Compressing deep neural networks by matrix product operators

Ze-Feng Gao,1 Song Cheng,2,3 Rong-Qiang He,1 Z. Y. Xie,1,‡ Hui-Hai Zhao,4,† Zhong-Yi Lu,1,‡ and Tao Xiang2,3,§

1Department of Physics, Renmin University of China, Beijing 100872, China
2Institute of Physics, Chinese Academy of Sciences, Beijing 100190, China
3University of Chinese Academy of Sciences, Beijing, 100049, China
4RIKEN Brain Science Institute, Wako-shi, Saitama, 351-0106, Japan

A deep neural network is a parameterization of a multi-layer mapping of signals in terms of many alternatively arranged linear and nonlinear transformations. The linear transformations, which are generally used in the fully-connected as well as convolutional layers, contain most of the variational parameters that are trained and stored. Compressing a deep neural network to reduce its number of variational parameters but not its prediction power is an important but challenging problem towards the establishment of an optimized scheme in training efficiently these parameters and in lowering the risk of overfitting. Here we show that this problem can be effectively solved by representing linear transformations with matrix product operators (MPO). We have tested this approach in five main neural networks, including FC2, LeNet-5, VGG, ResNet, and DenseNet on two widely used datasets, namely MNIST and CIFAR-10, and found that this MPO representation indeed sets up a faithful and efficient mapping between input and output signals, which can keep or even improve the prediction accuracy with dramatically reduced number of parameters.

I. INTRODUCTION

Deep neural networks1–12 are important tools of artificial intelligence. Their applications in many computing tasks, for examples, in the famous ImageNet Large Scale Visual Recognition Challenge (ILSVRC)13, large vocabulary continuous speech recognition14, and natural language processing15, have achieved great success. They have become the most popular and dominant machine learning approaches6 that are used in almost all recognition and detection tasks4–17, including but not limited to, language translation18, sentiment analysis19, segmentation and reconstruction20, drug activity prediction21, feature identification in big data22, and have attracted increasing attentions from almost all natural science and engineering communities, including mathematics23,25, physics26–30, biology21,23, and material science31.

A deep feedforward network sets up a mapping between a set of input signals, such as images, and a set of output signals, say categories, through a multi-layer transformation, $F$, which is represented as a composition of many alternatively arranged linear ($L$) and nonlinear ($N$) mappings32,33. More specifically, an $n$-layer neural network $F$ is a sequential product of alternating linear and nonlinear transformations:

$$F = N_n L_n \cdots N_2 L_2 N_1 L_1$$

The linear mappings contain most of the variational parameters that need to be determined. The nonlinear mappings, which contain almost no free parameters, are realized by some operations known as activations, including Rectified Linear Unit (ReLU), softmax, and so on.

A linear layer maps an input vector $x$ of dimension $N_x$ to an output vector $y$ of dimension $N_y$ via a linear transformation characterized by a weight matrix $W$

$$y = Wx + b.$$  

A fully-connected layer plays the role as a global linear transformation, in which each output element is a weighted summation of all input elements, and $W$ is a full matrix. A convolutional layer34 represents a local linear transformation, in the sense that each element in the output is a weighted summation of a small portion of the elements, which form a local cluster, in the input. The variational weights of this local cluster form a dense convolutional kernel, which is designated to extract some specific features. To maintain good performance, different kernels are used to extract different features. A graphical representation of $W$ is shown in Fig. 1(a).

Usually, the number of elements or neurons, $N_x$ and $N_y$, are very large, and thus there are a huge number of parameters to be determined in a fully-connected layer34–36. The convolutional layer reduces the variational parameters by grouping the input elements into many partially overlapped kernels, and one output element is connected to one kernel. The number of variational parameters in a convolutional layer is determined by the number of kernels and the size of each kernel. It could be much less than that in a fully-connected layer. However, the total number of parameters in all the convolutional layers can still be very large in a deep neural network which contains many convolutional layers11. To train and store these parameters raises a big challenge in this field. First, it is time consuming to train and optimize these parameters, and may even increase the probability of overfitting. This would limit the generalization power of deep neural networks. Second, it needs a big memory space to store these parameters. This would limit its applications where the space of hard disk is strongly confined, for example, on mobile terminals.

However, the linear transformations in a commonly used deep neural network have a number of features which may allow us to simplify their representations. In a fully-connected layer, for example, it is well known that the rank of the weight matrix is strongly restricted34–36, due to short-range correlations or entanglements among the input pixels. This suggests that we can safely use a lower-rank matrix to represent this layer without affecting its prediction power. In a convolutional layer, the correlations of imagines are embedded in the kernels, whose sizes are generally very small in comparison with the whole image size. This implies that the “extracted features” from this convolution can be obtained from very local clusters. In both cases, a dense weight matrix is not absolutely
Here, the one-dimensional coordinate $x$ of the input signal $x$ with dimension $N_x$ is reshaped into a coordinate in a $n$-dimensional space, labelled by $(i_1i_2\cdots i_n)$. Hence, there is a one-to-one mapping between $x$ and $(i_1i_2\cdots i_n)$. Similarly, the one-dimensional coordinate $y$ of the output signal $y$ with dimension $N_y$ is also reshaped into a coordinate in a $n$-dimensional space, and there is a one-to-one correspondence between $y$ and $(j_1j_2\cdots j_n)$. If $I_k$ and $J_k$ are the dimensions of $i_k$ and $j_k$, respectively, then

$$
\prod_{k=1}^{n} I_k = N_x, \quad \prod_{k=1}^{n} J_k = N_y.
$$

The index decomposition in Eq. (3) is not unique. One should in principle decompose the input and output vectors such that the test accuracy is the highest. However, to test all possible decompositions is time consuming. For the results presented in this work, we have done the decomposition just by convenience.

The MPO representation of $W$ is obtained by factorizing it into a product of $n$ local tensors

$$
W_{j_k\cdots j_2j_1i_1\cdots i_n} = \text{Tr} \left( w^{(1)}[j_1,i_1]w^{(2)}[j_2,i_2]\cdots w^{(n)}[j_n,i_n] \right)
$$

where $w^{(k)}[j_k,i_k]$ is a $D_{k-1} \times D_k$ matrix with $D_k$ the virtual basis dimension on the bond linking $w^{(k)}$ and $w^{(k+1)}$ with $D_0 = D_n = 1$. For convenience in the discussion below, we assume $D_k = D$ for all $k$ except $k = 0$ or $n$. A graphical representation of this MPO is shown in Fig. 1(b).

In this MPO representation, the tensor elements of $w^{(k)}$ are variational parameters. The number of parameters increases with the increase of the virtual bond dimension $D$. Hence $D$ serves as a tunable parameter that controls the expressive power.

### II. RESULTS

Here we show the results obtained with the MPO representation in five kinds of neural networks on two datasets, i.e., FC2 and LeNet-5 on the MNIST dataset, VGG, ResNet, and DenseNet on the CIFAR-10 dataset. Among them, FC2 and LeNet-5 are relatively shallow in the depth of network. VGG, Residual CNN (ResNet), and Dense CNN (DenseNet) are deeper neural networks.

For convenience, we use MPO-Net to represent a deep neural network with all or partial linear layers being represented by MPO. Moreover, we denote an MPO, defined by Eq. (5), as

$$
M^{I_{j_1,j_2,\cdots,j_n}}(D).
$$

To quantify the compressibility of MPO-net with respect to a neural network, we define its compression ratio $\rho$ as

$$
\rho = \frac{\sum_i N^{(i)}_{\text{ori}}}{\sum_i N^{(i)}_{\text{mpo}}}
$$

where $\sum_i$ is to sum over the linear layers whose transformation tensors are replaced by MPO. $N^{(i)}_{\text{ori}}$ and $N^{(i)}_{\text{mpo}}$ are the number of parameters in the $l$-th layer in the original and MPO representations, respectively. The smaller is the compression ratio, the more the compression. In this work, we have done the decomposition just by convenience.

The MPO representation of $W$ is obtained by factorizing it into a product of $n$ local tensors

$$
W_{j_k\cdots j_2j_1i_1\cdots i_n} = \text{Tr} \left( w^{(1)}[j_1,i_1]w^{(2)}[j_2,i_2]\cdots w^{(n)}[j_n,i_n] \right)
$$

where $w^{(k)}[j_k,i_k]$ is a $D_{k-1} \times D_k$ matrix with $D_k$ the virtual basis dimension on the bond linking $w^{(k)}$ and $w^{(k+1)}$ with $D_0 = D_n = 1$. For convenience in the discussion below, we assume $D_k = D$ for all $k$ except $k = 0$ or $n$. A graphical representation of this MPO is shown in Fig. 1(b).

In this MPO representation, the tensor elements of $w^{(k)}$ are variational parameters. The number of parameters increases with the increase of the virtual bond dimension $D$. Hence $D$ serves as a tunable parameter that controls the expressive power.
fewer number of parameters is used in the MPO representation.

Furthermore, to examine the performance of a given neural
network, we train the network \( m \) times independently to obtain
a test accuracy \( a \) with a standard deviation \( \sigma \) defined by

\[
a = \bar{a} \pm \sigma,
\]

(8)

\[
\sigma = \frac{1}{\sqrt{m-1}} \left( \sum_{i=1}^{m} (a_i - \bar{a})^2 \right)^{1/2}
\]

(9)

where \( a_i \) is the test accuracy of the \( i \)-th training procedure. \( \bar{a} \)
is the average of \( a_i \). The results presented in this work are
obtained with \( m = 5 \).

A. MNIST dataset

We start from the identification of handwritten digits in the
MNIST dataset\(^8\), which consists of 60,000 digits for training and
10,000 digits for testing. Each image is a square of 28×28
gray-scale pixels, and all the images are divided into 10 classes
corresponding to numbers 0~9, respectively.

1. FC2

We first test the MPO representation in the simplest text-
book structure of neural network, i.e., FC2\(^6\). FC2 consists of
only two fully-connected layers whose weight matrices have
784 × 256 and 256 × 10 elements, respectively. We replace
these two weight matrices respectively by \( M_{4,4,4,4}^{1,4,1,4}(D) \)
and \( M_{4,4,4,4}^{3,4,3,4}(4) \) in the corresponding MPO representation. Here
we fix the bond dimension in the second layer to 4, and only
allow the bond dimension to vary in the first layer.

![Figure 2. Performance of the MPO representations in FC2 on MNIST.](image)

The solid straight line denotes the test accuracy obtained by the normal FC2, 98.35% ± 0.2%, and the dashed straight lines are plotted to indicate its error bar.

2. LeNet-5

We further test MPO-Net with the famous LeNet-5 network\(^2\), which is the first instance of convolutional neural networks. LeNet-5 has five linear layers. Among them, the last convolutional layer and the two fully-connected layers contain the most of parameters. We represent these three layers by three MPOs, which are structured as \( M_{2,10,10,2}^{2,5,6,2}(4) \), \( M_{2,5,6,2}^{2,3,7,2}(4) \) and \( M_{2,3,7,2}^{1,5,2,1}(2) \), respectively. The compression ratio is \( \rho = 0.2 \).

Table I shows the results obtained with the original and MPO representations of LeNet-5. We find that the test accuracy of LeNet-5 can be faithfully reproduced by MPO-Net. Since LeNet-5 is the first and prototypical convolutional neural network, this success gives us confidence in using the MPO presentation in deeper neural networks.

| Dataset   | Network     | Original Rep | MPO-Net     |
|-----------|-------------|--------------|-------------|
|           | \( a \) (%) | \( a \) (%)  | \( \rho \)  |
| MNIST     | LeNet-5     | 99.17 ± 0.04 | 99.17 ± 0.08 | 0.2 |
| CIFAR-10  | VGG-16      | 93.13 ± 0.39 | 93.76 ± 0.16 | -0.0006 |
|           | VGG-19      | 93.36 ± 0.26 | 93.80 ± 0.09 | -0.0006 |

Figure 2 compares the results obtained with FC2 and the corresponding MPO-Net. The test accuracy of MPO-Net increases when the bond dimension \( D \) is increased. It reaches the accuracy of the normal FC2 when \( D = 16 \). Even for the \( D = 2 \) MPO-Net, which has only 1024 parameters, about 200 times less than the original FC2, the test accuracy is already very good. This shows that the linear transformations in FC2 are very local and can indeed be effectively represented by MPO. The compression ratio of MPO-Net decreases with increasing \( D \). But even for \( D = 16 \), the compression ratio is still below 8%, which indicates that the number of parameters to be trained can be significantly reduced without any accuracy loss.

B. CIFAR-10 dataset

CIFAR-10 is a more complex dataset\(^8\). It consists of 50,000 images for training and 10,000 images for testing. Each image is a square of 32×32 RGB pixels. All the images in this dataset are divided into 10 classes corresponding to airplane, automobile, ship, truck, bird, cat, deer, dog, frog, and horse, respectively. To have a good classification accuracy, deeper neural networks with many convolutional layers are used. In order to show the effectiveness of MPO representation, as a preliminary test, we use MPO only on the fully-connected layers and on some heavily parameter-consuming convolutional layers.
FIG. 3. Comparison of the test accuracy $a$ between the original and MPO representations of ResNet on CIFAR-10 with $k = 4$. The compression ratio of MPO-Net $\rho \sim 0.11$.

1. VGG

VGG\cite{10} is the first very deep neural network contructed. It won the first place in the localization task of the ILSVRC competition 2014. We have tested two well-established VGG structures, which have 16 and 19 layers, respectively. In both cases, there are many convolutional layers and three fully-connected layers. We represent the last two heaviest convolutional layers and all the three fully-connected layers respectively by MPO with the structures: $M_{2,8,8,2}^{4}(4), M_{2,8,8,2}^{5}(4), M_{4,4,4,2}^{4}(4), M_{4,4,4,2}^{5}(4), M_{4,4,4}^{4}(4), M_{4,4,4}^{8}(4), M_{4,4,4}^{8}(4)$, and $M_{4,4,4}^{10,1,1} (4)$. The result is summarized in Tab. I.

For both structures, the compression ratio of MPO-Net is about 0.0006. Hence the number of parameters used is much less than in the original representation. However, we find that the prediction accuracy of MPO-Net is even better than those obtained from the original networks. This is consistent with the results reported by Novikov\cite{43} for the ImageNet dataset\cite{44}. It results from two facts of MPO: First, since the number of variational parameters is greatly reduced in MPO-Net, the representation is more economical and it is easier to train the parameters. Second, the local correlations between input and output elements are more accurately represented by MPO. This can reduce the probability of overfitting.

2. ResNet

ResNet\cite{11} is commonly used to address the degradation problem with deep convolutional neural networks. It won the first place on the detection task in ILSVRC in 2015, and differs from the ordinary convolutional neural network by the so-called ResUnit structure, in which identity mappings are added to connect some of the input and output signals. The ResNet structure used in our calculation has a fully-connected layer realized by a weight matrix of $64k \times 10$. Here $k$ controls the width of the network. We represent this layer by an MPO of $M_{4,4,4}^{1,5,2,1}(3)$. In our calculation, $k = 4$ is use and the corresponding compression ratio is about 0.11.

Figure 3 shows the test accuracy as a function of the depth of layers of ResNet with $k = 4$. We find that MPO-Net has the same accuracy as the normal ResNet for all the cases we have studied. We also find that even the ResUnit can be compressed by MPO. For example, for the 56-layer ResNet, by representing the last heaviest ResUnit and the fully-connected layer with two $M_{2,4,4,4,4,4,4,2}^{2}(4)$ and one $M_{4,4,4}^{1,5,2,1}(3)$, we obtain the same accuracy as the normal ResNet. Similar observations are obtained for other $k$ values.

3. DenseNet

The last deep neural network we have tested is DenseNet\cite{12}. Constructed in the framework of ResNet, DenseNet modifies ResUnit to DenseUnit by adding more shortcuts in the units. This forms a wider neural network, allowing the extracted information to be more efficiently recycled. It also achieved great success in the ILSVRC competition, and drew much attention in the CVPR conference in 2017.

The DenseNet used in this work has a fully-connected layer with a weight matrix of $(n + 3km) \times 10$, where $m$ controls the total depth $L$ of the network, $L = 3m + 4$, and $k$ are the other two parameters that specify the network. Although there is only one fully-connected layer in DenseNet, it consumes about half of the total parameters in the network. We use MPO to reduce the parameter number in this layer. Corresponding to different $m, k,$ and $n$, we use different MPO representations.

Our results are summarized in Table II. For the four DenseNet structures we have studied, the fully-connected layer is compressed by more than 7 to 21 times. The corresponding compression ratios vary from 0.044 to 0.129. In the first three cases, we find that the test precisions obtained with MPO-Net agree with the DenseNet results within numerical errors. For the fourth case, the test accuracy obtained with MPO-Net is even slightly higher than that obtained with DenseNet.

III. DISCUSSION

Motivated by the success of MPO in the study of quantum many-body systems with short-range interactions, we propose to use MPO to represent linear transformation matrices in deep neural networks. This is based on the assumption that the correlations between pixels, or the inherent structures of information hidden in “images”, are essentially localized\cite{50, 51}. We have tested our approach with five different kinds of main neural networks on two datasets, and found that MPO can not only improve the efficiency in training and reduce the memory space, as originally expected, but also slightly improve the test accuracy using much fewer number of parameters than in the original networks. This, as already mentioned, may result from the fact that the variational parameters can be more accurately and efficiently trained due to the dramatic reduction of
TABLE II. Performance of MPO representations in DenseNet on CIFAR-10.

| Depth (n, m, k) | Test accuracy (%) | MPO structure | ρ   |
|----------------|-------------------|----------------|-----|
| (16, 12, 12)   | 93.56 ± 0.26      | M^{5,2,1}_{4,4,4}(4) | 0.129 |
| (16, 12, 24)   | 95.12 ± 0.15      | M^{5,2,1}_{5,5,11,11}(4) | 0.089 |
| (24, 32, 12)   | 95.36 ± 0.15      | M^{5,2,1}_{5,5,26}(4) | 0.070 |
| (96, 32, 24)   | 95.74 ± 0.09      | M^{5,2,1}_{5,8,12,5}(4) | 0.044 |

parameters in MPO-Net. The MPO representation emphasizes more on the local correlations of input signals. It puts a strong constraint to the linear transformation matrix and avoids the training data being trapped at certain local minima. We believe this can reduce the risk of overfitting.

MPO can be used to represent both fully-connected and convolutional layers. One can also use it just to represent the kernels in convolutional layers, as suggested by Garipov. However, it is more efficient in representing a fully-connected layer where the weight matrix is a fully dense matrix. This representation can greatly reduce the memory cost and shorten the training time in a deep neural network where all or most of the linear layers are fully-connected ones, such as in a recurrent neural network, which is used to dispose video data.

Tensor-network representation of deep neural network is actually not new. Inspired by the locality assumption about the correlations between pixels, matrix product representation has been already successfully used to characterize and compress images, and to determine the underlying generative models. Novikov et al. also used MPO to represent some fully-connected layers, not including the classifiers, in FC2 and VGG. Our work, however, demonstrates that all fully-connected layers, including the classifiers especially, as well as convolutional layers, can be effectively represented by MPO no matter how deep a neural network is.

There are also other mathematical structures that have been used to represent deep neural networks. For example, Kossshalki et al. used a Tucker-structure representation, which is a lower-rank approximation of a high-dimensional tensor, to represent a fully-connected layer and its input feature. Hallam et al. used a tensor network called multi-scale entangled renormalization ansatz and Liu et al. used an unitary tree tensor network to represent the entire map from the input to the output labels.

In this work, we have proposed to use MPO to compress the transformation matrices in deep neural networks. Similar ideas can be used to compress complex datasets, for example the dataset called ImageNet, in which each image contains about 224 × 224 pixels. In this case, it is matrix product states instead of MPO, that should be used. We believe this can reduce the cost in decoding each “image” in a dataset, and by combining with the MPO representation of the linear transformation matrices, can further compress deep neural networks and enhance its prediction power.

IV. METHODS

The tensor elements of $w^{(i)}$ in Eq. (5) are the variational parameters that need to be determined in the training procedure of deep neural networks. For an MPO whose structure is defined by Eq. (6), The total number of these variational parameters equals

$$N_{m_{mpo}} = \sum_{k=2}^{n-1} I_k J_k D_k^2 + I_1 J_1 D + I_n J_n D.$$  (10)

The strategy of training is to find a set of optimal $w$’s so that the following cost function is minimized

$$L = -\sum_i \frac{1}{m} \sum \log y_m + \frac{\alpha}{2} \sum_i |w^{(i)}|^2,$$  (11)

where $n$ is the label of images, $i$ is the label of all the parameters, including the local tensors in the MPO representations and the kernels in the untouched convolutional layers. $|w|$ represents the norm of parameter $w$, and $\alpha$ is an empirical parameter that is fixed prior to the training. The first term measures the cross entropy between prediction vectors $y$ and target label vectors $t$. The second term is a constraint, called the L2 regularization, added to alleviate overfitting.

To implement a training step, $L$ is evaluated using the known $w$’s, which are randomly initialized, and input dataset. The gradients of the cost function with respect to the variational parameters are determined by the standard back propagation. Parameters $w$’s are updated by the stochastic gradient descent with momentum algorithm. This training step is terminated when the cost function stops to drop.

To implement a training step, $L$ is evaluated using the known $w$’s, which are randomly initialized, and input dataset. The gradients of the cost function with respect to the variational parameters are determined by the standard back propagation. Parameters $w$’s are updated by the stochastic gradient descent with momentum algorithm. This training step is terminated when the cost function stops to drop.

The detailed structures of the neural networks we have studied are introduced in the supplemental material. The source code used in this work is available at https://github.com/zfgao66/deeplearning-mpo.

V. ACKNOWLEDGMENT

We thank Pan Zhang and Lei Wang for valuable discussions. This work was supported by the National Natural Science Foundation of China (Grants No. 11774420, 11774422, 11874421, 11888101), and by the National R&D Program of China (Grants No. 2016YFA0300503, 2017YFA0302900).

Author Contributions — Ze-Feng Gao, with the help of Song Cheng, carried out the numerical calculations. Hui-Hai
Zhao and Z. Y. Xie set up the framework of coding. Z. Y. Xie, Hui-Hai Zhao, Zhong-Yi Lu and Tao Xiang conceived the ideas and developed the algorithms. Z. Y. Xie, Hui-Hai Zhao, Rong-Qiang He, Zhong-Yi Lu and Tao Xiang analyzed the results and drafted the paper.

Competing Interests — The authors declare that they have no competing financial interests.
A. Novikov, D. Podoprikhin, A. Osokin, D. Vetrov, Tensorizing Neural Networks, NIPS (2015).

J. Eisert, M. Cramer, M. B. Plenio, Colloquium: Area laws for the entanglement entropy, Rev. Mod. Phys., 82, 277 (2010)

D. Poulin, A. Qarry, R. Somma, F. Verstraete, Quantum simulation of time-Dependent Hamiltonians and the convenient illusion of Hilbert space, Phys. Rev. Lett. 106, 170501 (2011)

See, e.g., http://www.tensorfly.cn/tfdoc/tutorials/mnist_beginners.html

The official website of MNIST is available at http://yann.lecun.com/exdb/mnist.

The official website of CIFAR is available at https://www.cs.toronto.edu/~kriz/cifar.html.

The official website of imagNet is available at http://www.image-net.org

J. I. Latorre, Image compression and entanglement, arXiv:quant-ph/0510031.

Some numerical analysis on the MNIST dataset based on the entropy calculation has been devoted to topic. E.g., see Song Cheng, Jing Chen, and Lei Wang, Information perspective to probabilistic modeling: Boltzmann machines versus Born machines, Entropy, 20, 583 (2018)

T. Garipov, D. Podoprikhin, A. Novikov, D. Vetrov, Ultimate tensorization: compressing convolutional and fully-connected layers alike, arXiv:1611.03214.

S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9, 1735 (1997)

Y. Yang, D. Krompass, V. Tresp, Tensor-Train recurrent neural networks for video classification, ICML (2017)

Z. Y. Han, J. Wang, H. Fan, L. Wang, P. Zhang, Unsupervised generative modeling using matrix product states, Phys. Rev. X 8, 031012 (2018)

J. Kossaifi, Z. Lipton, A. Khanna, T. Furlanello, A. Anandkumar, Tensor regression networks, arXiv:1707.08308.

A. Hallam, E. Grant, V. Stojnic, S. Severini, A. G. Green, Compact neural networks based on the multiscale entanglement renormalization ansatz, arXiv:1711.03357

G. Vidal, Entanglement renormalization, Phys. Rev. Lett., 99, 220405 (2007)

D. Liu, S. J. Ran, P. Wittek, C. Peng, R. B. Garcia, G. Su, and M. Lewenstein, Machine learning by two-dimensional hierarchical tensor networks: A quantum information theoretic perspective on deep architectures, arXiv:1710.04833

Y. Shi, L. M. Duan, and G. Vidal, Classical simulation of quantum many-body systems with a tree tensor network, Phys. Rev. A. 74, 022320 (2006)

M. Fannes, B. Nachtergaele, Finitely correlated states on quantum spin chains, Comm. Math. Phys., 144, 443 (1992)

I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, Cambridge (2016)

D. E. Rumelhart, G. E. Hinton, R. J. Williams, Nature, Learning representations by back-propagating errors, Nature, 323, 533 (1986)

K. P. Murphy, Machine learning: A probabilistic perspective, Cambridge (2012)
Supplementary Materials:
Structure Details of the Neural Networks

Ze-Feng Gao,1 Song Cheng,2, 3 Rong-Qiang He,1 Z. Y. Xie,1,* Hui-Hai Zhao,4, 1 Zhong-Yi Lu,1, 3 and Tao Xiang2, 3, §

1Department of Physics, Renmin University of China, Beijing 100872, China
2Institute of Physics, Chinese Academy of Sciences, Beijing 100190, China
3University of Chinese Academy of Sciences, Beijing, 100049, China
4RIKEN Brain Science Institute, Hirosawa, Wako-shi, Saitama, 351-0106, Japan
In this supplementary material, we give the detailed structure of the neural networks used in the paper titled as *Compressing deep neural networks by matrix product operators*. The corresponding source code used in this work is available at [https://github.com/zfgao66/deeplearning-mpo](https://github.com/zfgao66/deeplearning-mpo). The used structures of FC2, LeNet5, VGG, ResNet, DenseNet in this paper are summarized in Tab. II-VII, and their prototypes can be found in Ref. 1–5, respectively. In order to simplify the descriptions, we introduce some short-hands summarized in Tab. I which are used in this materials.

| Abbreviation | Meaning |
|--------------|---------|
| MaxPo        | a max-pooling layer |
| AvgPo        | an average-pooling layer |
| Conv         | a convolutional layer |
| ConvUnit     | a unit composed of convolutional layers |
| ResUnit      | a unit introduced in ResNet |
| ResBlock     | a block composed of several ResUnits |
| DenseUnit    | a unit introduced in DenseNet |
| BN           | batch normalization |
| \([w, h; c; s]\) | a convolutional layer with \(c\) kernels each with width \(w\) height \(h\) and stride \(s\) |
| \([w, h; s]\) | a pooling layer with pooling width \(w\) height \(h\) and stride \(s\) |
| \([w, h; c; s, t]\) | a ResUnit composed of two convolutional layers denoted as width \([w, h; c; s]\) and \([w, h; c; t]\) resp. |
| \(N_{\text{para}}\) | number of parameters in the linear layers |
| Represented  | whether this block of layers are represented by MPO in the preliminary test |

**TABLE I.** The short-hands used in this materials.

| No. | Layer name | Input size | Output size | Comment | \(N_{\text{para}}\) | Represented |
|-----|------------|------------|-------------|---------|-------------------|-------------|
| 1   | FC         | 28×28      | 256         |         | 200704            | Yes         |
|     | ReLu       |        | |         | | |
| 2   | FC         | 256       | 10          |         | 2560              | Yes         |
|     | Softmax    |        | |         | | |

**TABLE II.** The FC2 network structure used in this work. ReLu and Softmax are element-wise operations, whose details are not shown here and after.

| No. | Layer name | Input size | Output size | Comment | \(N_{\text{para}}\) | Represented |
|-----|------------|------------|-------------|---------|-------------------|-------------|
| 1   | Conv       | 28×28      | 28×28×6     | \([5,5,6; 1]\) | 150               |             |
|     | ReLu       |        | |         | | |
|     | MaxPo      | 28×28×6   | 14×14×6     | \([2,2; 2]\) | | |
| 2   | Conv       | 14×14×6   | 10×10×16    | \([5,5; 16; 1]\) | 400              |             |
|     | ReLu       |        | |         | | |
|     | MaxPo      | 10×10×16  | 5×5×16      | \([2,2; 2]\) | | |
| 3   | Conv       | 5×5×16    | 120         | \([5,5; 120; 1]\) | 3000            | Yes         |
|     | ReLu       |        | |         | | |
| 4   | FC         | 120       | 84          |         | 10080            | Yes         |
|     | ReLu       |        | |         | | |
| 5   | FC         | 84        | 10          |         | 840              | Yes         |
|     | Softmax    |        | |         | | |

**TABLE III.** The LeNet5 network structure used in this work. Here the subscript \(np\) is used to emphasize that no padding is used in convolutions there.
| No. | Layer name | Input size | Output size | Comment | $N_{para}$ | Represented |
|-----|------------|------------|-------------|---------|------------|-------------|
| 1   | ConvUnit   | 32×32×3    | 32×32×64    |         | 2×[3,3;64;1] | 1152        |
|     |            |            |             | ReLu    |            |             |
| 2   | ConvUnit   | 16×16×64   | 16×16×128   |         | 2×[3,3;128;1] | 2304        |
|     |            |            |             | ReLu    |            |             |
| 3   | ConvUnit   | 8×8×128    | 8×8×256     |         | 2×[3,3;256;1] | 4608        |
|     |            |            |             | ReLu    |            |             |
| 4   | Conv       | 8×8×256    | 8×8×256     | [1,1;256;1] | 256        |
|     |            |            |             | ReLu    |            |             |
| 5   | ConvUnit   | 4×4×256    | 4×4×512     |         | 2×[3,3;512;1] | 9216        |
|     |            |            |             | ReLu    |            |             |
| 6   | Conv       | 4×4×512    | 4×4×512     | [1,1;512;1] | 512        |
|     |            |            |             | ReLu    |            |             |
| 7   | ConvUnit   | 2×2×512    | 2×2×512     | [2,2;2]  |            |
|     |            |            |             | ReLu    |            |             |
| 8   | Conv       | 2×2×512    | 2×2×512     | [1,1;512;1] | 512        |
|     |            |            |             | ReLu    |            |             |
| 9   | FC         | 512        | 4096        |         |             |             |
|     |            |            |             | ReLu    |            |             |
| 10  | FC         | 4096       | 4096        |         |             |             |
|     |            |            |             | ReLu    |            |             |
| 11  | FC         | 4096       | 10          |         |             |             |

TABLE IV. The VGG-16 network structure used in this work. Here a ConvUnit denoted with $m \times$ means $m$ convolutional layers separated by ReLu.
| No. | Layer name   | Input size | Output size | Comment          | $N_{para}$ | Represented |
|-----|--------------|------------|-------------|------------------|-----------|-------------|
| 1   | ConvUnit     | 32×32×3    | 32×32×64    | 2×[3,3;64;1]     | 1152      |             |
|     |              |            |            | ReLu             |           |             |
|     | MaxPo        | 32×32×64   | 16×16×64   | [2,2;2]          |           |             |
| 2   | ConvUnit     | 16×16×64   | 16×16×128  | 2×[3,3;128;1]    | 2304      |             |
|     |              |            |            | ReLu             |           |             |
|     | MaxPo        | 16×16×128  | 8×8×128    | [2,2;2]          |           |             |
| 3   | ConvUnit     | 8×8×128    | 8×8×256    | 4×[3,3;256;1]    | 9216      |             |
|     |              |            |            | ReLu             |           |             |
|     | MaxPo        | 8×8×256    | 4×4×256    | [2,2;2]          |           |             |
| 4   | ConvUnit     | 4×4×256    | 4×4×512    | 4×[3,3;512;1]    | 18432     |             |
|     |              |            |            | ReLu             |           |             |
|     | MaxPo        | 4×4×512    | 2×2×512    | [2,2;2]          |           |             |
| 5   | ConvUnit     | 2×2×512    | 512        |                  |           |             |
|     |              |            |            | ReLu             |           |             |
|     | MaxPo        | 2×2×512    | 512        | [2,2;2]          |           |             |
| 6   | FC           | 512        | 4096       |                  | 2097152   | Yes         |
|     |              |            |            | ReLu             |           |             |
| 7   | FC           | 4096       | 4096       |                  | 16777216  | Yes         |
|     |              |            |            | ReLu             |           |             |
| 8   | FC           | 4096       | 10         |                  | 40960     | Yes         |
|     |              |            |            | Softmax          |           |             |

**TABLE V. The VGG-19 network structure used in this work. Here a ConvUnit denoted with $m\times$ means $m$ convolutional layers separated by ReLu. By partially we mean the last two convolutional layers in that ConvUnit was represented by MPO.**

| No. | Layer name  | Input size | Output size | Comment          | $N_{para}$ | Represented |
|-----|-------------|------------|-------------|------------------|-----------|-------------|
| 1   | Conv        | 32×32×3    | 32×32×16    | [3,3;16;1]       | 144       |             |
|     |             |            |            | BN + ReLu        |           |             |
| 2   | ResBlock    | 32×32×16   | 32×32×16k   | $m\times[3,3;16k;1,1]$ | 288km     |             |
|     |             |            |            | BN + ReLu        |           |             |
| 3   | ResUnit     | 32×32×16k  | 16×16×32k   | [3,3;32k;2,1]    | 576k      |             |
|     |             |            |            | BN + ReLu        |           |             |
| 4   | ResBlock    | 16×16×32k  | 16×16×32k (m-1)×[3,3;32k;1,1] | 576k(m-1) |           |             |
|     |             |            |            | BN + ReLu        |           |             |
| 5   | ResUnit     | 16×16×32k  | 8×8×64k     | [3,3;64k;2,1]    | 1152k     |             |
|     |             |            |            | BN + ReLu        |           |             |
| 5   | ResBlock    | 8×8×64k    | 8×8×64k (m-1)×[3,3;64k;1,1] | 1152k(m-1) | Partially |             |
|     |             |            |            | BN + ReLu        |           |             |
|     | AvgPo       | 8×8×64k    | 64k         | [8,8;8]          |           |             |
| 6   | FC          | 64k        | 10          |                  | 640k      | Yes         |
|     |             |            |            | Softmax          |           |             |

**TABLE VI. The ResNet network structure used in this work. The total depth $L$ is given by $L = 6m + 2$. Here in a single ResUnit, the two convolutional layers are separated by batch normalization and ReLu, and a ResBlock denoted with $m\times$ means $m$ ResUnits separated by batch normalization and ReLu. By partially we mean the last ResUnit in that ResBlock is represented by MPO.**
| No. | Layer name      | Input size          | Output size          | Comment                  | \( N_{\text{para}} \) | Represented |
|-----|-----------------|---------------------|----------------------|--------------------------|------------------------|-------------|
| 1   | Conv            | 32\times 32\times 3 | 32\times 32\times n  | [3,3;n;1]                | 9n                     |             |
|     |                 |                     |                      |                          |                        |             |
| 2   | DenseUnit       | 32\times 32\times n | 32\times 32\times (n+k)m | m\times[3,3;k;1] | 9m                     |             |
|     |                 |                     |                      |                          |                        |             |
| 3   | Conv            | 32\times 32\times (n+k)m | 32\times 32\times (n+k)m | [1,1;n+k;1]              | n+k                    |             |
|     |                 |                     |                      |                          |                        |             |
|     | AvgPo           | 32\times 32\times (n+k)m | 16\times 16\times (n+k)m | [2,2;2]                  |                        |             |
|     |                 |                     |                      |                          |                        |             |
| 4   | DenseUnit       | 16\times 16\times (n+k)m | 16\times 16\times (n+2k)m | m\times[3,3;k;1] | 9km                    |             |
|     |                 |                     |                      |                          |                        |             |
| 5   | Conv            | 16\times 16\times (n+2k)m | 16\times 16\times (n+2k)m | [1,1;n+2k;1]             | n+2km                  |             |
|     |                 |                     |                      |                          |                        |             |
|     | AvgPo           | 16\times 16\times (n+2k)m | 8\times 8\times (n+2k)m | [2,2;2]                  |                        |             |
|     |                 |                     |                      |                          |                        |             |
| 6   | DenseUnit       | 8\times 8\times (n+3k)m | 8\times 8\times (n+3k)m | m\times[3,3;k;1] | 9km                    |             |
|     |                 |                     |                      |                          |                        |             |
| 7   | FC              | n+3km               | 10                   | [8,8]                    |                        | Yes         |

TABLE VII. The DenseNet network structure used in this work. The total depth \( L \) is given by \( L = 3m + 4 \). A DenseUnit denoted with \( m\times \) means there are \( m \) convolutional layers separated by batch normalization and ReLu in this unit.

---

\[ \text{\url{http://www.tensorfly.cn/tfdoc/tutorials/mnist_beginners.html}} \]

1. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, *Proc. of The IEEE*, 86, 2278 (1998)
2. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, *ICLR* (2015)
3. K. M. He, X. Y. Zhang, S. Q. Ren, J. Sun, Deep residual learning for image recognition, *CVPR* (2016)
4. G. Huang, Z. Liu, L. van der Maaten, K. Q. Weinberge, Densely Connected Convolutional Networks, *CVPR* (2017)