Tensor network and neural networks

Junjie He
School of Information Science and Technology
ShanghaiTech University
Shanghai, China
hejj1@shanghaitech.edu.cn

Jiaqiong Zhang
School of Information Science and Technology
ShanghaiTech University
Shanghai, China
zhangjq@shanghaitech.edu.cn

I. Introduction

Main topic for our project are tensor methods and neural network.Deep neural networks currently demonstrate state-of-the-art performance in several domains.such as computer vision, speech recognition, text processing, etc.These advances have become possible because of algorithmic advances, large amounts of available data,and modern hardware. For example, convolutional neural networks (CNNs) [1] [2]show by a large margin superior performance on the task of image classification.These models have thousands of nodes and millions of learnable parameters and are trained using millions of images [3] on powerful Graphics Processing Units (GPUs).The necessity of expensive hardware and long processing time are the factors that complicate the application of such models on conventional desktops and portable devices. In tradition neural network,this layer has a linear transformation of a high dimension input signal to a high dimension output signal.For example,the data set CIFAR10 widely used in deep learning course is a collection of pictures.When it used as input signal into neural network,the pictures in it will be divided 32*32*3 pixels.particularly, 32*32 means a picture divided into 32*32 pixel blocks,and 3 means three channels(RGB).Then,the input signal will be reshaped a 32*32*3 dimensional vector .Obviously,such an operation will largely increase the dimension of input signal,and leads to further complexity of the calculations.The data set CIFAR10 [5] is already a very simple data set in the field of deep learning.However,in the convolutional neural network model used in practical application the dimensions of the input and output signals of the fully-connected layers are of the order of thousands, bringing the number of parameters of the fully-connected layers up to millions.This is undoubtedly a very demanding requirement for hardware facilities. Consequently, a large number of works tried to reduce both hardware requirements (e. g. memory demands) and running times.To solve this problem,We consider the most frequently used layer of neural network:fully-connected layer.We use a compact tensor train data set to represent the matrix of the fully-connected layers using few parameters while keeping enough flexibility to perform signal transformations [6].And,the layer transformed should be compatible with the existing training algorithms for neural network,because all the derivatives required by the back propagation algorithm [4] can be computed using the properties of Tensor train set. Tensors are natural multidimensional generalizations of matrices and have attracted tremendous interest in recent years.Multilinear algebra,tensor analysis, and the theory of tensor approximations play increasingly important roles in computational mathematics and numerical analysis [7] [8] [9] [10]. An efficient representation of a tensor (by tensor we mean only an array with d indices) by a small number of parameters may give us an opportunity and ability to work with d-dimensional problems, with d being as high as 10, 100, 1000 or even one million.Problems of such sizes cannot be handled by standard numerical methods due to the curse of dimensionality, since everything (memory, amount of operations) grows exponentially in d.So,Tensor train decomposition will a effective way to solve this problem. We will apply our method to popular network architectures proposed for data set CIFAR10.We will experimentally use the networks with tradition fully-connected layer and the tensor fully-connected layer to train a neural network model.Then,we will compare the performance of two models.

II. Preliminary ideas

In various fields, low-rank approximation was applied to reduce the computation cost and memory usage. In [11], they generalize the idea of low-rank. The authors do not find low-rank approximation of weight matrix in fully-connected layers, they treat the matrix as multidimensional tensor and employ Tensor Train decomposition [6] to accelerate the computation. Usally, wider neural network can achieve better performance than narrow neural network. But wide neural networks imply large dense matrix, amount of computation resources are used in per step when training neural networks. By using Tensor Train decomposition for weight matrix, wide neural network can be developed for applications with moderate computation cost and memory usage. [11] shows that wide and shallow neural networks has competitive performance with the state-of-art deep neural networks by traing a shallow network oin the outputs of a trained deep neural network. They report the improvement of performance with the increase of the layer size and used up to 30 000 hidden units while restricting the matrix rank of the weight matrix in order to be able to keep and to update it during the training. Restricting the TT-ranks of the weight matrix (in contrast to the matrix rank) allows to use much wider layers potentially leading to the greater expressive power of the model.

CP-decomposition algorithm was applied to compress convolution kernel in CNNs. And they also using properties of CP-decomposition to speed up the inference time. tp speed up computation of matrix-by-vector, properties of the Kronecker product of matrices was exploited. These matrices have the same structure as TT-matrices with unit TT-ranks. We can generalize this idea to formulate the weight matrix with TT-matrices with unit TT-ranks. The Tucker format and the canonical format will meet the curse of dimensionality, TT-format is immune to the cues of dimensionality and its algorithm are robust.

A d-dimensional array (tensor) $\mathcal{A}$ is said to be TT-format if for each dimension $k = 1, ..., d$ and for each possible value of the k-th dimension index $j_k = 1, ..., n_k$ there exists a matrix $\mathbf{G}_k[j_k]$ such that all elements of $\mathcal{A}$ can be computed as the following matrix product:

$$\mathcal{A}(j_1, ..., j_d) = \mathbf{G}_1[j_1]\mathbf{G}_2[j_2] \cdots \mathbf{G}_d[j_d] \tag{1}$$

All the matrices $\mathbf{G}_k[j_k]$ related to the same dimension k are restricted to be of the same size $r_{k-1} \times r_k$. The values r_0 and r_d equal to 1 in order to keep the matrix product (1) of size 1×1. In what follows we refer to the representation of a tensor in the TT-format as the TT-representation or d the TT-decomposition. The sequence $\{r_k\}_{k=0}^{d}$ is referred to as the TT-ranks of the TT-representation of $\mathbf{A}$ (or the ranks for short), its maximum – as the maximal TT-rank of the TT-representation n of $\mathcal{A} : r = \max_{k=0,...,d} r_k$. The collections of the matrices $(\mathbf{G}_k[j_k])_{j_k}^{n_k} = 1$ corresponding to the same dimension (technically, 3-dimensional arrays $\mathcal{G}_k$) are called the cores.

We use the symbols $\mathbf{G}_k[j_k](\alpha_{k-1}, \alpha_k)$ to denote the element of the matrix $\mathbf{G}_k[j_k]$ in the position (α_{k-1}, α_k),

where $\alpha_{k-1} = 1, ..., r_{k-1}, \alpha_k = 1, ..., r_k$. Equation (1) can be equivalently rewritten as the sum of the products of the elements of the cores:

$$\mathcal{A}(j_1, ..., j_d) = \sum_{\alpha_0, ..., \alpha_d} \mathbf{G}_1[j_1](\alpha_0, \alpha_1) \cdots \mathbf{G}_d[j_d](\alpha_{d-1}, \alpha_d) \tag{2}$$

The representation of a tensor $\mathcal{A}$ via the explicit enumeration of all its elements requires to store $\prod_{k=1}^{d} n_k$ numbers compared with $\sum_{k=1}^{d} n_k r_{k-1} r_k$ numbers if the tensor is stored in the TT-format. Thus, the TT-format is very efficient in terms of memory if the ranks are small.

An attractive property of the TT-decomposition is the ability to efficiently perform several types of operations on tensors if they are in the TT-format: basic linear algebra operations, such as the addition of a constant and the multiplication by a constant, the summation and the entrywise product of tensors (the results of these operations are tensors in the TT-format generally with the increased ranks); computation of global characteristics of a tensor, such as the sum of all elements and the Frobenius norm. See [17] for a detailed description of all the supported operations.

Neural networks are usually trained with the stochastic gradient descent algorithm where the gradient is computed using the back-propagation procedure. Back-propagation allows to compute the gradient of a loss-function L with respect to all the parameters of the network. The method starts with the computation of the gradient of L w.r.t. the output of the last layer and proceeds sequentially through the layers in the reversed order while computing the gradient w.r.t. the parameters and the input of the layer making use of the gradients computed earlier. Applied to tensorizing fully-connected layers the back-propagation method computes the gradients w.r.t. the input $\mathbf{x}$ and the parameters $\mathbf{W}$ and $\mathbf{b}$ given the gradients $\frac{\partial L}{\partial \mathbf{y}}$ w.r.t to the output $\mathbf{y}$:

$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{W}^T \frac{\partial L}{\partial \mathbf{y}}, \frac{\partial L}{\partial \mathbf{W}} = \mathbf{W}^T \frac{\partial L}{\partial \mathbf{y}} \mathbf{x}^T, \frac{\partial L}{\partial \mathbf{b}} = \mathbf{W}^T \frac{\partial L}{\partial \mathbf{y}} \tag{3}$$

In what follows we derive the gradients required to use the back-propagation algorithm with the tensorizing layers. To compute the gradient of the loss function w.r.t. the bias vector $\mathbf{b}$ and w.r.t. the input vector $\mathbf{x}$ one can use equations (3). The latter can be applied using the matrix-by-vector product (where the matrix is in the TT-format) with the complexity of $\mathcal{O}(dr^2 n \max\{m, n\}^d) = \mathcal{O}(dr^2 n \max\{M, N\})$. To perform a step of stochastic gradient descent, we can use traditional back-propagation in computational graph to compute gradient of loss function w.r.t the weight matrix $\mathbf{W}$, then we convert the gradient matrix into the TT-format using TT-SVD algorithm. Another way to learn the TensorNet parameters is to compute gradient of loss function w.r.t the cores of the TT-representations of $\mathbf{W}$.

For high-dimensional matrices, the TT-SVD algorithm will meet curse of dimensionality, i.e., computation cost will increase quickly such as exponentially. Then we have difficulty to employ TT-format in neural networks. To A Randomized Tensor Train Singular Value Decomposition Each of the existing TT decomposition algorithms, including the TT-SVD and randomized TT-SVD, is successful in the field, but neither can both accurately and efficiently decompose large-scale sparse tensors. [13] proposes a new quasi-best fast TT decomposition algorithm for large-scale sparse tensors with proven correctness and the upper bound of its complexity is derived. In numerical experiments, authors verify that the proposed algorithm can decompose sparse tensors faster than the TT-SVD, and have more speed, precision and versatility than randomized TT-SVD [14], and it can be used to decomposes arbitrary high-dimensional tensor without losing efficiency when the number of non-zero elements is limited. Faster TT-SVD algorithm can be integrated into tensorizing neural networks, and it should be more efficiently to solve the problem in large scale.

III. Experiments

In all experiments we will use MATLAB extension[1] of the MatConvNet framework[2]. For the operations related to the TT-format we use the TT-Toolbox[3] implemented in MATLAB as well. To show the properties of the TT-layer and compare different strategies for setting its parameters: dimensions of the tensors representing the input/output of the layer and the TT-ranks of the compressed weight matrix. We run the experiment on the MNIST dataset for the task of handwritten-digit recognition. As a baseline we use a neural network with two fullyconnected layers (1024 hidden units) and rectified linear unit (ReLU) and compute error on the test set. For more reshaping options we resize the original 28 × 28 images to 32 × 32.

Futhermore, we will train several networks differing in the parameters of the single TT-layer. The networks contain the following layers: the TT-layer with weight matrix of size 1024×1024, ReLU, the fully-connected layer with the weight matrix of size 1024 × 10. We test different ways of reshaping the input/output tensors and try different ranks of the TT-layer. As a simple compression baseline in the place of the TT-layer we use the fully-connected layer such that the rank of the weight matrix is bounded (implemented as follows: the two consecutive fully-connected layers with weight matrices of sizes 1024 × r and r ×1024, where r controls the matrix rank and the compression factor).

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems 25 (NIPS), 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition,"in International Conference on Learning Representations (ICLR), 2015.

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla,M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," International Journal of Computer Vision (IJCV), 2015.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors,"Nature, vol. 323, no. 6088, pp. 533–536, 1986.

[5] A. Krizhevsky, "Learning multiple layers of features from tiny images," Master's thesis, Computer Science Department, University of Toronto, 2009.

[6] I. V. Oseledets, "Tensor-Train decomposition," SIAM J. Scientific Computing, vol. 33, no. 5, pp. 2295–2317, 2011.

[7] L. de Lathauwer, B. de Moor, and J. Vandewalle, A multilinear singular value decomposition, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278.

[8] L. de Lathauwer, B. de Moor, and J. Vandewalle, On best rank-1 and rank-(R1, R2,...,RN) approximation of high-order tensors, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342.

[9] R. Bro, PARAFAC: Tutorial and applications, Chemometrics Intell. Lab. Syst., 38 (1997), pp. 149–171.

[10] L. Grasedyck, Existence and computation of low Kronecker-rank approximations for large systems in tensor product structure, Computing, 72 (2004), pp. 247–265.

[11] Novikov, A., Podoprikhin, D., Osokin, A., Vetrov, D. P. (2015). Tensorizing neural networks. In Advances in neural information processing systems (pp. 442-450).

[12] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in Advances in Neural Information Processing Systems 27 (NIPS), 2014, pp. 2654–2662.

[13] Li, L., Yu, W., Batselier, K. (2019). Faster Tensor Train Decomposition for Sparse Data. arXiv preprint arXiv:1908.02721.

[14] Huber, B., Schneider, R., Wolf, S. (2017). A randomized tensor train singular value decomposition. In Compressed Sensing and its Applications (pp. 261-290). Birkhäuser, Cham.

[1] https://github.com/Bihaqo/TensorNet

[2] http://www.vlfeat.org/matconvnet

[3] https://github.com/oseledets/TT-Toolbox