Extensions to Tree-Recursive Neural Networks for Natural Language Inference

Raghav Gupta
Department of Computer Science
Stanford University
rgupta93@stanford.edu

Nihit Desai
Department of Computer Science
Stanford University
nihit@stanford.edu

Abstract

Understanding textual entailment and contradiction is considered fundamental to natural language understanding. Tree-recursive neural networks, which exploit valuable syntactic parse information, achieve state-of-the-art accuracy among pure sentence encoding models for this task. In this course project for CS224D, we explore two extensions to tree-recursive neural networks - deep TreeLSTMs and attention mechanisms over TreeLSTMs - and evaluate our models on the Stanford Natural Language Inference (SNLI) corpus. Our best models show $\sim$2% improvement in classification accuracy compared to a pure sentence TreeLSTM encoder baseline.

1 Introduction

Natural Language Inference (NLI) or Recognizing Textual Entailment (RTE) is a popular natural language processing task. It is viewed as an important step towards achieving true natural language understanding (Katz 1972), via learning to determine the semantic relationship between two sentences. More specifically, it is the task of determining whether two natural language sentences are contradicting each other, not related, or whether the first sentence (the *premise*) entails the second sentence (the *hypothesis*). In its current form, it is conveniently modeled as a 3-way classification problem with two inputs - the premise and hypothesis sentences. Predictably, building good models for NLI can be very helpful for NLU/NLP tasks including but not limited to machine comprehension/question answering, text summarization, and machine translation.

Much like the Vauquois triangle for machine translation, traditional approaches to NLI have incorporated varied levels of sentence understanding, right from approaches deriving ideas from propositional logic and theorem provers, to hand-crafted feature-based classifiers relying on lexical and paraphrase matching. With the release of a new large dataset for NLI (see Section 3), it has become possible to train neural network models for NLI which have pushed the state of the art by at least 10% in accuracy). Most neural network models for NLI rely on learning distributed vector representations of sentence meaning using word vector representations. This final sentence representation is then used for the 3-way classification task. Common sentence encoders include sequence-based models such as Recurrent Neural Networks with Long Short-Term Memory (Hochreiter and Schmidhuber 1997) which accumulate information over the sentence sequentially, and tree-recursive neural networks (Socher et al. 2011) which propagate information up a binary parse tree.

Tree-RNNs are a more principled choice to combine vector representations, since meaning in sentences is known to be constructed recursively according to a tree structure. Our goal in this project is to explore linguistically-informed deep learning techniques that have been separately applied to this task, namely neural attention and tree-recursive neural networks (see Section 2 for descriptions), and build and experiment with new models incorporating these ideas. Specifically, we extend tree-recursive neural networks in two ways - neural attention and stacking. We evaluate the performance of our models on the Stanford Natural Language Inference (SNLI) dataset, and find that our best models show $\sim$2% improvement in classification accuracy compared to a pure sentence TreeLSTM encoder baseline.

2 Related Work

We here describe the major approaches taken on SNLI, starting from handcrafted classifiers to deep learning models, in addition to recursive LSTM networks/TreeLSTMs and their use in semantic tasks in general.

2.1 TreeRNNs for Semantic Tasks

TreeRNNs, introduced in Socher et al. 2011 and extended to use LSTM-like composition functions as TreeLSTMs in Tai, Socher, and C. Manning 2015, report improvements over vanilla LSTMs on many semantic tasks. Irsoy and Cardie 2014 present improvements by stacking multiple layers of tree-structured vanilla recursive NNs on sentiment classification. However, for naive implementations of the aforementioned models, each training example requires a different computational graph to run a tree-recursive NN on a sentence - preventing batch computation and scalability for recursive NNs. Bowman, Gauthier, et al. 2016 report a method to batch TreeLSTMs computations, taking ideas from shift-reduce parsing, and add a sequential component to the model, yielding the state-of-the-art for pure sentence encoding models. Section 4 presents more details.

2.2 Deep Learning and SNLI

The Excitement Open Platform in Magnini et al. 2014 includes models for 2-way entailment classification, thus not directly applicable. The first feature-based results on 3-way SNLI are in Bowman, Angeli, et al. 2015, who train supervised classifiers on unlexicalized and lexicalized features, obtaining 50.4% and 78.2% accuracy respectively. They also report a pure sentence encoding model with a 100d LSTM encoder encoding each sentence separately, and the concatenated sentence embeddings passed through a 3-layer multi-layer perceptron (MLP) for classification, which gets 77.6% accuracy. A similar approach with 300d hidden states gets 80.6%. Mou et al. 2015 present another pure sentence encoding approach: they build dependency tree-structured composition for local feature extraction, but use simpler pooling techniques to build sentence features efficiently.

2.2.1 Neural Attention and SNLI

One view of neural attention is as a soft-search over past inputs/outputs to find the ones most relevant for later inputs. The first major work on attention models for NLP is Bahdanau, Cho, and Bengio 2014 on machine translation. For NLI, Rocktäschel et al. 2015 apply neural attention over a linear LSTM encoder to build a hypothesis representation conditioned on the premise, by running a linear RNN over the existing linear LSTM hypothesis encoder, where the RNN sees as inputs the hypothesis hidden state at the current time step and ”attends over” all the hidden states generated by the premise LSTM, in addition to its own previous ”attention state”. The final attention and hypothesis hidden states are composed and fed alone as input to a classifier. Wang and Jiang 2015 make this recurrent attention-state connection an LSTM instead of a vanilla RNN. Cheng, Dong, and Lapata 2016 add to this model by generating premise and hypothesis representations with intra-sentence attention i.e. attending over the hidden states of the already-processed tokens within the sentence.

3 Dataset

To evaluate our models on the NLI task, we use the publicly-available Stanford Natural Language Inference (SNLI) corpus from Bowman, Angeli, et al. 2015. The SNLI corpus is a collection of 570k English sentence pairs manually labeled with the labels entailment, contradiction, and neutral, supporting the task of natural language inference. Each sentence pair has the judgments of five turkers as well as a consensus judgment. All sentences come with parses from the Stanford parser from Chen and C. Manning 2014. Some example sentence pairs from this corpus, with the associated consensus labels, are given in Table 1. This corpus is around two orders of magnitude larger than other natural language inference datasets, such as Sentences Involving Compositional Knowledge (SICK, Marelli et al. 2014). The larger size and good quality of SNLI make it a good candidate for training deep neural network models. Some statistics about this dataset are given below:

- Number of sentence pairs : 570152 (550152 train, 10000 dev, 10000 test)
- Number of classes : 3 (entailment, contradiction, neutral)
- Balanced classes: majority class is 37%
- Mean sentence lengths in tokens : premise - 14.1, hypothesis - 8.3

Premise	**Hypothesis**	**Judgment**
A black race car starts up in front of a crowd of people	A man is driving down a lonely road.	contradiction
A soccer game with multiple males playing	Some men are playing a sport.	entailment
A smiling costumed woman is holding an umbrella	A lady in a fairy costume holds an umbrella	neutral

Table 1: Example sentence pairs from the SNLI dataset

4 Approach

In this section we first briefly describe TreeLSTMs with tracking, the baseline model, and descriptions of our model extensions, namely deep TreeLSTMs and attention models over TreeLSTMs.

4.1 Basic model : TreeLSTMs with tracking

Bowman, Gauthier, et al. 2016 present a TreeLSTM implementation capable of batching (see 2.1). Specifically, their model lays out the nodes of a parse tree like in a postorder traversal and, at each time step, pushes word representations onto a stack or composes two node representations (those of the left and right children) and pushes this onto the stack, mimicking the push-merge sequence used to construct the original parse tree. The model's operation ensures that the top two elements on the stack at the time of a merge are the left and right children of the current node. They also include a low-dimensional sequential tracking LSTM which receives as input at each time step the 2 top representations in the stack, and the top word embedding in the buffer, conditioning the merge operation of the TreeLSTM with more context. Equations 1-3 describe the composition function for this model. Here, the TreeLSTM hidden states of the left and right children of the current node are denoted as $\vec{h}_l$ and $\vec{h}_r$, and $\vec{e}$ is the sequential tracking LSTM's hidden state.

$$\begin{bmatrix} \vec{i}^{(\eta)} \\ \vec{f}_l^{(\eta)} \\ \vec{f}_r^{(\eta)} \\ \vec{o}^{(\eta)} \\ \vec{g}^{(\eta)} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left(W_{\text{comp}} \begin{bmatrix} \vec{h}_l^{(\eta)} \\ \vec{h}_r^{(\eta)} \\ \vec{e} \end{bmatrix} + \vec{b}_{\text{comp}} \right) \tag{1}$$

$$\vec{c^{(\eta)}} = \vec{f}_l^{(\eta)} \odot \vec{c}_l^{(\eta)} + \vec{f}_r^{(\eta)} \odot \vec{c}_r^{(\eta)} + \vec{i}^{(\eta)} \odot \vec{g}^{(\eta)} \tag{2}$$

$$\vec{h}^{(\eta)} = \vec{o}^{(\eta)} \odot \vec{c}^{(\eta)} \tag{3}$$

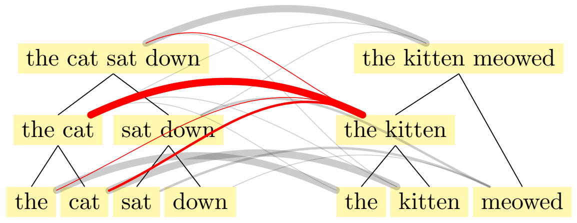
4.2 Deep TreeLSTMs

Our first extension to this model is to stack multiple TreeLSTM layers on top of each other, similar to Irsoy and Cardie 2014 stacking multiple TreeRNN layers. Our motivation for this model is to be able to capture different aspects of compositionality in language, with a deeper model, as has been shown in prior work for deep recursive as well as recurrent NN models. Equations 4-6 describe this model, where η is the layer number. Specifically, at layer η, the inputs to the composition function are the TreeLSTM hidden states of the left and right child of the current node at layer η, and the current node's representation from layer $\eta - 1$. The latter input is replaced by the tracking LSTM hidden state $\vec{e}$ for the lowest TreeLSTM layer. The highest-layer representation of the parse tree root is taken as the sentence embedding. Figure 1b illustrates the structure of a 2-layer TreeLSTM. Due to a lack of time, we train and evaluate models with upto 2 TreeLSTM layers only.

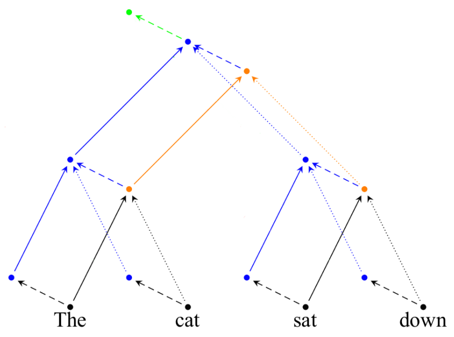
$$\begin{bmatrix} \vec{i}^{(\eta)} \\ \vec{f}_l^{(\eta)} \\ \vec{f}_r^{(\eta)} \\ \vec{o}^{(\eta)} \\ \vec{g}^{(\eta)} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left(W_{\text{comp}} \begin{bmatrix} \vec{h}_l^{(\eta)} \\ \vec{h}_r^{(\eta)} \\ \vec{h}^{(\eta-1)} \end{bmatrix} + \vec{b}_{\text{comp}} \right) \tag{4}$$

$$\vec{c^{(\eta)}} = \vec{f}_l^{(\eta)} \odot \vec{c}_l^{(\eta)} + \vec{f}_r^{(\eta)} \odot \vec{c}_r^{(\eta)} + \vec{i}^{(\eta)} \odot \vec{g}^{(\eta)} \tag{5}$$

$$\vec{h}^{(\eta)} = \vec{o}^{(\eta)} \odot \vec{c}^{(\eta)} \tag{6}$$



(a) Illustration of constituent attention for premise *"the cat sat down"* and hypothesis *"the kitten meowed"*. The width of premise-hypothesis connections show the contribution of the premise node in the attention representation of the hypothesis node, as learned by the LSTM. The connections for *"the kitten"* are in red



(b) Deep Tree-LSTMs on the sentence *"The cat sat down"*. The black/orange nodes denote layer 0 TreeLSTM representations, and the blue nodes layer 1 TreeLSTM representations. The top blue node is the final sentence embedding

Figure 1: Illustrations of our extensions to TreeLSTMs

4.3 Attention over constituents

Our main motivation is to extend the benefits of neural attention to phrase pairs instead of just word pairs in the premise and the hypothesis. In effect, current attention models, for each hypothesis token, compute its hidden state's similarity with all premise hidden states, corresponding to premise tokens. This enables an attention model to identify words which are semantically related (but possibly entailing or contradicting) in the premise and the hypothesis. For instance, for a premise-hypothesis pair *"The cat sat down"* and *"The kitten meowed"*, an attention model enables the identification of the entailment relation between *'cat'* and *'kitten'* and also the potentially contradictory relation between *'sat'* and *'meowed'*. Figure 1a shows an illustration of the same.

TreeLSTM computation can be viewed as a sequence of operations applied to the representations at each node in the constituency parse of the premise/hypothesis. If a sentence has N tokens, its constituency parse will have $2N-1$ nodes, each corresponding to a TreeLSTM time step, and also a phrase in the underlying sentence. Attention over TreeLSTM constituent representations, hence, can match hypothesis phrases to premise phrases.

Specifically, to perform *constituent-by-constituent* attention, we attend over Y^{st}, the matrix of representations that have appeared at the top of the stack during premise encoding, one for each premise constituent (as opposed to prior models doing attention over a matrix Y of word-in-context representations from the premise encoder), 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. It is worth noting here that these models strictly operate on the TreeLSTM encodings, final and intermediate, as inputs i.e. they are wholly on top of the TreeLSTM encoder. For either attention model, we use only the final hypothesis attention state for classification, discarding the premise or hypothesis pure sentence encodings.

4.3.1 Linear Constituent Attention Model (LinearAtt)

Roughly, our first attention model is an adaptation of Wang and Jiang 2015 to constituent attention. Let $Y^p \in \mathbb{R}^{k \times L_s}$ be the matrix of all premise constituent representations. The premise contains L_s constituents and k is the model dimension. Furthermore, $e_{L_s} \in R^{L_s}$ is a vector of 1s, and $\vec{h}_t$ is the hypothesis LSTM hidden state at time step t ($t \in (1, L_t)$; hypothesis has L_t tokens). For each hypothesis constituent x_t, the attention mechanism generates an attention weight-vector α_t over all premise constituents, and consequently attention-weighted representations of the premise for each hypothesis constituent. The attention state, $\vec{a}_t$, calculated as in equations 7-9, forms a recurrent connection since it depends on the attention state from the previous time step. The LSTM composition function takes the weighed premise representation $Y^p\vec{\alpha}_t^T$ as its input, and reuses its own previous state (exact equations similar to literature and skipped for brevity). W^p, W^h, W^a, w and the $LSTMComposition$ parameters are all trained.

$$M_t = tanh(W^p Y^p + (W^h \vec{h}_t + W^a \vec{a}_{t-1}) \otimes e_{L_s}), M_t \in \mathbb{R}^{k \times L_s} \tag{7}$$

$$\vec{\alpha}_t = softmax(w^T M_t), \vec{\alpha}_t \in \mathbb{R}^{L_s} \tag{8}$$

$$\vec{a}_t = LSTMComposition([Y^p \vec{\alpha}_t^T ; \vec{a}_{t-1}]), \vec{a}_t \in \mathbb{R}^k \tag{9}$$

4.3.2 Tree Constituent Attention Model (TreeAtt)

Given that we are dealing with tree-structured composition, a more natural choice for accumulating attention states is via a TreeLSTM running on top of the hypothesis encoding TreeLSTM. The previous model combines attention representations in parse tree post-order, whereas tree-attention follows the order of construction of the sentence encodings themselves. Reusing much of the notation from the previous section, the attention state is calculated as in equations 10-12. The TreeLSTM composition function is identical to the TreeLSTM composition defined in equations 1-3, except for the lack of an external state i.e. the tracking component. $\vec{a}_l$ and $\vec{a}_r$ denote the attention states for the left and right children respectively. $W^p, W^h, W^a_l, W^a_r, w$ and the $TreeLSTMComposition$ parameters are all trained.

$$M_t = tanh(W^p Y^p + (W^h \vec{h}_t + W^a_r \vec{a_r} + W^a_l \vec{a_l}) \otimes e_{L_s}), M_t \in \mathbb{R}^{k \times L_s} \tag{10}$$

$$\vec{\alpha_t} = softmax(w^T M_t), \vec{\alpha_t} \in \mathbb{R}^{L_s} \tag{11}$$

$$\vec{a_t} = TreeLSTMComposition([Y^p \vec{\alpha_t}^T ; (\vec{a_l}, \vec{a_r})]), \vec{r_t} \in \mathbb{R}^k \tag{12}$$

4.4 Classifying MLP

Once we contruct the sentence representation of premise and hypothesis, the next and final step is to classify the sentence representations/sentence pair representation into one of three classes. For this, we use a multilayer perceptron (MLP), the number of whose hidden layers is a hyperparameter. Our loss function is a softmax-cross-entropy loss over the 3 classes, added to the L2 loss for each of the trainable parameters.

Deep TreeLSTM: For Deep TreeLSTM (which is a pure sentence encoding model, hence the representations of premise and hypothesis are constructed independently), we run two copies of the model with shared parameters: one on the premise sentence and another on the hypothesis sentence. We then use their outputs (the $\vec{h}$ states at the top of the stack in the last layer) to construct a feature vector for the pair as follows (drawing from Mou et al. 2015):

$$\vec{x}_{\text{classifier}} = \begin{bmatrix} \vec{h}_{\text{premise}} \\ \vec{h}_{\text{hypothesis}} \\ \vec{h}_{\text{premise}} - \vec{h}_{\text{hypothesis}} \\ \vec{h}_{\text{premise}} \odot \vec{h}_{\text{hypothesis}} \end{bmatrix} \tag{13}$$

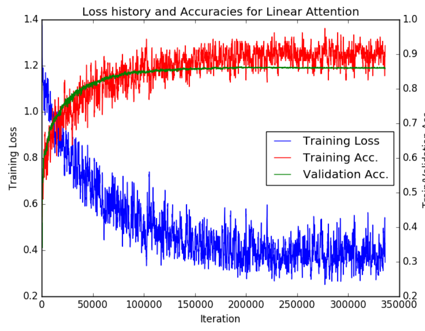
Neural Attention: In case of neural attention models (both tree attention and linear attention), we use the final attention representation of the hypothesis conditioned on the premise as feature vector for the classifier, ignoring the premise and hypothesis pure sentence encodings.

Following Bowman, Angeli, et al. 2015, this feature vector is then passed to a series of 1024D ReLU-activated neural network layers (i.e., a multilayer perceptron), and then passed into a affine transformation, and then finally passed to a softmax layer. The output of softmax layer gives a distribution over class labels. To regularize our model, we use dropout (Srivastava et al. 2014) and $L2$ regularization. Additionally, we use batch normalization (Ioffe and Szegedy 2015) as it has been shown to improve training speed of neural networks by reducing covariate shift.

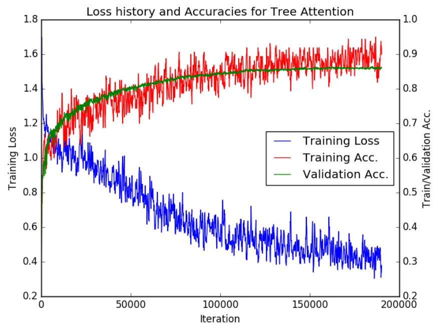
5 Implementation and Experiments

We extend the TreeLSTM implementation in Bowman, Gauthier, et al. 2016[1] with our models, written in Theano. The main reason for this is that the implementation allows batching and represents the state-of-the-art on pure sentence encodings with its tracking component. All our models use hidden and cell state dimensions 300, for a total model dimension of 600. For word embeddings, we use pretrained 300d GloVe embeddings (Pennington, Socher, and Manning 2014). We do not update these embeddings but instead learn a linear projection layer that projects these GloVe embeddings to a 600d space, including both the hidden and cell TreeLSTM states for the leaf nodes in the sentence parses.

[1] `https://github.com/stanfordnlp/spinn`



(a) Loss history and accuracies for LinearAtt



(b) Loss history and accuracies for TreeAtt

Figure 2: Training loss history and train/val accuracies during training for our attention models

For evaluation, we mainly plan to use training loss curves as a yardstick to ensure proper training of our NN models, and overall accuracy to compare models, and to check for over/underfitting, since

1. The SNLI dataset is fairly balanced (the largest class, neutral, is about 37% of the dataset)
2. Existing results in the prior-art are reported in terms of accuracy too

We use RMSProp (Dauphin et al. 2015) with $\rho = 0.9$ for parameter updates. The hyperparameters we tune, along with the ranges we try out for these, are as follows. The ranges have been arrived at by doing coarse grid search and some manual tuning after that. For each hyperparameter, we select values uniformly along a scale, mentioned for each of these hyperparameters. Finally, we randomly selected 12 combinations of these hyperparameters (4 for the Deep TreeLSTM owing to a resource crunch), and reported the test accuracies on the model with the best dev accuracy.

1. Learning rate : $(2e\text{-}4, 1e\text{-}2)$, log scale, decay 0.75 every 10k steps
2. L2 λ : $(1e\text{-}6, 1e\text{-}4)$, log scale
3. Dropout keep rate for word embeddings : (0.8, 0.95), linear scale
4. Dropout keep rate for classifying MLP : (0.8, 0.95), linear scale
5. Number of hidden layers in classifying MLP : 2 or 3, chosen randomly

Our models take approximately 20 epochs to converge, which translates to roughly 200k iterations, with a batch size of 32. Unfortunately, larger batch sizes turned out to be too heavy on memory requirements, and our jobs kept crashing owing to segfaults. On a single NVIDIA Tesla K40 GPU, each of our runs took about 6 days to train. Figure 2 shows the training loss and training and validation accuracies during the training of our LinearAtt and TreeAtt models. We used these plots mainly for debugging our neural network models and hyperparameter choices.

6 Results

In this section, we summarize and discuss our experimental results, presenting quantitative accuracies on SNLI, and some qualitative analysis of our models and the task in general.

Model $\downarrow$ Data $\rightarrow$	Paper	No. of params	Training	Development	Test
300D LSTM	Bowman, Gauthier, et al. 2016	3.0m	83.9	82.4	80.6
300D LSTM + Wang-Jiang Attention	Wang and Jiang 2015	1.9m	92.0	86.9	86.1
300D TreeLSTM with tracking	Bowman, Gauthier, et al. 2016	3.7m	89.2	84.8	83.2
300D 2-layer Deep TreeLSTM	Our result	5.1m	87.1	84.0	83.0
300D TreeLSTM + LinearAtt	Our result	5.0m	91.4	86.3	84.8
300D TreeLSTM + TreeAtt	Our result	5.9m	88.6	86.4	85.0

Table 2: Accuracy on SNLI for different models

6.1 Experimental Results & Discussion

We evaluate our new models on SNLI as described in Section 5, and summarize our results in Table 2. Note that our model extensions are on top of the TreeLSTM with tracking model introduced in Bowman, Gauthier, et al. 2016 and as such, we use their results (third row of 2) as our baseline. The first row describes results from a 300D LSTM encoder model, and the second row describes a linear attention model built on top of that. We include these figures to serve as reference points for the marginal gain attention has yielded for the NLI task on other models.

Our deep TreeLSTM model gets a test accuracy of 83.0%, which is a little less than the SPINN baseline of 83.2%. However, we still find this result promising, since we were able to launch only 4 runs for this model (owing to a crunch on computational resources), and hence we could not perform exhaustive hyperparameter optimization.

Just as with linear LSTMs (first two rows of 2), we find that attention models also yield improvements in case of tree-structured LSTMs albeit the magnitude of improvement is smaller. Our best attention models show $\sim$2% improvement in classification accuracy compared a pure sentence TreeLSTM encoder baseline. Additionally, we find that tree-attention mechanism marginally outperforms linear-attention. This is expected - since we are dealing with tree-structured composition, a more natural choice for accumulating attention states is via a TreeLSTM running on top of the hypothesis encoding TreeLSTM. While linear attention combines attention representations in parse tree postorder, tree-attention follows the order of construction of the sentence encodings themselves.

Even though our attention models outperform the baseline, we believe there is room for improving both our models - deep TreeLSTMs as well as attention models. Particularly, the size of our extended models ($\sim$5M parameters each) meant our models took longer and were harder to train compared to the baseline. Possibly, further hyperparameter optimization may yield even better results. Interestingly, the training and test accuracy gap strongly suggests overfitting, but when we tried a higher L2 λ range, it consistently reduced our model accuracy.

6.2 Qualitative Results & Error Analysis

Classwise performance: Figure 4a shows a t-SNE (Van der Maaten and Hinton 2008) plot of the sentence pair representations obtained by our deep TreeLSTM on the dev set. We observe that just with the elementwise difference and products of these sentence embeddings, contradiction and entailment class examples (red and green respectively) are fairly well separated, while the neutral class (blue) lies in between, and it can be hard to distinguish neutral examples from the others. This is also reflected by the classwise F1 scores- 0.8 for neutral & 0.86 for the others, and also by the confusion matrix in Figure 4b. Given the methodology used for annotating SNLI, the neutral class occupies everything not perfectly entailing or contradicting - a wider and more ambiguous region in the semantic space.

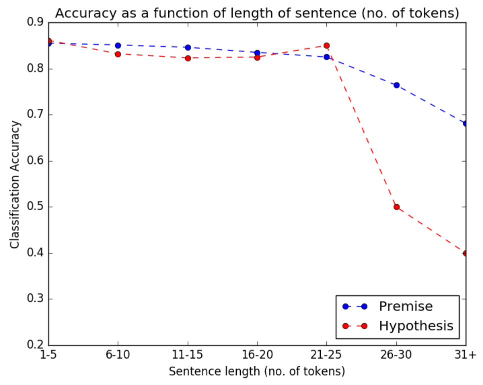
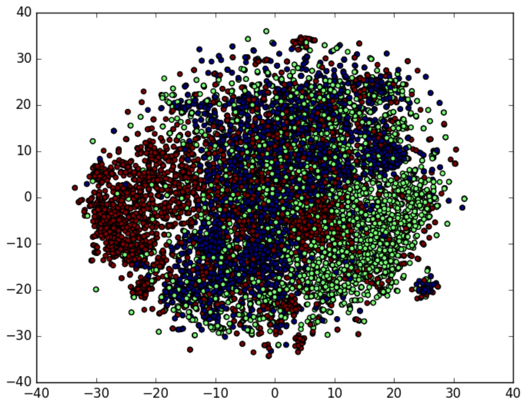


Figure 3: Classification accuracy as a function of number of tokens in premise and hypothesis

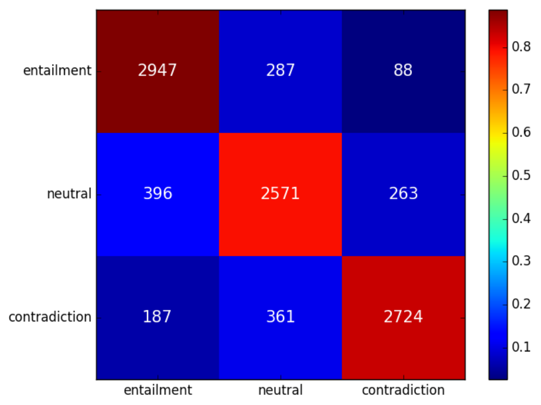
Error Analysis: We here describe some common types of errors that our models make. For instance, for the premise-hypothesis pair *"A black limousine in front of a marble building with a crowd of people."* and *"A black limousine is behind a marble building."*, actually contradiction but predicted as entailing, the system falls prey to the premise-hypothesis phrase overlap and predicts entailment, failing to capture a crucial modifier, often a single preposition/adjective or sometimes a noun; in this example, *'behind'* plays such a thwarting role. Most contradictory/neutral examples predicted as entailing belong to this category of errors.

Another common error is to classify contradictory examples as neutral. This probably arises from noise in the dataset, since the definitions of contradiction and neutral labels for sentence pairs as unrelated as *"A group of people are gathered in the woods around trees and a ladder."* and *"Miners work hard at gathering coal."* can be ambiguous for annotators. Another such example would be the pair *"A woman in a gray shirt working on papers at her desk."* and *"Lady showing result of anger on her boss scolding"*, labeled as contradiction but predicted neutral.

A third type of error we observed sentence pairs with long premise or hypothesis sentences (as measured by



(a) t-SNE visualization of the elementwise difference and product of the premise and hypothesis encodings produced on the development set. Points with labels blue, red and green represent neutral, contradicting and entailing examples



(b) Confusion matrix for the deep TreeLSTM model. Rows indicate true class, columns indicate predicted class

Figure 4: Visualizations of our NLI models on the SNLI development set

number of tokens). Figure 3 shows classification accuracy as a function of the number of tokens in the premise and hypothesis. We observe that our model performance in general decreases as the length of the premise and hypothesis increases. This is reasonable to expect because long-range dependencies have been hard to model in sentence encodings. TreeLSTMs have been shown to do better than linear LSTMs at this phenomenon (Tai, Socher, and C. Manning 2015), but the current model uses only syntactic cues and leaves out semantics altogether.

Consider the pair *"A woman sitting outside weaving a long piece of red, white, and blue cloth."* and *"She is working with cloth."*. Here, the hypothesis features a pronoun with the antecedent lying in the premise. Many incorrectly classified examples feature this pronoun coreference property which the model may not be able to figure out. Even with attention models, it is hard to infer whether a hypothesis pronoun corefers to a premise noun/nominal.

7 Conclusion and Future Work

As part of this project, we explored TreeLSTMs for the natural language inference task, and two extensions to TreeLSTMs - namely deep TreeLSTMs and neural constituent-by-constituent attention models - inspired from prior work on the task. We discovered that attention models improve NLI accuracy significantly, and deeper models, while harder to train, show potential as well. Due to a lack of time and, towards the end, computing resources, however, we could not pursue certain research directions (in the recursive neural network space) which we would have liked to pursue, and are enlisted here. These fall into minor variations and more ambitious endeavors:

- Minor variations:
  1. Given that phrase overlap between the premise and the hypothesis is detrimental to model performance, one very simple heuristic would be to try and account for these by removing these from the sentences.
  2. Given that the tracking LSTM gives vanilla TreeLSTM accuracies a big boost, it might be worthwhile trying to include the tracking LSTM state in the computation of the higher layers in our deep TreeLSTM.
  3. Deeper TreeLSTM networks might be harder to train, but are potentially more powerful.
- Long-term enhancements:
  1. The current batchable TreeLSTM implementation works only with constituency parses. It would be worthwhile to experiment with sentence encodings constructed using a dependency parse-driven recursive NN. Combining constituency and dependency information to produce embeddings could also be interesting.
  2. Differentiable push-merge sequences using differentiable memory like in Grefenstette et al. 2015, could allow us to learn parses better suited to the downstream semantic task than the provided syntax-only parses.

Note: Ideas behind the attention models described here find mention in the first author (Raghav)'s past research. However, attention results, plus everything for the deep TreeLSTM, constitute work done solely for this project.

References

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[Bow+15] Samuel Bowman, Gabor Angeli, et al. “A large annotated corpus for learning natural language inference”. In: *EMNLP* (2015).

[Bow+16] Samuel Bowman, Jon Gauthier, et al. “A Fast Unified Model for Parsing and Sentence Understanding”. In: *arXiv preprint arXiv:1603.06021* (2016).

[CDL16] Jianpeng Cheng, Li Dong, and Mirella Lapata. “Long short-term memory-networks for machine reading”. In: *arXiv preprint arXiv:1601.06733* (2016).

[CM14] Danqi Chen and Christopher Manning. “A Fast and Accurate Dependency Parser using Neural Networks”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 740–750.

[Dau+15] Yann N Dauphin et al. “RMSProp and equilibrated adaptive learning rates for non-convex optimization”. In: *arXiv preprint arXiv:1502.04390* (2015).

[Gre+15] Edward Grefenstette et al. “Learning to transduce with unbounded memory”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 1819–1827.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “LSTM can solve hard long time lag problems”. In: *Advances in neural information processing systems* (1997), pp. 473–479.

[IC14] Ozan Irsoy and Claire Cardie. “Deep recursive neural networks for compositionality in language”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 2096–2104.

[IS15] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[Kat72] Jerrold J Katz. “Semantic theory”. In: (1972).

[Mag+14] Bernardo Magnini et al. “The Excitement Open Platform for Textual Inferences.” In: 2014.

[Mar+14] Marco Marelli et al. “Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment”. In: *SemEval-2014* (2014).

[Mou+15] Lili Mou et al. “Recognizing Entailment and Contradiction by Tree-based Convolution”. In: *arXiv preprint arXiv:1512.08422* (2015).

[PSM14] Jeffrey Pennington, Richard Socher, and Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543.

[Roc+15] Tim Rocktäschel et al. “Reasoning about Entailment with Neural Attention”. In: *arXiv preprint arXiv:1509.06664* (2015).

[Soc+11] Richard Socher et al. “Parsing natural scenes and natural language with recursive neural networks”. In: (2011), pp. 129–136.

[Sri+14] Nitish Srivastava et al. “Dropout: A simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[TSM15] Kai Sheng Tai, Richard Socher, and Christopher Manning. “Improved semantic representations from tree-structured long short-term memory networks”. In: *arXiv preprint arXiv:1503.00075* (2015).

[VH08] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of Machine Learning Research* 9.2579-2605 (2008), p. 85.

[WJ15] Shuohang Wang and Jing Jiang. “Learning Natural Language Inference with LSTM”. In: *arXiv preprint arXiv:1512.08849* (2015).