weight to the ground-truth operation.

2.5 Model 4

Model 4 modifies Model 3 by removing the direct supervision signal from the operation classifier (blue), instead forcing the operation classifier to learn solely from the gradient provided by the downstream supervision task. It may be possible

[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.

to accelerate or otherwise improve Model 4 training by initializing it with parameters learned by Model 3.

By removing an external parser signal, we allow Model 4 to fully exploit the soft stack representation. It is free to predict soft parse operations in the case of ambiguous parses. This distinguishes it from Model 3, which is encouraged to replicate the 100%-certain ground truth parse predictions provided by the external parser.

3 Other possible model features

3.1 Encoding the contents of the stack and buffer

The tracking LSTM (red) needs access to the top of the buffer and the top two elements of the stack in order to make even minimally informed decisions about whether to shift or reduce. It could benefit further from additional information about broader sentential context. This can be provided by running new LSTMs along the elements of each of the stack and the buffer (following **?**) and feeding the result into the tracking LSTM.

3.2 Contextually-informed composition

The composition function in the basic model (green) combines only the top elements of the stack, without using any further information. It may be possible to encourage the composition function to learn to do some amount of context-sensitive interpretation/disambiguation by adding a connection from the tracking LSTM (red) directly into the composition function.

For Model 0, no tracking LSTM is needed for the ordinary operation of the model, but it would be possible to add one for this purpose, taking as inputs the top two values of the stack at each time point and emitting as output a context vector that can be used to condition the composition function.

3.3 Typed REDUCE operations

Shift-reduce parsers for natural language typically operate with a restricted set of typed REDUCE operations (also known as "arc" operations). These operations specify the precise relation between the elements being merged. It would be possible to train any of the parse-supervised models (1–3) to learn such typed arc operations, expanding the op set dramatically to something like {SHIFT, REDUCE-NP, REDUCE-S, REDUCE-PP, ...} (in the case of constituency parse supervision). The model can then learn a distinct composition function depending on the relation of the two elements being merged.

4 Implementation notes

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 $N-2$.

4.1 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 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

5 Experiments

5.1 Step time

Comparing model step time to the plain RNN of **?**. We use the small **?** sentiment corpus that they use, and train with identical hyperparameters: ...

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

5.2 Sentiment

Learning to jointly parse and to predict logical relations between sentences over SST (**?**).

Evaluation metrics: accuracy, F1 (for all models but 0).

5.3 Natural language inference

Learning to jointly parse and to predict logical relations between sentences over SNLI (**?**).

Evaluation metrics: accuracy, F1 (for all models but 0).

6 Discussion

6.1 Inferred Model 4 parses?

Acknowledgments

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