Long Short-Term Memory-Networks for Machine Reading

Ege Ersü
Department of Computer Engineering
Koc University
Istanbul, Turkey
eersu16@ku.edu.tr

Abstract

This is a technical report introducing the Long Short-Term Memory Network (LSTMN) [1], a sequence level network that is inspired by the mechanisms of human language processing. It is an extension of the vanilla Long Short-Term Memory architecture, which makes use of an additional external memory tape and a hidden tape. The LSTMN also uses neural attention to model relations among tokens. Experiments on Sentiment Analysis and Natural Language Inference show that the model outperforms the vanilla LSTM. The report also includes a comparison of our results with the original authors' results.

1 Introduction

The model is called a machine reader since it can be used to solve many different natural language processing tasks. The report will first define LSTMNs as a general-purpose reading simulator, and later propose LSTMN architectures that focus on specific tasks, handling different input structures.

A Recurrent Neural Network (RNN) treats each sentence as a sequence of words and tries to build a useful representation of its meaning in an iterative fashion. It starts from the leftmost word and computes a hidden state (h_1) that hopefully is a useful representation of the meaning of the sentence up until that word. Once the model starts processing the next word, it takes into account the current word it is on (x_t), represented as a vector, and the hidden state computed at the previous time-step (h_{t-1}). By using these two inputs the RNN computes a new hidden state (h_t) which it then passes on to the next time-step. The RNN keeps computing hidden states until the sentence is over and in the end we hope that the final hidden state is a useful representation of the meaning of the entire sentence, which we can then use to solve many different tasks. A Long Short-Term Memory (LSTM) [2] works just like the RNN, but at each time-step it additionally takes into account a memory vector (c_{t-1}), which is

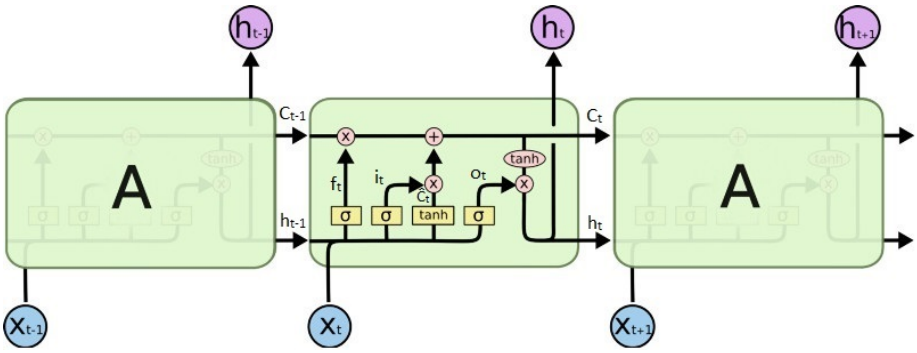

Figure 1: LSTM

also used in computing the next hidden (h_t) and memory vector (c_t). For most classification tasks, a common approach is to feed hidden states into a softmax function, and get a probability distribution over all target classes (p_t).

The formal definition of the LSTM is as follows:

$$f_t = \sigma([h_{t-1}, x_t] \odot W_f + b_f) \tag{1}$$

$$i_t = \sigma([h_{t-1}, x_t] \odot W_i + b_i) \tag{2}$$

$$\hat{C}_t = \tanh([h_{t-1}, x_t] \odot W_C + b_C) \tag{3}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t \tag{4}$$

$$o_t = \sigma([h_{t-1}, x_t] \odot W_o + b_o) \tag{5}$$

$$h_t = o_t \odot \tanh(C_t) \tag{6}$$

$$p_t = softmax(h_t * W_v) \tag{7}$$

Recurrent Neural Networks (RNN) were shown to be successful language modeling tools for a variety of different tasks. They achieved noticeable accuracy on tasks such as machine translation (Bahdanau et al., 2014), sentence compression (Hermann et al., 2015), ontology learning (Poon and Domingos, 2010) and textual entailment (Rocktaschelet et al., 2016)

2 The Model

The LSTMN, in addition to its core LSTM weights, maintains two sets of vectors, stored in a hidden tape $[h_t \forall t]$, and a memory tape $[c_t \forall t]$. At each time-step t, the produced hidden (h_t) and cell state (c_t) are stored in their corresponding tapes, and then passed on to the next time-step. Therefore, each token (x_t) is associated with a hidden vector and a memory vector. At time-step t, the model

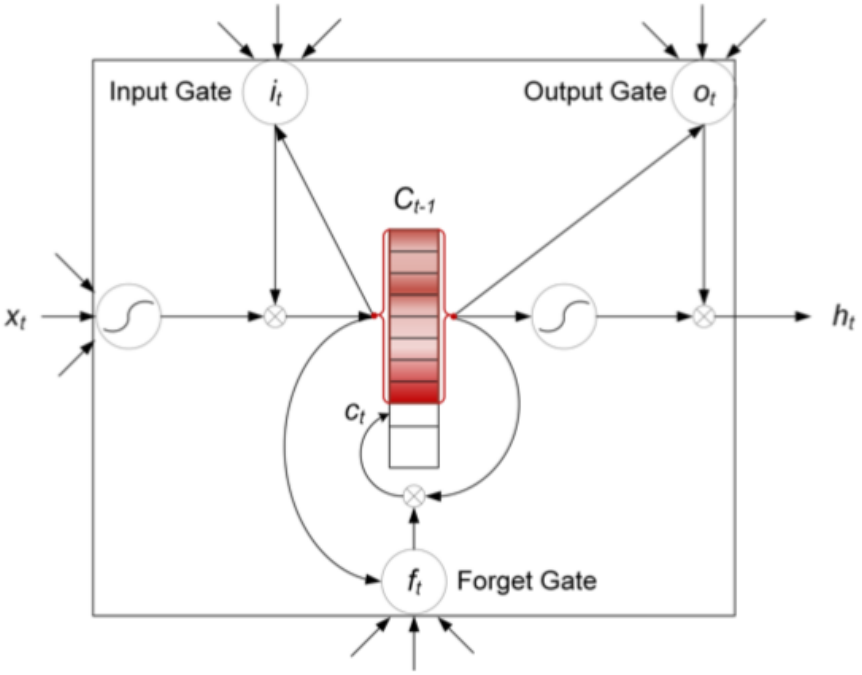

Figure 2: LSTMN Cell

computes the relation between x_t and x_1,...,x_{t-1} through h_1 ... h_{t-1} through the attention layer:

$$a_i^t = v^T tanh(W_h h_i + W_x x_t + W_{\tilde{h}} \tilde{h}_{t-1}) \tag{8}$$

$$s_i^t = softmax(a_i^t) \tag{9}$$

where v, W_h, W_x, $W_{\tilde{h}}$, $W_{\tilde{h}}$ are parameters that are being learned by the model. The computations yields a probability distribution over all hidden state vectors of previous tokens. The model then computes an intermediate hidden state and cell state by making use of these two tapes and the score vector (s_i^t).

$$[\tilde{h}_t, \tilde{c}_t] = \sum_{i=1}^{t-1} s_i^t \cdot [h_i, c_i] \tag{10}$$

Once these intermediary vectors are computed, we pass them into our vanilla LSTM module as if they were the hidden (h_{t-1}) and cell state (c_{t-1}) coming in from the previous time-step. But instead, they are a linear combination of every previous hidden and cell state produced by the model, where the constants come

from the score function. Before moving on to the next time-step, these newly computed values are stored in the hidden and memory tapes, so that they can be used by future time-steps to compute their own score vectors.

$$[i_t, f_t, o_t, \tilde{c}_t] = [\sigma, \sigma, \sigma, tanh] \cdot W \cdot [\tilde{h}_t, x_t] \tag{11}$$

$$c_t = f_t \odot \tilde{c}_t + i_t \odot \hat{c}_t \tag{12}$$

$$h_t = o_t \odot tanh(c_t) \tag{13}$$

The attention mechanism is used to induce more implicit relations between tokens, which might not be caught by the vanilla LSTM weights. The attention module is treated as a submodule that is being optimized within the larger network, to which no direct supervision independent of the entire network is given.

3 Encoder-Decoder Architecture for LSTMNs

In order to solve tasks that require processing two distinct sequences such as machine translation or textual entailment, the architecture should be able to model both of those sequences. The proposed architecture is an encoder-decoder architecture, making use of two distinct LSTMNs with distinct weights. While the LSTMNs apply attention for intra-relation reasoning on individual sentences, as described in the previous section; the encoder-decoder network also has an additional attention module that is learning the inter-alignment between these two sequences.

The model first computes a score vector by making use of 3 different parameters that are being learned by the model: w, W^y, W^h. Y corresponds to our hidden tape at end of the sequence, and h_N is the last output vector after the premise and hypothesis were processed by the two LSTMNs respectively. Once we send the score vector through the softmax function, we get a probability distribution over all hidden states in our memory.

$$s = w^T tanh(W^y Y + W^h h_N) \tag{14}$$

$$\alpha = softmax(s) \tag{15}$$

We then use these attention weights (α) to compute a weighted representation (r) of the premise using:

$$r = Y\alpha^T \tag{16}$$

The model then computes the final sentence-pair representation from a linear combination of the attention weighted representation (r) of the premise and the last output vector (h_N). We make use of two additional parameters W^p and W^x which are used as projection matrices to give us our final hidden state,

which can then be used for purposes such as classification, via a separate neural network.

$$h^* = tanh(W^p r + W^x h_N) \tag{17}$$

4 Experiments

4.1 Sentiment Analysis

The first task I have experimented with is predicting the sentiment labels of sentences. I used the Stanford Sentiment Treebank (Socher et al., 2013a), where each sentence is assigned one of the five classes: very negative, negative, neutral, positive, very positive. I used 8,544 sentences for training, 1,101 sentences for validation and 2,210 sentences for testing. Next to Fine-Grained Sentiment Classification (5 classes) I have also experimented with Binary Sentiment Classification (2 classes). In order to do this, every sentence with a neutral label has been removed, very-negative and negative labels were mapped to negative, and very-positive and positive labels were mapped to positive. After the modification the dataset ended up with 6,920 sentences for training, 872 for validation and 1,821 for testing. Here are the results on both binary and fine-grained Sentiment Classification tasks, compared with the results of Cheng et al.:

Model Accuracy (%) on Sentiment Treebank		
Model	Fine-Grained	Binary
LSTM	**46.22**	**84.78**
LSTMN	**46.25**	**86.08**
LSTMN (Cheng et al.)	47.6	86.3

In order to determine the sentiment label of the sentence, the last hidden state is passed on to a 2-layer neural network classifier with ReLU as the activation function. The memory and hidden tape sizes were set to 72 for both tasks. For word embeddings, pre-trained 300-D Glove 42B vectors (Pennington et al., 2014) were used. The unknown words were initialized using Knet's xavier function and were learned through training. Mapping unknown words to "UNK" tokens resulted in lower accuracy.

I used Knet's Adam for optimization with the two momentum parameters set to 0.9 and 0.999 respectively. The initial learning rate was set to 2E-3. Regularization was applied with a constant of 1E-4. The mini-batch size was set to 5. A dropout rate of 0.5 was applied to the final neural network at each layer.

The results in the table show that for both Binary and Fine-grained sentiment classification, the LSTMN outperformed the vanilla LSTM. By trying out different custom reviews, I found out that the model showed very little attention to any phrase that came before "but", and mostly took into account what came after it. The model also fails on sentences involving "not" or "not the case that", showing that it fails to recognize negation. The difference between my

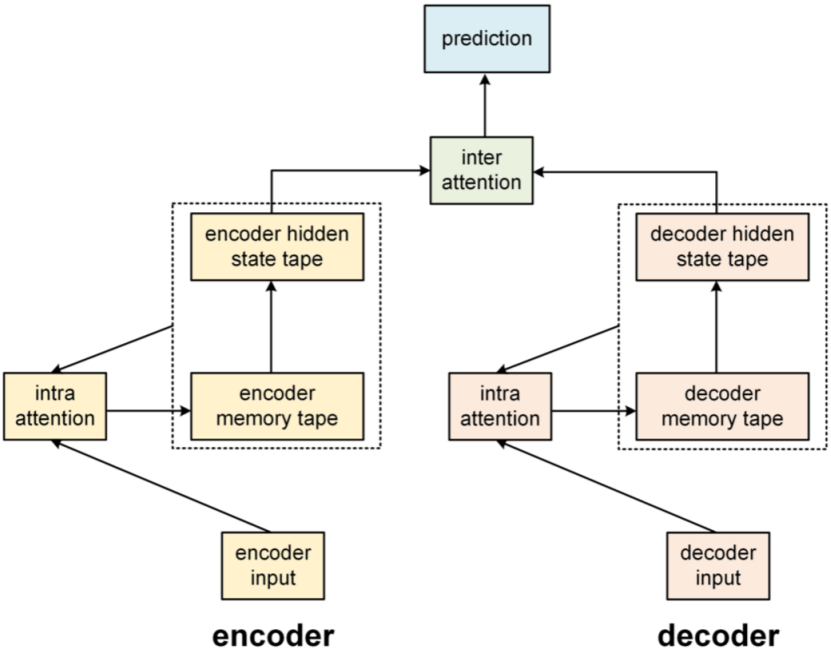

Figure 3: Encoder-Decoder Network With Inter-Attention

results and of Cheng et al., could be explained by our Glove embedding vector vocabulary size being different due to memory issues: mine being of size 42B, theirs of size 840B.

4.2 Natural Language Inference

The second experiment conducted using the LSTMN is recognizing textual entailment between a premise and a hypothesis. This is equivalent to classifying the relation between two input sentences as one of the 3 labels: "Entailment", "Neutral" and "Contradictory". For the experiment, I have used the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015), which contains premise hypothesis pairs and labels indicating the relation between the two. The dataset was split into 549,369 pairs for training, 9,842 for validation and 9,824 for testing. The vocabulary size of the entire dataset was 36,809, with an average sentence length of 22.

The architecture used for this task was described in section 3: Encoder-Decoder Architecture for LSTMNs. In order to determine the label of the relation, the last hidden state is passed on to a 2-layer neural network classifier with ReLU as the activation function. The embeddings were initialized with pre-trained 300-D Glove 42B embeddings (Pennington et al., 2014) if available,

and the rest were initialized with Knet's xavier function. The dropout rate was selected uniformly from [0.1,0.2,0.3,0.4] for all neural networks in the model. The model was trained with Adam, two momentum parameters set to 0.9 and 0.999 respectively. The initial learning rate was set to 1E-3. Mini-batch size was set to 32. The hidden and memory tape size was set to 72 for all LSTMNs used to experiment.

Model Accuracy (%) on SNLI	
Model	Accuracy
LSTM	**75.94**
LSTMN	**79.42**
LSTMN with inter-attention	**83.58**
LSTMN with inter-attention (Cheng et al.)	84.3

References

[1] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. arXiv preprint arXiv:1601.06733, 2016.

[2] Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. Neural computation, 9(8):1735–1780.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In Proceedings of the 2014 ICLR, Banff, Alberta.

[4] Karl Moritz Hermann, Tomas Ko- cisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teach- ing machines to read and comprehend. In Advances in Neural Information Processing Systems, pages 1684– 1692.

[5] Tim Rocktaschel, Edward Grefenstette, Karl Moritz Hermann, Tomas Kocisk y, and Phil Blunsom. 2016. Reasoning about entailment with neural attention. In Proceedings of ICLR

[6] 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 (EMNLP).

[7] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, An- drew Ng, and Christopher Potts. 2013a. Recursive deep models for semantic compositionality over a sen- timent treebank. In Proceedings of the 2013 EMNLP, pages 1631–1642, Seattle, Washington.

[8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.