Neural Networks Compression for Language Modeling

Artem M. Grachev[1,2](✉), Dmitry I. Ignatov[2], and Andrey V. Savchenko[3]

[1] Samsung R&D Institute Rus, Moscow, Russia
grachev.art@gmail.com
[2] National Research University Higher School of Economics, Moscow, Russia
[3] Laboratory of Algorithms and Technologies for Network Analysis, National Research University Higher School of Economics, Nizhny Novgorod, Russia

Abstract. In this paper, we consider several compression techniques for the language modeling problem based on recurrent neural networks (RNNs). It is known that conventional RNNs, e.g., LSTM-based networks in language modeling, are characterized with either high space complexity or substantial inference time. This problem is especially crucial for mobile applications, in which the constant interaction with the remote server is inappropriate. By using the Penn Treebank (PTB) dataset we compare pruning, quantization, low-rank factorization, tensor train decomposition for LSTM networks in terms of model size and suitability for fast inference.

Keywords: LSTM · RNN · Language modeling · Low-rank factorization · Pruning · Quantization

1 Introduction

Neural network models can require a lot of space on disk and in memory. They can also need a substantial amount of time for inference. This is especially important for models that we put on devices like mobile phones. There are several approaches to solve these problems. Some of them are based on sparse computations. They also include pruning or more advanced methods. In general, such approaches are able to provide a large reduction in the size of a trained network, when the model is stored on a disk. However, there are some problems when we use such models for inference. They are caused by high computation time of sparse computing. Another branch of methods uses different matrix-based approaches in neural networks. Thus, there are methods based on the usage of Toeplitz-like structured matrices in [1] or different matrix decomposition techniques: low-rank decomposition [1], TT-decomposition (Tensor Train decomposition) [2,3]. Also [4] proposes a new type of RNN, called uRNN (Unitary Evolution Recurrent Neural Networks).

In this paper, we analyze some of the aforementioned approaches. The material is organized as follows. In Sect. 2, we give an overview of language modeling methods and then focus on respective neural networks approaches. Next we describe different types of compression. In Sect. 3.1, we consider the simplest methods for neural networks compression like pruning or quantization. In Sect. 3.2, we consider approaches to compression of neural networks based on different matrix factorization methods. Section 3.3 deals with TT-decomposition. Section 4 describes our results and some implementation details. Finally, in Sect. 5, we summarize the results of our work.

2 Language Modeling with Neural Networks

Consider the language modeling problem. We need to compute the probability of a sentence or sequence of words $(w_1, \dots, w_T)$ in a language L.

$$\begin{aligned} \mathsf{P}(w_1, \dots, w_T) &= \mathsf{P}(w_1, \dots, w_{T-1})\mathsf{P}(w_T | w_1, \dots, w_{T-1}) \\ &= \prod_{t=1}^{T} \mathsf{P}(w_t | w_1, \dots, w_{t-1}) \end{aligned} \tag{1}$$

The use of such a model directly would require calculation $\mathsf{P}(w_t|w_1, \dots, w_{t-1})$ and in general it is too difficult due to a lot of computation steps. That is why a common approach features computations with a fixed value of N and approximate (1) with $\mathsf{P}(w_t|w_{t-N}, \dots, w_{t-1})$. This leads us to the widely known N-gram models [5,6]. It was very popular approach until the middle of the 2000s. A new milestone in language modeling had become the use of recurrent neural networks [7]. A lot of work in this area was done by Thomas Mikolov [8].

Consider a recurrent neural network, RNN, where N is the number of timesteps, L is the number of recurrent layers, x_ℓ^{t-1} is the input of the layer ℓ at the moment t. Here $t \in \{1, \dots, N\}$, $\ell \in \{1, \dots, L\}$, and x_0^t is the embedding vector. We can describe each layer as follows:

$$z_\ell^t = W_\ell x_{\ell-1}^t + V_\ell x_\ell^{t-1} + b_l \tag{2}$$

$$x_\ell^t = \sigma(z_\ell^t), \tag{3}$$

where W_ℓ and V_ℓ are matrices of weights and σ is an activation function. The output of the network is given by

$$y^t = \text{softmax}\left[W_{L+1} x_L^t + b_{L+1}\right]. \tag{4}$$

Then, we define

$$\mathsf{P}(w_t|w_{t-N}, \dots, w_{t-1}) = y^t. \tag{5}$$

While N-gram models even with not very big N require a lot of space due to the combinatorial explosion, neural networks can learn some representations of words and their sequences without memorizing directly all options.

Now the mainly used variations of RNN are designed to solve the problem of decaying gradients [9]. The most popular variation is Long Short-Term Memory

(LSTM) [7] and Gated Recurrent Unit (GRU) [10]. Let us describe one layer of LSTM:

$$i_\ell^t = \sigma\left[W_l^i x_{l-1}^t + V_l^i x_l^{t-1} + b_l^i\right] \qquad \text{input gate} \qquad (6)$$

$$f_\ell^t = \sigma\left[W_l^f x_{l-1}^t + V_l^f x_l^{t-1} + b_l^f\right] \qquad \text{forget gate} \qquad (7)$$

$$c_\ell^t = f_l^t \cdot c_l^{t-1} + i_l^t \tanh\left[W_l^c x_{l-1}^t + U_l^c x_l^{t-1} + b_l^c\right] \qquad \text{cell state} \qquad (8)$$

$$o_\ell^t = \sigma\left[W_l^o x_{\ell-1}^t + V_l^o x_l^{t-1} + b_l^o\right] \qquad \text{output gate} \qquad (9)$$

$$x_\ell^t = o_\ell^t \cdot \tanh[c_l^t], \qquad (10)$$

where again $t \in \{1, \ldots, N\}$, $\ell \in \{1, \ldots, L\}$, c_ℓ^t is the memory vector at the layer ℓ and time step t. The output of the network is given the same formula 4 as above.

Approaches to the language modeling problem based on neural networks are efficient and widely adopted, but still require a lot of space. In each LSTM layer of size $k \times k$ we have 8 matrices of size $k \times k$. Moreover, usually the first (or zero) layer of such a network is an embedding layer that maps word's vocabulary number to some vector. And we need to store this embedding matrix too. Its size is $n_{vocab} \times k$, where n_{vocab} is the vocabulary size. Also we have an output softmax layer with the same number of parameters as in the embedding, i.e. $k \times n_{vocab}$. In our experiments, we try to reduce the embedding size and to decompose softmax layer as well as hidden layers.

We produce our experiments with compression on standard PTB models. There are three main benchmarks: Small, Medium and Large LSTM models [11]. But we mostly work with Small and Medium ones.

3 Compression Methods

3.1 Pruning and Quantization

In this subsection, we consider maybe not very effective but still useful techniques. Some of them were described in application to audio processing [12] or image-processing [13,14], but for language modeling this field is not yet well described.

Pruning is a method for reducing the number of parameters of NN. In Fig. 1. (left), we can see that usually the majority of weight values are concentrated near zero. It means that such weights do not provide a valuable contribution in the final output. We can set some threshold and then remove all connections with the weights below it from the network. After that we retrain the network to learn the final weights for the remaining sparse connections.

Quantization is a method for reducing the size of a compressed neural network in memory. We are compressing each float value to an eight-bit integer representing the closest real number in one of 256 equally-sized intervals within the range.

Pruning and quantization have common disadvantages since training from scratch is impossible and their usage is quite laborious. In pruning the reason

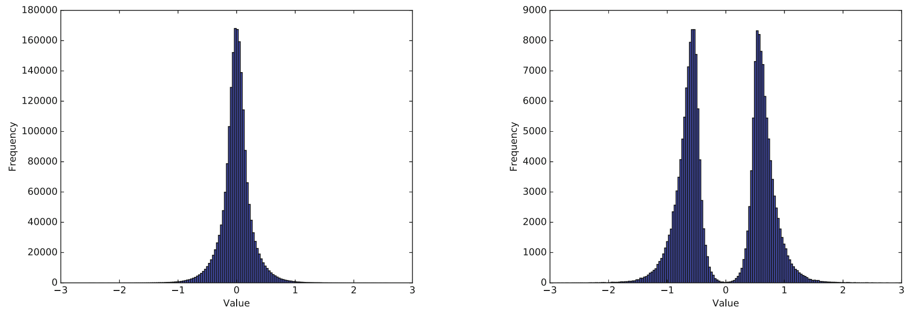

Fig. 1. Weights distribution before and after pruning

is mostly lies in the inefficiency of sparse computing. When we do quantization, we store our model in an 8-bit representation, but we still need to do 32-bits computations. It means that we have not advantages using RAM. At least until we do not use the tensor processing unit (TPU) that is adopted for effective 8- and 16-bits computations.

3.2 Low-Rank Factorization

Low-rank factorization represents more powerful methods. For example, in [1], the authors applied it to a voice recognition task. A simple factorization can be done as follows:

$$x_l^t = \sigma\left[W_\ell^a W_\ell^b x_{\ell-1}^t + U_l^a U_l^b x_\ell^{t-1} + b_l\right] \tag{11}$$

Following [1] require $W_l^b = U_{\ell-1}^b$. After this we can rewrite our equation for RNN:

$$x_l^t = \sigma\left[W_l^a m_{l-1}^t + U_l^a m_l^{t-1} + b_l\right] \tag{12}$$

$$m_l^t = U_l^b x_l^t \tag{13}$$

$$y_t = \text{softmax}\left[W_{L+1} m_L^t + b_{L+1}\right] \tag{14}$$

For LSTM it is mostly the same with more complicated formulas. The main advantage we get here from the sizes of matrices W_l^a, U_l^b, U_l^a. They have the sizes $r \times n$ and $n \times r$, respectively, where the original W_l and V_l matrices have size $n \times n$. With small r we have the advantage in size and in multiplication speed. We discuss some implementation details in Sect. 4.

3.3 The Tensor Train Decomposition

In the light of recent advances of tensor train approach [2,3], we have also decided to apply this technique to LSTM compression in language modeling.

The tensor train decomposition was originally proposed as an alternative and more efficient form of tensor's representation [15]. The TT-decomposition (or TT-representation) of a tensor $\boldsymbol{A} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ is the set of matrices $G_k[j_k] \in \mathbb{R}^{r_{k-1} \times r_k}$, where $j_k = 1, \ldots, n_k$, $k = 1, \ldots, d$, and $r_0 = r_d = 1$ such that each of the tensor elements can be represented as $\boldsymbol{A}(j_1, j_2, \ldots, j_d) = G_1[j_1] G_2[j_2] \ldots G_d[j_d]$. In the same paper, the author proposed to consider the input matrix as a multidimensional tensor and apply the same decomposition to it. If we have matrix A of size $N \times M$, we can fix d and such $n_1, \ldots, n_d$, $m_1, \ldots, m_d$ that the following conditions are fulfilled: $\prod_{j=1}^{d} n_j = N$, $\prod_{i=1}^{d} m_i = M$. Then we reshape our matrix A to the tensor $\boldsymbol{A}$ with d dimensions and size $n_1 m_1 \times n_2 m_2 \times \ldots \times n_d m_d$. Finally, we can perform tensor train decomposition with this tensor. This approach was successfully applied to compress fully connected neural networks [2] and for developing convolution TT layer [3].

In its turn, we have applied this approach to LSTM. Similarly, as we describe it above for usual matrix decomposition, here we also describe only RNN layer. We apply TT-decomposition to each of the matrices W and V in Eq. 2 and get:

$$z_\ell^t = \mathrm{TT}(W_i) x_{\ell-1}^t + \mathrm{TT}(V_l) x_\ell^{t-1} + b_\ell. \tag{15}$$

Here $\mathrm{TT}(W)$ means that we apply TT-decomposition for matrix W. It is necessary to note that even with the fixed number of tensors in TT-decomposition and their sizes we still have plenty of variants because we can choose the rank of each tensor.

4 Results

For testing pruning and quantization we choose Small PTB Benchmark. The results can be found in Table 1. We can see that we have a reduction of the size with a small loss of quality.

For matrix decomposition we perform experiments with Medium and Large PTB benchmarks. When we talk about language modeling, we must say that the embedding and the output layer each occupy one third of the total network size. It follows us to the necessity of reducing their sizes too. We reduce the output layer by applying matrix decomposition. We describe sizes of **LR LSTM 650-650** since it is the most useful model for the practical application. We start with basic sizes for W and V, 650×650, and 10000×650 for embedding. We reduce each W and V down to 650×128 and reduce embedding down to 10000×128. The value 128 is chosen as the most suitable degree of 2 for efficient device implementation. We have performed several experiments, but this configuration is near the best. Our compressed model, **LR LSTM 650-650**, is even smaller than **LSTM 200-200** with better perplexity. The results of experiments can be found in Table 2.

In TT decomposition we have some freedom in way of choosing internal ranks and number of tensors. We fix the basic configuration of an LSTM-network with two 600-600 layers and four tensors for each matrix in a layer. And we perform a grid search through different number of dimensions and various ranks.

We have trained about 100 models with using the Adam optimizer [16]. The average training time for each is about 5–6 h on GeForce GTX TITAN X

Table 1. Pruning and quantization results on PTB dataset

Model	Size	No. of params	Test PP
LSTM 200-200 (Small benchmark)	18.6 Mb	4.64 M	117.659
Pruning output layer 90% w/o additional training	5.5 Mb	0.5 M	149.310
Pruning output layer 90% with additional training	5.5 Mb	0.5 M	121.123
Quantization (1 byte per number)	4.7 Mb	4.64 M	118.232

Table 2. Matrix decomposition results on PTB dataset

	Model	Size	No. of params	Test PP
PTB benchmarks	LSTM 200-200	18.6 Mb	4.64 M	117.659
	LSTM 650-650	79.1 Mb	19.7 M	82.07
	LSTM 1500-1500	264.1 Mb	66.02 M	78.29
Ours	LR LSTM 650-650	16.8 Mb	4.2 M	92.885
	TT LSTM 600-600	50.4 Mb	12.6 M	168.639
	LR LSTM 1500-1500	94.9 Mb	23.72 M	89.462

(Maxwell architecture), but unfortunately none of them has achieved acceptable quality. The best obtained result (**TT LSTM 600-600**) is even worse than **LSTM-200-200** both in terms of size and perplexity.

5 Conclusion

In this article, we have considered several methods of neural networks compression for the language modeling problem. The first part is about pruning and quantization. We have shown that for language modeling there is no difference in applying of these two techniques. The second part is about matrix decomposition methods. We have shown some advantages when we implement models on devices since usually in such tasks there are tight restrictions on the model size and its structure. From this point of view, the model **LR LSTM 650-650** has nice characteristics. It is even smaller than the smallest benchmark on PTB and demonstrates quality comparable with the medium-sized benchmarks on PTB.

Acknowledgements. This study is supported by Russian Federation President grant MD-306.2017.9. A.V. Savchenko is supported by the Laboratory of Algorithms and Technologies for Network Analysis, National Research University Higher School of Economics. D.I. Ignatov was supported by RFBR grants no. 16-29-12982 and 16-01-00583.

References

1. Lu, Z., Sindhwan, V., Sainath, T.N.: Learning compact recurrent neural networks. In: Acoustics, Speech and Signal Processing (ICASSP) (2016)
2. Novikov, A., Podoprikhin, D., Osokin, A., Vetrov, D.P.: Tensorizing neural networks. In: Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, pp. 442–450 (2015)
3. Garipov, T., Podoprikhin, D., Novikov, A., Vetrov, D.P.: Ultimate tensorization: compressing convolutional and FC layers alike. CoRR/NIPS 2016 Workshop: Learning with Tensors: Why Now and How? abs/1611.03214 (2016)
4. Arjovsky, M., Shah, A., Bengio, Y.: Unitary evolution recurrent neural networks. In: Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, pp. 1120–1128 (2016)
5. Jelinek, F.: Statistical Methods for Speech Recognition. MIT Press, Cambridge (1997)
6. Kneser, R., Ney, H.: Improved backing-off for m-gram language modeling. Proc. IEEE Int. Conf. Acoust. Speech Sig. Process. **1**, 181–184 (1995)
7. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
8. Mikolov, T.: Statistical Language Models Based on Neural Networks. Ph.D. thesis, Brno University of Technology (2012)
9. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: Kremer, S.C., Kolen, J.F. (eds.) A Field Guide to Dynamical Recurrent Neural Networks (2001)
10. Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint 2014f (2014). arXiv:1409.1259
11. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization. Arxiv preprint (2014)
12. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In: Acoustics, Speech and Signal Processing (ICASSP) (2016)
13. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kaut, J.: Pruning convolutional neural networks for resource efficient transfer learning. arXiv preprint (2016). arXiv:1611.06440
14. Rassadin, A.G., Savchenko, A.V.: Compressing deep convolutional neural networks in visual emotion recognition. In: Proceedings of the International Conference on Information Technology and Nanotechnology (ITNT), vol. 1901, pp. 207–213. CEUR-WS (2017)
15. Oseledets, I.V.: Tensor-train decomposition. SIAM J. Sci. Computing **33**(5), 2295–2317 (2011)
16. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. In: The International Conference on Learning Representations (ICLR) (2015)