Learning Semantic Entailment and Contradiction

Ameera Chowdhury Vaibhav Gandhi
Robin Heinonen
University of California, San Diego
La Jolla, CA, 92093
anchowdh@ucsd.edu, vrgandhi@.ucsd.edu, rheinone@ucsd.edu

Abstract

We build and train models to learn the logical relations between pairs of sentences—that is, semantic entailment and contradiction—in the Stanford Natural Language Inference (SNLI) corpus. Similar to Bowman et al. *(2015) [1], we construct a baseline model upon handcrafted unlexicalized and lexicalized features and compare it to deep-learning methods based on LSTMs. As shown by Bowman* et al.*, we confirm that LSTM-based models can perform competitively or better on the SNLI corpus and achieve a best accuracy rate of 76.12% on the test set. We also experiment with certain variations on the architecture.*

1. Introduction

The problem of natural language inference (NLI), that is, automatically determining semantic entailment and contradiction, is an important and challenging benchmark for the larger problem of natural language understanding [9]. In recent years, the problem has seen increased attention and many advances [2].

Semantic entailment essentially means one sentence implies another. If a human reading premise A would infer that hypothesis B is most likely true, then we say that premise A *entails* hypothesis B. Similarly, we say A *contradicts* B if a human reading premise A would infer that hypothesis B is most likely false.

For example, consider premise A, "A man and a child play catch with a baseball." This premise reasonably implies the hypothesis B_1, "Two people of different ages are playing a game," so A entails B_1. However, most readers would agree that hypothesis B_2, "The man is sleeping," contradicts A, and that hypothesis B_3, "A dog is present," is neither implied nor contradicted by A and is thus neutral in relation to A.

Given a premise A and a hypothesis B, our task is to classify whether A entails B, A contradicts B, or whether A and B are neutral; that is A neither entails nor contradicts B. This task is called natural language inference (NLI) or recognizing textual entailment (RTE).

2. Description of the Project

We use the Stanford Natural Language Inference (SNLI) corpus [1, 2] to learn semantic entailment and contradiction. The SNLI corpus consists of 570K pairs of human-written English language sentences with one of three labels—entailment, contradiction, and neutral. All sentence pairs and labels in the SNLI corpus were written by humans and were carefully generated to balance the sentence pairs equally among the three labels. Moreover, for 57K sentence pairs, four additional judgments for each label were collected in a separate validation phase. Of these, 98% of sentence pairs had a three-annotator consensus, and 58% had a unanimous consensus from all five annotators. This suggests that the SNLI corpus is a high quality data set for machine learning.

Model	Train (% acc)	Test (% acc)
Unlexicalized Classifier	49.4	50.4
Lexicalized Classifier	99.7	78.2
100D LSTM encoders	84.8	77.6

Table 1. Baselines reported in Bowman *et al.* [1]

Bowman *et al.* [1] reported two baselines for their SNLI corpus, which are recorded in Table 1. The first and best-performing baseline, with an accuracy rate of 78.2% on the test set, was a linear classifier that used both unlexicalized and lexicalized features. An *unlexicalized* feature is one that does not directly make use of the words in the sentences; for example, the length of the sentence would be an unlexicalized feature. In contrast, a *lexicalized* feature is one which does directly use the words in the sentences.

Due to its large size, the SNLI corpus was the first NLI corpus that permitted training of a neural network, and Bowman *et al.* were the first to demonstrate that neural network-based models could perform competitively on

Publication	Model	Train (% acc)	Test (% acc)
Zhiguo Wang *et al.* [16]	BiMPM Ensemble	93.2	88.8
Yichen Gong *et al.* [6]	448D DIIN Ensemble	92.3	88.9
Qian Chen *et al.* [3]	KIM Ensemble	93.6	89.1
Ghaeini *et al.* [5]	450D DR-BiLSTM Ensemble	94.8	89.3
Yi Tay *et al.* [14]	300D CAFE Ensemble	92.5	89.3

Table 2. Current state of the art accuracy rates on the SNLI corpus

NLI questions. By using an LSTM to embed both the premise and hypothesis sentences into a 100-dimensional vector space and by then feeding these vectors into a three-layer neural network, Bowman *et al.* achieved a 77.6% accuracy rate on the test set.

The current state of the art accuracy rates on the SNLI corpus are listed in Table 2. These accuracy rates are achieved by ensemble methods, which use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone.

3. Experiments

Taking inspiration from the models reported in [1], we implement several models to test the hypothesis that, on the SNLI corpus, Deep Neural Networks can perform as well as traditional Machine Learning models that use handcrafted features. We implemented a baseline model based on handcrafted unlexicalized and lexicalized features. With this as a benchmark, we compare the results with those of two more sophisticated LSTM-based models. In the following subsections we describe the various models we tested and the results we obtained therefrom.

3.1. Baseline Model

For training and testing, we used all sentence pairs in the SNLI training (respectively test) set, except for those sentence pairs that had been evaluated under the separate validation phase and did not achieve the three-annotator consensus required for a gold label. Following Bowman *et al.* [1], our baseline model utilizes the following features extracted from the SNLI dataset:

1. BLEU score of hypothesis with respect to premise,
2. Length difference between hypothesis and premise,
3. Word overlap between hypothesis and premise, as a percentage of possible overlap,
4. An indicator for the 10000 most common unigrams in the hypotheses.

The BLEU score [10] of a premise with respect to a hypothesis is a metric that evaluates to what extent the hypothesis is a perfect translation of the premise. The BLEU score is always between 0 and 1, with 1 indicating a perfect translation.

A unigram refers to a single word. To implement the last feature, we first built a vocabulary space of the 10000 most common words occurring in the hypotheses. We then converted each hypothesis into a 0-1 vector of length 10000 with a 1 in coordinate i indicating that the i^{th} word in the vocabulary space occurs in the hypothesis. Observe that the last feature of our classifier is lexicalized because it makes direct reference to the vocabulary from which the hypotheses are drawn. In contrast, the first three features of our classifier are unlexicalized.

We used NLTK [8] to remove stopwords and NLTK's PorterStemmer to stem our words before extracting the first, third, and last features. This allowed for a denser feature representation.

3.1.1 Results

In Scikit-learn [11], we used these features to train various classifiers, namely a naive Gaussian Bayes model, a support vector machine (SVM), and a logistic regression model. We could only use the naive Gaussian Bayes model on the first three features because this method in Scikit-learn requires dense matrices. Once we implemented our last feature, we had to store our training vectors in a 549367×10003 CSR sparse matrix. The accuracy rates of each of these classifiers is reported in Table 3. We report accuracy rates on both the training and test sets to judge overfitting.

System	Train	Test
Unlexicalized Gaussian Naive Bayes	46.5	47.3
Unlexicalized Support Vector Machine	42.8	46.0
Unlexicalized Logistic Regression	45.9	46.5
Unlexicalized + Unigram Indicator Support Vector Machine	62.9	60.9
Unlexicalized + Unigram Indicator Logistic Regression	66.4	**66.4**

Table 3. Accuracy Rates for Different Classifiers

For the unlexicalized features, the naive Gaussian Bayes classifier achieved an accuracy rate of 47.3% and thus performed better than either a support vector machine or logistic regression, which had accuracy rates of 46.0% and 46.5% respectively. Our accuracy rate of 47.3% almost matches the accuracy rate of 50.4% for an unlexicalized classifier reported by Bowman *et al.* [1] in Table 1. The reason for the slight difference in accuracy rates is that, in the implementation of the third feature, Bowman *et al.* used the overlap between words in the premise and hypothesis, both as an absolute count and as a percentage of possible overlap, and both over all words and over just nouns, verbs, adjectives, and adverbs.

When we implemented the unigram indicator, our last feature, we found that Logistic Regression yielded an accuracy rate of 66.4% and thus performed better than a Support Vector Machine, which yielded an accuracy rate of 60.9%. The confusion matrix for our Logistic Regression model, reported in Table 4, allows us to better analyze our results. In the confusion matrix, the true labels of the premise–hypothesis pairs are given by the rows and the predicted labels are given by the columns. The confusion matrix shows that our classifier is equally accurate at identifying each of the three labels. Moreover, when our classifier incorrectly classifies a premise–hypothesis pair, neither of the two remaining labels is excessively favored over the other.

	Contradiction	Entailment	Neutral
Contradiction	2166	573	498
Entailment	464	2479	425
Neutral	684	657	1878

Table 4. Confusion Matrix for Logistic Regression

Our accuracy rate of 66.4% falls short of the accuracy rate of 78.2% for a lexicalized classifier that was reported by Bowman *et al.* [1] in Table 1. This is because, on top of the four features that we implemented, Bowman *et al.* implemented the following additional features

1. An indicator for every unigram and bigram in the hypothesis.
2. Cross-unigrams: for every pair of words across the premise and hypothesis which share a POS tag, an indicator feature over the two words.
3. Cross-bigrams: for every pair of bigrams across the premise and hypothesis which share a POS tag on the second word, an indicator feature over the two bigrams.

Due to time constraints and lack of expertise, we were not able to implement the remaining features of the classifier in Bowman *et al.* The code for our baseline classifier can be found on Github.

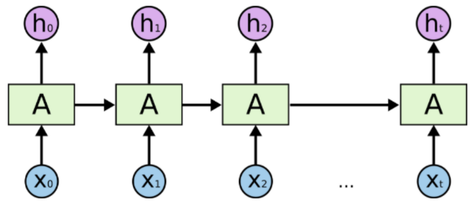

Figure 1. Basic schematic of an LSTM, showing inputs and outputs into cells and the relationships between neighboring cells (from http://colah.github.io/posts/2015-08-Understanding-LSTMs)

3.2. LSTM Models

We can also use a long short-term memory network (LSTM) to classify the sentence pairs. An LSTM [7] is a kind of recurrent neural network (RNN) whose cells feature a *forget gate* which controls the extent to which weights from the previous cell are 'remembered' by the next cell. A simple schematic is shown in Fig. (1). The structure is as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{bmatrix} \tag{1}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}^i\mathbf{H} + \mathbf{b}^i) \tag{2}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}^f\mathbf{H} + \mathbf{b}^f) \tag{3}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^o\mathbf{H} + \mathbf{b}^o) \tag{4}$$

$$\mathbf{A}_t = \mathbf{f}_t \circ \mathbf{A}_{t-1} + \mathbf{i}_t \circ \phi(\mathbf{W}^a\mathbf{H} + \mathbf{b}^a) \tag{5}$$

$$\mathbf{h}_t = \mathbf{o}_t \circ \phi(\mathbf{A}_t) \tag{6}$$

Here, $\mathbf{A}_t$ are the cells, $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t$ are input, forget, and output gates, respectively, σ is the sigmoid function, ϕ is a nonlinear activation function such as tanh or ReLU, and $\circ$ denotes element-wise products. The $\mathbf{W}$'s and $\mathbf{b}$'s are parameters to be learned.

LSTMs are capable of learning long-term dependencies between words in sentences and are thus well-suited to the problem of natural language inference. We implemented them in two distinct ways ('Model 1' and 'Model 2'), which we now describe in detail. The repository for the code can be found on Github.

3.2.1 Model 1

Figure 2 shows the architecture for LSTM Model 1. In this model, we first encode each word from both premise and the hypothesis to a 300-dimensional vector of floating point values using the GloVe algorithm [12]. This is an unsupervised learning algorithm that uses co-occurrence frequencies of words in the English language to map them into spaces of 50–300 real-valued dimensions. In particular, we used the highest-quality GloVe embeddings—300D vectors trained on 840B tokens and a vocabulary size of 2.2M.

We then concatenate the two sequences of vectors (for the premise and the hypothesis) with a stop token to separate them. The result is a sequence of 300D vectors that is of length $2 \times T + 1$ where we choose T to be 30. We truncate the premise or the hypothesis if they are more than T words long. Thus, we get a single sequence that is of shape ($2 \times T + 1$, 300). We then use an LSTM network to process this sequence of vectors and encode it to a single 600D vector. A 3-layer multi-layer perceptron (MLP) is used on top of this to classify the vector to one of three classes - entailment, contradiction and neutral.

In detail, we trained the network according to the cross-entropy loss across the three classes (entailment, contradiction, neutral). We experimented with both rectified linear unit (ReLU) and tanh activation functions in the LSTM and MLP, as well as a hybrid approach with a tanh activation in the LSTM and ReLU in the MLP. For each layer, we use batch normalization, L2 regularization and dropout (with dropout rate of 20%). These measures reduce covariance shift, increase stability, and discourage overfitting. The optimization algorithm that we found to work best was RMSprop [15], a version of adaptive learning rate gradient descent in which the learning rate is damped by a weighted average of the square of gradients. We used an initial learning rate of 0.001 and a moving average parameter of $\eta = 0.9$. Early stopping when the accuracy on the validation set did not improve for four consecutive epochs was implemented to prevent overfitting.

The key architectural choice here is that of concatenating and processing the premise and the hypothesis in succession. The intuition behind this is that doing so allows the LSTM to encode the hypothesis in the context of the premise. This is because the hidden state after processing the premise is passed on to the network when it starts to encode the hypothesis. This allows the network to allocate its fixed representational power to encode the differences/similarities between the premise and the hypothesis rather than encode the hypothesis in a general manner.

3.2.2 Model 2

Figure 3 shows the architecture for LSTM Model 2. In this model, we again first encode each word from both premise and the hypothesis to a 300 dimensional vector of floating point values. We use GloVe 300D embeddings ($840B$ tokens, $2.2M$ vocab). We don't train the embeddings again. We thus have two sequences corresponding to the premise and the hypothesis each of shape (T, 300). Unlike in Model 1 however, we don't concatenate these. Instead, we use an LSTM network to separately process the two sequences of vectors and encode the premise and the hypothesis to two

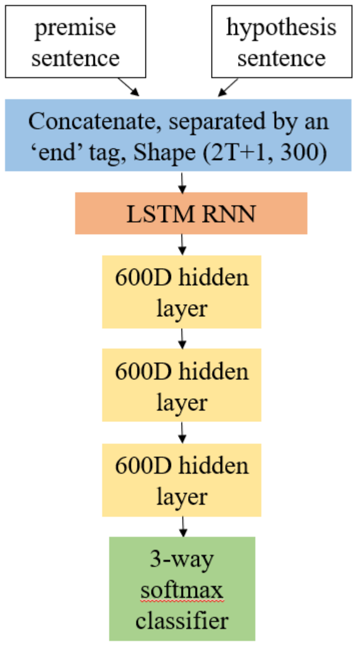

Figure 2. Architecture of LSTM Model 1

300D vectors. We then concatenate the two vectors to form a single 600D vector, and at this stage the model is passed into an MLP with identical architecture to that of Model 1.

We use a Siamese architecture for the LSTM network, meaning that we share the weights of the LSTMs used to process the premise and the hypothesis so that that both the premise and the hypothesis can be processed equivalently. This helps regularize the model by limiting the number of parameters. We validate this hypothesis by comparing the accuracy of the model with and without weight-sharing.

The key architectural difference between this model and Model 1 is that we do not process the premise and the hypothesis as one long sequence and instead process them separately in two small sequences. This has two possible advantages:

1. By reducing the length of the sequence that the LSTM has to process, we no longer need to pass the gradient for a large number of timesteps. As adept as LSTMs are at processing long sequences, it always helps to limit the timesteps if possible.

2. By reducing the timesteps we are also reducing the information that needs to be remembered by the hidden state at timesteps $T > 30$. This is important because the size of the hidden states is constant and is indeed the bottleneck in Model 1.

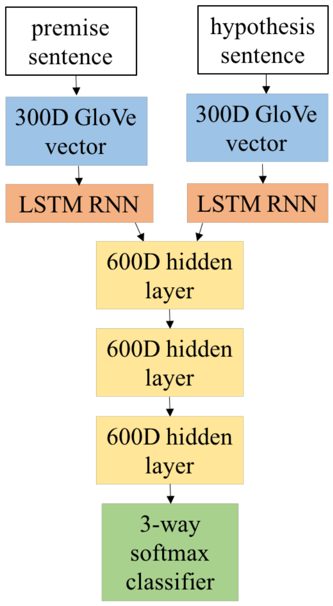

Figure 3. Architecture of LSTM Model 2

3.2.3 Results

A plot of the training and validation accuracy vs. training epoch for one instance each of Model 1 and Model 2 is shown in Fig. 4; convergence typically occurred after around 6 epochs. The accuracy rates for the various LSTM models we tested are displayed in Table 5. Model 2, wherein the premise and hypothesis are processed separately, was found to strongly outperform Model 1 on both the training and validation sets.

We verified that sharing the LSTM weights between the premise and hypothesis in Model 2 is superior to processing them in separate LSTMs. Moreover, we found that using the hybrid approach to the activation function yielded better performance than a pure tanh or ReLU approach. An accuracy rate of 76.12% was achieved on the validation set in the best case. This result is comparable to the 77.6% achieved by Bowman *et al.* using 100D GloVe representations on a similar architecture.

A confusion matrix for our best model is shown in Table 6. Comparing to Table 4 for the baseline model, we see the LSTM is especially better at classifying neutral pairs and very rarely confuses contradiction for entailment.

4. Conclusion

Our results have demonstrated the viability of LSTMs for the purpose of natural language inference. The LSTM-based models are seen to outperform our handcrafted feature-based model, though it is likely that including additional lexicalized features such as bigrams and cross-n-grams (i.e. part-of-speech pairs across premis and hypothesis) in the baseline model could bring its performance in line with the LSTM models that we used here.

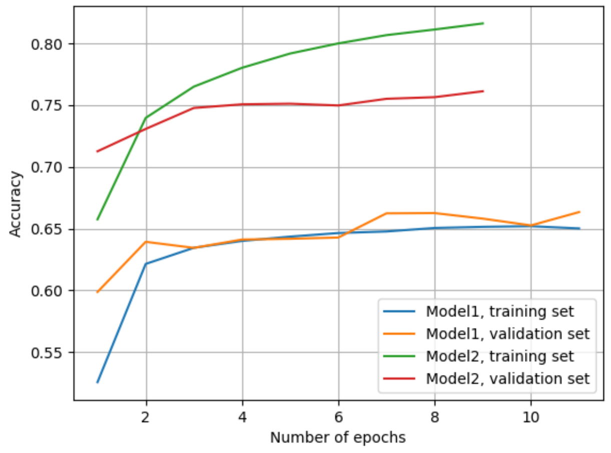

Figure 4. Plot of training and validation accuracy vs. number of epochs for LSTM models.

Model	Train	Validation
Model 1 w/ tanh activation	65.19%	66.25%
Model 2 w/ hybrid act., shared LSTM	81.62%	**76.12%**
Model 2 w/ hybrid act., separate LSTMs	78%	74%
Model 2 w/ tanh, shared LSTM	77.2%	74.3%
Model 2 w/ ReLU, shared LSTM	76.1%	73.45%

Table 5. Accuracy rates for various LSTM models on training set and validation set. Model 2 generally outperforms Model 1.

	Entailment	Contradiction	Neutral
Entailment	2779	168	393
Contradiction	411	2386	475
Neutral	562	336	2332

Table 6. Confusion matrix for our best LSTM model. Rows are true labels, columns are predictions.

Moreover, we have seen that encoding sentences separately in a single LSTM is superior to encoding them both together. This result we interpret as a demonstration of the principle that fewer model parameters tend to lead to superior models; clearly there is redundancy in many of the extra parameters needed to process the premise and hypothesis in a single sweep.

A shortcoming of the LSTM approach is a limited ability to learn relations between words across the hypothesis and premise. Similar, but more sophisticated approaches such as word-by-word attention models [13] are likely to achieve even greater performance. The model can be further improved using a bidirectional LSTM that can read the sentences both forwards and backwards [4].

References

[1] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[2] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. The Stanford Natural Language Inference (SNLI) Corpus. `https://nlp.stanford.edu/projects/snli/`, 2015. Accessed: 2018-02-03.

[3] Q. Chen, X. Zhu, Z.-H. Ling, and D. Inkpen. Natural language inference with external knowledge. *arXiv preprint arXiv:1711.04289*, 2017.

[4] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes. Supervised learning of universal sentence representations from natural language inference data. *CoRR*, abs/1705.02364, 2017.

[5] R. Ghaeini, S. A. Hasan, V. Datla, J. Liu, K. Lee, A. Qadir, Y. Ling, A. Prakash, X. Z. Fern, and O. Farri. DR-BiLSTM: Dependent reading bidirectional LSTM for natural language inference. *arXiv preprint arXiv:1802.05577*, 2018.

[6] Y. Gong, H. Luo, and J. Zhang. Natural language inference over interaction space. *arXiv preprint arXiv:1709.04348*, 2017.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–80, 12 1997.

[8] E. Loper and S. Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*, pages 63–70. Association for Computational Linguistics, 2002.

[9] B. MacCartney and C. D. Manning. Natural logic for textual inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 193–200, 2007.

[10] K. Papineni, S. Roukos, T. Ward, , and W.-J. Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, July 2002.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[13] T. Rocktäschel, E. Grefenstette, K. M. Hermann, T. Kociský, and P. Blunsom. Reasoning about entailment with neural attention. *CoRR*, abs/1509.06664, 2015.

[14] Y. Tay, L. A. Tuan, and S. C. Hui. A compare-propagate architecture with alignment factorization for natural language inference. *arXiv preprint arXiv:1801.00102*, 2017.

[15] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[16] Z. Wang, W. Hamza, and R. Florian. Bilateral multi-perspective matching for natural language sentences. *arXiv preprint arXiv:1702.03814*, 2017.