Adaptive Mixture of Low-Rank Factorizations for Compact Neural Modeling

Ting Chen[1][*], Ji Lin[2][*], Tian Lin[3], Song Han[2], Chong Wang[3], Dengyong Zhou[3]
[1] University of California, Los Angeles, [2]Massachusetts Institute of Technology, [3]Google
`tingchen@cs.ucla.edu, {jilin, songhan}@mit.edu,`
`{tianlin,chongw,dennyzhou}@google.com`

Abstract

Modern deep neural networks have a large amount of weights, which make them difficult to deploy on computation constrained devices such as mobile phones. One common approach to reduce the model size and computational cost is to use low-rank factorization to approximate a weight matrix. However, performing standard low-rank factorization with a small rank can hurt the model expressiveness and significantly decrease the performance. In this work, we propose to use a mixture of multiple low-rank factorizations to model a large weight matrix, and the mixture coefficients are computed dynamically depending on its input. We demonstrate the effectiveness of the proposed approach on both language modeling and image classification tasks. Experiments show that our method not only improves the computation efficiency but also maintains (sometimes outperforms) its accuracy compared with the full-rank counterparts.

1 Introduction

Modern neural networks usually contain millions of parameters [5, 9], and they are difficult to be deployed on mobile devices with limited computation resources. To solve this problem, model compression techniques are proposed in recent years. Low-rank factorization is a popular way of reducing the matrix size. It has been extensively explored in the literature [6, 7, 4, 11]. Mathematically, a large weight matrix $W \in \mathbb{R}^{m \times n}$ is factorized to two small rank-d matrices $U \in \mathbb{R}^{m \times d}$, $V \in \mathbb{R}^{n \times d}$ with $W = UV^T$. Since both U and V are dense, no sparsity support is required from specialized hardware. It naturally fits the general-purpose, off-the-shelf CPUs and GPUs.

To significantly reduce the model size and computation, the rank d in the low-rank factorization needs to be small. However, a small rank can limit the expressiveness of the model [10] and lead to worse performance. To understand the limitations, given a n-dim feature vector h, we observe that $V^T h$, as in $U(V^T h)$, is a linear projection from a high-dimensional space (n dims) to a low-dimensional space (d dims). This can lead to a significant loss of information. The conflict between the rank d and the model expressiveness prevents us from obtaining a both *compact* and *accurate* model.

To address the dilemma, we propose to increase the expressiveness by learning an adaptive, input-dependent factorization, rather than performing a fixed factorization of a weight matrix. To do so, we use a mixture of multiple low-rank factorizations. The mixing weights are computed based on the input. This creates an adaptive linear projection from a high-dimensional space to a low-dimensional space. Compared to the conventional low-rank factorization, the proposed approach can significantly improve its performance while only introducing a small additional cost.

[*]Equal Contribution. Work done at Google.

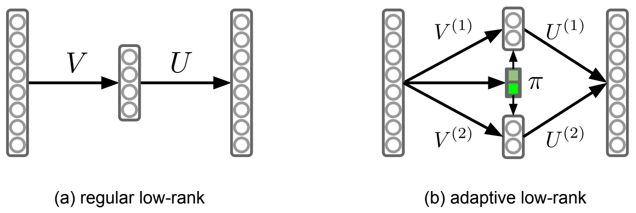


Figure 1: (a) regular factorization and (b) adaptive mixture of low-rank factorizations. First compute $z_k = \pi_k(h)((V^{(k)})^T h)$ and then $h' = \sum_k U^{(k)} z_k$, where z can be treated as the middle layer. Techniques like pooling can be applied to compute π to make it efficient.

2 Adaptive mixture of low-rank factorizations

We propose to use an unnormalized learned mixture of low-rank factorizations whose mixing weights are computed adaptively based on the input. More specifically, denoting the input by h and the number of mixture components by K, we decompose a large weight matrix by

$$W(h) = \sum_{k=1}^{K} \pi_k(h) U^{(k)} \big(V^{(k)}\big)^\top, \tag{1}$$

where $\pi(\cdot) : \mathbb{R}^n \to \mathbb{R}^K$ is the function which maps each input to its mixture coefficients. For example, π can be a small neural network. This introduces a small amount of extra parameters and computation. We will later discuss the details of efficient ways to implement the mixture function π.

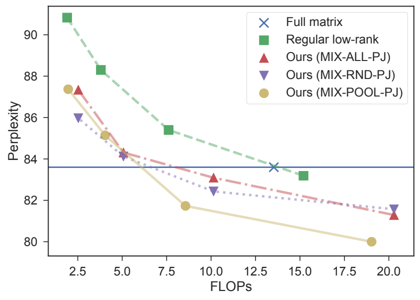
If π_k, $k = 1, ..., K$, is chosen to be constant (input independent), it can be absorbed into either $U^{(k)}$ or $V^{(k)}$. Thus, the proposed method reduces to the low-rank factorization. This is evidenced by rewriting W as $W = [\pi_1 U^{(1)}, ..., \pi_K U^{(K)}][V^{(1)}, ..., V^{(K)}]^\top$. In other words, the conventional low-rank factorization can be considered as a special case of our method. Figure 1 depicts the proposed framework.

Adaptive mixing weights $\pi(h)$. The mixing weights can encode important information that we can use to increase the expressiveness of the projected low-dimensional space. Under our framework, the generation of the mixing weights $\pi(h)$ is flexible. A straight-forward approach is to use a non-linear transformation of the input to the weight matrix. For example, $\pi(h) = \sigma(Ph)$, where σ is a non-linear transformation, such as sigmoid or hyperbolic tangent function, and $P \in \mathbb{R}^{K\times n}$ is an extra weight matrix. This adds some extra parameters and computation to the model since the linear projection that we construct is $\mathbb{R}^n \to \mathbb{R}^K$. To further reduce the parameter and computation in the mixing weights π, we propose the following strategies.

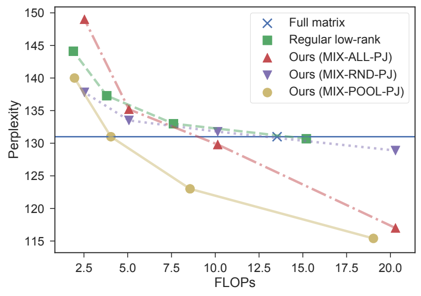
Pooling before projection. We do not require the whole input to compute the mixture function π. Instead, we can apply pooling to the input h before projection. For example, a global average pooling can be applied if the input is a 3D tensor (for images); for a 1D vector, we can segment the vector and average each segmentations. By applying pooling, we can both save the computation and better capture the global information.

Random projection. To reduce the number of parameters in the linear projection of h, we can use a random matrix P_{random} in place of a fully adjustable P, i.e. $\pi(h) = \sigma(P_{\text{random}} h)$. Note that we can simply save a seed to recover the random matrix, but it still requires the same amount of memory and computation as the fully adjustable linear projection of h.

Increased expressiveness. The adaptive mixing weights introduce a non-linear transformation into the high-to-low-dimensional projection that can be more expressive. Since each $W(h)$ is a data-dependent low-rank matrix, there is no constant linear weight independent to the input (even a full-rank matrix) that can mimic the transformation $W(h)$. Generating the whole weight matrices can be very expensive. Our method can be seen as a swift approach to generate the weights by adaptively adjusting mixing weights for the linear bottleneck. It assigns weights into groups and dynamically controls them at the group level.



(a) Penn Tree Bank



(b) Text8

Figure 2: FLOPs vs. perplexity. The horizontal line is the full LSTM's baseline accuracy. We also compare variants of the proposed approaches with regular low-rank factorization, indicated by different colors and markers. Lower perplexity is better.

Table 1: Performance for different networks on ImageNet. With negligible FLOPs increase, adaptive low-rank factorizations outperforms regular ones.

Network	Top 1	Params	MACs
MobileNet	70.6	4.2M	575M
Low-rank MobileNet (0.75)	68.8	2.6M	209M
Adaptive Low-rank MobileNet (0.75)	**70.5**	2.8M	209M
Low-rank MobileNet	71.7	3.4M	300M
Adaptive Low-rank MobileNet	**73.1**	3.7M	300M

3 Experiments

Recurrent neural networks for language modeling. Recurrent neural networks (RNNs) are widely used in language modeling, machine translation and sequence modeling in general. We adopt the same Long Short Term Memory (LSTM) models and follow the settings from a previous state-of-the-art model [12] for language modeling, and use Penn Tree Bank (PTB) as well as Text8 datasets. More specifically, we use the medium sized model introduced in [12].

We test three variants of the proposed model against regular low-rank factorization, each with different ways of computing mixing weights, namely (1) MIX-ALL-PJ: direct linear projection of the input vector h, (2) MIX-POOL-PJ: linear projection after segment-based mean pooling of the input vector h, and (3) MIX-RND-PJ: use a random projection for the input vector h. Among these adaptive projection methods, MIX-ALL-PJ has a large amount of extra parameters, MIX-POOL-PJ has a small amount of extra parameters, and MIX-RND-PJ has no extra parameters. We compute the FLOPs of a single time-step of applying LSTM, and the perplexity associated to different settings.

The results are shown in Figure 2. Firstly, with adaptive mixtures, the low-rank factorization model achieved 40% reduction in FLOPs, and even surpassed the performance of the full matrix baseline by decreasing the perplexity by 1.7 points. Secondly, the use of adaptive mixtures can significantly improve the performance compared with regular, non-adaptive low-rank factorization. Thirdly, using pooling before projection can be a good choice for computing the mixing weights π. It not only reduces the computation and parameter size, but can better capture the global information and achieve better accuracy.

CNN for image recognition. We further demonstrate the effectiveness of the proposed approach on compressing CNN models on ImageNet [2]. We chose to use modern compact CNN models as the baseline (which are harder to compress), rather than using the bulky CNN models (which is easier to compress). Specifically, we choose to compress the point-wise convolution in *depth-wise separable convolutions* [1], MobileNet [3, 8] in particular.

Table 1 shows the comparison of different state-of-art compact convolutional models. We observed that compared to the regular low-rank factorization of MobileNet model (a.k.a. MobileNet V2), the proposed method achieves significantly better results (2.5% and 2% for two different Low-rank MobileNet settings, respectively), while only adding negligible extra FLOPs (less than 1%).

References

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

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[3] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[4] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

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

[6] Zhiyun Lu, Vikas Sindhwani, and Tara N Sainath. Learning compact recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5960–5964. IEEE, 2016.

[7] Preetum Nakkiran, Raziel Alvarez, Rohit Prabhavalkar, and Carolina Parada. Compressing deep neural networks using a rank-constrained topology. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv preprint arXiv:1801.04381*, 2018.

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

[10] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations*, 2018.

[11] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

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