FACTORIZATION TRICKS FOR LSTM NETWORKS

Oleksii Kuchaiev
NVIDIA
Santa Clara, CA 95050, USA
okuchaiev@nvidia.com

Boris Ginsburg
NVIDIA
Santa Clara, CA 95050, USA
bginsburg@nvidia.com

ABSTRACT

Large Long Short-Term Memory (LSTM) networks have tens of millions of parameters and they are very expensive to train. We present two simple ways of reducing the number of parameters in LSTM network: the first one is "matrix factorization by design" of LSTM matrix into the product of two smaller matrices, and the second one is partitioning of LSTM matrix, its inputs and states into the independent groups. Both approaches allow us to train large LSTM networks significantly faster to the state-of the art perplexity. On the One Billion Word Benchmark we improve single model perplexity down to 24.29.

1 INTRODUCTION

LSTM networks (Hochreiter & Schmidhuber, 1997) and it's many variants have been successfully used in language modeling (Jozefowicz et al., 2016; Shazeer et al., 2017), speech recognition (Xiong et al., 2016), machine translation (Wu et al., 2016), image annotation (Vinyals et al., 2016), reinforcement learning (Lample & Chaplot, 2016) and many other tasks. However, these networks have tens of millions of parameters in recurrent layers, and they often require weeks of training on multi-GPU systems (Jozefowicz et al., 2016).

We introduce two modifications of LSTM cell: factorized LSTM (F-LSTM) and group LSTM (G-LSTM), both providing significant reduction in the number of parameters and much faster training. The first method approximates big LSTM matrix with a product of 2 smaller matrices. The second method partitions LSTM cell into the independent groups.

We test new F-LSTM and G-LSTM architectures on the task of language modeling using One Billion Word Benchmark (Chelba et al., 2013). As a baseline, we used BIGLSTM model without CNN inputs described by Jozefowicz et al. (2016). We train all networks for 1 week on a DGX-1 system with 8 Tesla P100 GPUs, after which BIGLSTM's eval perplexity was 31.0, while both F-LSTM and G-LSTM models got to much better perplexity: 28.11 for F-LSTM and 28.17 for G-LSTM. For details see Section 3. Further, we trained G-LSTM with 4 group for an additional week on a single DGX-1 and achieved new single-model state-of-the-art perplexity of 24.29.

Our code is available at `https://github.com/okuchaiev/f-lm`

1.1 LONG SHORT-TERM MEMORY OVERVIEW

We use the following notation. RNN layer, at any given time step t, takes some input x_t and it's previous state h_{t-1}, to generate a new state $RNN : h_{t-1}, x_t \rightarrow h_t$. Usually, RNNs are trained using gradient back-propagation through time and thus, learning long-range dependencies is challenging due to the vanishing and exploding gradient problems (Bengio et al., 1994; Pascanu et al., 2013). To address this issue, the concept of memory c_t has been introduced in the LSTM cell (Hochreiter & Schmidhuber, 1997), with recurrent computations described by:

$$LSTM : h_{t-1}, c_{t-1}, x_t \rightarrow h_t, c_t. \tag{1}$$

For LSTM cell with projection of size p, transformation 1 is computed as follows (Sak et al., 2014; Zaremba et al., 2014). First, cell gates (i, f, o, g) are computed:

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} T \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \tag{2}$$

where $x_t \in R^p$, and $h_t \in R^p$, $T : R^{2p} \rightarrow R^{4n}$ is an affine transform $T = W * [x_t, h_{t-1}] + b$, and $W \in R^{4n,2p}$, $b \in R^{4n}$.

Next state $h_t \in R^p$ and memory $c_t \in R^n$ are computed using following equations:

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = P(o \odot tanh(c_t))$$

where $P : R^n \rightarrow R^p$ is a linear projection. Hence, the major part of LSTM cell computation is spent computing affine transform T because it involves multiplication with $4n \times 2p$ matrix W. Our cell modifications focus on reducing the number of parameters in the transform T.

1.2 Related Work

The partition of layer into parallel groups have been introduced by Krizhevsky et al. (2012) in AlexNet, where some convolutional layers have been divided into two groups to split the model between two GPUs. After that, multi-group convoluitonal networks have been widely used to reduce network weights and required compute, for example Esser et al. (2016). This multi-group approach was extended to the extreme by Chollet (2016) in Xception architecture, with one input channel per group.

The idea of factorization of large convolutinal layer into the stack of layers with smaller filters was successfully used in, for example, VGG networks (Simonyan & Zisserman, 2014) and in ResNet "bottleneck design" (He et al., 2016).

2 Models

2.1 Factorized LSTM cell

Factorized LSTM (F-LSTM) cell simply follows "factorization by design" principle and replaces transform T with one big matrix W, by another transform T' which contains two smaller matrices that essentially try to approximate W as $W = W2 * W1$, where $W1$ is $r \times 2p$, $W2$ is $4n \times r$, and $r < p <= n$ is a factorization rank. The key underlying assumption here is that W can be well approximated by the matrix of rank r and the factorization is also learned during network training. Hence, equations (1) - (3) remain intact for F-LSTM, except for the new affine transform formula:

$$T'(d) = W2 * W1 * d + b \tag{3}$$

Such approximation contains less LSTM parameters than original model - $(r * 2p + r * 4n)$ versus $(2p * 4n)$ and, therefore, can be computed faster and synchronized faster in the case of mutli-gpu or distributed training. While one might go further and try to approximate T' using arbitrary feed forward neural network with $2p$ inputs and $4n$ outputs, during our initial experiments we did not see immediate benefits of doing so. Hence, it remains a topic of future research.

2.2 Group LSTM cell

Another approach to speeding up LSTM computation goes beyond simple approximation of T. Here, inspired by groups in convolutional neural networks Krizhevsky et al. (2012), we postulate that some parts of the input x_t and hidden state h_t can be thought of as independent feature groups. For example, if we use two groups, then both x_t and h_t are effectively split into two vectors concatenated together $x_t = (x_t^1, x_t^2)$ and $h_t = (h_t^1, h_t^2)$ with h_t^i only dependent on x_t^i, h_{t-1}^i and cell's memory

state. Therefore, for k groups equation 2 changes to:

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \left(\begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} T^1 \begin{pmatrix} x_t^1 \\ h_{t-1}^1 \end{pmatrix}, ..., \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} T^k \begin{pmatrix} x_t^k \\ h_{t-1}^k \end{pmatrix} \right) \tag{4}$$

with the rest of equations remaining unchanged. Here, T^j is a group j's affine transform from $R^{2p/k}$ to $R^{4n/k}$, and k of them have $k * \frac{4n*2p}{k*k}$ parameters. Note, that this cell architecture is well suited for model parallelism since every group computation is independent.

An alternative interpretation of G-LSTM layers is demonstrated in the Figure 1 (c). While this might look similar to ensemble (Shazeer et al., 2017) or multi-tower (Ciregan et al., 2012) models, the key differences are: (1) input to different groups is different and assumed independent, and (2) instead of computing ensemble output, it is concatenated into independent pieces.

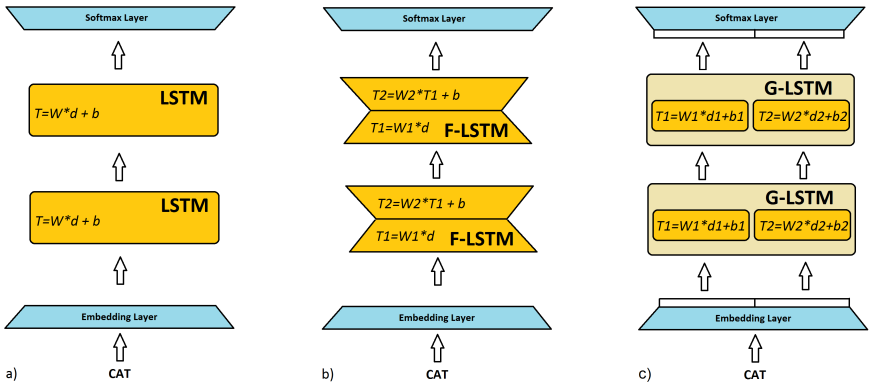

Figure 1: Language model using: (a) 2 regular LSTM layers, (b) 2 F-LSTM layers, and (c) 2 G-LSTM layers with 2 group in each layer. Equations inside cells show what kind of affine transforms are computed by those cells at each time step.

3 Experiments and Results

We test our new cells on the task of learning the joint probabilities over word sequences of arbitrary lengths n:

$$P(w_1, ..., w_n) = \prod_{i=1}^{n} P(w_i | w_1, ..., w_{i-1})$$

An intuitive explanation of this task is being able to assign high probabilities to "real" sentences and low probabilities to random sequences of words. Due to their sequential nature, recurrent neural networks are well suited for such problem and currently achieve state-of-the-art results on this task (Jozefowicz et al., 2016; Shazeer et al., 2017). Figure 1 (a) shows the simplest LSTM-based language model architecture. First, the words are embedded into the low dimensional dense input for RNN, then the "context" is learned using RNNs via pre-specified number of steps and, finally, the softmax layer converts RNN output into the probability distribution $P(w_1, ..., w_n)$.

We perform our experiments using DGX-1 machine with 8 Pascal-based Tesla P100 GPUs. Each model is allowed to run for exactly 1 week using all 8 GPUs. We test the following configurations:

- BIGLSTM - model introduced by Jozefowicz et al. (2016), without CNN inputs
- BIG F-LSTM F512 - with rank of 512 for LSTM matrix W
- BIG G-LSTM G-4, with 4 groups in both layers
- BIG G-LSTM G-16, with 16 groups in both layers.

Table 1: One Billion Words benchmark evaluation results after 1 week of training using DGX-1 (8 GPUs) for every model.

Model	Perplexity	Step	Number of RNN parameters	Words per second
BIGLSTM baseline	31.001	584.6K	83,951,616	20.3K
BIG F-LSTM F512	**28.11**	1.217M	51,445,760	42.9K
BIG G-LSTM G-4	**28.17**	1.128M	33,619,968	41.1K
BIG G-LSTM G-16	34.789	850.4K	21,037,056	41.7K

We train using synchronous Adagrad optimizer, mini-batch of 128 per GPU, dropout keep probability of 0.9 and 0.2 learning rate. Table 1 summarizes our experiments. Judging from the training loss plots 2, it is clearly visible that at the same step count, the model with more parameters wins. However, in practice, this extra complexity might not be worth the extra time it takes to train. More specifically, while the difference between BIGLSTM and BIG G-LSTM-G4 is clearly visible, BIG G-LSTM-G4 contains 2.4 less RNN parameters than BIGLSTM, trains 2 times faster and, as a results, achieves better evaluation perplexity within the same training time budget (1week).

Since BIG G-LSTM-G4 result looks almost as good as BIG F-LSTM-F512 but it has fewer parameters, we let it run for 1 more week (approximately 28*2 epochs in total) to achieve the perplexity of **24.29** which, we believe, is the new state-of-the-art perplexity for single model on this benchmark.

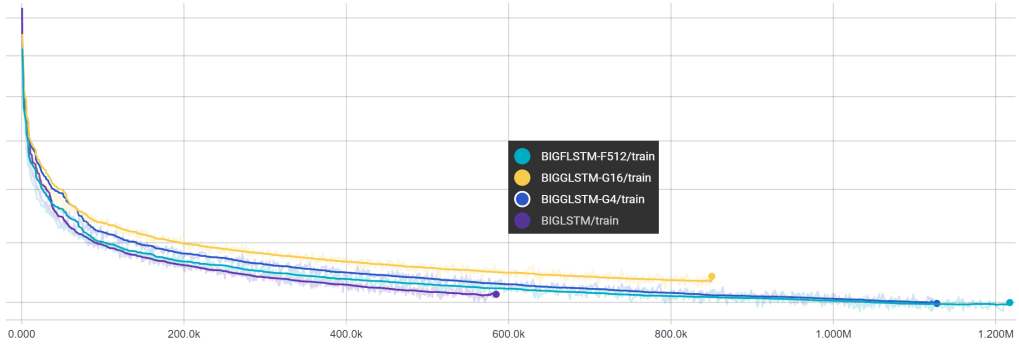

Figure 2: Y-axis: training loss log-scale, X-axis: step, or mini-batch count. BIGLSTM baseline, BIG G-LSTM-G4, BIG G-LSTM-G16, and BIG F-LSTM-F512. It is clearly visible that at the same step count, the model with more parameters wins. On the other hand, factorized models can do significantly more iterations in the given amount of time.

3.1 Future research

It might be possible to reduce the number of RNN parameters even further by stacking G-LSTM layers with increasing group counts on top of each other. In our second, smaller experiment, we replace the second layer of BIG G-LSTM-G4 network by the layer with 8 groups instead of 4, and call it BIG G-LSTM-G4-G8. We let both BIG G-LSTM-G4 and BIG G-LSTM-G4-G8 ran for 1 week on 4 GPUs each and achieved perplexities of 29.09 and 28.47 correspondingly. Hence, the model with "hierarchical" groups did not lose much accuracy, ran faster and got better perplexity. Such "hierarchical" group layers look intriguing as they might provide a way for learning different levels of abstractions but this remains a topic for future research.

References

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3642–3649. IEEE, 2012.

Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, pp. 201604850, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

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

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. *arXiv preprint arXiv:1609.05521*, 2016.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Interspeech*, pp. 338–342, 2014.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE transactions on pattern analysis and machine intelligence*, 2016.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.