Predicting Entailment through Neural Attention and Binary Parsing

Matthew Katzman
Department of Mathematics
Stanford University
Stanford, CA 94305
mkatzman@stanford.edu

Natalie Ng
School of Engineering
Stanford University
Stanford, CA 94305
nng1@stanford.edu

Christina Ramsey
Department of Computer Science
Stanford University
Stanford, CA 94305
cmramsey@stanford.edu

Abstract

This paper focuses primarily on the problem of textual entailment, working to build models that determine whether pairs of natural language sentence entail, contradict, or are neutral towards each other. It expands on the work done by Rocktäschel et al [2], in which progress was made towards creating an end-to-end differentiable system that achieves state-of-the-art accuracy on a textual entailment dataset. Here in this paper, the authors use a basic LSTM, an LSTM with general attention, an LSTM with word-by-word attention, and an LSTM with an extra depth feature to explore the problem of recognizing textual entailment. Based off of preliminary results, the LSTM with the added depth feature obtains the highest accuracy in recognizing textual entailment.

1 Introduction

1.1 Motivation for Recognizing Textual Entailment

Of all the problems that arise in the field of Natural Language Processing, that of recognizing textual entailment (RTE) is of paramount importance. While to some it may seem contrived, the ability to discern entailment is a subproblem of almost every NLP task. Tasks such as information retrieval and extraction, text summarization, machine translation, and semantic parsing all either rely explicitly on, or would benefit significantly from, the ability to recognize textual entailment [1]-[2]. Even many solutions to problems in computer vision and image recognition are beginning to rely on such methods to improve the results above those obtained relying on vision techniques alone [3].

1.2 Approach

In this paper, the authors utilize the SNLI dataset (section 2.1) to explore a variety of models aimed at RTE. Comparing the results to a bag-of-words baseline neural network model, explored designs include a vanilla LSTM, an LSTM incorporating general attention, and an LSTM incorporating word-by-word attention, as well as the addition of features derived from the binary and constituency parses included in the dataset (described in more detail in section 2.1).

2 Background

2.1 The SNLI dataset

In 2015, Bowman et al. created the Stanford Natural Language Inference (SNLI) Corpus in order to allow for the design and development of models proficient in (RTE) [1]. At a size of 570,152 sentence pairs, SNLI was orders of magnitude larger than all other RTE datasets at the time, and remains one of the most comprehensive such collections to date. The premise sentences were written as photograph captions, and the hypothesis sentences (one each of an *entailment*, a *contradiction*, and a *neutral* sentence) were written by workers viewing the captions but not the photographs. The premise-hypothesis pairs were then validated by five annotators, from whom a majority label was used as the gold label. The data was then parsed using the Stanford PCFG parser trained on the standard training set, as well as the Brown corpus.

2.2 Related work

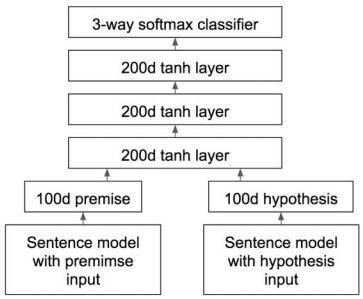
This extremely well-known problem has attracted a number of publications since the development of the SNLI corpus in 2015, and ideas for solutions have ranged from the vanilla feature-based model to (most recently) complex neural network models relying on 45 million parameters. Almost all published approaches have achieved test accuracies between 80% and 90%. Of these results, the primary motivation for this paper stems from Rocktschel et al., the first model implementing an LSTM with word-by-word attention, which attains a test accuracy of 83.5%.

3 Approach

After implementing a baseline bag-of-words neural network model for comparison, the authors first implemented a long-short term memory (LSTM) neural network, before introducing both general and word-by-word attention, as well as considering some additional features. The three LSTM models were directly inspired by those employed by Rocktäschel et al [2]. These models are explained below. In all models discussed, GloVe vectors are used as the word embeddings.

3.1 Baseline bag-of-words

Figure 1: Baseline bag-of-words model



In the baseline model, both the premise and hypothesis sentences were embedded as the average of their word embeddings. These embeddings were concatenated and sent through a number of hidden layers before the output was fed through a three-way softmax classifier, as shown in Figure 1. In equations, with a premise of n_p words and a hypothesis of n_h words in which $\boldsymbol{E}_{w_p^{(i)}}$ represents the GloVe embedding of the i^{th} word of the premise sentence:

$$\boldsymbol{e}_p = \frac{1}{n_p}\sum_{i=1}^{n_p} \boldsymbol{E}_{w_p^{(i)}} \in \mathbb{R}^d;\ \boldsymbol{e}_h = \frac{1}{n_h}\sum_{i=1}^{n_h} \boldsymbol{E}_{w_h^{(i)}} \in \mathbb{R}^d$$

$$\boldsymbol{e} = \begin{bmatrix} \boldsymbol{e}_p \\ \boldsymbol{e}_h \end{bmatrix} \in \mathbb{R}^{2d}.$$

Using this concatenated average embeddings vector $\boldsymbol{e}$ as input, the hidden layers were implemented as

$$\boldsymbol{h}_1 = \tanh\left(\boldsymbol{W}_1\boldsymbol{e} + \boldsymbol{b}_1\right) \in \mathbb{R}^{h_1};\ \boldsymbol{h}_2 = \tanh\left(\boldsymbol{W}_2\boldsymbol{h}_1 + \boldsymbol{b}_2\right) \in \mathbb{R}^{h_2}$$

$$\boldsymbol{h}_3 = \tanh\left(\boldsymbol{W}_3\boldsymbol{h}_2 + \boldsymbol{b}_3\right) \in \mathbb{R}^{h_3}$$

for $\boldsymbol{W}_1 \in \mathbb{R}^{h_1 \times 2d}, \boldsymbol{W}_2 \in \mathbb{R}^{h_2 \times h_1}, \boldsymbol{W}_3 \in \mathbb{R}^{h_3 \times h_2}$ (initialized using xavier initialization) and $\boldsymbol{b}_1 \in \mathbb{R}^{h_1}, \boldsymbol{b}_2 \in \mathbb{R}^{h_2}, \boldsymbol{b}_3 \in \mathbb{R}^{h_3}$ (initialized to $\mathbf{0}$). Finally, the output was decided by

$$\hat{\boldsymbol{y}} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}_3 + \boldsymbol{b}_4\right) \in \mathbb{R}^3$$

with $\boldsymbol{U} \in \mathbb{R}^{3 \times h_3}, \boldsymbol{b}_r \in \mathbb{R}^3$ initialized as above [1]. The model was trained to minimize cross-entropy loss:

$$J(\theta) = CE\left(\boldsymbol{y}, \hat{\boldsymbol{y}}\right) = -\sum_{i=1}^{3} \boldsymbol{y}_i \log \hat{\boldsymbol{y}}_i$$

and optimized using TensorFlow's Adam Optimizer. Here, the hyperparameters $d = 50, h_1 = h_2 = h_3 = 200$ were used, along with an initial learning rate of 0.0005 and a dropout rate of 0.25 applied at each step (not shown in the equations above). In the implementation of this model, the word-vectors were not tuned during training.

3.2 Vanilla LSTM

In the basic LSTM implementation, two LSTM's were constructed, one to process the premise sentence and the other to process the hypothesis. Because of this, separate weights could be used to process the separate sentences. However, the memory state of the hypothesis LSTM was initialized to the final memory state of the premise LSTM in order to maintain the important information from the premise across neural networks. Additionally, the hypothesis sentence was prepended by a `Start` symbol in order to process the initialized memory state. In equations (for s taking values p or h),

$$\boldsymbol{h}_p^{(0)} = \boldsymbol{c}_p^{(0)} = \mathbf{0} \in \mathbb{R}^h \qquad \boldsymbol{h}_h^{(0)} = \boldsymbol{h}_p^{(n)} \in \mathbb{R}^h, \boldsymbol{c}_h^{(0)} = \mathbf{0} \in \mathbb{R}^h$$

$$\boldsymbol{H}_s^{(t)} = \begin{bmatrix} \boldsymbol{E}_{w_s^{(t)}} \\ \boldsymbol{h}_s^{(t-1)} \end{bmatrix} \in \mathbb{R}^{d+h}$$

$$\boldsymbol{i}_s^{(t)} = \sigma\left(\boldsymbol{W}_s^i \boldsymbol{H}_s^{(t)} + \boldsymbol{b}_s^i\right) \in \mathbb{R}^h$$

$$\boldsymbol{f}_s^{(t)} = \sigma\left(\boldsymbol{W}_s^f \boldsymbol{H}_s^{(t)} + \boldsymbol{b}_s^f\right) \in \mathbb{R}^h$$

$$\boldsymbol{o}_s^{(t)} = \sigma\left(\boldsymbol{W}_s^o \boldsymbol{H}_s^{(t)} + \boldsymbol{b}_s^o\right) \in \mathbb{R}^h$$

$$\boldsymbol{c}_s^{(t)} = \boldsymbol{f}_s^{(t)} \circ \boldsymbol{c}_s^{(t-1)} + \boldsymbol{i}_s^{(t)} \circ \tanh\left(\boldsymbol{W}_s^c \boldsymbol{H}_s^{(t)} + \boldsymbol{b}_s^c\right) \in \mathbb{R}^h$$

$$\boldsymbol{h}_s^{(t)} = \boldsymbol{o}_s^{(t)} \circ \tanh\left(\boldsymbol{c}_s^{(t)}\right) \in \mathbb{R}^h$$

with $\boldsymbol{W}_s^i, \boldsymbol{W}_s^f, \boldsymbol{W}_s^o, \boldsymbol{W}_s^c \in \mathbb{R}^{h \times (d+h)}$ (initialized using xavier initialization) and $\boldsymbol{b}_s^i, \boldsymbol{b}_s^f, \boldsymbol{b}_s^o, \boldsymbol{b}_s^c \in \mathbb{R}^h$ (initialized to $\mathbf{0}$). The final output obtained the same as in the baseline model:

$$\hat{\boldsymbol{y}} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}_h^{(n_h)} + \boldsymbol{b}\right) \in \mathbb{R}^3$$

with $\boldsymbol{U} \in \mathbb{R}^{3 \times h}, \boldsymbol{b} \in \mathbb{R}^3$ initialized as above [2]. In this case, the parameters used were $d = 100, h = 200$ with an initial learning rate of 0.005, and a maximum sentence-length clipped at 60 words.

In addition to training to minimize cross-entropy loss and optimizing using TensorFlow's Adam Optimizer, a number of additional components were applied to this model. Gradient clipping was introduced to keep the gradient from exploding, allowing for magnitudes up to 5. A decaying learning rate was used, as well as both ℓ_1- and ℓ_2-regularization (both with coefficients of 0.00001). As opposed to the baseline model, the word embeddings were also tuned during training for this model.

3.3 LSTM with general attention

Adding a basic notion of attention to the LSTM model described in section 3.2 required only a minor modification between running the hypothesis LSTM and outputting the softmax result. Because the vanilla model required the premise LSTM to output a single representation vector before even seeing the hypothesis sentence, it was unable to give more emphasis to words that would turn out to be relevant. With attention, in contrast, every hidden state from the premise LSTM is given a fair chance to impact the result. This was achieved by constructing a vector α of attention weights and instead directing a nonlinear combination of the weighted premise output vectors and the hypothesis output vector, as seen in equations:

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{h}_p^{(1)} & \dots & \boldsymbol{h}_p^{(n_p)} \end{bmatrix} \in \mathbb{R}^{h \times n_p}$$

$$\boldsymbol{M} = \tanh\left(\boldsymbol{W}^y \boldsymbol{Y} + \boldsymbol{W}^h \boldsymbol{h}_h^{(n_h)} \otimes \mathbf{1}\right) \in \mathbb{R}^{h \times n_p}$$

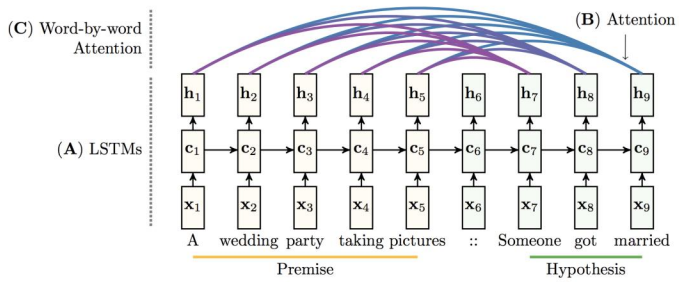
$$\alpha = \text{softmax}\left(\boldsymbol{w}^T \boldsymbol{M}\right) \in \mathbb{R}^{n_p};\ \boldsymbol{r} = \boldsymbol{Y}\alpha^T \in \mathbb{R}^h$$

$$\boldsymbol{h}^* = \tanh\left(\boldsymbol{W}^p \boldsymbol{r} + \boldsymbol{W}^x \boldsymbol{h}_h^{(n_h)}\right) \in \mathbb{R}^h$$

$$\hat{\boldsymbol{y}} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^* + \boldsymbol{b}\right) \in \mathbb{R}^3$$

with $\boldsymbol{W}^y, \boldsymbol{W}^h, \boldsymbol{W}^p, \boldsymbol{W}^x \in \mathbb{R}^{h \times h}$ initialized as above, $\boldsymbol{w} \in \mathbb{R}^h$ initialized to $\mathbf{0}$, and $\mathbf{1} \in \mathbb{R}^{n_p}$, with $\otimes$ the outer product (so $\boldsymbol{W}^h \boldsymbol{h}_h^{(n_h)} \otimes 1$ is the matrix $\begin{bmatrix} \boldsymbol{W}^h \boldsymbol{h}_h^{(n_h)} & \dots & \boldsymbol{W}^h \boldsymbol{h}_h^{(n_h)} \end{bmatrix} \in \mathbb{R}^{h \times n_p}$) [2].

Figure 2: Recognizing textual entailment using two LSTMs [2]



3.4 LSTM with word-by-word attention

To add word by word attention requires adding additional inputs to each LSTM cell as described in section 3.2. Specifically, we pass matrix $\boldsymbol{Y}$, as described in section 3.3, to each LSTM cell in the hypothesis sentence. In essence, this adds attention from the premise sentence to each word in the hypothesis sentence. The was achieved though a modification of the LSTM with general attention model, as seen in equations:

$$\boldsymbol{M} = \tanh\left(\boldsymbol{W}^y \boldsymbol{Y} + (\boldsymbol{W}^h \boldsymbol{h}_t + \boldsymbol{W}^r r_{t-1}) \otimes \mathbf{1}\right)$$

$$\alpha = \text{softmax}\left(\boldsymbol{w}^T \boldsymbol{M}\right) \in \mathbb{R}^{n_p};\ \boldsymbol{r} = \boldsymbol{Y}\alpha^T + \tanh(\boldsymbol{W}^t \boldsymbol{r}_{(t-1)}) \in \mathbb{R}^h$$

$$\boldsymbol{h}^* = \tanh\left(\boldsymbol{W}^p \boldsymbol{r} + \boldsymbol{W}^x \boldsymbol{h}_N\right) \in \mathbb{R}^h$$

$$\hat{\boldsymbol{y}} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^* + \boldsymbol{b}\right) \in \mathbb{R}^3$$

3.5 Depth feature

In order to provide more structure to the input, the authors considered augmenting the models above with features auxiliary to the word embeddings themselves, but decided only to supply one additional feature to prevent overfitting in a model with such a large parameter count. Ultimately, the depth of the binary parse was the feature included in the model, as it provides the most global structure on the sentence without requiring a sparse representation (the state of the stack in the provided constituency parse would have been more informative, but each stack arrangement would occur very rarely and likely would not have provided much additional data in training).

Armed with the binary parses provided by the Stanford PCFG parser, on a sentence such as

A person on a horse jumps over a broken down airplane.

the binary parse provides more structure to the sentence, grouping phrases as shown:

(((A person) (on (a horse))) ((jumps (over (a (broken (down airplane))))) .))

The input to the LSTM would then contain a word-depth tuple:

(a,3)	(a,4)	(over,4)	(down,7)
(person,3)	(horse,4)	(a,5)	(airplane,7)
(on, 3)	(jumps,3)	(broken,6)	(.,2)

Furthermore, each depth was represented by a trained, dense vector which was concatenated onto the word embedding:

$$\boldsymbol{H}_s^{(t)} = \begin{bmatrix} \boldsymbol{E}_{w_s^{(t)}} \\ \boldsymbol{D}_{d_s^{(t)}} \\ \boldsymbol{h}_s^{(t-1)} \end{bmatrix} \in \mathbb{R}^{d+h'}$$

in which $\boldsymbol{D}_{d_s^{(t)}}$ represents the embedding of the depth of the word in sentence s at position t, a vector in $\mathbb{R}^{h'-h}$ (here chosen to be $\mathbb{R}^{20}$) initialized normally at random with mean 0 and standard deviation 0.9 (the same as unknown word vector embeddings).

The primary motivation for using this feature lies in noticing very small pieces of sentences that completely transform the essence. For example, the sentence

they are trying not to fall in the water

is completely changed by the presence or absence of the word "not". In order to highlight the importance of this, the "not" is at a shallower depth than the action being avoided:

(they (are (trying (not (to (fall (in (the water))))))))

As "not" is at depth 4 while "to fall in the water" is all deeper, the model has an added ability to recognize that "not" is modifying "to fall in the water", and has the ability to better understand the sentence meaning.

4 Experiments

4.1 Tuning Hyperparameters

The authors ran various simulations of the LSTM with attention to identify the best hyperparameters for model training. Hyperparameters investigated include regularization, dropout rate, and learning rate. To identify whether the LSTM with attention overfits on the SNLI dataset, training was performed both with and without L1 + L2 regularization (both with coefficients of 0.00001). Dropout rate was set between 0.05 and 0.25. The impact of constant versus exponentially decaying learning rates were also investigated.

Table 1 shows the train and devevelopment accuracies from the models trained for hyperparameter tuning with no regularization. We can see that in all experiments, the train accuracy far exceeded the

Table 1: Accuracy for Learning Rate vs Dropout Rate - No Regularization

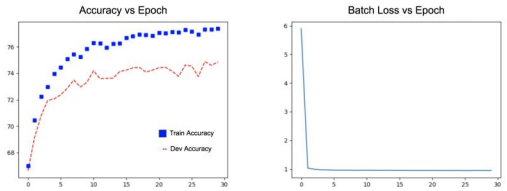
LR	Dropout Rate		
	0.25	0.1	0.05
Flat 0.001	Max Dev: 74.96 Max Train: 87.94 Epoch: 5	Max Dev: 75.1 Max Train: 85.21 Epoch: 4	Max Dev: 75.22 Max Train: 82.50 Epoch: 3
0.01	Max Dev: 66.15 Max Train: 68.90 Epoch: 0	Max Dev: 65.53 Max Train: 67.96 Epoch: 0	Max Dev: 66.82 Max Train: 70.13 Epoch: 1
0.005	Max Dev: 72.16 Max Train: 84.71 Epoch: 5	Max Dev: 73.03 Max Train: 83.15 Epoch: 4	Max Dev: 72.72 Max Train: 80.58 Epoch: 2
0.001	Max Dev: 75.74 Max Train: 85.29 Epoch: 4	Max Dev: 74.95 Max Train: 84.99 Epoch: 4	Max Dev: 74.76 Max Train: 85.35 Epoch: 4

development accuracy, and that the maximum development accuracy was attained at an early epoch. This indicates that the model overfits to the data without regularization. Table 2 shows the results with regularization. The results show less evidence of overfitting, as the train and development accuracies are much closer, and with one exception, the maximum development accuracy does not occur until after epoch 14. Using an exponentially decaying learning rate of 0.005 give the best results. Through parameter tuning, the authors were able to increase the development accuracy of an LSTM with attention from 67% to approximately 75%.

Table 2: Accuracy for Learning Rate vs Dropout Rate - Regularization

LR	Dropout Rate		
	0.25	0.1	0.05
Flat 0.001	Max Dev: 68.36 Max Train: 68.7 Epoch: 2	Max Dev: 74.25 Max Train: 77.83 Epoch: 14	Max Dev: 74.62 Max Train: 78.12 Epoch: 14
0.01	Max Dev: 72.05 Max Train: 72.91 Epoch: 9	Max Dev: 71.97 Max Train: 72.58 Epoch: 11	Max Dev: 72.38 Max Train: 73.39 Epoch: 12
0.005	Max Dev: 75.05 Max Train: 76.96 Epoch: 14	Max Dev: 74.57 Max Train: 76.58 Epoch: 13	Max Dev: 74.28 Max Train: 76.62 Epoch: 14
0.001	Max Dev: 74.64 Max Train: 77.08 Epoch: 12	Max Dev: 74.42 Max Train: 77.33 Epoch: 13	Max Dev: 74.75 Max Train: 77.96 Epoch: 14

Figure 3: Learning Rate: 0.005; Dropout Rate: 0.1



4.2 Model comparisons

All models reported were run to convergence (an example of running to convergence is shown in Figure 3). The epoch with the best development set was selected and evaluated on the test set. The final hyperparameters used in model training were an exponentially decaying learning rate, initialized at 0.005, and a dropout of 0.1.

Table 3: Accuracies and F1 Scores by Model

	Train	Dev	Test	F1
Baseline Bag of Words	51.85	51.81	52.5	52.5
Vanilla LSTM	76.79	74.87	74.26	74.26
Vanilla LSTM with Attention	77.75	75.26	75.05	75.05
Vanilla LSTM with Word-by-Word Attention*	72.90	72.57	72.18	72.18
Vanilla LSTM with Attention and Depth Features	81.95	77.62	**77.35**	**77.35**

*Model did not run to convergence

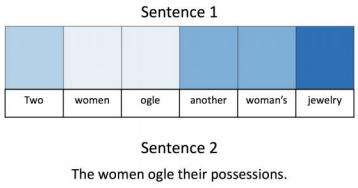
The F1 score was calculated by globally counting the number of total true positives, false negatives, and false positives. The reported F1 score is the harmonic mean between the precision and recall computed using these counts. As observed, the F1 score is almost identical to the test accuracy (Table 3). Additionally, as visualized by example confusion matrices (Table 4), the model classifies with similar success across the three labels.

Table 4: Confusion Matrices

	Vanilla LSTM Prediction			LSTM w. Attention Prediction		
	E	C	N	E	C	N
Entailment	2619	261	488	2792	203	373
Contradiction	342	2415	480	449	2353	435
Neutral	496	462	2261	637	354	2228

4.3 Impact of attention

Figure 4: Attention alpha vector visualization



Alpha vectors outputted by attention were visualized to ensure that attention was focused on important words in the context sentence with respect to the hypothesis sentence. A representative example is shown in Figure 4. The alpha vector focuses on the word "jewelry," which relates to the word "possession" in the hypothesis sentence. This assists in correct classification of this example by the LSTM with attention. In the dataset overall, the added attention parameters increase the test accuracy by 0.75% in the test set.

4.4 Impact of depth parameters

Depth parameters cause a noticeable improvement in test accuracy. Here are a few examples that the LSTM with attention models classifies incorrectly, but the LSTM with depth parameters classifies correctly.

Sentence 1:	*A man and a woman walk at a flea market.*
Binary Parse:	*((((A man) and) (a woman)) ((walk(at (a (flea market)))) .))*
Sentence 2:	*The man and women are at a high priced store.*
Binary Parse:	*((((The man) and) women) ((are (at (a ((high priced) store)))) .))*
Gold Label:	Contradiction
Attention Prediction:	Entailment

In this case, it is likely that with the depth feature, the model is able to directly compare the context word "flea" with "high priced", which would lead to the correct classification of the two sentences. However, without the depth features, we would lose this direct comparison and predict entailment, as the sentences describe relatively similar situations.

Sentence 1:	*Two people using a water buffalo to cultivate a watery field.*
Binary Parse:	*(((Two people) ((using (a (water buffalo))) (to (cultivate (a (watery field))))))) .)*
Sentence 2:	*Two people are outside with animals.*
Binary Parse:	*((Two people) (((are outside) (with animals)) .))*
Gold Label:	Entailment
Attention Prediction:	Contradiction

Similarly, in this example, the LSTM with attention may declare contradiction due to the many words relating to water in the first sentence, while the second sentence relates more to animals. With the depth parameters, the model can identify that animals and buffalo are at the same depth, aiding in correct classification.

4.5 Model Shortcomings

The model performs well at detecting similar or contradictory context clues to identify entailments and contradictions. However, there are instances where a similar context clue can lead to a contradiction. Consider the following example:

Sentence 1:	*A person attempt to rope a black cow while riding a horse.*
Sentence 2:	*The person is riding on a camel.*
Gold Label:	Contradiction
Model Prediction:	Entailment

The context clues suggest that in the premise sentence, the person is working with riding and working with some form of animal. The model likely picked this up and saw similar context clues in the hypothesis sentence and declared an entailment.

We also see that the model struggles with identifying multi-word context clues. In the following sentence, the context clue in the first sentence comprises of speaking on a microphone, while the second sentence also has a multi-word context clue of addressing a crowd.

Sentence 1:	*A public speaker in a plaid shirt speaks on the microphone.*
Sentence 2:	*A person addresses the crowd.*
Gold Label:	Entailment
Model Prediction:	Neutral

5 Conclusion

This report describes a successful impelementation of using LSTMs to detect sentence entailment and contradiction. The results for all implemented LSTMs far exceed the baseline model. The feature that is of the most interest to the authors is that of the aded depth features. Adding this feature increased model accuracy in the test set by 3%, and qualitative analysis of the depth feature confirms that it captures information that a LSTM without this feature misses. The authors believe that further research into the depth parameter is of particular interest and can be implemented into current state of the art methods to boost model performance further. Next steps also include implementing more complex LSTM structures, including re-read LSTMs [4] and Cafe ensembles [5].

Acknowledgments

The authors would like to thank Kevin Clark and Huseyin Inan for their assistance throughout the project. They would additionally like to thank Richard Socher for his excellent instruction without which this paper would not have been possible. The authors would also like to acknowledge Christopher Yeh for his assistance with TensorFlow debugging and William Bradbury for his support in brainstorming the inclusion of additional features. Finally, the authors would like to thank Microsoft for their sponsorship of the class and their generous donation of Azure credits.

References

[1] Bowman, S.R., Angeli, G., Potts, C. & Manning, C.D. (2015) A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[2] Rocktäschel, T., Grefenenstette, E., Hermann, K.M., Kočiský, T., Blunsom, P. (2015) Reasoning about Entailment with Neural Attention. In *ICLR 2016.*

[3] Vendrov, I., Kiros, R., Fidler, S., Urtasun, R. (2015) Order-Embeddings of Images and Language. In *ICLR 2016.*

[4] Sha, L., Chang B., Sui, Z., Li S. Reading and Thinking: Re-read LSTM Unit for Textual Entailment Recognition. In *COLING 2016.*

[5] Tay Y., Tuan, L.A., Hui, S.C. A Compare-Propagate Architecture with Alignment Factorization for Natural Language Inference. In *ArXiv 2017.*