Long Short-Term Memory-Networks for Machine Reading

Ege Ersü
Department of Computer Engineering
Koç University
Istanbul, Turkey
egeersu.github.io

Abstract

This is a technical report introducing the Long Short-Term Memory Network (LSTMN), a machine reading simulator that implements two additional mechanisms on top of the vanilla Long-Short Memory Network (LSTM). First, the memory cell is replaced by a memory network to store the contextual representation of each input token. Second, an attention mechanism is implemented to induce relations between tokens. Although the LSTMN can only process a single sequence, it is explained how one could integrate it within an encoder-decoder architecture. Experiments on Sentiment Analysis and Natural Language Inference show that the model outperforms the vanilla LSTM. The report also includes a comparison of experiment results with the original author.

GitHub Repository: github.com/egeersu/LSTMN

1 Word Representation

The model is dependent on a Vocabulary which contains a dictionary from words to their keys. To construct the Vocabulary, the entire training data is preprocessed such that each sentence is turned into a list of lowercase words and punctuation marks. Each unique word is then assigned to a unique integer which serves as that word's key. A word's key will later be used to retrieve the representation of that word.

The LSTMN uses pre-trained 300-D Glove 840B vectors [1] as representations. We assume that using a word's 300 dimensional vector representation will be functionally equivalent to that word's meaning. The reducibility of semantics to vector spaces is unfortunately beyond the scope of this paper. An embedding matrix is constructed by retrieving the representation of each unique word in our Vocabulary, and concatenating those vectors such that the key of a word will correspond to the column number where that word's

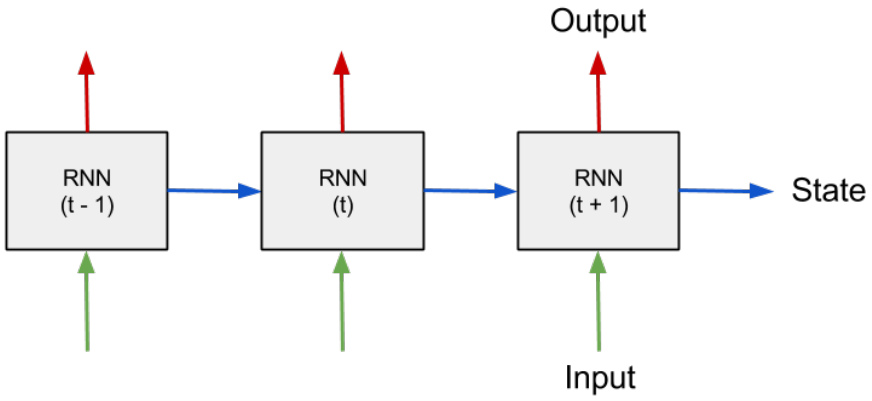

Figure 1: RNN

representation is stored. In Knet.jl, it is sufficient to turn the resulting matrix into a KnetArray so that the representations used in forward computation will be updated during backpropagation.

2 The Model

The report will first formally define LSTMN as a general-purpose reading simulator, and later propose LSTMN architectures that focus on specific language processing and interpreting tasks. The overall model can be designed to handle different numbers and types of inputs or outputs. The LSTMN serves a similar functional role with RNNs. When LSTMNs are used as building blocks in larger and more complex architectures, they will usually appear at similar locations with RNNs.

2.1 Recurrent Neural Network

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 vector representation of the current word it is on (x_t), 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. In the

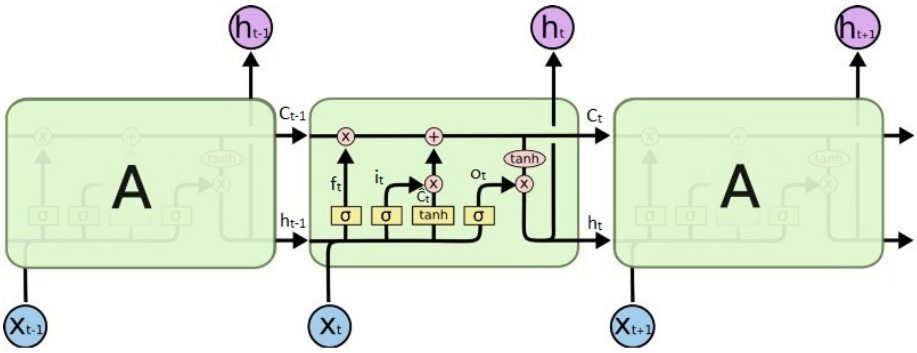

Figure 2: LSTM

end, we hope that the final hidden state of the RNN will turn out to be a useful representation of the meaning of the entire sentence. This final hidden vector can then be used as an input to other layers. For most classification tasks, a common approach is to feed the final hidden state into a linear layer followed by a softmax function, which computes a probability distribution over all target classes (p_t).

2.2 Long Short-Term Memory

A Long Short-Term Memory (LSTM) [2] works just like the RNN, but at each time-step (t), the LSTM computes two vectors: a hidden state (h_t) and a cell state (c_t). To compute the next cell state (c_t), the LSTM makes use of the representation of the word its currently processing (w_t), the previous hidden state (h_{t-1}) and the previous cell state (c_{t-1}). This computation uses three different gates: i, f and o.

It is theorized that if the gates properly learn what they are supposed to do, the cell state (c_t) will either store information or forget information as it is needed. For example, it could be the case that the incoming cell somehow represents the gender of the subject. If we are using the LSTM as a language model, once it generates the pronoun of the subject it should compute the next cell state such that it no longer prioritizes that information. It is usually imagined as a conveyor belt running through the entire LSTM, once we unroll it in time [3]. To compute the next hidden state (h_t), the LSTM makes use of the cell state it just computed (c_t) and the output gate (o_t).

The formal definition of the LSTM is given below:

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

$$f_t = \sigma([h_{t-1}, x_t] \odot W_f + b_f) \tag{2}$$

$$o_t = \sigma([h_{t-1}, x_t] \odot W_o + b_o) \tag{3}$$

$$\hat{c}_t = \tanh\left([h_{t-1}, x_t] \odot W_c + b_c\right) \tag{4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t \tag{5}$$

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

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

2.3 LSTMN

In order to motivate the LSTMN [4], I will first explain two crucial assumptions that are being made by the LSTM. The LSTMN will seek to ameliorate the problems that arise when those assumptions are rejected. A fundamental property of the LSTM is that the next state is always computed from the current state. More formally: given the current state h_t, the next state h_{t+1} is conditionally independent of states h_1 ... h_{t-1} and tokens x_1 ... x_t. In the context of language modeling, this is equivalent to saying that the current state (h_t) alone summarizes the meaning of the sentence up to the current word (x_t). This assumption will surely fail as sentences get longer since the meaning of the entire sentence must be compressed into a single dense vector. The proposed solution is to implement a dynamic memory network into the LSTM, which will store the hidden states and cell states produced at each time-step. It will start as empty and will keep growing as time passes and will stop once the memory span (a hyper-parameter of the model) is reached. This will enable the model to access hidden and cell states that were produced at any given time-step, increasing the number of states that could possibly influence the final meaning representation of the sentence.

The second assumption concerns how the input is processed, which in turn determines how the representations are constructed. Since the LSTM processes the input on a token-by-token basis in sequential order from left to right, the model cannot take into account the words that are to the left or to the right; unless the required information is already modeled by the current state (h_t). By implementing an attention mechanism over the memory network, we enable the model to read previous hidden states and use them as it computes the next hidden state. The model will learn to

I will now give the formal definition of LSTMN. The model still makes use of a LSTM at its core, which has its own set of weights as specified above. The model also maintains two sets of vectors: the 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)

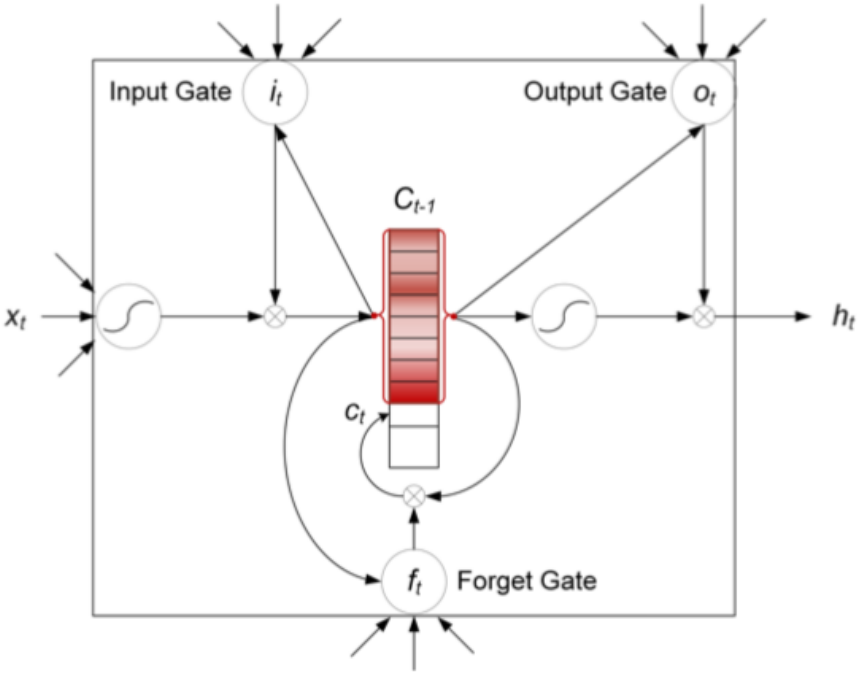

Figure 3: LSTMN Cell

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 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 of the LSTMN. 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, they are passed into the vanilla LSTM module as if they were the previous hidden state (h_{t-1}) and the previous 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 are specified by 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 sub-module that is being optimized within the larger network, which is not directly supervised.

3 Encoder-Decoder Architecture for LSTMNs

3.1 Natural Language Inference

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 [5,6,7]. 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 the hidden tape at end of the sequence, and h_N is the last output vector after the premise and hypothesis are processed by the two LSTMNs respectively. Once the score vector is sent through the softmax function, it computes a probability distribution over all hidden states in our memory.

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

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

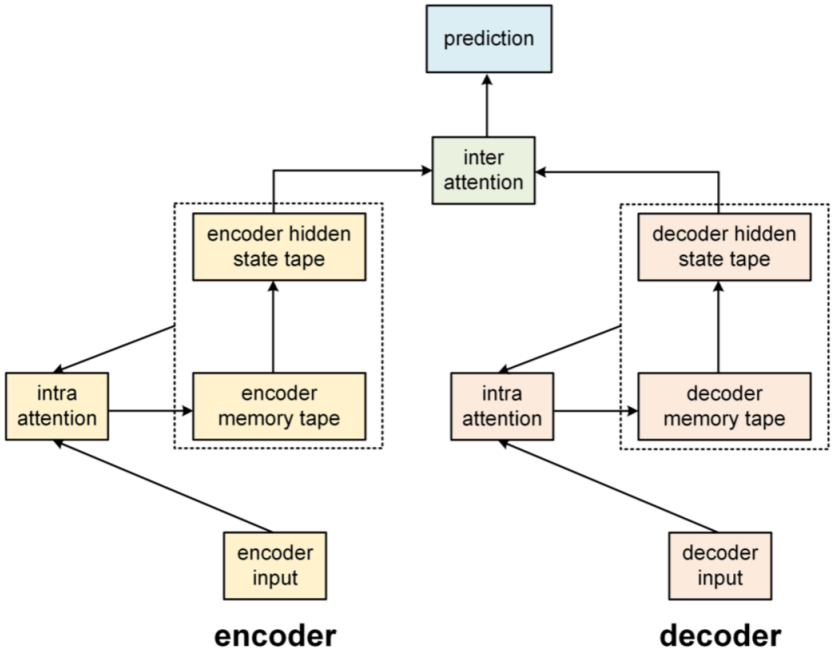

Figure 4: Encoder-Decoder Network With Inter-Attention

The attention weights (α) are then used to 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). Two additional parameters W^p and W^x are used as projection matrices to compute the 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

Sentiment Analysis is the task of assigning a sentiment value to each sentence in the dataset. The Stanford Sentiment Treebank was used to train and test

the LSTMN [8]. The first experiment was Binary Sentiment Classification, assigning one of two sentiment classes: ["positive", "negative"]. The second experiment was Fine-Grained Sentiment Classification, assigning one of five sentiment classes: ["very negative", "negative", "neutral", "positive", "very positive"]. For Fine-Grained Sentiment Classification, the dataset was partitioned such that 8,544 sentences were used for training, 1,101 sentences were used for validation and 2,210 sentences were used for testing the model. For Binary Sentiment Classification, every sentence with a neutral label was removed from the dataset, "very negative" labels were mapped to "negative", and "very positive" labels were mapped to "positive". After this modification, the dataset ended up with 6,920 sentences reserved 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 original 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.

Knet.jl's Adam optimizer was used with 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 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 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.

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", "Contradictory"]. The Stanford Natural Language

Inference (SNLI) dataset [9] was used for training and testing. A single data point consists of a premise hypothesis pair and label 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) and the unknown words 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 sizes were set to 72 for all LSTMNs used in the 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] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

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

[3] C. Olah, "Understanding LSTM Networks," Understanding LSTM Networks – colah's blog, 27-Aug-2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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).