International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Modified Long Short-Term Memory Recurrent Neural Network Architectures

Manish Rana[1], Shubham Mishra[2]

[1]Professor, Department of Computer Science, TCET, Mumbai University, India

[2]Student, Department of Computer Science, TCET, Mumbai University, India

Abstract: ***Long Short-Term Memory (LSTM) is a specific recurrent neural network (RNN) architecture that was designed to model temporal sequences and their long-range dependencies more accurately than conventional RNNs. In this paper, we explore LSTM RNN architectures and made some changes for its better performance. LSTM RNNs are more effective than DNNs. Here, we have changed the gates calculation and also have removed some unnecessary features of standard LSTM architecture. This architecture makes more effective use of model parameters than the others considered, converges quickly, and outperforms a deep feed forward neural network having an order of magnitude more parameters.***

Keywords: Long Short-Term Memory, LSTM, recurrent neural network, RNN.

1. Introduction

The Deep Neural Network (DNN) is an extremely expressive model that can learn highly complex vector-to-vector mappings. The Recurrent Neural Network (RNN) is a DNN that is adapted to sequence data, and as a result the RNN is also extremely expressive. RNNs maintain a vector of activations for each time step, which makes the RNN extremely deep. Their depth, in turn, makes them difficult to train due to the exploding and the vanishing gradient problems [3] [13] [14].

There have been a number of attempts to address the difficulty of training RNNs. Vanishing gradients were successfully addressed by Hochreiter & Schmidhuber (1997), who developed the Long Short-Term Memory (LSTM) architecture, which is resistant to the vanishing gradient problem. The LSTM turned out to be easy to use, causing it to become the standard way of dealing with the vanishing gradient problem. Other attempts to overcome the vanishing gradient problem include the use of the powerful second order optimization algorithms [18] [19] and regularization of the RNN's weights that ensures that the gradient does not vanish [23], giving up on learning the recurrent weights altogether [15] [16] and a very careful initialization of RNN's parameters [25] [26] Unlike the vanishing gradient problem, the exploding gradient problem turned out to be relatively easy to address by simply enforcing a hard constraint over the norm of the gradient [20] [23].

A criticism of the LSTM architecture is that it is ad-hoc and that it has a substantial number of components whose purpose is not immediately apparent. As a result, it is also not clear that the LSTM is an optimal architecture, and it is possible that better architectures exist.

Motivated by this criticism, we attempted to determine whether the LSTM architecture is optimal by means of an extensive evolutionary architecture search. We found specific architectures similar to the Gated Recurrent Unit (GRU) [6] That outperformed the LSTM and the GRU by on most tasks, although an LSTM variant achieved the best results whenever dropout was used. In addition, by adding a bias of 1 to the LSTM's forgetting gate. We can close the gap between the LSTM and the better architectures.

2. Long Short-Term Memory

In this section we will briefly explain LSTM architecture. The figure 1 is the traditional LSTM architecture. Standard RNNs suffer from both exploding and vanishing gradients [3] [13]. Both problems are caused by the RNN's iterative nature, whose gradient is essentially equal to the recurrent weight matrix raised to a high power. These iterated matrix powers cause the gradient to grow or to shrink at a rate that is exponential in the number of time steps. The exploding gradients problem is relatively easy to handle by simply shrinking gradients whose norms exceed a threshold, a technique known as gradient clipping [20] [23]. While learning would suffer if the gradient is reduced by a massive factor too frequently, gradient clipping is extremely effective whenever the gradient has a small norm the majority of the time.

The full LSTM's definition includes circuitry for computing ΔSt and circuitry for decoding information from ΔSt. Unfortunately; different practitioners use slightly different LSTM variants. In this work, we use the LSTM architecture that is precisely specified below. It is similar to the architecture of [10] but without peep-hole connections:

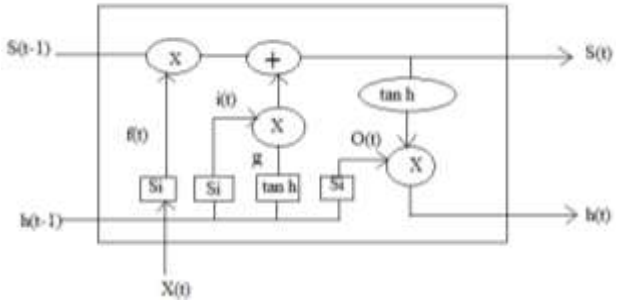

Figure 1: LSTM architecture

In the above LSTM architecture the symbols are defined as,

S (t-1): Previous cell status
h (t-1): Previous cell hidden state
f (t): Forget gate
i (t): Information gate
Si: Sigmoid function
X (t): Current input
X: Vector multiplication, in this paper it is represented by * notation.
O (t): Output
S (t): Current cell status
h (t): Current cell hidden state

In this LSTM architecture the cell status store cell status. Based on current input LSTM takes decision that how much past information is to delete. This action is performed with the help of Forget gate. Once past information is deleted then new information is added to the cell using Information gate. The equations are,

- i (t) = Si ($W_{xi} X_t + W_{hi} h_{t-1} + b_i$)
- f (t) = Si ($W_{xf} X_t + W_{hf} h_{t-1} + b_f$)
- g = tanh ($W_{xg} X_t + W_{hg} h_{t-1} + b_g$)
- (t) = Si ($W_{xo} X_t + W_{ho} h_{t-1} + b_o$)
- S (t) = S (t-1) * f (t) + i (t) * g
- h (t) = tanh (S(t)) * O(t)

'W' is the weight vector initialized randomly. 'b' is the bias value also initialized randomly. All the weight vectors are updated after each iteration.

LSTM architecture learn more and more by training and work good for both long term as well as for short term memory.

3. Methodology

To work on this architecture we are creating input data in the program. The created dataset will be in a range defined by the standard for LSTM architecture. For better performance of the architecture we made two changes in the standard architecture which helps the LSTM to work even more efficiently.

Firstly, In standard architecture amount of past information to delete and new information to add was decided separately hence was missed out some use full information. In the new architecture the amount of information to delete is calculated based on the amount of new information required to add. Hence in the new Architecture initially information gate will calculate new information to add and based on information gate output forget gate will calculate amount of information to forget.

Secondly, Due to the use of 'tanh' function while calculating h(t) some use full information was lost hence we decided to remove this 'tanh' function. After removing 'tanh' from the h(t) equation the architecture become more accurate and error rate has been reduced. The new modified architecture is,

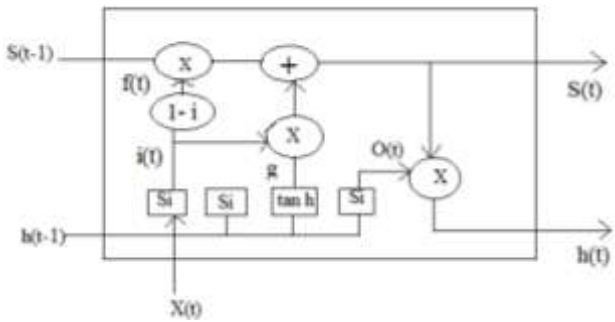

Figure 2: Modified LSTM architecture

After modification new equations are highlighted as,

- i (t) = Si ($W_{xi} X_t + W_{hi} h_{t-1} + b_i$)
- **f (t) = (1 – i)**
- g = tanh ($W_{xg} X_t + W_{hg} h_{t-1} + b_g$)
- (t) = Si ($W_{xo} X_t + W_{ho} h_{t-1} + b_o$)
- S (t) = S (t-1) * f (t) + i (t) * g
- **h (t) = S(t) * O(t)**

Modified algorithm/architecture will not only work more accurately but also will reduce total execution time.

4. Result and Discussion

We have implemented LSTM in python language and executed for multiple changes but this modified version of architecture gives us maximum accuracy. We also worked on the multiple modified architectures by changing normalization functions, changing equations and reconnecting different gates with different gates but all the architecture performed poorly compared to the standard architecture.

Standard LSTM is executed for a set of data for 0 to 99 iterations and the result is shown as the screen shot of the output. This screen short only was showing last part of the output with final loss at the end of the output.

```
cur iter: 98
y_pred[0] : -0.500087106205
y_pred[1] : 0.200498204588
y_pred[2] : 0.0994746051746
y_pred[3] : -0.499579491524
loss: 7.08662403825e-07
cur iter: 99
y_pred[0] : -0.500096072911
y_pred[1] : 0.200463300125
y_pred[2] : 0.0995059890975
y_pred[3] : -0.499595630139
loss: 6.31438766408e-07
```

Figure 3: Standard LSTM Output

In figure 3 after last iteration the final loss is $6.31438e^{-07}$. This loss is less than all the other loss of diffrent architecture except figure 2 modified LSTM architecture's loss.

Now, modified LSTM when execucate for the same data input the result is shown as the screen shot of the output. This screen short only was showing last part of the output.

```
cur iter: 98
y_pred[0] : -0.499712732281
y_pred[1] : 0.199925730777
y_pred[2] : 0.100122240059
y_pred[3] : -0.500299187737
loss: 1.92494594059e-07
cur iter: 99
y_pred[0] : -0.499731288362
y_pred[1] : 0.19992779384
y_pred[2] : 0.100116124116
y_pred[3] : -0.500280730381
loss: 1.69714030648e-07
```

Figure 4: Modified LSTM Output

In figure 4 after last iteration the final loss is $1.6971403e^{-07}$. Modified LSTM's final loss is less than that of the standard LSTM architecture final loss.

Table 1: Loss Difference

	Standard LSTM	Modified LSTM
Loss	$6.31438e^{-07}$	$1.6971403e^{-07}$

5. Conclusion

Standard LSTM architecture works better then the RNN by handling the vanishing gradient problem. The LSTM architecture is not perfect. To make it more accurate we experimented on it by changing its architecture and hence come up with a new LSTM architecture which works better than that of the standard architecture. Thus, modifying standard LSTM architecture by changing the forget gate structure and removing unnecessary normalization function improved the LSTM performance.

References

[1] Amit, Daniel J. Modeling brain function: The world of attractor neural networks. Cambridge University Press, 1992.
[2] Bayer, Justin, Wierstra, Daan, Togelius, Julian, and Schmidhuber, J¨urgen. Evolving memory cell structures for sequence learning. In Artificial Neural Networks–ICANN 2009, pp. 755–764. Springer, 2009.
[3] Bengio, Yoshua, Simard, Patrice, and Frasconi, Paolo. Learning long-term dependencies with gradient descent is difficult. Neural Networks, IEEE Transactions on, 5 (2):157–166, 1994.
[4] Bergstra, James, Breuleux, Olivier, Bastien, Fr´ed´eric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In Proceedings of the Python for Scientific Computing Conference (SciPy), June 2010.
[5] Boulanger-Lewandowski, Nicolas, Bengio, Yoshua, and Vincent, Pascal. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and the transcription. The arXiv preprint arXiv:1206.6392, 2012.
[6] Cho, Kyunghyun, van Merrienboer, Bart, Gulcehre, Caglar, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using rnn encoderdecoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.
[7] Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.
[8] Collobert, Ronan, Kavukcuoglu, Koray, and Farabet, Cl´ement. Torch7: A matlab-like environment for machine learning. In BigLearn, NIPS Workshop, number EPFL-CONF-192376, 2011.
[9] Gers, Felix A, Schmidhuber, J¨urgen, and Cummins, Fred. Learning to forget: Continual prediction with lstm. Neural computation, 12(10):2451–2471, 2000.
[10] Graves, Alex. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850, 2013.
[11] Greff, Klaus, Srivastava, Rupesh Kumar, Koutn´ık, Jan, Steunebrink, Bas R, and Schmidhuber, J¨urgen. Lstm: A search space odyssey. arXiv preprint arXiv:1503.04069, 2015.
[12] Hinton, Geoffrey E and Shallice, Tim. Lesioning an attractor network: investigations of acquired dyslexia. Psychological review, 98(1):74, 1991.
[13] Hochreiter, Sepp. Untersuchungen zu dynamischen neuronalen netzen. Master's thesis, Institut fur Informatik, Technische Universitat, Munchen, 1991.
[14] Hochreiter, Sepp and Schmidhuber, J¨urgen. Long shortterm memory. Neural computation, 9(8):1735–1780, 1997.
[15] Jaeger, Herbert. The echo state approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, 148:34, 2001.
[16] Jaeger, Herbert and Haas, Harald. The Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science, 304(5667):78–80, 2004.
[17] Marcus, Mitchell P, Marcinkiewicz, Mary Ann, and Santorini, Beatrice. Building a large annotated corpus of english: The penn treebank. Computational linguistics, 19(2):313–330, 1993.
[18] Martens, James. Deep learning via the hessian free optimization. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp. 735–742, 2010.
[19] Martens, James Sutskever and Ilya. Learning recurrent neural networks with hessian-free optimization. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 1033–1040, 2011.
[20] Mikolov, Tom´a˘s. Statistical Language Models based on Neural Networks. PhD thesis, PhD thesis, Brno University of Technology, 2012. -, 2012.
[21] Mikolov, Tomas, Karafi´at, Martin, Burget, Lukas, Cernock` y, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010, pp. 1045–1048, 2010.
[22] Mikolov, Tomas, Joulin, Armand, Chopra, Sumit, Mathieu, Michael, and Ranzato, Marc'Aurelio. Learning longer memory in recurrent neural networks. arXiv preprint arXiv:1412.7753, 2014.

[23] Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. arXiv preprint arXiv:1211.5063, 2012.
[24] Plaut, David C. Semantic and associative priming in a distributed attractor network. In Proceedings of the 17th annual conference of the cognitive science society, volume 17, pp. 37–42. Pittsburgh, PA, 1995.
[25] Sutskever, Ilya, Martens, James, Dahl, George, and Hinton, Geoffrey. On the importance of initialization and momentum in deep learning. In Proceedings of the 30th International Conference on Machine Learning (ICML- 13), pp. 1139–1147, 2013.
[26] Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc VV. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems, pp. 3104–3112, 2014.
[27] Zaremba, Wojciech and Sutskever, Ilya. Learning to execute. arXiv preprint arXiv:1410.4615, 2014.
[28] Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent neural network regularization. arXiv preprint arXiv:1409.2329, 2014.