LSTM Sentiment Analyis Project

Alex Trott

July 20, 2016

1 Introduction

This work was completed as part of the coding task associated with an interview at MetaMind. The task was to replicate the baseline results of Sheng Tai et al. (2015)[1] on the Sentiment Analysis data set available here[2], specifically those results related to the performance of the chain LSTM RNN variants. This includes the standard LSTM and the Bi-directional LSTM. The project was completed using the Theano deep learning framework, the code for which is available online[3]

2 Sentiment analysis

The data set used for this project includes over 10,000 snippets of movie reviews collected from Rotten Tomatoes. Each review is scored between 0 and 1 to describe its overall sentiment (0 being the most negative, and 1 the most positive).

2.1 Classification tasks

The task of sentiment classification was approached at two levels of granularity. Fine-grained sentiment classification was performed after discretizing the scores into 5 bins, such that the task was to classify the sentiment bin (i.e. polarity and strength) of each review. Alternatively, the scores were binarized to represent either positive or negative sentiment, and the task was to perform this binary classification. In the latter case, reviews with sentiment scores in the domain $[0.4, 0.6)$ were removed from the data set. For the current project, this resulted in train/test/dev sizes of 8114/2125/1044 for the fine-grained classification task and 6568/1749/825 for the binary classification task, respectively. (Some reviews were excluded because they could not be matched to any scored phrases in the raw data set.)

2.2 Vector representations

To make this data set suitable for a deep learning application, the movie reviews had to first be converted into a numerical representation. A standard approach

[1] Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks

[2] http://nlp.stanford.edu/sentiment/

[3] https://github.com/alextrott16/sentimiento

is to tokenize each review and replace each token by a learned vector embedding. This project took advantage of the pre-trained corpus of word vectors produced by the GloVe algorithm[4], which provides a dictionary of 300-dimensional word vectors for roughly 2.2M cased tokens. Movie reviews were converted into vector sequences according to this dictionary; unrecognized tokens were represented as a vector of zeros.

3 Models and training

The two models considered here were the standard LSTM RNN and the Bi-directional variant.

3.1 LSTM

The LSTM RNN has served as a popular model for dealing with sequential problems due to its architecture, which modifies the typical RNN by including a memory cell for each hidden unit. The contents of each memory cell are updated according to accompanying input and forget gates, and the memory cell is read out to influence the hidden state according to yet another output gate. Specifically, for this application, the hidden state of the LSTM at each sequence position, h_t was calculated as:

$$i_t = \sigma(W^{ix}x_t + W^{ih}h_{t-1} + b^i) \tag{1}$$
$$f_t = \sigma(W^{fx}x_t + W^{fh}h_{t-1} + b^f) \tag{2}$$
$$o_t = \sigma(W^{ox}x_t + W^{oh}h_{t-1} + b^o) \tag{3}$$
$$u_t = tanh(W^{ux}x_t + W^{uh}h_{t-1} + b^u) \tag{4}$$
$$c_t = i_t \odot u_t + f_t \odot c_{t-1} \tag{5}$$
$$h_t = o_t \odot tanh(c_t) \tag{6}$$

Where, x_t specifies the input (that is, as a word vector) at position t and σ the sigmoid nonlinearity. For a sequence of T tokens, classification was performed as

$$p(S = 1|x_{1:T}) = \sigma(Wh_T + b) \tag{7}$$

for the binarized scores and as

$$p(S = s|x_{1:T}) = \frac{e^{W^s h_T + b^s}}{\Sigma_{s'} e^{W^{s'} h_T + b^{s'}}} \tag{8}$$

for the fine-grained scores, where $p(S = s|x_{1:T})$ indicates the probability that the sentiment score for a review with vector representation $x_{1:T}$ takes on some value $s \in 1, ..., 5$.

[4]http://nlp.stanford.edu/projects/glove/

3.2 Bi-directional LSTM

The bi-directional variant of the LSTM employs all the same mechanisms as the standard LSTM. The difference is that each bi-directional layer is essentially the concatenation of two LSTMs, one that processes the input sequence as $x_{1:T}$ and the other that processes it backwards as $x_{T:1}$. Because this project did not experiment with network depth, it was sufficient for the classification task to define the output of a bi-directional LSTM as

$$h_T = [h_T^F, h_T^B]$$

where h_T^F is the output of the forward LSTM at the end of the input sequence and h_T^B is the output of the backwards LSTM at the end of the reversed input sequence and brackets indicate concatenation. As with the standard LSTM, classification was carried out according to Equation 7 or 8 depending on the task.
For consistency with Sheng Tai et al. (2015), the weights of the forwards and backwards LSTMs were tied.

3.3 Training

Models performing the binary and fine-grained classification tasks were trained according to the binary cross-entropy loss and categorical cross-entropy loss, respectively. This project utilized the Adam optimization method, using a learning rate of 0.005 for the standard LSTM and 0.001 for the bi-directional variant. Orthogonal initialization was applied to the LSTM weights.
Training was terminated by early-stopping: performance on the development split was periodically evaluated to detect when training no longer produced improvement on these examples. After roughly 10 epochs of training without improvement of development performance, training was terminated and the final parameters were then used to evaluate test performance.
Regularization was performed through two methods. First, an $L2$ penalty was added to the training objective. This penalty was weighted by 10^{-4} and ignored all bias parameters. Second, dropout was applied both to the inputs and the hidden activations at a rate of 0.5. For each portion of the network where dropout was applied, the same dropout mask was used throughout a batch and was refreshed after each iteration of gradient descent. For a given iteration, the dropout mask did not change from one sequence position to the next. The batch size was set to 25 throughout training.

4 Performance

This project applied the two LSTM variants described above to the two sentiment classification tasks. For comparison with Sheng Tai et al., the LSTM components had a dimensionality of 168. Since the forwards and backwards components of the bi-directional LSTM were tied, the two variants had similar numbers of parameters. However, the read-out layers of the bi-directional variants used twice as many parameters since the concatenated h_T is twice as long.

Architecture	Model performance	Sheng Tai et al.
Standard LSTM	**87.4**	84.9 (0.6)
Bi-directional LSTM	85.6	**87.5 (0.5)**

Table 1: Model performance achieved during this project (middle) and published performance (right) for the **binary** classification task. Values correspond to percent correct classification. Values inside parentheses show standard deviation measured across NUM training runs.

Architecture	Model performance	Sheng Tai et al.
Standard LSTM	**47.8**	46.4 (1.1)
Bi-directional LSTM	45.8	**49.1 (1.0)**

Table 2: Performance for the **fine-grained** task. Same conventions as Table 1

Model performance and published baselines for the binary sentiment classification task are reported in Table 1. Training the standard LSTM using the methods described above improved the test-set classification accuracy compared to the published baseline. The most likely source of this difference is hyperparameter and/or optimization settings since this project adopted an identical LSTM architecture as Sheng Tai et al.

Interestingly, the same hyperparameter and optimization settings used to produce improved performance with the standard LSTM led to weaker performance when training a bi-directional LSTM. To be certain, the bi-directional variant should be at least as expressive as the standard LSTM. To demonstrate this, I initialized the bi-directional LSTM with the parameters learned for the standard LSTM. After a few rounds of training to allow the readout weights to adjust, this initialization yielded comparable performance to the standard LSTM (>87% correct), but further training never continued to improve development/test accuracy. Therefore, the reduced performance reported above likely reflects the added difficulty of finding a solution that generalizes well when using the bi-directional architecture.

The same trend describes the performance on the fine-grained classification task (Table 2). As before, the current training procedure yields improved test performance relative to Sheng Tai et al. when training with the standard LSTM but this improvement is not observed when training with the bi-directional LSTM. Again, this cannot be a constraint of the model architecture since, for both tasks, bi-directional LSTM performance could be almost immediately set to standard LSTM performance by initializing with the weights learned for the latter. Instead, this result is likely the outcome of bi-directional LSTMs (at least those with tied forward/backward parameters) being more sensitive to hyperparameter settings. Some effort was applied to fine-tune these hyperparameters for the bi-directional LSTM, but it is likely that a more exhaustive search would be required to fully replicate the performance observed by Sheng Tai et al.