Context Encoding LSTM
CS224N Course Project

Abhinav Rastogi
arastogi@stanford.edu
Supervised by - Samuel R. Bowman

December 7, 2015

Abstract

This project uses ideas from greedy transition based parsing to build neural network models that can jointly learn to parse sentences and use those parses to guide semantic composition. The model is used for sentence encoding for tasks like Sentiment classification and Entailment. The performance is evaluated on Stanford Sentiment Treebank(SST) and Stanford Natural Language Inference (SNLI) corpus.

1 Introduction

Sentence encoding models are used to obtain distributed representation of a sentence. These models map a sentence to a feature space and then a classifier operating on these features can classify the sentences for a particular task e.g, Sentiment classification, Question answering, Inference classification etc. A popular approach to obtain the distributed sentence representation is to start with a distributed representation of sentence tokens obtained from a word embedding and combine them, often using a recurrent architecture, to form a single vector.

Recurrent models like LSTM-RNN (Palangi, 2015), obtain the sentence encoding by combining the tokens in sequential order. On the other hand, tree structured models like Tree LSTM (Tai et al., 2015) combine the sentence tokens in an order dictated by the parse tree. Such models are a linguistically attractive option due to their relation to syntactic interpretations of the tree structures. However, such models generate a non-static graph structure (the tree structure depends on the sentence). The proposed model exploits the correspondence between a binary parse tree and stack based shift-reduce parsing to propose a model exploiting semantic compositionality as in a TreeLSTM whilst using a static graph structure that can take advantage of existing neural network libraries like Theano for both automatic differentiation and highly optimized matrix computations.

The proposed model can be easily adapted to jointly learn the parsing and interpretation when sentence parses for training data are not known.

2 Overview

This section describes the correspondence between shift-reduce parsing and a tree structured model through a simple example. The example also illustrates the working of the model from a high level without going much into details.

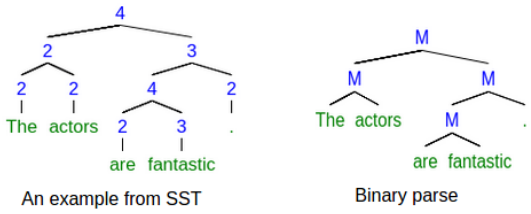
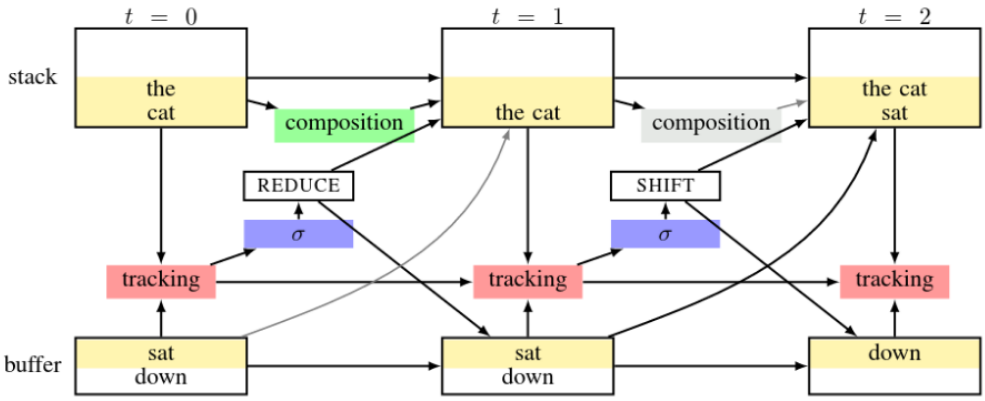


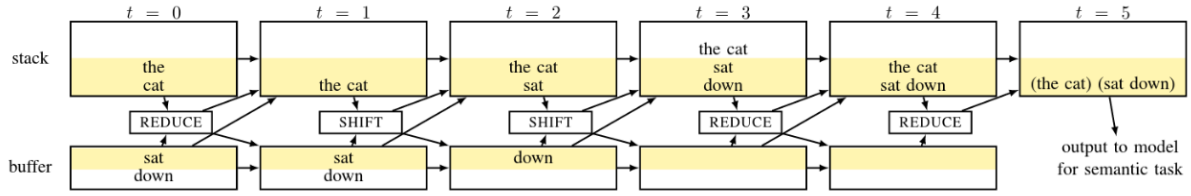
Figure 1: An example from Stanford Sentiment treebank (SST) containing the sentiment scores of intermediate tree nodes. These trees are preprocessed to obtain binary parse trees as shown on right

Instr.	Stack and Buffer
-	[], [”The”, ”actors”, ”are”, ”fantastic”, ”.”]
S	[”The”], [”actors”, ”are”, ”fantastic”, ”.”]
S	[”The”, ”actors”], [”are”, ”fantastic”, ”.”]
R	[”The actors”], [”are”, ”fantastic”, ”.”]
S	[”The actors”, ”are”], [”fantastic”, ”.”]
S	[”The actors”, ”are”, ”fantastic”], [”.”]
R	[”The actors are”, ”fantastic”], [”.”]
S	[”The actors are”, ”fantastic”, ”.”], []
R	[”The actors are”, ”fantastic .”], []
R	[”The actors are fantastic .”], []

Table 1: Sequence of operations corresponding to the parse tree shown in Figure 1 and the state of Stack and Buffer after each step is executed. Word (phrase) vectors have been represented as the word (phrase) they correspond to for clarity.



(a) 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 at test time.



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

Figure 2: 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 unit and to the composition unit. Thin gray arrows indicate connections which are blocked by a gating function, and so contribute no information.

If each node of a binary parse tree is considered to be a vector, the tree structure can be used to define an order in which these vectors should be combined. The procedure starts with the word embedding vectors corresponding to the leaf nodes. In a bottom-up fashion, the vectors corresponding to the children of a non-terminal node n can be composed together using a TreeLSTM unit to obtain the vector representation of the node n.

The same algorithm can be described recursively as- "To obtain the vector representation of a node n, obtain the vector representation of the left subtree l, then its right subtree r and then combine the representations of l and r using a TreeLSTM unit." This recursive definition is easily amenable to a stack based implementation as shown in Table 1. The model is initialized with an empty stack and a buffer consisting of word-vectors of all the tokens in the sentence. Two kinds of operations are supported on the Stack and the Buffer -

1. SHIFT - Pop one element from the top of the buffer and push it on the stack.
2. REDUCE - Pop top two vectors from the stack, use a TreeLSTM to combine them, and push the result back on the stack.

Starting with the state of the stack and the buffer as described above, the structure of a binary parse tree maps to a unique sequence of SHIFT and REDUCE operations. After executing these instructions, the element located on the top of the stack corresponds to the vector representation of the entire sentence.

3 Model Architecture

The proposed model has been depicted in Figure 2. It is a recurrent model which can be unrolled to K steps. The model consists of the following components-

- **Stack** - A $B \times D \times K$ tensor where B is the

batch size D is the dimension of the word embedding and K is the maximum number of transitions allowed. This tensor is initialized to zeros.

- **Buffer** - A $B \times D \times K$ tensor pre-populated with word-embeddings. The remaining slots which don't contain words are filled with zeros. The reason for keeping extra space is discussed later.
- **Tracking Unit** - This unit (depicted in red) combines views of the stack and buffer (the top element of the buffer and the top two elements of the stack, highlighted in yellow) from the previous time-step. This unit also has a recurrent connection from the previous time step. This connection is hoped to encode the "context" as described later. The output of the tracking unit is fed into a sigmoid operation classifier (blue) which chooses between the SHIFT and REDUCE operations.

  Two different implementations of the Tracking unit have been used in the experiments. (i) A Linear layer (ii) LSTM.
- **Composition Unit** - This unit (depicted in green) sees the top two elements of the stack and combines them into a single vector. This unit also has a recurrent connection from the previous time step which has not been depicted in Figure 2. This unit is made up of a TreeLSTM unit.

The dynamics of the model are very similar to a Shift-Reduce parser. At each step, the Tracking unit decides the transition to be made based on the inputs from the previous state. 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 the composition unit (green), with the result placed back on top of the stack.

However, there is a key difference between this model and Shift-Reduce (SR) parser, which is a greedy transition based parser. In SR parsing, the decision made by the tracking unit is solely based on the top two elements of the Stack and the top element of the Buffer. In the presented model, recurrent connections exist between the Tracking unit. It is hoped that these connections can convey the information regarding the tokens observed in previous steps and hence the predicted transitions are no longer greedy.

4 Data preparation

All training data must be parsed in advance into an unlabeled binary constituency tree. The models presented here don't need parses to be available during test time. The model has been tested for two datasets -

1. **Stanford Sentiment Treebank (SST)** - The training data consists of a binary parse of 8544 sentences corresponding to movie reviews. The sentences are classifier into 5 sentiment classes 0 to 4, 0 being a highly negative and 4 being a highly positive sentiment.
2. **Stanford Natural Language Inference Corpus (SNLI)** - It consists of binary parse of 550,153 sentence pairs. Each sentence pair belongs to one of the three classes - Contradiction, Entailment or Neutral.

For both SST and SNLI we use the parses included with the corpus distributions whenever parses are needed. A pre-trained word embedding (GloVe) is used for obtaining the distributed word representations. The words in the training data are then replaced by the word indices in the embedding. The parse structure is then decomposed into two lists - one corresponding to the list of token indices as seen in the sentence from left to right and second corresponding to the sequence of transitions obtained from the parse structure.

For example, the parse tree "((the cat) (sat down))" is converted to the token index sequence [1, 45, 9073, 4949] where the four indices correspond to the four words and the transition sequence [0, 0, 1, 0, 0, 1, 1], where 0 corresponds to SHIFT and 1 to REDUCE.

For efficient batching, it is desirable that all token index sequences be of same length M and all transition sequence should be of same length K across all examples. For evaluation, $K = 100$ has been used while training and $K = 150$ while testing. Sentences having fewer than K transitions are padded with SHIFT transitions in the beginning and special NULL tokens having a token index of 0 are inserted in the beginning of the token

index list. If the number of transitions in a sentence is larger than K, it is cropped and the corresponding word indices are also removed.

In a sentence of x words, a correct binary parse is expected to contain x SHIFT operations and $x - 1$ REDUCE operations. However, during test time, the model predicts its own transitions, hence this constraint is not guaranteed to hold. So, $M = K$ is chosen to protect the model from breaking down in case the tracking unit wrongly predicts all the transitions to be SHIFT.

5 Training

5.1 Supervision

The model is trained using two objective functions simultaneously - **(i) Semantic objective function** - It 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). **(ii) Syntactic objective function** - It 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. The gradient updates are propagated using Stochastic Gradient Descent using RMS Propagation strategy.

5.2 Training configuration

The presented model has been trained in two different configurations described below. The evaluation phase is the same for both these models.

1. **Model 1** - 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. **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.

   Since Model 2 is exposed to the results of its own decisions during training, it is encouraged 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.

5.3 Regularization

Regularization is a standard technique used to prevent high capacity models from over-fitting the training data by restricting the degrees of freedom of the weights. Two standard regularization techniques have been used - (i) L2 regularization (ii) Dropout - In the connections from the word embedding and within the semantic task classifier

6 Results

	Classification Accuracy	Transition Accuracy	Training Updates
Model 1	0.4854	0.6967	110000
Model 2	0.4462	0.6410	46500

(a) SST

	Classification Accuracy	Transition Accuracy	Training Updates
Model 1	0.727183	0.675695	48900
Model 2	0.652045	0.745967	22000

(b) SNLI

Table 2: Performance of the two models on SST and SNLI validation data. These are the best results across several hyper-parameter settings.

Table 2 presents the results for Model 1 and 2 on the validation set of SST and SNLI data. A boolean dataset consisting of random boolean expressions made up of AND and OR operators was

[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).

used for debugging purposes and the two models converged fairly quickly on it, achieving a transition accuracy of > 0.98 in 750 updates.

For five class classification task on SST corpus, the state of the art classification accuracy is 0.51 using Constituency Tree LSTM (Tai et al., 2015). However, this model requires sentence parse to be present at test time (produced using Stanford PCFG parser). For SNLI, the state of the art classification accuracy for non-attention based models is 0.776 using LSTM encoders (Bowman et al., 2015).

7 Experiments and Error Analysis

7.1 LSTM vs Linear Layer in Tracking unit

Figure 3 shows the learning curves (validation set classification accuracy vs number of updates) on SST data for two different implementations of the Tracking unit - Linear layer and LSTM. It is observed that LSTM is slower to train in the beginning but outperforms the Linear layer after training longer. This is expected because LSTM's are better suited to learn long-term dependencies as they combine the states additively. The plots cor-

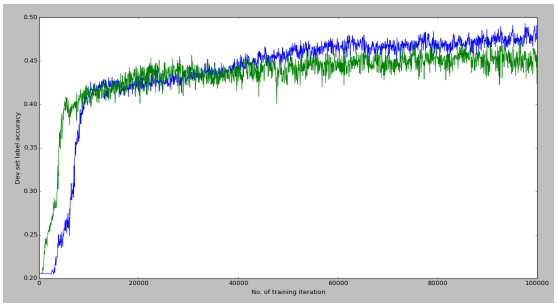


Figure 3: The learning curves for tracking unit made up of Linear layer (green) and LSTM (blue)

respond to the same dimensionality

7.2 Parsing Performance

The quality of binary parse obtained by the two models can be quantified using the metrics listed below. These metrics are calculated using a binary bracketed data evaluation tool called Evalb.The results obtained for SST dev set are shown in Table 3. The parsing results for SNLI were observed to be quite poor (they even broke the evaluation tool) despite having a decent transition accuracy.

- **Bracketing precision** = (Number of correct constituents) / (Total number of constituents) Complete match
- **Average crossing** = (Number of constituents crossing in a gold constituent) / (Total Number of sentences)
- **2 or less crossing** = ratio of sentences having ≤ 2 crossing brackets

Metric	Model 1	Model 2
Valid Sentence	0.9891	0.4727
Bracketing Precision	0.6046	0.1879
Complete match	0.0340	0.0096
Average crossing	0.0722	0.1328
2 or less crossing	0.1892	0.0597

Table 3: Various parsing metrics for SST dev set containing 1101 sentences.

These results also show that there is not a clear relation between transition and parsing accuracy. It seems that not all transitions are equally important in determining the accuracy of the parse.

7.3 Label Confusion

The label confusion matrix for SST data using Model 1 is shown in Table 4. The matrix lists the counts of (original label, predicted label) pairs and is helpful in visualizing the label pairs that are confused more frequently. As the numbers indicate, the most confused sentiment labels for a label l are $l-1$ or $l+1$. This indicates that the mistakes are not too severe.

	0	1	2	3	4
0	46	76	7	10	0
1	28	183	43	33	2
2	7	91	68	61	2
3	2	44	35	171	27
4	0	8	12	84	61

Table 4: Label confusion matrix for SST Model 1 on dev set. The rows correspond to original labels and the columns correspond to predicted labels

Table 5 shows the confusion matrix for SNLI dev set obtained using Model 1. It is observed that the wrongly predicted labels are almost uniformly distributed among the other two labels.

	Entail	Neutral	Contradict
Entail	2556	453	320
Neutral	574	2220	441
Contradict	398	580	2300

Table 5: Label confusion matrix for SNLI Model 1 on dev set. The rows correspond to original labels and the columns correspond to predicted labels

7.4 Variation with Sentence Length

Table 6 shows the variation of parsing metrics with maximum allowed sentence length. This analysis is useful in figuring out how does the model perform on longer sentences. It is expected that the model will lose performance as the sentence length is increased because the longer sentences require the network to propagate the information across larger timesteps.

The obtained results are as expected. The percentage of valid sentences drops, percentage of complete matches drops, average crossing increases and percentage of sentences having ≤ 2 crossing brackets also drops. So all metrics except the bracketing precision seem to be getting worse as the sentence length increases.

Metric	10	20	40
No. of sentences	200	633	1086
Valid Sentence	1.000	0.994	0.989
Bracketing Precision	0.667	0.629	0.697
Complete match	0.155	0.057	0.035
Average crossing	0.021	0.045	0.070
2 or less crossing	0.645	0.316	0.192

Table 6: Variation of parsing metrics for SST dev set with the maximum allowed sentence length (10, 20 and 40)

8 Future Work

There are several directions of future work we have in mind-

- **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 (Dyer et al., 2015)) and feeding the result into the tracking LSTM.

- **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.

- **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 the parse-supervised models 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.

- **Differentiable Stack and Buffer** - If the stack and buffer in the model can be made differentiable (Grefenstette et al., 2015), gradients may be propagated through them. It will be interesting to see what the model learns when fractional push and pop instructions are allowed.

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.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China, July. Association for Computational Linguistics.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. *arXiv preprint arXiv:1506.02516*.

Hamid Palangi, et al. 2015. Deep sentence embedding using long short-term memory networks. *arXiv*.

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