Learning Natural Language Inference with LSTM

Shuohang Wang
School of Information Systems
Singapore Management University
shwang.2014@phdis.smu.edu.sg

Jing Jiang
School of Information Systems
Singapore Management University
jingjiang@smu.edu.sg

Abstract

Natural language inference (NLI) is a fundamentally important task in natural language processing that has many applications. The recently released Stanford Natural Language Inference (SNLI) corpus has made it possible to develop and evaluate learning-centered methods such as deep neural networks for natural language inference (NLI). In this paper, we propose a special long short-term memory (LSTM) architecture for NLI. Our model builds on top of a recently proposed neural attention model for NLI but is based on a significantly different idea. Instead of deriving sentence embeddings for the premise and the hypothesis to be used for classification, our solution uses a match-LSTM to perform word-by-word matching of the hypothesis with the premise. This LSTM is able to place more emphasis on important word-level matching results. In particular, we observe that this LSTM remembers important mismatches that are critical for predicting the contradiction or the neutral relationship label. On the SNLI corpus, our model achieves an accuracy of 86.1%, outperforming the state of the art.

1 Introduction

Natural language inference (NLI) is the problem of determining whether from a premise sentence P one can infer another hypothesis sentence H (MacCartney, 2009). NLI is a fundamentally important problem that has applications in many tasks including question answering, semantic search and automatic text summarization. There has been much interest in NLI in the past decade, especially surrounding the PASCAL Recognizing Textual Entailment (RTE) Challenge (Dagan et al., 2005). Existing solutions to NLI range from shallow approaches based on lexical similarities (Glickman et al., 2005) to advanced methods that consider syntax (Mehdad et al., 2009), perform explicit sentence alignment (MacCartney et al., 2008) or use formal logic (Clark and Harrison, 2009).

Recently, Bowman et al. (2015) released the Stanford Natural Language Inference (SNLI) corpus for the purpose of encouraging more learning-centered approaches to NLI. This corpus contains around 570K sentence pairs with three labels: *entailment*, *contradiction* and *neutral*. The size of the corpus makes it now feasible to train deep neural network models, which typically require a large amount of training data. Bowman et al. (2015) tested a straightforward architecture of deep neural networks for NLI. In their architecture, the premise and the hypothesis are each represented by a sentence embedding vector. The two vectors are then fed into a multi-layer neural network to train a classifier. Bowman et al. (2015) achieved an accuracy of 77.6% when long short-term memory (LSTM) networks were used to obtain the sentence embeddings.

A more recent work by Rocktäschel et al. (2016) improved the performance by applying a neural attention model. While their basic architecture is still based on sentence embeddings for the premise and the hypothesis, a key difference is that the embedding of the premise takes into consideration the alignment between the premise and the hypothesis. This so-called *attention-weighted* representation of the premise was shown to help push the accuracy to

83.5% on the SNLI corpus.

A limitation of the aforementioned two models is that they reduce both the premise and the hypothesis to a single embedding vector before matching them; i.e., in the end, they use two embedding vectors to perform sentence-level matching. However, not all word or phrase-level matching results are equally important. For example, the matching between stop words in the two sentences is not likely to contribute much to the final prediction. Also, for a hypothesis to *contradict* a premise, a single word or phrase-level mismatch (e.g., a mismatch of the subjects of the two sentences) may be sufficient and other matching results are less important, but this intuition is hard to be captured if we directly match two sentence embeddings.

In this paper, we propose a new LSTM-based architecture for learning natural language inference. Different from previous models, our prediction is not based on whole sentence embeddings of the premise and the hypothesis. Instead, we use an LSTM to perform *word-by-word* matching of the hypothesis with the premise. Our LSTM sequentially processes the hypothesis, and at each position, it tries to match the current word in the hypothesis with an attention-weighted representation of the premise. Matching results that are critical for the final prediction will be "remembered" by the LSTM while less important matching results will be "forgotten." We refer to this architecture a match-LSTM, or *m*LSTM for short.

Experiments show that our *m*LSTM model achieves an accuracy of 86.1% on the SNLI corpus, outperforming the state of the art. Furthermore, through further analyses of the learned parameters, we show that the *m*LSTM architecture can indeed pick up the more important word-level matching results that need to be remembered for the final prediction. In particular, we observe that good word-level matching results are generally "forgotten" but important mismatches, which often indicate a *contradiction* or a *neutral* relationship, tend to be "remembered."

2 Model

In this section, we first review LSTM. We then review the word-by-word attention model by Rocktäschel et al. (2016), which is their best performing model. Finally we present our *m*LSTM architecture for natural language inference.

2.1 Background

LSTM: Let us first briefly review LSTM (Hochreiter and Schmidhuber, 1997). LSTM is a special form of recurrent neural networks (RNNs), which process sequence data. LSTM uses a few gate vectors at each position to control the passing of information along the sequence and thus improves the modeling of long-range dependencies. While there are different variations of LSTMs, here we present the one adopted by Rocktäschel et al. (2016). Specifically, let us use $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ to denote an input sequence, where $\mathbf{x}_k \in \mathbb{R}^l$ $(1 \leq k \leq N)$. At each position k, there is a set of internal vectors, including an input gate $\mathbf{i}_k$, a forget gate $\mathbf{f}_k$, an output gate $\mathbf{o}_k$ and a memory cell $\mathbf{c}_k$. All these vectors are used together to generate a d-dimensional hidden state $\mathbf{h}_k$ as follows:

$$\begin{aligned}
\mathbf{i}_k &= \sigma(\mathbf{W}^{\mathrm{i}}\mathbf{x}_k + \mathbf{V}^{\mathrm{i}}\mathbf{h}_{k-1} + \mathbf{b}^{\mathrm{i}}), \\
\mathbf{f}_k &= \sigma(\mathbf{W}^{\mathrm{f}}\mathbf{x}_k + \mathbf{V}^{\mathrm{f}}\mathbf{h}_{k-1} + \mathbf{b}^{\mathrm{f}}), \\
\mathbf{o}_k &= \sigma(\mathbf{W}^{\mathrm{o}}\mathbf{x}_k + \mathbf{V}^{\mathrm{o}}\mathbf{h}_{k-1} + \mathbf{b}^{\mathrm{o}}), \\
\mathbf{c}_k &= \mathbf{f}_k \odot \mathbf{c}_{k-1} + \mathbf{i}_k \odot \tanh(\mathbf{W}^{\mathrm{c}}\mathbf{x}_k + \mathbf{V}^{\mathrm{c}}\mathbf{h}_{k-1} + \mathbf{b}^{\mathrm{c}}), \\
\mathbf{h}_k &= \mathbf{o}_k \odot \tanh(\mathbf{c}_k), \qquad (1)
\end{aligned}$$

where σ is the sigmoid function, $\odot$ is the elementwise multiplication of two vectors, and all $\mathbf{W}^* \in \mathbb{R}^{d \times l}$,$\mathbf{V}^* \in \mathbb{R}^{d \times d}$ and $\mathbf{b}^* \in \mathbb{R}^d$ are weight matrices and vectors to be learned.

Neural Attention Model: For the natural language inference task, we have two sentences $\mathbf{X}^{\mathrm{s}} = (\mathbf{x}^{\mathrm{s}}_1, \mathbf{x}^{\mathrm{s}}_2, \ldots, \mathbf{x}^{\mathrm{s}}_M)$ and $\mathbf{X}^{\mathrm{t}} = (\mathbf{x}^{\mathrm{t}}_1, \mathbf{x}^{\mathrm{t}}_2, \ldots, \mathbf{x}^{\mathrm{t}}_N)$, where $\mathbf{X}^{\mathrm{s}}$ is the premise and $\mathbf{X}^{\mathrm{t}}$ is the hypothesis. Here each $\mathbf{x}$ is an embedding vector of the corresponding word. The goal is to predict a label y that indicates the relationship between $\mathbf{X}^{\mathrm{s}}$ and $\mathbf{X}^{\mathrm{t}}$. In this paper, we assume y is one of *entailment*, *contradiction* and *neutral*.

Rocktäschel et al. (2016) first used two LSTMs to process the premise and the hypothesis, respectively, but initialized the second LSTM (for the hypothesis) with the last cell state of the first LSTM (for the premise). Let us use $\mathbf{h}^{\mathrm{s}}_j$ and $\mathbf{h}^{\mathrm{t}}_k$ to denote the resulting hidden states corresponding to $\mathbf{x}^{\mathrm{s}}_j$ and

$\mathbf{x}^{\mathrm{t}}_k$, respectively. The main idea of the word-by-word attention model by Rocktäschel et al. (2016) is to introduce a series of attention-weighted combinations of the hidden states of the premise, where each combination is for a particular word in the hypothesis. Let us use $\mathbf{a}_k$ to denote such an *attention vector* for word $\mathbf{x}^{\mathrm{t}}_k$ in the hypothesis. Specifically, $\mathbf{a}_k$ is defined as follows[1]:

$$\mathbf{a}_k = \sum_{j=1}^{M} \alpha_{kj} \mathbf{h}^{\mathrm{s}}_j, \tag{2}$$

where α_{kj} is an attention weight that encodes the degree to which $\mathbf{x}^{\mathrm{t}}_k$ in the hypothesis is aligned with $\mathbf{x}^{\mathrm{s}}_j$ in the premise. The attention weight α_{kj} is generated in the following way:

$$\alpha_{kj} = \frac{\exp(e_{kj})}{\sum_{j'} \exp(e_{kj'})}, \tag{3}$$

where

$$e_{kj} = \mathbf{w}^{\mathrm{e}} \cdot \tanh(\mathbf{W}^{\mathrm{s}}\mathbf{h}^{\mathrm{s}}_j + \mathbf{W}^{\mathrm{t}}\mathbf{h}^{\mathrm{t}}_k + \mathbf{W}^{\mathrm{a}}\mathbf{h}^{\mathrm{a}}_{k-1}). \tag{4}$$

Here $\cdot$ is the dot-product between two vectors, the vector $\mathbf{w}^{\mathrm{e}} \in \mathbb{R}^d$ and all matrices $\mathbf{W}^* \in \mathbb{R}^{d\times d}$ contain weights to be learned, and $\mathbf{h}^{\mathrm{a}}_{k-1}$ is another hidden state which we will explain below.

The attention-weighted premise $\mathbf{a}_k$ essentially tries to model the relevant parts in the premise with respect to $\mathbf{x}^{\mathrm{t}}_k$, i.e., the k^{th} word in the hypothesis. Rocktäschel et al. (2016) further built an RNN model over $\{\mathbf{a}_k\}_{k=1}^N$ by defining the following hidden states:

$$\mathbf{h}^{\mathrm{a}}_k = \mathbf{a}_k + \tanh(\mathbf{V}^{\mathrm{a}}\mathbf{h}^{\mathrm{a}}_{k-1}), \tag{5}$$

where $\mathbf{V}^{\mathrm{a}} \in \mathbb{R}^{d\times d}$ is a weight matrix to be learned. We can see that the last $\mathbf{h}^{\mathrm{a}}_N$ aggregates all the previous $\mathbf{a}_k$ and can be seen as an attention-weighted representation of the whole premise. Rocktäschel et al. (2016) then used this $\mathbf{h}^{\mathrm{a}}_N$, which represents the whole premise, together with $\mathbf{h}^{\mathrm{t}}_N$, which can be approximately regarded as an aggregated representation of the hypothesis[2], to predict the label y.

2.2 Our Model

Although the neural attention model by Rocktäschel et al. (2016) achieved better results than Bowman et al. (2015), we see two limitations. First, the model still uses a single vector representation of the premise, namely $\mathbf{h}^{\mathrm{a}}_N$, to match the entire hypothesis. We speculate that if we instead use each of the attention-weighted representations of the premise for matching, i.e., use $\mathbf{a}_k$ at position k to match the hidden state $\mathbf{h}^{\mathrm{t}}_k$ of the hypothesis while we go through the hypothesis, we could achieve better matching results. This can be done using an RNN which at each position takes in both $\mathbf{a}_k$ and $\mathbf{h}^{\mathrm{t}}_k$ as its input and determines how well the overall matching of the two sentences is up to the current position. In the end the RNN will produce a single vector representing the matching of the two entire sentences.

The second limitation is that the model by Rocktäschel et al. (2016) does not explicitly allow us to place more emphasis on the more important matching results between the premise and the hypothesis and down-weight the less critical ones. For example, matching of stop words is presumably less important than matching of content words. Also, some matching results may be particularly critical for making the final prediction and thus should be remembered. For example, consider the premise "*A dog jumping for a Frisbee in the snow.*" and the hypothesis "*A cat washes his face and whiskers with his front paw.*" When we sequentially process the hypothesis, once we see that the subject of the hypothesis *cat* does not match the subject of the premise *dog*, we have a high probability to believe that there is a contradiction. So this mismatch should be remembered.

Based on the two observations above, we propose to use an LSTM to sequentially match the two sentences. At each position the LSTM takes in both $\mathbf{a}_k$ and $\mathbf{h}^{\mathrm{t}}_k$ as its input. Figure 1 gives an overview of our model in contrast to the model by Rocktäschel

[1]We present the word-by-word attention model by Rocktäschel et al. (2016) in a different way but the underlying model is the same. Our $\mathbf{h}^{\mathrm{a}}_k$ is their $\mathbf{r}_t$, our $\mathbf{H}^{\mathrm{s}}$ (all of $\mathbf{h}^{\mathrm{s}}_j$) is their $\mathbf{Y}$, our $\mathbf{h}^{\mathrm{t}}_k$ is their $\mathbf{h}_t$, and our α_k is their α_t. Our presentation is close to the one by Bahdanau et al. (2015), with our attention vectors $\mathbf{a}$ corresponding to the context vectors $\mathbf{c}$ in their paper.

[2]Strictly speaking, in the model by Rocktäschel et al. (2016), $\mathbf{h}^{\mathrm{t}}_N$ encodes both the premise and the hypothesis because the two sentences are chained. But $\mathbf{h}^{\mathrm{t}}_N$ places a higher emphasis on the hypothesis given the nature of RNNs.

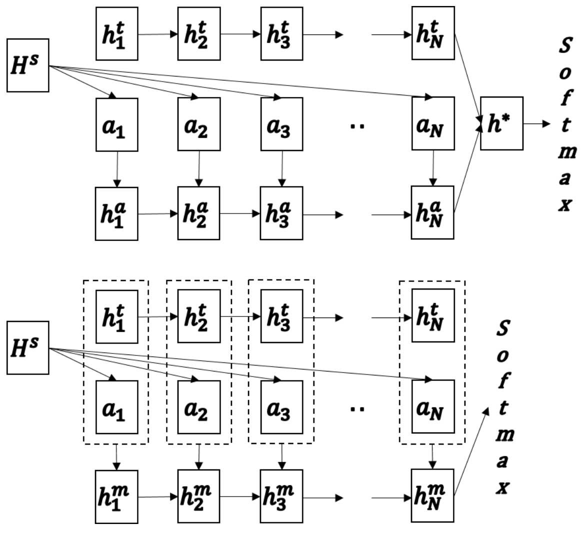

Figure 1: The top figure depicts the model by Rocktäschel et al. (2016) and the bottom figure depicts our model. Here $\mathbf{H}^{\mathrm{s}}$ represents all the hidden states $\mathbf{h}_j^{\mathrm{s}}$. Note that in the top model each $\mathbf{h}_k^{\mathrm{a}}$ represents a weighted version of the premise only, while in our model, each $\mathbf{h}_k^{\mathrm{m}}$ represents the matching between the premise and the hypothesis up to position k.

et al. (2016).

Specifically, our model works as follows. First, similar to Rocktäschel et al. (2016), we process the premise and the hypothesis using two LSTMs, but we do not feed the last cell state of the premise to the LSTM of the hypothesis. This is because we do not need the LSTM for the hypothesis to encode any knowledge about the premise but we will match the premise with the hypothesis using the hidden states of the two LSTMs. Again, we use $\mathbf{h}_j^{\mathrm{s}}$ and $\mathbf{h}_k^{\mathrm{t}}$ to represent these hidden states.

Next, we generate the attention vectors $\mathbf{a}_k$ similarly to Eqn (2). However, Eqn (4) will be replaced by the following equation:

$$e_{kj} = \mathbf{w}^{\mathrm{e}} \cdot \tanh(\mathbf{W}^{\mathrm{s}}\mathbf{h}_j^{\mathrm{s}} + \mathbf{W}^{\mathrm{t}}\mathbf{h}_k^{\mathrm{t}} + \mathbf{W}^{\mathrm{m}}\mathbf{h}_{k-1}^{\mathrm{m}}). \quad (6)$$

The only difference here is that we use a hidden state $\mathbf{h}^{\mathrm{m}}$ instead of $\mathbf{h}^{\mathrm{a}}$, and the way we define $\mathbf{h}^{\mathrm{m}}$ is very different from the definition of $\mathbf{h}^{\mathrm{a}}$.

Our $\mathbf{h}_k^{\mathrm{m}}$ is the hidden state at position k generated from our *m*LSTM. This LSTM models the *matching* between the premise and the hypothesis. Important matching results will be "remembered" by the LSTM while non-essential ones will be "forgotten." We use the concatenation of $\mathbf{a}_k$, which is the attention-weighted version of the premise for the k^{th} word in the hypothesis, and $\mathbf{h}_k^{\mathrm{t}}$, the hidden state for the k^{th} word itself, as input to the *m*LSTM.

Specifically, let us define

$$\mathbf{m}_k = \begin{bmatrix} \mathbf{a}_k \\ \mathbf{h}_k^{\mathrm{t}} \end{bmatrix}. \quad (7)$$

We then build the *m*LSTM as follows:

$$\begin{aligned}
\mathbf{i}_k^{\mathrm{m}} &= \sigma(\mathbf{W}^{\mathrm{mi}}\mathbf{m}_k + \mathbf{V}^{\mathrm{mi}}\mathbf{h}_{k-1}^{\mathrm{m}} + \mathbf{b}^{\mathrm{mi}}), \\
\mathbf{f}_k^{\mathrm{m}} &= \sigma(\mathbf{W}^{\mathrm{mf}}\mathbf{m}_k + \mathbf{V}^{\mathrm{mf}}\mathbf{h}_{k-1}^{\mathrm{m}} + \mathbf{b}^{\mathrm{mf}}), \\
\mathbf{o}_k^{\mathrm{m}} &= \sigma(\mathbf{W}^{\mathrm{mo}}\mathbf{m}_k + \mathbf{V}^{\mathrm{mo}}\mathbf{h}_{k-1}^{\mathrm{m}} + \mathbf{b}^{\mathrm{mo}}), \\
\mathbf{c}_k^{\mathrm{m}} &= \mathbf{f}_k^{\mathrm{m}} \odot \mathbf{c}_{k-1}^{\mathrm{m}} + \mathbf{i}_k^{\mathrm{m}} \odot \tanh(\mathbf{W}^{\mathrm{mc}}\mathbf{m}_k + \mathbf{V}^{\mathrm{mc}}\mathbf{h}_{k-1}^{\mathrm{m}} \\
&\quad + \mathbf{b}^{\mathrm{mc}}), \\
\mathbf{h}_k^{\mathrm{m}} &= \mathbf{o}_k^{\mathrm{m}} \odot \tanh(\mathbf{c}_k^{\mathrm{m}}). \quad (8)
\end{aligned}$$

With this *m*LSTM, finally we use only $\mathbf{h}_N^{\mathrm{m}}$, the last hidden state, to predict the label y.

2.3 Implementation Details

Besides the difference of the LSTM architecture, we also introduce a few other changes from the model by Rocktäschel et al. (2016). First, we insert a special word *NULL* to the premise, and we allow words in the hypothesis to be aligned with this *NULL*. This is inspired by common practice in machine translation. Specifically, we introduce a vector $\mathbf{h}_0^{\mathrm{s}}$, which is fixed to be a vector of 0s of dimension d. This $\mathbf{h}_0^{\mathrm{s}}$ represents *NULL* and is used with other $\mathbf{h}_j^{\mathrm{s}}$ to derive the attention vectors $\{\mathbf{a}_k\}_{k=1}^N$.

Second, we use word embeddings trained from GloVe (Pennington et al., 2014) instead of word2vec vectors. The main reason is that GloVe covers more words in the SNLI corpus than word2vec[3].

Third, for words which do not have pre-trained word embeddings, we take the average of the embeddings of all the words (in GloVe) surrounding the unseen word within a window size of 9 (4 on the left and 4 on the right) as an approximation of the embedding of this unseen word. Then we do not update any word embedding when learning our model. Although this is a very crude approximation, it reduces

[3]The SNLI corpus contains 37K unique tokens. Around 12.1K of them cannot be found in word2vec but only around 4.1K of them cannot be found in GloVe.

the number of parameters we need to update, and as it turns out, we can still achieve better performance than Rocktäschel et al. (2016).

3 Experiments

3.1 Experiment Settings

Data: We use the SNLI corpus to test the effectiveness of our model. The original data set contains 570,152 sentence pairs, each labeled with one of the following relationships: *entailment*, *contradiction*, *neutral* and –, where – indicates a lack of consensus from the human annotators. We discard the sentence pairs labeled with – and keep the remaining ones for our experiments. In the end, we have 549,367 pairs for training, 9,842 pairs for development and 9,824 pairs for testing. This follows the same data partition used by Bowman et al. (2015) in their experiments. We perform three-class classification and use accuracy as our evaluation metric.

Parameters: We use the Adam method (Kingma and Ba, 2014) with hyperparameters β_1 set to 0.9 and β_2 set to 0.999 for optimization. The initial learning rate is set to be 0.001 with a decay ratio of 0.95 for each iteration. The batch size is set to be 30. We experiment with $d = 150$ and $d = 300$ where d is the dimension of all the hidden states.

Methods for comparison: We mainly want to compare our model with the word-by-word attention model by Rocktäschel et al. (2016) because this model achieved the state-of-the-art performance on the SNLI corpus. To ensure fair comparison, besides comparing with the accuracy reported by Rocktäschel et al. (2016), we also re-implemented their model and report the performance of our implementation. We also consider a few variations of our model. Specifically, the following models are implemented and tested in our experiments:

- Word-by-word attention ($d = 150$): This is our implementation of the word-by-word attention model by Rocktäschel et al. (2016), where we set the dimension of the hidden states to 150. The differences between our implementation and the original implementation by Rocktäschel et al. (2016) are the following: (1) We also add a *NULL* token to the premise for matching. (2) We do not feed the last cell state of the LSTM for the premise to the LSTM for the hypothesis, to keep it consistent with the implementation of our model. (3) For word representation, we also use the GloVe word embeddings and we do not update the word embeddings. For unseen words, we adopt the same strategy as described in Section 2.3.
- *m*LSTM ($d = 150$): This is our *m*LSTM model with d set to 150.
- *m*LSTM with bi-LSTM sentence modeling ($d = 150$): This is the same as the model above except that when we derive the hidden states $\mathbf{h}^{\text{s}}_j$ and $\mathbf{h}^{\text{t}}_k$ of the two sentences, we use bi-LSTMs (Graves, 2012) instead of LSTMs. We implement this model to see whether bi-LSTMs allow us to better align the sentences.
- *m*LSTM ($d = 300$): This is our *m*LSTM model with d set to 300.
- *m*LSTM with word embedding ($d = 300$): This is the same as the model above except that we directly use the word embedding vectors $\mathbf{x}^{\text{s}}_j$ and $\mathbf{x}^{\text{t}}_k$ instead of the hidden states $\mathbf{h}^{\text{s}}_j$ and $\mathbf{h}^{\text{t}}_k$ in our model. In this case, each attention vector $\mathbf{a}_k$ is a weighted sum of $\{\mathbf{x}^{\text{s}}_j\}_{j=1}^{M}$. We experiment with this setting because we hypothesize that the effectiveness of our model is largely related to the *m*LSTM architecture rather than the use of LSTMs to process the original sentences.

3.2 Main Results

Table 1 compares the performance of the various models we tested together with some previously reported results.

We have the following observations: (1) First of all, we can see that when we set d to 300, our model achieves an accuracy of 86.1% on the test data, which to the best of our knowledge is the highest on this data set. (2) If we compare our *m*LSTM model with our implementation of the word-by-word attention model by Rocktäschel et al. (2016) under the same setting with $d = 150$, we can see that our performance on the test data (85.7%) is higher than that of their model (82.6%). We also tested statistical significance and found the improvement to be statistically significant at the 0.001 level. (3) The performance of *m*LSTM with bi-LSTM sentence modeling compared with the model with standard LSTM sentence modeling when d is set to 150 shows that using bi-LSTM to process the original sentences helps

Model	d	$\lvert\theta\rvert_{\text{W+M}}$	$\lvert\theta\rvert_{\text{M}}$	Train	Dev	Test
LSTM [Bowman et al. (2015)]	100	10M	221K	84.4	-	77.6
Classifier [Bowman et al. (2015)]	-	-	-	99.7	-	78.2
LSTM shared [Rocktäschel et al. (2016)]	159	3.9M	252K	84.4	83.0	81.4
Word-by-word attention [Rocktäschel et al. (2016)]	100	3.9M	252K	85.3	83.7	83.5
Word-by-word attention (our implementation)	150	340K	340K	85.5	83.3	82.6
*m*LSTM	150	544K	544K	91.0	86.2	85.7
*m*LSTM with bi-LSTM sentence modeling	150	1.4M	1.4M	91.3	86.6	86.0
*m*LSTM	300	1.9M	1.9M	92.0	**86.9**	**86.1**
*m*LSTM with word embedding	300	1.3M	1.3M	88.6	85.4	85.3

Table 1: Experiment results in terms of accuracy. d is the dimension of the hidden states. $|\theta|_{\text{W+M}}$ is the total number of parameters and $|\theta|_{\text{M}}$ is the number of parameters excluding the word embeddings. Note that the five models in the last section were implemented by us while the other results were taken directly from previous papers. Note also that for the five models in the last section, we do not update word embeddings so $|\theta|_{\text{W+M}}$ is the same as $|\theta|_{\text{M}}$. The three columns on the right are the accuracies of the trained models on the training data, the development data and the test data, respectively.

	ground truth		
prediction	*N*	*E*	*C*
N	2628	286	255
E	340	3005	159
C	250	77	2823

Table 2: The confusion matrix of the results by *m*LSTM with $d = 300$. *N*, *E* and *C* correspond to *neutral*, *entailment* and *contradiction*, respectively.

(86.0% vs. 85.7% on the test data), but the difference is small and the complexity of bi-LSTM is much higher than LSTM. Therefore when we increased d to 300 we did not experiment with bi-LSTM sentence modeling. (4) Interestingly, when we experimented with the *m*LSTM model using the pre-trained word embeddings instead of LSTM-generated hidden states as initial representations of the premise and the hypothesis, we were able to achieve an accuracy of 85.3% on the test data, which is still better than previously reported state of the art. This suggests that the *m*LSTM architecture coupled with the attention model works well, regardless of whether or not we use LSTM to process the original sentences.

Because the NLI task is a three-way classification problem, to better understand the errors, we also show the confusion matrix of the results obtained by our *m*LSTM model with $d = 300$ in Table 2. We can see that there is more confusion between *neutral* and *entailment* and between *neutral* and *contradiction* than between *entailment* and *contradiction*. This shows that *neutral* is relatively hard to capture.

3.3 Further Analyses

To obtain a better understanding of how our proposed model actually performs the matching between a premise and a hypothesis, we further conduct the following analyses. First, we look at the learned word-by-word alignment weights α_{kj} to check whether the soft alignment makes sense. This is the same as what was done by Rocktäschel et al. (2016). We then look at the values of the various gate vectors of the *m*LSTM. By looking at these values, we aim to check (1) whether the model is able to differentiate between more important and less important word-level matching results, and (2) whether the model forgets certain matching results and remembers certain other ones.

To conduct the analyses, we choose three examples and display the various learned parameter values. These three sentence pairs share the same premise but have different hypotheses and different relationship labels. They are given in Table 3. The values of the alignment weights and the gate vectors are plotted in Figure 2.

Besides using the three examples, we will also give some overall statistics of the parameter values to confirm our observations with the three examples.

	ID	sentence	label
Premise		A dog jumping for a Frisbee in the snow.	
Hypothesis	Example 1	An animal is outside in the cold weather, playing with a plastic toy.	*entailment*
	Example 2	A cat washed his face and whiskers with his front paw.	*contradiction*
	Example 3	A pet is enjoying a game of fetch with his owner.	*neutral*

Table 3: Three examples of sentence pairs with different relationship labels. The second hypothesis is a contradiction because it mentions a completely different event. The third hypothesis is neutral to the premise because the phrase "with his owner" cannot be inferred from the premise.

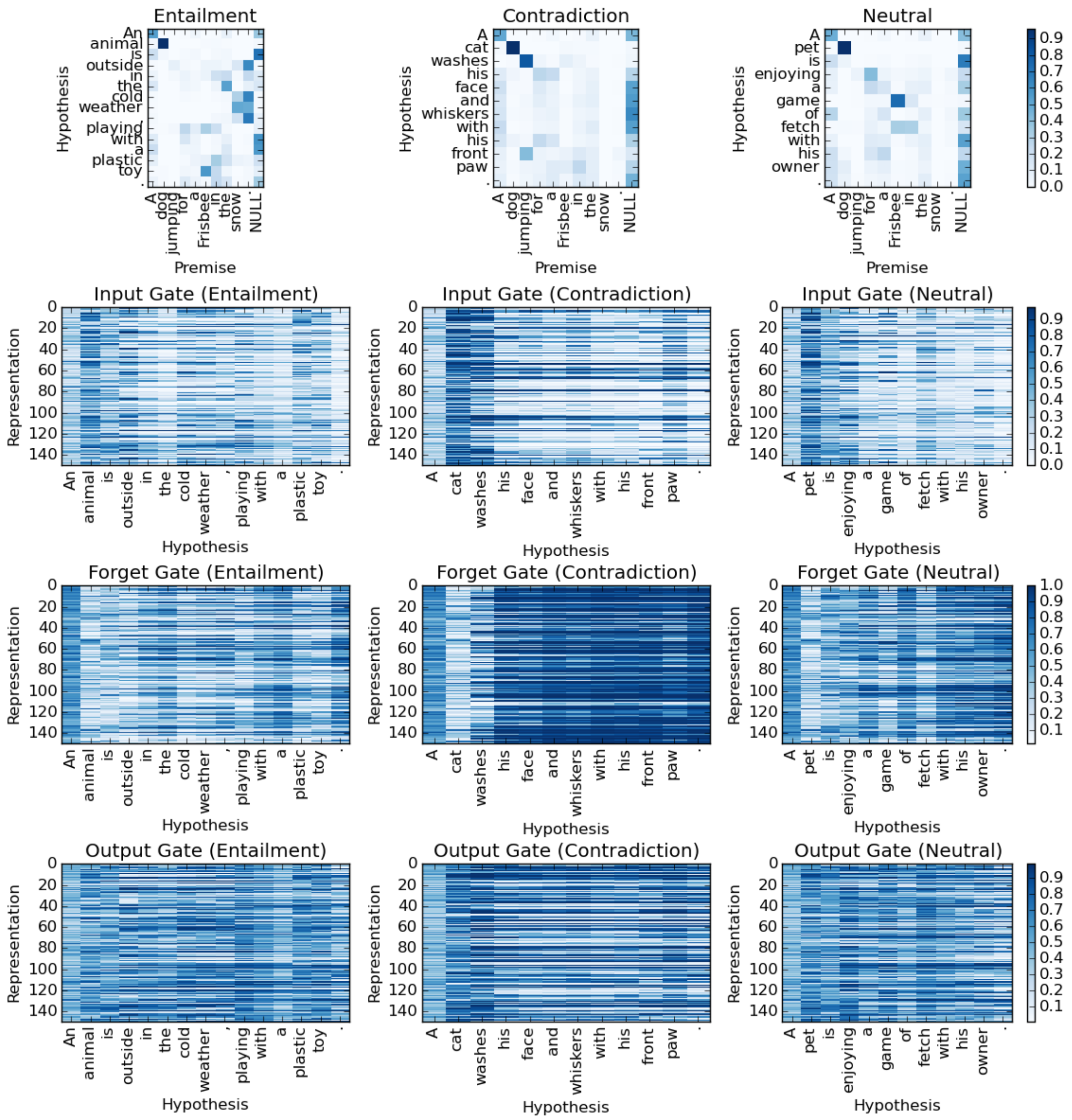

Figure 2: The alignment weights and the gate vectors of the three examples.

Word Alignment

First, let us look at the top-most plots of Figure 2. These plots show the alignment weights α_{kj} between the hypothesis and the premise, where a darker color corresponds to a larger value of α_{kj}. Recall that α_{kj} is the degree to which the k^{th} word in the hypothesis is aligned with the j^{th} word in the premise. Also recall that the weights α_{kj} are configured such that for the same k all the α_{kj} add up to 1. This means the weights in the same row in these plots add up to 1. From the three plots we can see that the alignment weights generally make sense. For example, in Example 1, "animal" is strongly aligned with "dog" and "toy" aligned with "Frisbee." The phrase "cold weather" is aligned with "snow." In Example 3, we also see that "pet" is strongly aligned with "dog" and "game" aligned with "Frisbee."

In Example 2, "cat" is strongly aligned with "dog" and "washes" is aligned with "jumping." It may appear that these matching results are wrong. However, "dog" is likely the best match for "cat" among all the words in the premise, and as we will show later, this match between "cat" and "dog" is actually a strong indication of a contradiction between the two sentences. The same explanation applies to the match between "washes" and "jumping."

We also observe that some words are aligned with the *NULL* token we inserted. For example, the word "is" in the hypothesis in Example 1 does not correspond to any word in the premise and is therefore aligned with *NULL*. The words "face" and "whiskers" in Example 2 and "owner" in Example 3 are also aligned with *NULL*. Intuitively, if some important content words in the hypothesis are aligned with *NULL*, it is more likely that the relationship label is either contradiction or neutral.

Values of Gate Vectors

Next, let us look at the values of the learned gate vectors of our *m*LSTM for the three examples. We show these values under the setting where d is set to 150. Each row of these plots corresponds to one of the 150 dimensions. Again, a darker color indicates a higher value.

An input gate controls whether the input at the current position should be used in deriving the final hidden state of the current position. From the three plots of the input gates, we can observe that generally for stop words such as prepositions and articles the input gates have lower values, suggesting that the matching of these words is less important. On the other hand, content words such as nouns and verbs tend to have higher values of the input gates, which also makes sense because these words are generally more important for determining the final relationship label.

To further verify the observation above, we compute the average input gate values for stop words and the other content words. We find that the former has an average value of 0.287 with a standard deviation of 0.084 while the latter has an average value of 0.347 with a standard deviation of 0.116. This shows that indeed generally stop words have lower input gate values. Interestingly, we also find that some stop words may have higher input gate values if they are critical for the classification task. For example, the negation word "not" has an average input gate value of 0.444 with a standard deviation of 0.104.

Overall, the values of the input gates confirm that the *m*LSTM helps differentiate the more important word-level matching results from the less important ones.

Next, let us look at the forget gates. Recall that a forget gate controls the importance of the *previous* cell state in deriving the final hidden state of the current position. Higher values of a forget gate indicate that we need to remember the previous cell state and pass it on whereas lower values indicate that we should probably forget the previous cell. From the three plots of the forget gates, we can see that overall the colors are the lightest for Example 1, which is an *entailment*. This suggests that when the hypothesis is an entailment of the premise, the *m*LSTM tends to forget the previous matching results. On the other hand, for Example 2 and Example 3, which are *contradiction* and *neutral*, we see generally darker colors. In particular, in Example 2, we can see that the colors are consistently dark starting from the word "his" in the hypothesis until the end. We believe the explanation is that after the *m*LSTM processes the first three words of the hypothesis, "A cat washes," it sees that the matching between "cat" and "dog" and between "washes" and "jumping" is a strong indication of a contradiction, and therefore these matching results need to be remembered until the end of the *m*LSTM for the final prediction.

We have also checked the forget gates of the other sentence pairs in the test data by computing the average forget gate values and the standard deviations for *entailment*, *neutral* and *contradiction*, respectively. We find that the values are 0.446 ± 0.123, 0.507±0.148 and 0.536±0.170, respectively. For *contradiction* and *neutral*, the forget gates start to have higher values from certain positions of the hypotheses.

Based on the observations above, we hypothesize that the way the *m*LSTM works is as follows. It remembers important mismatches, which are useful for predicting the *contradiction* or the *neutral* relationship, and forgets good matching results. At the end of the *m*LSTM, if no important mismatch is remembered, the final classifier will likely predict *entailment* by default. Otherwise, depending on the kind of mismatch remembered, the classifier will predict either *contradiction* or *neutral*.

For the output gates, we are not able to draw any important conclusion except that the output gates seem to be positively correlated with the input gates but they tend to be darker than the input gates.

4 Related Work

There has been much work on natural language inference. Shallow methods rely mostly on lexical similarities but are shown to be robust. For example, Bowman et al. (2015) experimented with a lexicalized classifier-based method, which only uses lexical information and achieves an accuracy of 78.2% on the SNLI corpus. More advanced methods use syntactic structures of the sentences to help matching them. For example, Mehdad et al. (2009) applied syntactic-semantic tree kernels for recognizing textual entailment. Because inference is essentially a logic problem, methods based on formal logic (Clark and Harrison, 2009) or natural logic (MacCartney, 2009) have also been proposed. A comprehensive review on existing work can be found in the book by Dagan et al. (2013).

The work most relevant to ours is the recently proposed neural attention model-based method by Rocktäschel et al. (2016), which we have detailed in previous sections. Neural attention models have recently been applied to some natural language processing tasks including machine translation (Bahdanau et al., 2015), abstractive summarization (Rush et al., 2015) and question answering (Hermann et al., 2015). Rocktäschel et al. (2016) showed that the neural attention model could help derive a better representation of the premise to be used to match the hypothesis, whereas in our work we also use it to derive representations of the premise that are used to sequentially match the words in the hypothesis.

The SNLI corpus is new and so far it has only been used in a few studies. Besides the work by Bowman et al. (2015) themselves and by Rocktäschel et al. (2016), there are two other studies which used the SNLI corpus. Vendrov et al. (2015) used a *Skip-Thought* model proposed by Kiros et al. (2015) to the NLI task and reported an accuracy of 81.5% on the test data. Mou et al. (2015) used tree-based CNN encoders to obtain sentence embeddings and achieved an accuracy of 82.1%.

5 Conclusions and Future Work

In this paper, we proposed a special LSTM architecture for the task of natural language inference. Based on a recent work by Rocktäschel et al. (2016), we first used neural attention models to derive attention-weighted vector representations of the premise. We then designed a match-LSTM that processes the hypothesis word by word while trying to match the hypothesis with the premise. The last hidden state of this *m*LSTM can be used for predicting the relationship between the premise and the hypothesis. Experiments on the SNLI corpus showed that the *m*LSTM model outperformed the state-of-the-art performance reported so far on this data set. Moreover, closer analyses on the gate vectors revealed that our *m*LSTM indeed remembers and passes on important matching results, which are typically mismatches that indicate a *contradiction* or a *neutral* relationship between the premise and the hypothesis.

With the large number of parameters to learn, an inevitable limitation of our model is that a large training data set is needed to learn good model parameters. Indeed some preliminary experiments applying our *m*LSTM to the SICK corpus (Marelli et al., 2014), a smaller textual entailment benchmark data set, did not give very good results. We believe that this is because our model learns everything from scratch except using the pre-trained word embeddings. A future direction would be to incorporate other resources such as the paraphrase database (Ganitkevitch et al., 2013) into the learning process so that such prior knowledge can be utilized.

References

Dzmitry Bahdanau, HyungHyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*.

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*.

Peter Clark and Phil Harrison. 2009. An inference-based approach to recognizing entailment. In *Proceedings of the Text Analysis Conference*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL Recognising Textual Entailment Challenge. In *Proceedings of the PASCAL Challenges Workshop on Recognizing Textual Entailment*.

Ido Dagan, Dan Roth, Mark Sammons, and Fabio Massimo Zanzotto. 2013. *Recognizing Textual Entailment: Models and Applications*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. PPDB: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics*.

Oren Glickman, Ido Dagan, and Moshe Koppel. 2005. Web based probabilistic textual entailment. In *Proceedings of the PASCAL Challenges Workshop on Recognizing Textual Entailment*.

Alex Graves. 2012. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations*.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in Neural Information Processing Systems*.

Bill MacCartney, Michel Galley, and Christopher D Manning. 2008. A phrase-based alignment model for natural language inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Bill MacCartney. 2009. *Natural Language Inference*. Ph.D. thesis, Stanford University.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*.

Yashar Mehdad, Alessandro Moschitti1, and Fabio Massiomo Zanzotto. 2009. SemKer: Syntactic/semantic kernels for recognizing textual entailment. In *Proceedings of the Text Analysis Conference*.

Lili Mou, Men Rui, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. 2015. Recognizing entailment and contradiction by tree-based convolution. *arXiv preprint arXiv:1512.08422*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, and Phil Blunsom. 2016. Reasoning about entailment with neural attention. In *Proceedings of the International Conference on Learning Representations*.

Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. 2015. Order-embeddings of images and language. *arXiv preprint arXiv:1511.06361*.