Recurrent Residual Network

2016/09/23

Abstract

This work briefly introduces the recurrent residual network which is a combination of the residual network and the long short term memory network(LSTM). The residual network is featured by residual blocks and the LSTM as a variant of RNN, is featured by the recurrent structure and long short term-memory cells. We modify the LSTM by adding residual links between non-adjacent layers. Experiments on several tasks shows the effectiveness of combining two models together.

1 Introduction

Dealing with inputs of variant lengths is a challenge for neural network models. The recurrent neural network is develop as a ad-hoc technology of processing such inputs. It essentially cuts inputs into short chunks of fixed length and turns the problem of dealing with variant lengths to a problem of dealing with fixed length inputs. The recurrent network is widely employed since it is proposed.

Researchers found there are some drawbacks with RNN. One of them is the gradient vanishing/exploding problem. In the back-propagation of a recurrent neural network, the gradient is multiplied a large number of times (as many as the number of time steps) by the weight matrix which connects neighbouring layers in the model. This means that, the magnitude of weights in the transition matrix can have a strong impact on the learning process. If the weights in this matrix are small, or more precisely, if the leading eigenvalue of the weight matrix is smaller than 1.0, it can lead to the gradients vanishing problem, which means the gradient signal gets so small that learning either becomes very slow or just impossible. It can also make more difficult the task of learning long-term dependencies in the data. On the other hand, if the weights in this matrix are large, or if the leading eigenvalue of the weight matrix is larger than 1.0, it can lead to a situation where the gradient signal is so large that it can cause learning to diverge. This is often referred to as exploding gradients.

To address this problem, researchers introduce the new long short-term memory cells in neural network models [Hochreiter and Schmidhuber(1997)]. A memory cell is composed of four main elements: an input gate, a neuron with a self-recurrent connection, a forget gate and an output gate. The input gate controls the impact of the input value on the state of the memory cell and the output gate controls the impact of the state of the memory cell on the output. The self-recurrent connection controls the evolution of the state of the memory cell and the forget gate decides whether to keep or reset the histories of the memory cell's states. These elements serve different purposes and work together to make LSTM cells much more powerful than previous neural cells. LSTM is widely used in various tasks in NLP and other machine learning fields [Schmidhuber et al.(2002)Schmidhuber, Gers, and Eck, Sundermeyer et al.(2012)Sundermeyer, Schlüter, and Ney, Sutskever et al.(2014)Sutskever, Vinyals, and Le, Dyer et al.(2015)Dyer, Ballesteros, Ling, Matthews, and Smith].

And there is another way of easing the problem of exploding gradients as is shown by the deep residual network [He et al.(2016)He, Zhang, Ren, and Sun] which is regarded as an improvement of the recurrent neural networks (RNN).

The deep residual network has two significant characters compared with RNN. The first one is the residual learning. In a neural network models, normally the data is passed from one layer to the adjacent layer. In the residual network, an additional layer is used to connect layers that are far away. During the back propagation, errors can be passed from a higher layer to lower layer

Figure 1: A LSTM memory cell

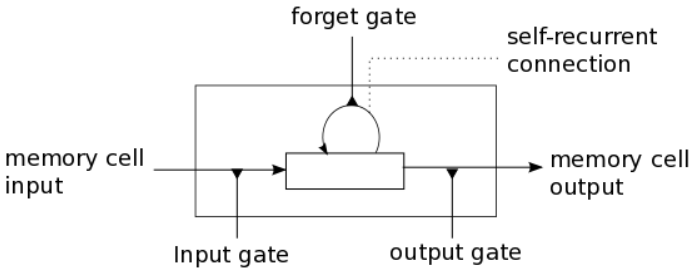


Figure 2: Operations in a LSTM Node

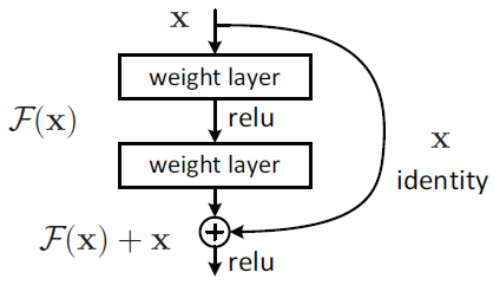
$$\begin{cases} i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ \widetilde{C_t} = tanh(W_c x_t + U_c h_{t-1} + b_c) \\ f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ C_t = i_t \widetilde{C_t} + f_t C_{t-1} \\ out_t = \sigma(W_o x_t + U_o h_{t-1} + V_o C_t + b_o) \\ h_t = out_t * tanh(C_t) \end{cases} \tag{1}$$

Here W_*, U_*, V_* are weight matrices. b_* are bias vectors. x_t is the input and $output_t$ is the output at time t. f_*, h_*, C_* are some internal states.

directly. It helps ease the vanishing gradient or exploding gradient problem, which is widely believed to be a reason of its superb performances [He et al.(2016)He, Zhang, Ren, and Sun]. Fig 3 shows a building block of deep residual network. The second one is the depth of such models. A typical residual network has hundreds of layers, which is much deeper than most existing models. Thus the training of such a model becomes a challenge. Besides, given the number of the layers, the number of parameters also exceeds most networks. When trained on a small dataset, a deep residual network may suffer from over-fitting [He et al.(2016)He, Zhang, Ren, and Sun].

The residual neural network has been proved useful in capturing information from images for classification. A number of variants have been introduced for a series of tasks. We do not go in to the details due to the space limitation. In the following section, we will show how to employ the residual network in the targeted task.

Figure 3: A Residual Learning Block [He et al.(2016)He, Zhang, Ren, and Sun]. Compared with a traditional MLP, The change is that some layers that used to be non-adjacent are connected.



2 Recurrent Residual Network

A recurrent residual network is a combination two of them together. To be specific, we add skip con-

nections in the LSTM model. In fact, the residual connections can be combined with RNN/GRU or some other models. Here we chose LSTM for it is powerful than others in a series of tasks.

The proposed model is shown in Fig 4. The right part shows the structure of a recurrent residual node which is constructed using a LSTM cell. As stated, other neural nodes can also be employed such as the simple perceptron or GRU. The left is the same with the recurrent model except that here each node represents a residual lstm node.

We do not elaborate the details as it is simple and clear enough.

We compare this model with the original LSTM. We firstly implemented the LSTM and then add the skip connections and linear transformations. We want to exclude all the un necessary impacts and make sure the results reflect the effectiveness of the modifications.

3 Experiment

We compare the performances using a char-level rnn [1]. A char-level rnn is similar to an rnn except that it is on the character level. One may doubt the meaning of such a model. We will move to morphemes and other tasks such as classification or sequence-to-sequence tasks later. Here we focus on the performance comparison. We use the program implemented using chainer [2].

Results are shown in the following tables.

The rrn contains 7 layers, (50,80), (80,80), (80,160), (160,320), (320,800), (800,50), (50,50), and in the lstm, the size of a node is set to be 50.

The results are reported on a GPU server with the following specs:
CPU: Intel(R) Xeon(R) CPU E5-2695 v3 @ 2.30GHz
GPU: Tesla K80 * 4 (12G) "reported by nvidia-smi"
Memory: 337G (Reported by "free")
Chainer: 1.9.1

As we can see, the time cost of the proposed model is about 5 times that of LSTM and the train loss is reduced by about 4%.

$$4\% = \frac{Avg\_Loss(LSTM) - Avg\_Loss(RRN)}{Avg\_Loss(LSTM)} \quad (2)$$

[1] https://github.com/karpathy/char-rnn

[2] https://github.com/yusuketomoto/chainer-char-rnn

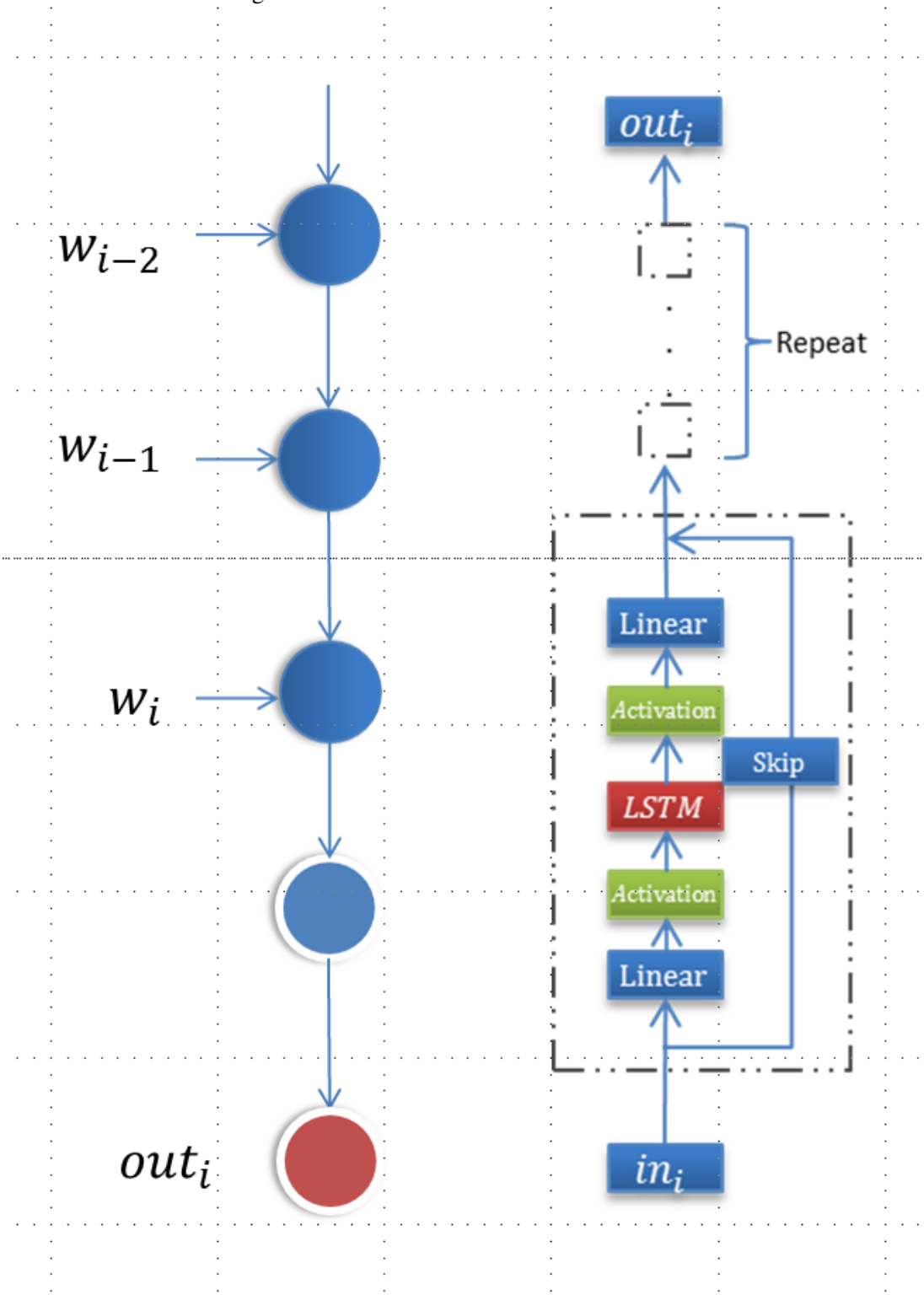
Table 1: Results of RRN

Iteration	Results	Time
423/22307	train_loss = 2.4386780262	time = 1.48
424/22307	train_loss = 2.41346979141	time = 1.48
425/22307	train_loss = 2.36787247658	time = 1.46
426/22307	train_loss = 2.38671374321	time = 1.48
427/22307	train_loss = 2.35455560684	time = 1.49
428/22307	train_loss = 2.38408517838	time = 1.46
429/22307	train_loss = 2.38966989517	time = 1.48
430/22307	train_loss = 2.39387822151	time = 1.48
431/22307	train_loss = 2.38929510117	time = 1.46
432/22307	train_loss = 2.4487323761	time = 1.49
433/22307	train_loss = 2.40984630585	time = 1.49
434/22307	train_loss = 2.3900475502	time = 1.49
435/22307	train_loss = 2.4499399662	time = 1.47
436/22307	train_loss = 2.35769629478	time = 1.49
437/22307	train_loss = 2.30001759529	time = 1.49
438/22307	train_loss = 2.38338518143	time = 1.46
439/22307	train_loss = 2.36612725258	time = 1.49
440/22307	train_loss = 2.30233669281	time = 1.49
441/22307	train_loss = 2.36858820915	time = 1.46
442/22307	train_loss = 2.43151712418	time = 1.48
443/22307	train_loss = 2.3438103199	time = 1.49
444/22307	train_loss = 2.46831035614	time = 1.46
445/22307	train_loss = 2.39090275764	time = 1.48
446/22307	train_loss = 2.40528726578	time = 1.48

Table 2: Results of LSTM

Iteration	Results	Time
423/22307	train_loss = 2.50547766685	time = 0.30
424/22307	train_loss = 2.46356320381	time = 0.30
425/22307	train_loss = 2.46316218376	time = 0.30
426/22307	train_loss = 2.49023270607	time = 0.30
427/22307	train_loss = 2.46731638908	time = 0.30
428/22307	train_loss = 2.47279858589	time = 0.31
429/22307	train_loss = 2.4947271347	time = 0.30
430/22307	train_loss = 2.50045156479	time = 0.31
431/22307	train_loss = 2.44542241096	time = 0.32
432/22307	train_loss = 2.50137901306	time = 0.30
433/22307	train_loss = 2.48271608353	time = 0.30
434/22307	train_loss = 2.48008418083	time = 0.30
435/22307	train_loss = 2.51200246811	time = 0.31
436/22307	train_loss = 2.48871541023	time = 0.30
437/22307	train_loss = 2.40931391716	time = 0.30
438/22307	train_loss = 2.45529794693	time = 0.30
439/22307	train_loss = 2.47192335129	time = 0.30
440/22307	train_loss = 2.4405977726	time = 0.31
441/22307	train_loss = 2.47446131706	time = 0.30
442/22307	train_loss = 2.49214029312	time = 0.31
443/22307	train_loss = 2.46333169937	time = 0.30
444/22307	train_loss = 2.5029528141	time = 0.30
445/22307	train_loss = 2.46809458733	time = 0.30
446/22307	train_loss = 2.42945790291	time = 0.30

Figure 4: A Recurrent Residual Network Node



4 Conclusion

The RRN is a good model if we were willing to afford the time cost.

5 Acknowledgement

The char-rnn is from *http://karpathy.github.io/2015/05/21/rnn-effectiveness/* and the LSTM implementation comes from *https://github.com/yusuketomoto/chainer-char-rnn*. We thanks these authors for their contributions.

References

[Dyer et al.(2015)Dyer, Ballesteros, Ling, Matthews, and Smith] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075* .

[He et al.(2016)He, Zhang, Ren, and Sun] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 770–778.

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

[Schmidhuber et al.(2002)Schmidhuber, Gers, and Eck] Jurgen Schmidhuber, Felix A Gers, and Douglas Eck. 2002. Learning nonregular languages: a comparison of simple recurrent networks and lstm. *Neural Computation* 14(9):2039–2041.

[Sundermeyer et al.(2012)Sundermeyer, Schlüter, and Ney] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. Lstm neural networks for language modeling. In *INTERSPEECH*. pages 194–197.

[Sutskever et al.(2014)Sutskever, Vinyals, and Le] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. pages 3104–3112.