Variational Tensor Neural Networks for Deep Learning

Saeed S. Jahromi[1, 2, 3, *] and Román Orús[2, 3, 4]

[1] Department of Physics, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan 45137-66731, Iran
[2] Donostia International Physics Center, Paseo Manuel de Lardizabal 4, E-20018 San Sebastián, Spain
[3] Multiverse Computing, Paseo de Miramón 170, E-20014 San Sebastián, Spain
[4] Ikerbasque Foundation for Science, Maria Diaz de Haro 3, E-48013 Bilbao, Spain

Deep neural networks (NN) suffer from scaling issues when considering a large number of neurons, in turn limiting also the accessible number of layers. To overcome this, here we propose the integration of tensor networks (TN) into NNs, in combination with variational DMRG-like optimization. This results in a scalable tensor neural network (TNN) architecture that can be efficiently trained for a large number of neurons and layers. The variational algorithm relies on a local gradient-descent technique, with tensor gradients being computable either manually or by automatic differentiation, in turn allowing for hybrid TNN models combining dense and tensor layers. Our training algorithm provides insight into the entanglement structure of the tensorized trainable weights, as well as clarify the expressive power as a quantum neural state. We benchmark the accuracy and efficiency of our algorithm by designing TNN models for regression and classification on different datasets. In addition, we also discuss the expressive power of our algorithm based on the entanglement structure of the neural network.

I. INTRODUCTION.

Machine learning algorithms based on deep learning have proven very successful in tasks such as identification [1–3], classification [4–6], regression [7, 8], clustering [9–11], and many other artificial intelligence applications. In particular, deep learning algorithms based on neural networks (NN) are of high interest for their enhanced abilities in feature learning [12, 13] and decision-making [14, 15], both in supervised and unsupervised learning. In addition, NNs have also found practical applications in condensed matter and statistical physics [16–18]. As an example, it has been shown that NNs have expressive power for representing complex many-body wave functions [19–23]. Moreover, ideas for detecting phase transitions in quantum many-body systems with fully connected NNs and convolutional neural networks (CNN) have also been successfully put forward [24–27].

Recently, the connection between tensor networks (TN), which are efficient ansatz for representing quantum many-body wave functions [28, 29], and neural networks, has also been established. It has been shown that the trainable weights of NNs are closely related to many-body wave functions, henceforth can be replaced by TNs and trained using variational optimization techniques [30, 31]. Efficient TN algorithms for classification [30], anomaly detection [32], and clustering [33] have been proposed. On top of their expressive power, TNs also provide efficient schemes for data compression based on tensor factorization [34–37]. As an example, the number of parameters in NN models can substantially be compressed by keeping only the most relevant degrees of freedom by discarding those that involve lower correlations. Tensor neural networks (TNN) [31] and tensor convolutional neural networks (TCNN) [38] are examples of NNs where the weight tensors of the hidden layers are replaced by tensor network structures using, e.g., the singular value decomposition (SVD). Recent studies actually confirm that, for such a reduced parameter space, TNNs have better performance and accuracy than standard NNs [31, 34].

In most of the current TNN developments, the tensorization takes place only at the level of hidden layers (trainable weights) [31, 36]. However, training of the model is generically performed by optimizing the contracted trainable weights of each layers based on standard optimization techniques such as gradient descent and automatic differentiation. Despite being efficient and accurate, these optimization schemes only target the global minimum of a loss function while being blind to the correlations and entanglement structure between the model parameters, in addition to being hard to scale. The behavior of a loss function monitors the training convergence in these approaches, and distinguishing local minima from actual global minima is in principle very difficult. Moreover, it is also difficult to infer meaningful information from such updated weight tensors.

In this paper we resolve these issues by fully integrating NNs with TNs. More specifically, we introduce an efficient TN layer in NN structures such that the trainable weights are represented by matrix product operators (MPO) [31]. The TN layer can replace the fully connected (`Dense`) hidden layers of a NN. The resulting TNN is scalable and can have any desired number of TN layers to form a truly deep NN. We further introduce an entanglement-aware variational DMRG-like training algorithm for the resulting TNN. In contrast to previously-developed TN machine learning algorithms, which are single-layer and goal-specific [30, 32], our TNN is an instance of a deep neural network that can be used for different data analysis tasks such as regression and classification. The DMRG-like training algorithm provides

* saeed.jahromi@iasbs.ac.ir

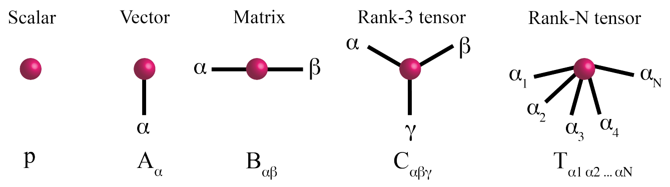

FIG. 1. [Color online] Diagrammatic representation of tensors. In this notation, a rank-n tensor is represented by a shape with n lines such that scalers, vectors, matrices, and rank-n tensors will be shapes with respectively, zero, one, two, and n lines.

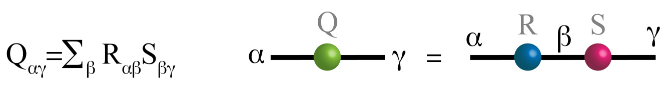
$Q_{\alpha\gamma}=\sum_\beta R_{\alpha\beta}S_{\beta\gamma}$
Q
R
S
=

FIG. 2. [Color online] Contraction of tensors, equivalent to tensor trace over their shared indices. This is represented graphically by connecting the shared links. Here, R and S tensors are connected along the shared leg β. This contraction operation is equivalent to matrix multiplication.

direct access to the entanglement spectrum of the MPO trainable weights, in turn giving rise to a clear insight on the correlations in the parameters of the machine learning model. Moreover, the entanglement structure of the MPOs and their expressive power as a quantum neural state can also be assessed by typical quantum information quantities such as, e.g., the entanglement entropy of a bipartition.

In order to demonstrate the efficiency and accuracy of our technique, we use our TN layer combined with different loss and activation functions to design different deep learning models. We use such algorithms for, e.g., linear and non-linear regression as well as classification. Next, we use our DMRG-like algorithm [39, 40] to train the models. Our findings show that the TNN is remarkably accurate and efficient and can be used as a reliable NN algorithm for different data analysis tasks with a full general purpose. We also show how the non-linearity in the correlations amongst input data reflects itself in the entanglement spectrum and entanglement entropy of the MPO trainable weights. Our findings suggest that the DMRG-like training algorithm is an efficient numerical tool for providing the TN representation of quantum neural states.

The paper is organized as follows: In Sec. II we provide a brief overview of tensor networks and how a standard neural network composed of fully connected `Dense` layers can be tensorized to obtain a TNN. Next in Sec. III we introduce our DMRG-like training algorithm, together with the details of local tensor optimization updates based on gradient descent. In Sec. V we build TNN models for regression and use them for fitting linear- and non-linear random data. More examples of applications of TNNs for classification of labeled data are provided in Sec. IV. Finally, in Sec. VI we wrap up with our conclusions and discussions about further possible work.

II. TENSOR NEURAL NETWORKS

Tensor networks [29, 41] where originally developed in physics with the aim of providing efficient representations for quantum many-body wave functions. They are also the basis of well-established numerical techniques such as density matrix renormalization group (DMRG) [39, 40] and time-evolving block decimation (TEBD) [42, 43]. However, due to its high potential for efficient data representation and compression, novel applications of TNs are emerging in different branches of data science such as machine learning and optimization. In this section we show how TNs can enhance deep neural networks by providing an efficient representation of the trainable weights of classical neural networks. To this end, we first review some basic concepts on TNs in order to establish basic notation and concepts widely used in the physics' context.

A. Tensor Network Basics

A tensor is a multi-dimensional array of complex numbers represented by $T_{\alpha\beta\gamma\dots}$ in which the subscripts denote the tensor dimensions. The number of tensor dimensions further corresponds to the rank of the tensor. Tensors and tensor operations can alternatively be described by using tensor network diagrams [28] as illustrated in Fig. 1. Within this graphical representation, a rank-n tensor is a shape with n connected links (legs) so that a scalar, a vector, and a matrix are objects with zero, one, and two connected links, respectively. This diagrammatic notation is then generalized to rank-n tensors to shapes with n legs, each corresponding to a tensor index.

TN diagrams not only represent the tensors but also represent tensor contractions, which is the generalization of matrix multiplication to rank-n tensors. For example, contraction of two rank-2 tensors, i.e., matrices $R_{\alpha\beta}$ and $S_{\beta\gamma}$ can be represented diagrammatically by connecting the two tensors along their shared link β, as shown in Fig. 2. This operation can further be represented mathematically as

$$Q_{\alpha\gamma} = \mathrm{tTR}(R_{\alpha\beta}S_{\beta\gamma}) = \sum_\beta R_{\alpha\beta}S_{\beta\gamma}, \qquad (1)$$

where the tTR is the tensor trace over shared indices (tensor legs).

Large matrices and multi-dimensional arrays with a massive number of parameters can further be represented efficiently by tensor networks. Fig. 3 shows examples of two well-known tensor networks, namely, matrix product states (MPS) (also known as a tensor trains) and matrix product operators (MPO) [29], which are used for

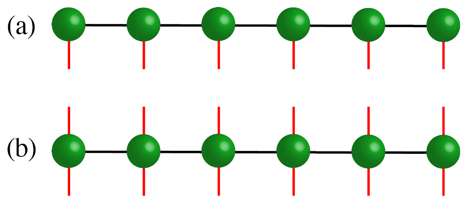

FIG. 3. [Color online] Examples of well-known one dimensional TNs. (a) Matrix product states (MPS) and (b) matrix product operators (MPO). The latter will be used for representing the trainable weights of `TNLayers`.

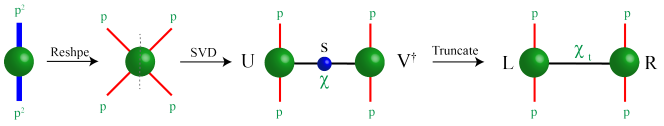

FIG. 4. [Color online] Singular value decomposition of a $p^2 \times p^2$ matrix M into the L and R tensors of a 2-site MPO, see text for details.

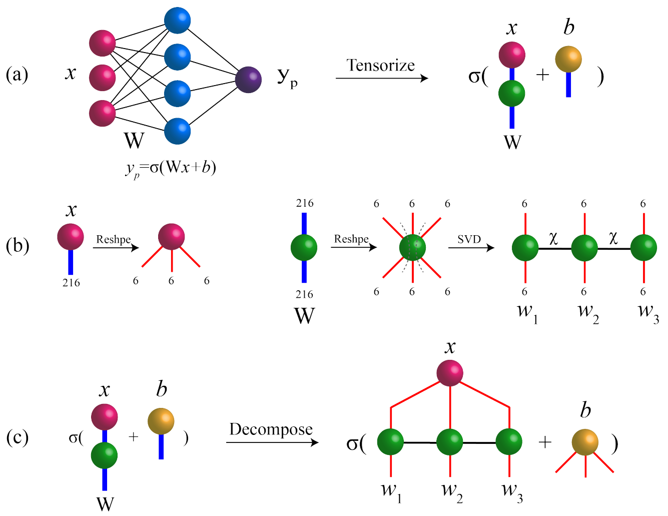

FIG. 5. [Color online] Tensorizing a NN with one fully connected dense layer: (a) TN representation of the NN; (b) MPO decomposition of the weight matrix W; (c) The resulting tensor neural network with MPO trainable weights.

representing one-dimensional quantum many-body wave functions and operators. In particular, one can provide efficient MPO representations of large matrices by reducing the number of parameters in a controlled way. More specifically, a matrix can be decomposed as an MPO by applying singular value decomposition (SVD) and truncating the negligible singular values. An example of MPO factorization for a matrix $M_{p^2\times p^2}$ is shown in Fig. 4. Reshaping M into a rank-4 tensor and applying an appropriate SVD, one can represent matrix M as a 2-site MPO with tensors $L_{pp\chi}$ and $R_{\chi pp}$:

$$M = USV^\dagger = LR, \qquad L = U\sqrt{S}, \qquad R = \sqrt{S}V^\dagger, \quad (2)$$

where U and $V^\dagger$ are unitary matrices reshaped into rank-3 tensors and S is a diagonal matrix of singular values. For the $p^2 \times p^2$ matrix M there exist at most $\chi = p^2$ singular values. The non-zero singular values quantify the amount of entanglement (correlation) between the L and R MPO tensors. In a weakly correlated system, most of the singular values are close to zero and can be discarded so that only the $\chi_t \leq \chi$ largest singular values are kept. One can, therefore, reduce the number of parameters along the so-called *virtual* tensor dimensions (black link) by keeping only the most relevant degrees of freedom for the correlations between L and R. In turn, this also implies that the interconnecting tensor dimensions which emerge in the decomposition capture the relevant degrees of freedom quantifying correlations in the tensor network.

B. Tensorizing Standard Neural Networks

In this subsection, we show how to integrate tensor networks into standard neural networks to build a TNN. Fig. 5-(a) shows the structure of a classic NN with one fully connected `Dense` layer of hidden units (neurons). The network prediction, y_p, is obtained by feeding the input feature vector x to the model as

$$y_p = \sigma(Wx + b), \quad (3)$$

where W is the weight matrix, b is some bias vector and σ is the activation function (e.g., `ReLu` or `Sigmoid`). Training the NN amounts to finding the optimum values for the parameters of the W weight matrix such that the y_p corresponds the actual label of the data.

Depending on the size of the problem, the weight matrices can be considerably large. This introduces computational bottlenecks to the NN model from the point of view of the required memory for storing such matrices, and also due to the large training time. The problem becomes more dramatic when dealing with deep NNs with many hidden layers. Optimizing the parameters of such huge weights will reduce the accuracy and efficiency of the model. It is therefore a necessity to reduce the number of model parameters, without sacrificing accuracy. To this end, we should resort to a controllable truncation scheme such that we only discard the least important information and keep the most relevant one. An example of such clever data compression schemes based on TN and MPO decomposition has already been introduced in the previous subsection. An efficient representation of the weight matrices can therefore be obtained by replacing W weights with MPOs [31, 34]. Figure 5 shows how the weight matrix of a fully connected `Dense` layer can be replaced by its MPO form obtained from consecutive applications of SVDs to the W matrix. The new tensorized layer, called `TNLayer`, now has several trainable weights w_i represented by MPOs. Reshaping the feature vector x to match the MPO dimensions, the network prediction y_p is obtained by contracting the resulting tensor net-

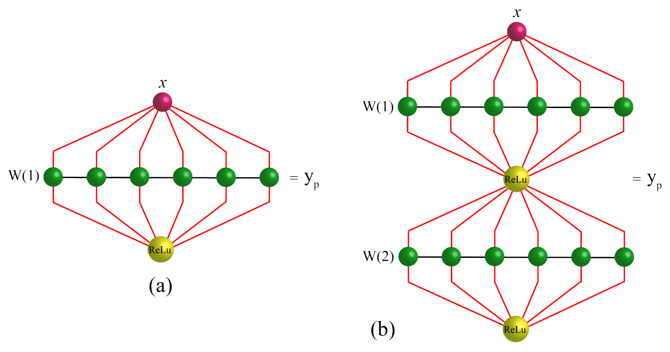

FIG. 6. [Color online] Examples of fully tensorized TNNs with (a) one `TNLayer` and (b) with two `TNLayers`.

work as shown in 5-(c). The resulting tensorized neural network is then called a TNN.

While the idea of a TNN is generic, direct tensorization can be applied to models whose feature vector x is factorable to integers, so that it could match the size and number dimensions of the MPO weights. An example of a valid-size feature vector x with 216 entries is shown in Fig. 5-(b). Such a vector can be reshaped into a rank-3 tensor with dimension size 6 that is can be contracted to a `TNLayer` with three MPO weights, each with input size 6. The features that are not factorizable to match the MPOs in the `TNLayer`, can rather be transformed in the preprocessing of the machine learning task so that their new size matches the input size of the `TNLayer`. Otherwise, one can introduce a dummy non-trainable `Dense` layer with size $N_F \times N_T$ in front of the `TNLayer` to compensate for the size mismatch. Here N_F is the size of the feature vector and N_T is the input size of the contracted `TNLayer`.

Let us further stress that applying the activation on the TN layer is a non-linear operation and, therefore, cannot be tensorized or be applied to each MPO tensor individually. We, therefore, have to first contract the MPOs along their virtual legs to obtain a single weight matrix and then apply the activation on this matrix to obtain the network prediction, i.e.,

$$y_p = \sigma(\text{tTR}[x_{ijk\ldots} w_i w_j w_k \ldots] + b_{ijk\ldots}), \qquad (4)$$

where the subscripts $ijk\ldots$ run over the dimensions of the feature tensor x. Nevertheless, we showed this operation with a single activation tensor as illustrated in Fig. 6 for examples of TNN with one and two `TNLayers`. In practice, one should first contract the features and MPOs, apply the activation function and then reshape the resulting tensor to match the inputs of the next layer. This process is continued until the whole network is contracted to obtain the prediction y_p.

The mandatory contraction of MPO weights enforced by the activation function can in principle be useful because one can connect the contracted `TNLayer` to `Dense` layers and design hybrid TNN models for different ML tasks. As exemplified in Secs. V and IV, we will use combinations of `TNLayer` and `Dense` layers to build different TNN models for regression and classification.

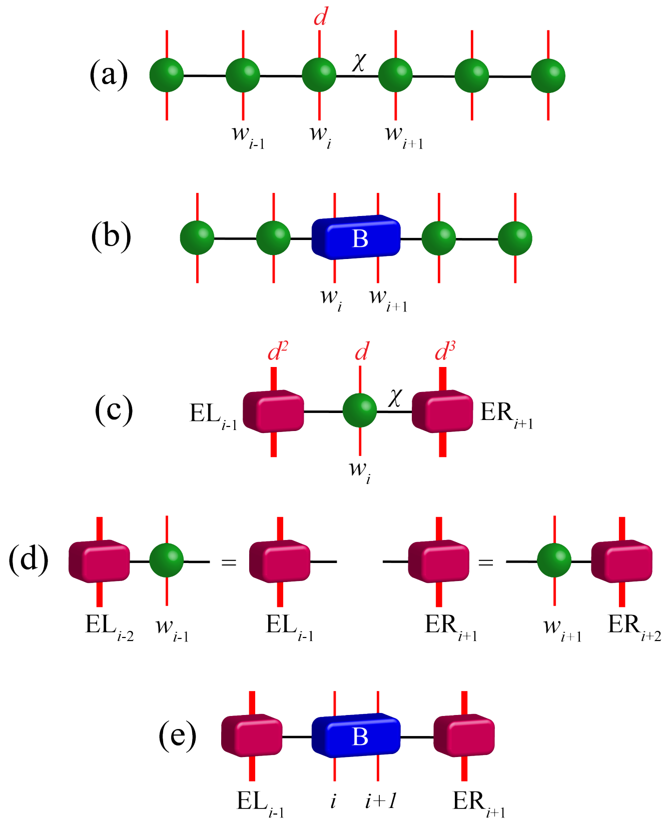

FIG. 7. [Color online] Defining bond and environment tensors: (a) MPO factorized weights; (b) Bond tensor $B_{i,i+1}$ (shown in blue) which is obtained from contracting a pair of neighboring MPO tensors, w_i, w_{i+1}; (c) Left and right environment tensors of each w_i MPO tensor; (d) A single step of obtaining environment tensors; (e) Alternative representation of MPO weights in terms of bond tensor $B_{i,i+1}$ and its surrounding environment.

III. DMRG-LIKE ALGORITHM FOR TNNS

In the previous section we introduced the tensor neural networks and how to build them from `TNLayers`. In this section, we provide details about the DMRG-like training algorithm for multi-layer TNN models. Similar to generic optimizers that can be found in any ML library such as `TensorFlow` [44] or `PyTorch` [45], our DMRG-like algorithm is also of generic purpose and can be used for training different models ranging from regression to classification.

A. Local gradient-descent update

The generic idea of training a neural network is to find the optimum values for the weight matrices of the NN

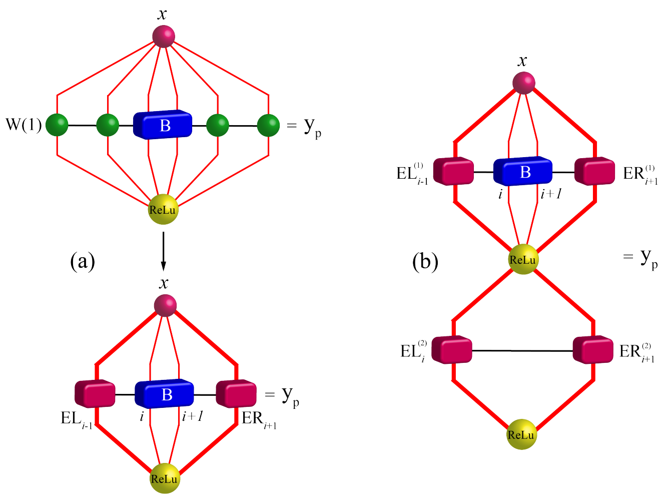

FIG. 8. [Color online] Alternative representation of a TNN in terms of bond tensor $B_{i,i+1}$ and environments for (a) a TNN with one `TNLayer` and (b) two `TNLayers`.

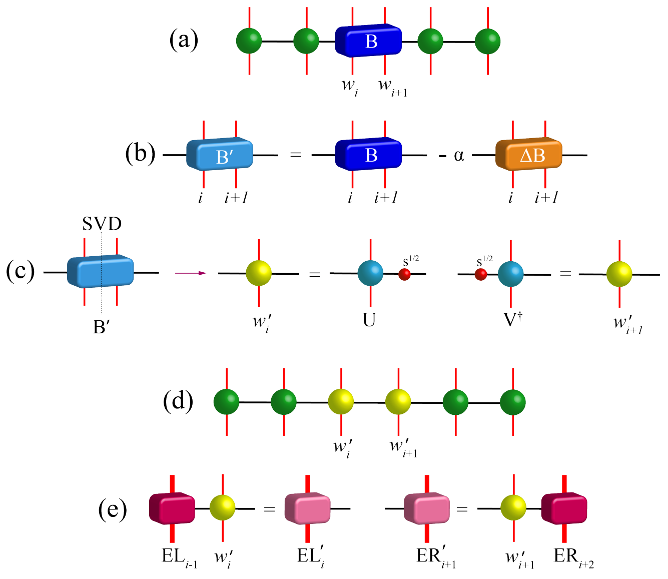

FIG. 9. [Color online] Graphical representation of the DMRG-like sweeping algorithm for training TNNs: (a) MPO weights with bond tensor $B_{i,i+1}$; (b) Updating the bond tensor with LGD step; (c) Calculating the updated local weights w'_i and w'_{i+1} from the updated bond tensor $B'_{i,i+1}$ by SVD and truncation; (e) Updating the left and right environments.

layers by minimizing a loss function. In a feed-forward NN, all the data or batches of it are fed to the network to calculate the y_p in the forward path. Next, the gradients of trainable weights with respect to the loss function are calculated in the backward path, and the weights are updated accordingly with a gradient-descent (GD) step. This whole process is iterated until a convergence criterion is met. Once trained, the model is tested to predict on unseen test data. The performance of the model is further evaluated by measuring different accuracy metrics.

In contrast to classic NNs with `Dense` layers, the TNN is composed of `TNLayers` that have multiple trainable MPO tensors. While a similar GD approach can be used for training the MPOs [31], here we use a DMRG-like technique to update the MPO tensors by pairs, with a sweeping local gradient-descent (LGD) algorithm. However, before we present the detail of the update, the following remarks are in order: i) Given a pair of neighboring MPO tensors, w_i, w_{i+1}, a *bond* tensor $B_{i,i+1}$ is obtained by contracting the w_i and w_{i+1} along their shared virtual dimension (see Fig. 7-(a,b)). ii) For every MPO tensor w_i, the contraction of all tensors at the right side and left side of w_i are called environment. These are denoted by EL_{i-1}, ER_{i+1} for the left and right environments, respectively (see Fig. 7-(c,d)). iii) For every bond tensor $B_{i,i+1}$, the `TNLayer` and further the overall TNN can be represented in terms of the bond tensor and its left and right environments, as shown in Fig. 7-(e) and Fig. 8. Given the fact that the TNN has to be contracted for every $B_{i,i+1}$ pair, which is required for updating the MPO weights, introducing the environment tensor can substantially reduce the computational cost of the network contraction.

Having the aforementioned remarks in mind, we train the TNN by sweeping over the MPO pairs of tensors of each `TNLayer` as follows:

1. Do for all MPO tensor pairs $\{w_i, w_{i+1}\}$ where $i \in [1, N_{\text{MPO}} - 1]$ (left to right sweep):
   (a) Build the local bond tensor $B_{i,i+1}$ by contracting w_i and w_{i+1}.
   (b) Calculate the gradient of $B_{i,i+1}$, i.e., $\Delta B_{i,i+1}$ with respect to a loss function (see Sec. III B, III C).
   (c) Update the bond tensor with learning rate α: $B'_{i,i+1} = B_{i,i+1} - \alpha \Delta B_{i,i+1}$.
   (d) Split the updated bond tensor by SVD: $B'_{i,i+1} = USV^{\dagger}$.
   (e) Truncate the matrices to keep the desired number of singular values: $B'_{i,i+1} \approx U'S'V'^{\dagger}$.
   (f) Update the weights: $w'_i = U'\sqrt{S'}$ and $w'_{i+1} = \sqrt{S'}V'^{\dagger}$.
   (g) Update the left and right environment tensors: $EL'_i = EL_{i-1}w'_i$ and $ER'_{i+1} = w'_{i+1}ER_{i+2}$.
2. Sweep back from right to left and repeat the same pair update.

Details about the LGD algorithm in terms of TN diagrams are also provided in Fig. 9.

B. Tensor gradient for linear activations

The DMRG-like sweeping algorithm for training the TNN is a gradient-based optimization approach in which

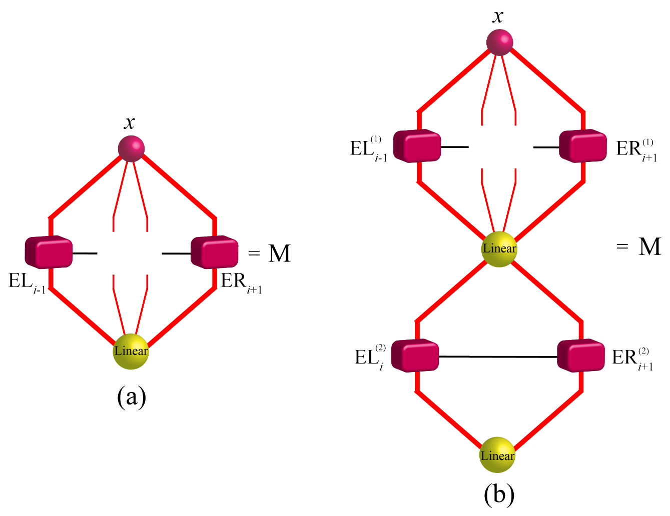

FIG. 10. [Color online] The M tensor required for manual calculation of the gradient $\Delta B_{i,i+1}$ for a TNN with (a) a single `TNLayer`, and (b) two `TNLayers`, and with linear activation functions.

the local bond tensors, $B_{i,i+1}$, are updated towards a global minimum of a loss function by a gradient descent step with learning rate α. Gradient of the bond tensors with respect to the loss, $\Delta B_{i,i+1}$, are therefore required for updating the weights. Given the fact that the TNN is a tensor network, gradients of the tensors in such a network with respect to a desired loss function are simply the network itself, with the corresponding tensor removed from the network. More specifically, consider a desired loss function such as the mean-squared error (MSE),

$$L_{\text{MSE}} = \frac{1}{N_s}\sum_{j=0}^{N_s}(y_p - y_j)^2, \tag{5}$$

where N_s is the number of training samples (input features) and y_js are their labels. Defining $y_p = MB$, where M is the contraction of all tensors in the TNN excluding the bond tensor B (see Fig. 10), the L_{MSE} alternatively reads

$$L_{\text{MSE}} = \frac{1}{N_s}\sum_{j=0}^{N_s}(MB - y_j)^2. \tag{6}$$

Taking the derivative with respect to B, the gradient of local bond tensors is given by

$$\Delta B = -\frac{\partial L_{\text{MSE}}}{\partial B} = \frac{2}{N_s}\sum_{n=1}^{N_s}(y_j - y_p)M. \tag{7}$$

Note that while both the M tensor and y_p involve the contraction of the TNN, we do not need to do it twice. In practice, we first calculate the M tensor and then we contract it to the bond tensor B to obtain the y_p prediction.

The above procedure for manual calculation of the gradient is only valid for linear activations or any other activation function that can be applied to the individual MPO weight tensors and not the contracted network. The reason is that the M tensor in Fig. 10 is obtained by removing the bond tensor B from the whole network after the application of the activation function. It has already been pointed out that for non-linear activations, one has to first contract the MPOs and the input from the previous layer and then apply the activation. It is obvious that the B tensor can not be removed from a contracted network. Therefore, no M tensor can be formed to calculate the tensor gradients manually.

Let us further remark that while the gradient in Eq. (7) and, subsequently, the LGD update (introduced in the previous subsection) are local, the DMRG-like sweep will restore the global features and correlations once iterated sufficiently.

C. Automatic gradient

Although the previous tensor gradient approach is suitable for TNN models fully composed of `TNLayers` and linear activation functions, it will not be efficient and flexible once dealing with models with hybrid architectures containing a mixture of `Dense` and `TNLayers` and nonlinear activation functions. Given the fact that our TNN is a feed-forward neural network, we can use automatic differentiation schemes and obtain the gradient of the TNN trainable weights with back-propagation (see Ref. [46, 47] for a review on automatic differentiation and back-propagation).

While one can implement the whole process manually, implementing the `TNLayer` in one of the ML libraries that support the automatic differentiation is highly recommended. Here we used the `TensorFlow` library and one of its useful features, i.e., `GradientTape` [44] to record all the mathematical operations in the forward path. These include the contraction of all tensors and calculation of the network prediction, y_p, as well as the loss, ex. Eq.(5). The `GradientTape` uses the operation records of the forward path and calculates the gradient of all trainable vectors, matrices and tensors in the backward move. Once the gradients are obtained, every trainable weights of the TNN can be updated. The MPOs are updated as prescribed in steps (c)-(g) of the DMRG-like algorithm of Sec. III A. The bias vectors, b, and the weight matrices of the `Dense` layers W can further be updated as

$$W' = W - \alpha\Delta W, \qquad b' = b - \alpha\Delta b.$$

For hybrid TNN models composed of multiple `TNLayers` and `Dense` layers one can in principle update the weights with different strategies, i.e., layer-by-layer (LbL) or partial-all-layer (PaL). In the LbL approach, we start updating from the first layer until we reach the last output layer. The `Dense` layers are updated according to Eq. (8) and the MPO weights of `TNLayers` are updated

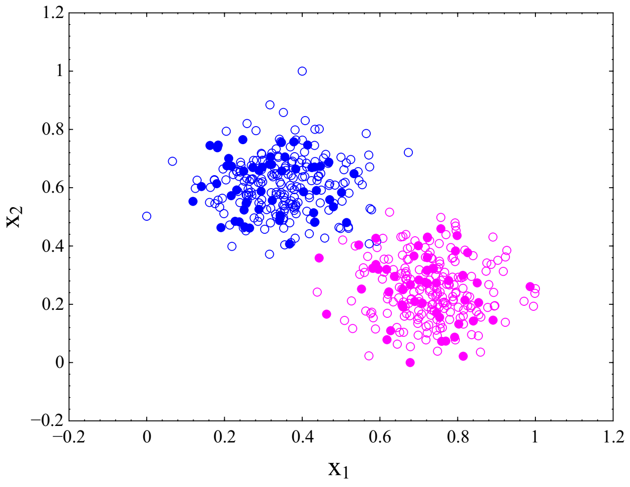

FIG. 11. [Color online] Binary classification of isotropic Gaussian blobs with TNN I. The empty (filled) circles represent training (test) data.

TABLE I. TNN architecture for binary classification of random Gaussian blobs

$N_s = 500$, Sweep $= 2000$, $\alpha = 0.1$, Batch $= 500$
$\mathcal{I} = \Big\{ \{x_1, x_2\}, \{y\} \Big\vert\ 0 \leq x_1, x_2 \leq 1,\ y = [0, 1] \Big\}$
$\downarrow \begin{cases} \texttt{Dense}\{\text{Units} = 64, \text{trainable} = \texttt{False}\} \\ \text{Activation: } \texttt{None} \end{cases}$
$\downarrow \begin{cases} \texttt{TNLayer}\{\text{Units} = 64, N_{\text{MPO}} = 6\} \\ \text{Activation: } \texttt{ReLU} \end{cases}$
$\downarrow \begin{cases} \texttt{Dense}\{\text{Units} = 2\} \\ \text{Activation: } \texttt{Softmax} \end{cases}$
$\mathcal{O} = [0{,}1]$
Loss Function: `BinaryCrossEntropy`

by a few sweeps say $N_{\text{sweep}} \leq 10$ according to the DMRG-like algorithm of Sec. III A. On the other hand, the PaL approach targets all the layers at once. In this scheme, we consider a large $N_{\text{sweep}} \geq 2000$ and update the MPO pairs $\{w_i^l, w_{i+1}^l\}$ of all `TNLayers` (l is the layer index), i.e, we do a multi-layer sweep. With each MPO pair update, we update the dense layers as well. This will help to reflect the local update of MPO weights in the matrix weights of the `Dense` layers during the sweep. The multi-layer sweep is performed until the loss converges to a certain threshold or until the N_{sweep} is reached.

It is also worth noting that for training the example models that we provide in the next sections, we observed that the PaL approach with large sweeps works much better from the point of view of convergence and accuracy.

D. Entanglement Entropy

In classic NN algorithms, the training convergence is checked by monitoring the loss function or some accuracy metric. The main drawback of these approaches is that they do not provide any information about the nature of the weights, the correlation among their parameters, and their expressive power. Thanks to the DMRG-like update, we have access to the singular values at every step of the sweep iterations, and this provides valuable information about the correlation (entanglement in quantum case) between MPO tensors. The individual singular values along the virtual dimensions of MPO tensors or their accumulative behavior in terms of the entropy, $S = \sum_i \lambda_i^2 \log \lambda_i^2$, reveal the degree of correlation (entanglement) between pieces of trainable weights. While weakly correlated states are distinguished by zero or very small entanglement entropy with only a few non-zero singular values, entangled states will have non-vanishing S and many non-zero singular values. Moreover, observing the behavior of entanglement entropy and singular values during the training can be used as another convergence measure, on top of the loss, which directly sheds light on the convergence of the MPO parameters.

IV. TNN CLASSIFIER

After introducing the TNN and the DMRG-like sweeping algorithm, let us see the performance and accuracy of the technique for classification tasks. In what follows, we present two examples for the classification of labeled data, one for the isotropic Gaussian blobs and another for the spiral distribution, and test the TNN in action.

A. Gaussian Blobs

As first example, we use the TNN to model a binary classification and train it over the random isotropic Gaussian blobs as shown with empty circles in Fig. 11. Each sample in the dataset has two position features denoted by x_1, x_2 and a label $y = [0, 1]$. The architecture of the TNN model used for the binary classification of the Gaussian blobs is detailed in Table I. The model is composed of a `TNLayer` and two `Dense` layers. The first `Dense` is non-trainable and has only been added to compensate for the size mismatch between the features and the next `TNLayer` (see Sec. II B). The `TNLayer` has six MPO trainable weights, each with input and output dimensions $d = 2$, and virtual dimension D, and `ReLu` activation function. Finally the output layer is a `Dense` layer with `Softmax` activation which delivers the predicted probabilities of both labels as one-hot-encoded (OHE) vectors. The prediction y_p can then be read form the `argmax` of the OHE probabilities for each sample.

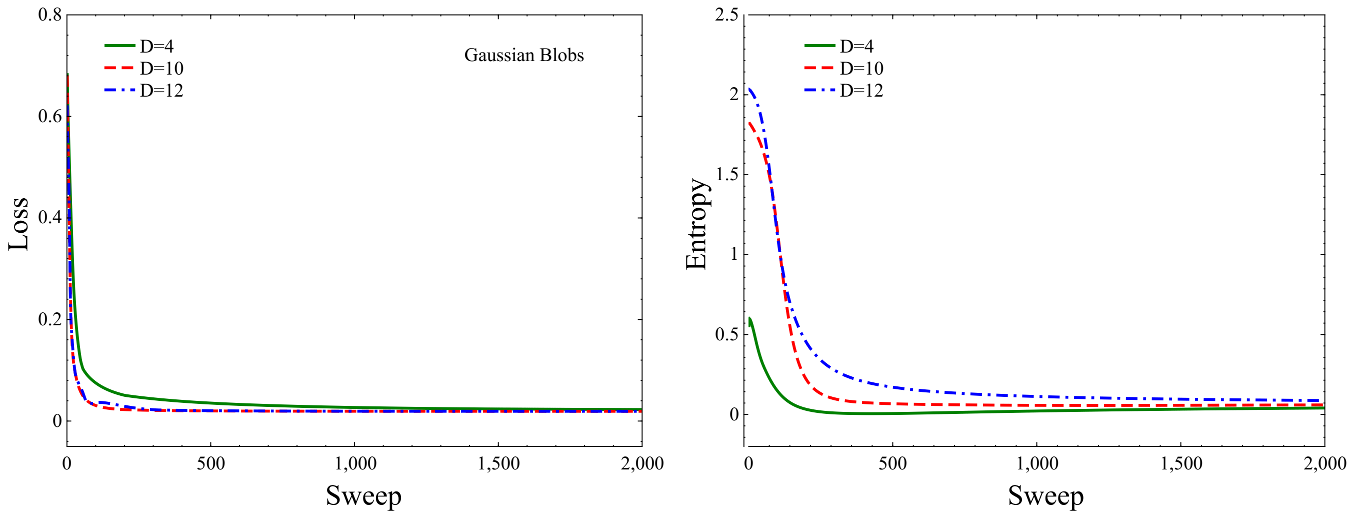

FIG. 12. [Color online] (left) Training loss and (right) the S entanglement entropy as a function of DMRG-like sweeps for training the TNN model I used for classification of Gaussian blobs.

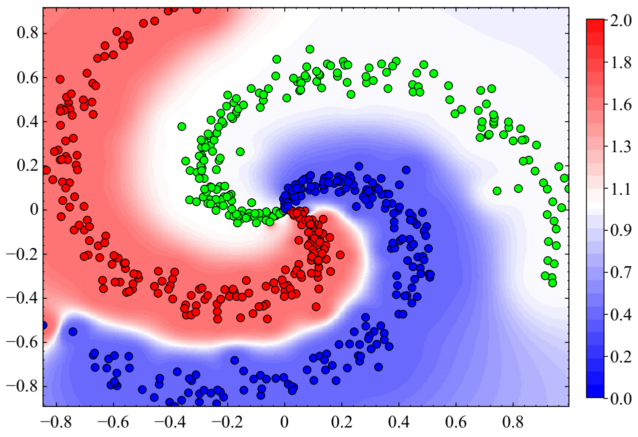

FIG. 13. [Color online] Ternary classification of random spiral distribution with three classes. The red, blue and green filled circles distinguish the training data in each class. The red, white and blue shaded regions further distinguish the decision boundary obtained from training the TNN model II.

TABLE II. TNN architecture for the classification of random spiral distributions

$N_s = 600$, Sweep $= 3000$, $\alpha = 0.1$, Batch $= 600$
$\mathcal{I} = \left\{ \{x_1, x_2\}, \{y\} \middle| -1 \leq x_1, x_2 \leq 1,\ y = [0, 1, 2] \right\}$
$\downarrow \left\{ \begin{array}{l} \texttt{Dense}\{\text{Units} = 64, \text{trainable} = \texttt{False}\} \\ \text{Activation: } \texttt{None} \end{array} \right.$
$\downarrow \left\{ \begin{array}{l} \texttt{TNLayer}\{\text{Units} = 64, N_{\text{MPO}} = 6\} \\ \text{Activation: } \texttt{ReLU} \end{array} \right.$
$\downarrow \left\{ \begin{array}{l} \texttt{Dense}\{\text{Units} = 3\} \\ \text{Activation: } \texttt{Softmax} \end{array} \right.$
$\mathcal{O} = [0,1,2]$
Loss Function: `BinaryCrossEntropy`

Fig. 11 shows the distribution of the two Gaussian blobs clusters. The empty circles denote the training data and the filled circles represent the test data. We performed the training for virtual dimension $D \in [2, 12]$ over 500 training data and tested the model over 100 unseen data. The average accuracy for classifying Gaussian blobs with the TNN model I is 99% (see Table I for training hyper-parameters).

Fig. 12 further shows the loss function and entanglement entropy as a function of the sweep iterations for different bond dimensions D. While both plots provide a measure for the training convergence, the S suggests that the MPO weights of the `TNLayer` represent a weekly entangled state with a small correlation among the parameters. This is best confirmed by very small entanglement entropy (close to zero) which is a typical behavior for product states. Looking at the distribution of blobs that are categorized into two localized clusters with almost no overlap, it is expected of the MPO weights be very close to that of the product states. The TNN therefore, not only classifies the blobs but also provides expressive insight into the nature of MPO weights.

B. Spiral Distribution

For the second example, we consider a random spiral distribution with three classes as illustrated in Fig. 13. We use the TNN model described in Table II for the ternary classification of spiral data. The model is simi-

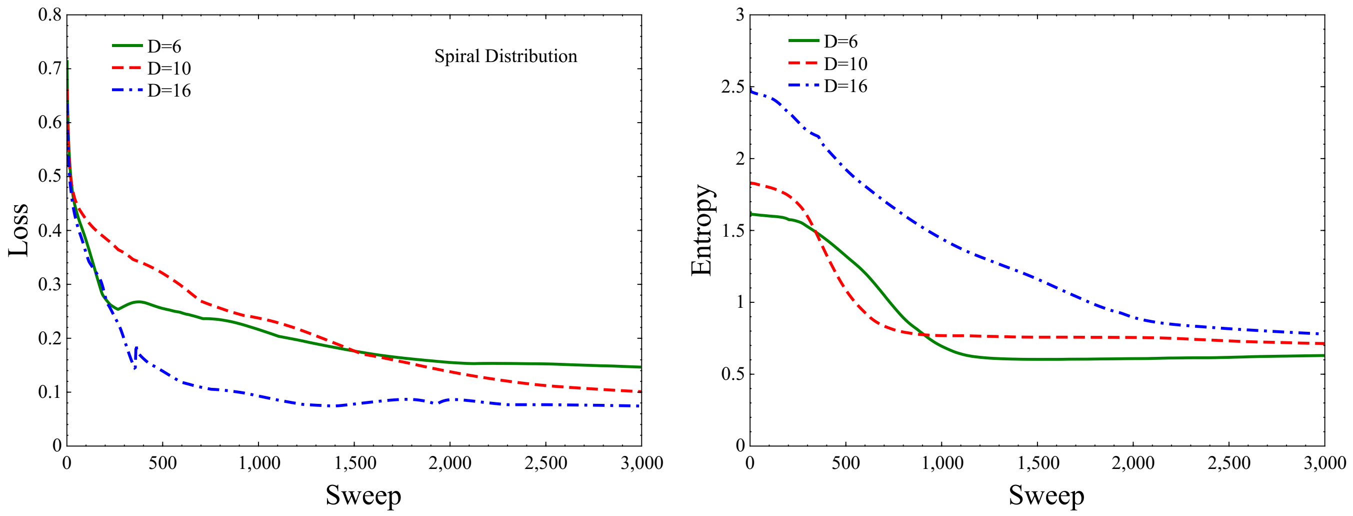

FIG. 14. [Color online] (left) Training loss and (right) the S entanglement entropy as a function of DMRG-like sweeps for training the TNN model II used for classification of spiral distibution.

lar to the previous example except that now the output `Dense` layer has three outputs for the probabilities of each class. We trained the model over 600 samples, 200, for each class and obtained the decision boundary of the spiral distribution as depicted in the shaded regions with red, white, and blue colors in Fig. 13. The average accuracy of the model from different runs is 95% (see Table II for training hyper-parameters).

The loss function and entanglement entropy during the training have also been displayed in Fig. 14 for different bond dimensions. Compared to gaussian blobs, one can clearly see that the behavior of loss and entanglement is totally different for the spiral distribution. Looking at the distribution of points in Fig. 13, the spiral dataset is expected to be more correlated and therefore, more challenging to be trained. The behavior of the loss as well as the non-vanishing entanglement entropy indeed confirms the higher degree of entanglement among the parameters of the MPOs for the spiral distribution. The S for the spiral distribution is approaching ≈ 0.75 signaling an entangled structure with more correlated parameters than that of the Gaussian blobs.

V. TNN REGRESSOR

In this section, we challenge our TNN for regression tasks. Here we present two regression examples one for fitting a line to a random linear distribution and another one for fitting to the random points around the nonlinear sine function.

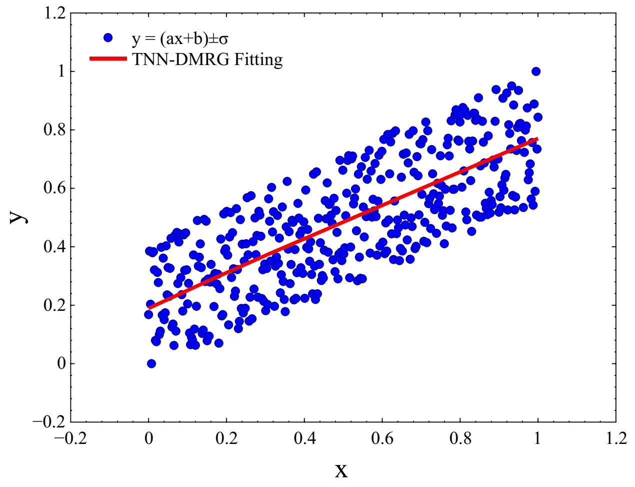

FIG. 15. [Color online] Linear regression of the random data around the line $y = ax + b$ with $a = 0.55$ and $b = 0.20$. The red line is the fit obtained from training the TNN model III. The predicted slope and shift of the red fitted line are $a = 0.57$ and $b = 0.19$.

A. Linear Regression

In order to test the TNN for a linear regression problem, we generate our data by adding uniform random noise to the points out of linear function $y = ax + b$ with $(a = 0.55, b = 0.20)$for $0 \leq x \leq 1$. The distribution of the linear random points has been depicted with blue circles in Fig. 15. In order to fit a line to the points, we designed a linear TTN model as described in Table III. The model is composed of a single trainable `TNLayer` with `ReLu` activation and a `Dense` output layer with no activation. As usual, we put a non-trainable `Dense` layer in front of the

TABLE III. TNN architecture for linear regression

$N_s = 400$, Sweep $= 2000$, $\alpha = 0.1$, Batch $= 400$
$\mathcal{I} = \left\{ \{x\}, \{y\} \middle\vert\ 0 \leq x, y \leq 1 \right\}$
$\downarrow \begin{cases} \texttt{Dense}\{\text{Units} = 64, \text{trainable} = \texttt{False}\} \\ \text{Activation: } \texttt{None} \end{cases}$
$\downarrow \begin{cases} \texttt{TNLayer}\{\text{Units} = 64, N_{\text{MPO}} = 6\} \\ \text{Activation: } \texttt{ReLU} \end{cases}$
$\downarrow \begin{cases} \texttt{Dense}\{\text{Units} = 1\} \\ \text{Activation: } \texttt{None} \end{cases}$
$\mathcal{O} = y_p$
Loss Function: `MeanSquareError`

`TNLayer` to compensate for the shape mismatch between the features and the `TNLayer`.

Training the linear TNN model for 400 random points and 2000 sweeps, we obtain a linear line that is perfectly fitted to the random data as shown in red in Fig. 15. Reading the slope and data shift from the predicted fitted line, we obtain $a_p = 0.57$ and $b_p = 0.19$ which is in good agreement with the original a and b parameters.

B. Non-linearRegression

Lastly, we challenge our TNN and DMRG-like algorithm for a non-linear regression task to predict a fit to the non-linear random points obtained by uniform noise to the $y = sinx$ function for $0 \leq x \leq 2\pi$. To this end, we introduce a non-linear TNN model with three `TNLayers` as detailed in Table IV. Neglecting the initial dummy `Dense` layer, the first `TNLayer` has a `Linear` activation, and the two others have `Sigmoid` activation functions. Adding to this structure an output `Dense` layer with `ReLu` activation, the resulting TNN model is fully capable of capturing non-linear correlation of the input features.

Fig. 16 shows 400 non-linear random *sine* shape points that have been fitted perfectly by the predicted red line from the trained TNN model. The TNN architecture of model IV is an example of a multi-layer deep TNN which is a mixture of both `Dense` and `TNLayers`. Indeed other complicated architectures with more layers and different activations can be designed for generic ML tasks based on neural networks. The choice of these examples and hyper-parameters was to showcase the performance, accuracy, and flexibility of designing generic models with the TNN and DMRG sweeping algorithm. Indeed by changing the architecture or hyper-parameter tuning of the aforementioned models, one can improve the quality of the results. However, this was not the purpose of this study.

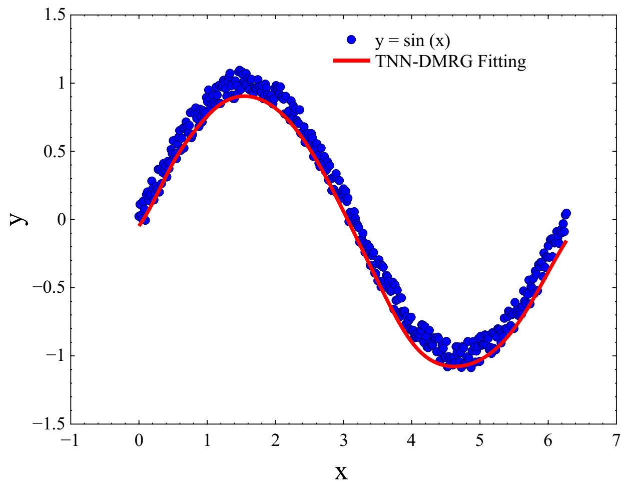

FIG. 16. [Color online] Non-Linear regression of the random data introduced to the function $y = \sin x$ for $0 \leq x \leq 2\pi$. The red line is the fit obtained from training the TNN model IV.

TABLE IV. TNN architecture for non-linear regression

$N_s = 400$, Sweep $= 2000$, $\alpha = 0.1$, Batch $= 400$
$\mathcal{I} = \left\{ \{x\}, \{y\} \middle\vert\ 0 \leq x \leq 2\pi,\ -1 \leq y \leq 1 \right\}$
$\downarrow \begin{cases} \texttt{Dense}\{\text{Units} = 64, \text{trainable} = \texttt{False}\} \\ \text{Activation: } \texttt{None} \end{cases}$
$\downarrow \begin{cases} \texttt{TNLayer}\{\text{Units} = 64, N_{\text{MPO}} = 6\} \\ \text{Activation: } \texttt{Linear} \end{cases}$
$\downarrow \begin{cases} \texttt{TNLayer}\{\text{Units} = 64, N_{\text{MPO}} = 6\} \\ \text{Activation: } \texttt{Sigmoid} \end{cases}$
$\downarrow \begin{cases} \texttt{TNLayer}\{\text{Units} = 64, N_{\text{MPO}} = 6\} \\ \text{Activation: } \texttt{Sigmoid} \end{cases}$
$\downarrow \begin{cases} \texttt{Dense}\{\text{Units} = 1\} \\ \text{Activation: } \texttt{ReLu} \end{cases}$
$\mathcal{O} = y_p$
Loss Function: `MeanSquareError`

VI. CONCLUSIONS AND OUTLOOK

In this paper, we introduced a fully tensorized neural network model for deep learning. By replacing the trainable weight matrices of the fully connected dense layers of classic NNs with matrix product operators, we obtained tensor neural networks that are capable of modelling different machine learning tasks ranging from classification to regression. We further introduced a new entanglement-aware training algorithm based on DMRG and local gradient-descent updates for training the TNN models which act on a reduced parameter subspace obtained from the tensorization of trainable weights, and is quite fast and accurate. Our TNNs are generic-purpose, i.e., they can be used for automatizing different ML models such as regression and classification. Our implementation further allows to construct hybrid architectures with a mixture of `Dense` and `TNLayers` to build real instances of deep learning models.

In order to show the performance and accuracy of the TNNs and DMRG-like training algorithm, we considered several deep learning models for linear and non-linear regression, as well as models for the classification of labeled data. Our findings suggest that the TNNs and DMRG-like training algorithm are performing efficiently and accurately for different ML tasks. Most importantly, the DMRG-like training algorithm provides direct access to the singular values along the virtual dimensions of the trainable MPOs of `TNLayers`, from which a measure of entanglement (correlation) between the features and model parameters can be computed.

Our TNN and DMRG-like algorithm suggest that tensor networks are closely related to neural networks. In fact, our approach opens the door for designing new numerical techniques for obtaining neural network representations of a quantum state and is a valuable tool to study the expressive power of quantum neural states. Last but not least, The ideas developed here can further be extended to other deep learning architectures such as convolutional neural networks and their training algorithm.

VII. ACKNOWLEDGEMENTS

S.S.J. acknowledges the support from Institute for Advanced Studies in Basic Sciences (IASBS), Donostia International Physics Center (DIPC) and Multiverse Computing.

[1] X. Zhou, W. Gong, W. Fu, and F. Du, Proceedings - 16th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2017 , 631 (2017).

[2] J. Sonoda and T. Kimoto, Asia-Pacific Microwave Conference Proceedings, APMC **2018-November**, 1298 (2019).

[3] Y. H. Chang, P. L. Chung, and H. W. Lin, Proceedings of 4th IEEE International Conference on Applied System Innovation 2018, ICASI 2018 , 66 (2018).

[4] S. Li, W. Song, L. Fang, Y. Chen, P. Ghamisi, and J. A. Benediktsson, IEEE Transactions on Geoscience and Remote Sensing **57**, 6690 (2019), arXiv:1910.12861.

[5] C. Affonso, A. L. D. Rossi, F. H. A. Vieira, and A. C. P. d. L. F. de Carvalho, Expert Systems with Applications **85**, 114 (2017).

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Communications of the ACM **60**, 84 (2017).

[7] S. Lathuiliere, P. Mesejo, X. Alameda-Pineda, and R. Horaud, IEEE Transactions on Pattern Analysis and Machine Intelligence **42**, 2065 (2020), arXiv:1803.08450.

[8] B. Ramsundar and R. B. Zadeh, *TensorFlow for deep learning : from linear regression to reinforcement learning* (O'Reilly Media, Inc., 2018).

[9] K. Tian, S. Zhou, and J. Guan, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **10535 LNAI**, 809 (2017).

[10] E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long, IEEE Access **6**, 39501 (2018).

[11] E. Aljalbout, V. Golkov, Y. Siddiqui, M. Strobel, and D. Cremers, (2018), 10.48550/arxiv.1801.07648, arXiv:1801.07648.

[12] D. Yu, M. L. Seltzer, J. Li, J. T. Huang, and F. Seide, 1st International Conference on Learning Representations, ICLR 2013 - Conference Track Proceedings (2013), 10.48550/arxiv.1301.3605, arXiv:1301.3605.

[13] L. Jing and Y. Tian, IEEE Transactions on Pattern Analysis and Machine Intelligence **43**, 4037 (2021), arXiv:1902.06162.

[14] S. Dehaene and J. P. Changeux, Progress in Brain Research **126**, 217 (2000).

[15] T. Hill, L. Marquez, M. O'Connor, and W. Remus, International Journal of Forecasting **10**, 5 (1994).

[16] E. Bedolla, L. C. Padierna, and R. Castañeda-Priego, Journal of Physics: Condensed Matter **33**, 053001 (2020), arXiv:2005.14228.

[17] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, Nature Reviews Physics 2021 3:6 **3**, 422 (2021).

[18] P. Mehta, M. Bukov, C. H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, Physics Reports **810**, 1 (2019), arXiv:1803.08823.

[19] O. Sharir, A. Shashua, and G. Carleo, (2021), 10.48550/arxiv.2103.10293, arXiv:2103.10293.

[20] G. Pescia, J. Han, A. Lovato, J. Lu, and G. Carleo, Physical Review Research **4**, 023138 (2022), arXiv:2112.11957.

[21] I. L. Gutiérrez and C. B. Mendl, Quantum **6**, 627 (2022), arXiv:1912.08831v4.

[22] G. Pescia, J. Han, A. Lovato, J. Lu, and G. Carleo, Physical Review Research **4**, 023138 (2022), arXiv:2112.11957.

[23] Y. Wu, J. Yao, P. Zhang, and H. Zhai, Physical Review Research **3**, L032049 (2021), arXiv:2101.04273.

[24] J. Carrasquilla and R. G. Melko, Nature Physics 2017 13:5 **13**, 431 (2017), arXiv:1605.01735.

[25] E. Greplova, A. Valenti, G. Boschung, F. Schäfer, N. Lörch, and S. D. Huber, New Journal of Physics **22**, 045003 (2020).

[26] A. Tanaka and A. Tomiya, http://dx.doi.org/10.7566/JPSJ.86.063001 **86** (2017), 10.7566/JPSJ.86.063001, arXiv:1609.09087.

[27] Y. Zhang, R. G. Melko, and E. A. Kim, Physical Review B **96**, 245119 (2017).

[28] R. R. Orús, Annals of Physics **349**, 117 (2014), arXiv:1306.2164.

[29] F. Verstraete, V. Murg, and J. Cirac, Advances in Physics **57**, 143 (2008).

[30] E. M. Stoudenmire and D. J. Schwab, (2016), 10.48550/arxiv.1605.05775, arXiv:1605.05775.

[31] R. G. Patel, C.-W. Hsing, S. S. Ahin, S. S. Jahromi, S. Palmer, S. Sharma, C. Michel, V. Porte, M. Abid, S. Aubert, P. Castellani, C.-G. Lee, S. Mugel, and R. Orús, (2022), 10.48550/arxiv.2208.02235, arXiv:2208.02235.

[32] J. Wang, C. Roberts, G. Vidal, and S. Leichenauer, (2020), 10.48550/arxiv.2006.02516, arXiv:2006.02516.

[33] E. M. Stoudenmire, Quantum Science and Technology **3**, 034003 (2018).

[34] Y. Panagakis, J. Kossaifi, G. G. Chrysos, J. Oldfield, M. A. Nicolaou, A. Anandkumar, and S. Zafeiriou, Proceedings of the IEEE **109**, 863 (2021), arXiv:2107.03436.

[35] M. Bahri, Y. Panagakis, and S. Zafeiriou, IEEE Transactions on Pattern Analysis and Machine Intelligence **41**, 2365 (2019), arXiv:1801.06432.

[36] J. Kossaifi, A. Khanna, Z. Lipton, T. Furlanello, and A. Anandkumar, IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops **2017-July**, 1940 (2017), arXiv:1706.00439.

[37] J. Kossaifi, Z. C. Lipton, A. Kolbeinsson, A. Khanna, T. Furlanello, and A. Anandkumar, Journal of Machine Learning Research **21**, 1 (2020).

[38] P. Martin-Ramiro, U. S. de la Maza, M. Alvarez-Cascos, G. J. Alvarez, O. H. Caballer, S. S. Jahromi, R. Orus, and S. Mugel, In prepration (2022).

[39] S. R. White, Physical Review Letters **69**, 2863 (1992).

[40] S. R. White, Physical Review B **48**, 10345 (1993).

[41] R. Orús and G. Vidal, Physical Review B - Condensed Matter and Materials Physics **78**, 155117 (2008), arXiv:0711.3960.

[42] G. Vidal, Physical Review Letters **91**, 147902 (2003), arXiv:0301063 [quant-ph].

[43] G. Vidal, Physical Review Letters **93**, 040502 (2004), arXiv:0310089 [quant-ph].

[44] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," (2015), software available from tensorflow.org.

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, in *Advances in Neural Information Processing Systems 32* (Curran Associates, Inc., 2019) pp. 8024–8035.

[46] C. C. Margossian, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **9**, e1305 (2019), arXiv:1811.05031.

[47] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, Journal of Machine Learning Research **18**, 1 (2018).