arXiv:1803.10908v1 [cond-mat.stat-mech] 29 Mar 2018

Matrix Product Operators for Sequence to Sequence Learning

Chu Guo,[1,2] Zhanming Jie,[3] Wei Lu,[3] and Dario Poletti[1]

[1]*Engineering Product Development Pillar, Singapore University of Technology and Design, 8 Somapah Road, 487372 Singapore*
[2]*Zhengzhou Information Science and Technology Institute, Zhengzhou 450004, China*
[3]*Information Systems Technology and Design, Singapore University of Technology and Design, 8 Somapah Road, 487372 Singapore*

(Dated: November 8, 2019)

The method of choice to study one-dimensional strongly interacting many body quantum systems is based on matrix product states and operators. Such method allows to explore the most relevant, and numerically manageable, portion of an exponentially large space. Here we show how to construct a machine learning model in which matrix product operators are trained to implement sequence to sequence prediction, i.e. given a sequence at a time step, it allows one to predict the next sequence. We then apply our algorithm to cellular automata (for which we show exact analytical solutions in terms of matrix product operators), and to nonlinear coupled maps. We show advantages of the proposed algorithm when compared to conditional random fields and bidirectional long short-term memory neural network. To highlight the flexibility of the algorithm, we also show how it can perform classification tasks.

I. INTRODUCTION

The last few years have witnessed a great shift of interest of society (individuals, industries and governmental organizations) towards machine learning and its wide range of applications, from images classification to translation and more. Of particular importance are machine learning models which, given an input sequence, can predict an output sequence, i.e. models of sequence to sequence learning. These sequences can be lists of either continuous or discrete values, or of words (e.g. a phrase), and hence the variety of possible applications and their impact are significant.

In recent years we have also witnessed an increasing activity at the intersection between machine learning and quantum physics. This includes further studies on quantum machine learning [1–3], use of machine learning for materials study [4], glassy physics [5], for phases recognition [6–12] and to numerically study quantum systems [13–22]. To be noted are studies on physical analogies and reasons for effectiveness of machine learning [23–25].

Another research direction, which we follow here, has been to apply tools developed in many body quantum physics to typical machine learning tasks. Recent examples are [26–28], where algorithms based on matrix product states (MPS), also known as tensor trains, were successfully used for supervised classification or unsupervised generative modelling. MPS based algorithms were also used for classification [29], as predictive modeling of stochastic processes [30], for language modeling [31] and compared to Boltzmann machines [32, 33]. Usefuleness of tensor representations and matrix product states was also noted in [34].

In physics, MPS are used to represent wave-functions, probability distributions or density matrices as a product of tensors [35–39]. Building on the density matrix renormalization group [40], matrix product states are very succesfully used in many body quantum physics to study ground or steady states and time evolution of Hamiltonian or dissipative systems [41–47]. As an extension, quantum mechanical operators are represented by matrix product operators (MPO), which are composed of tensors, and map an MPS to another one.

In this work we implement a quantum-inspired matrix product operator model for sequence to sequence prediction. This significantly increases the type of machine learning applications for matrix product states/operators based algorithms. Given a set of input and output vectors, we are going to train an MPO which can accurately reproduce the transformation from input to output vectors and which can be then used for vectors not used in the training. As exemplary applications we are going to train an MPO to predict the evolution of dynamical systems, in particular cellular automata and coupled maps. We will show that it is possible to train an MPO to predict exactly the evolution of cellular automata, as long as the matrix product operators are large enough. It is even possible to write analytically an MPO description for cellular automata which we discuss both in the main text and in the appendix. We will then show that the predictions of the trained MPO for the evolution of cellular automata can be perfect even in presence of a large portion of errors in the training data. For this problem we compare the MPO based method to conditional random fields (CRF) [48], showing some advantages of the MPO model. Later we apply the MPO algorithm to the prediction of the evolution of a nonlinear coupled map. Despite the lack of exact analytical solutions for the MPO, we show that it can be used to predict the evolution of such systems as well. When compared to our implementation of a bidirectional long short-term memory (LSTM) neural network (both one and two layers) [49–53], we show that the MPO algorithm gives more accurate results for the problem analyzed.

To exemplify the generality of the algorithm, we also describe how to apply it to classification problems, using as an example the MNIST handwritten digits data set [54].

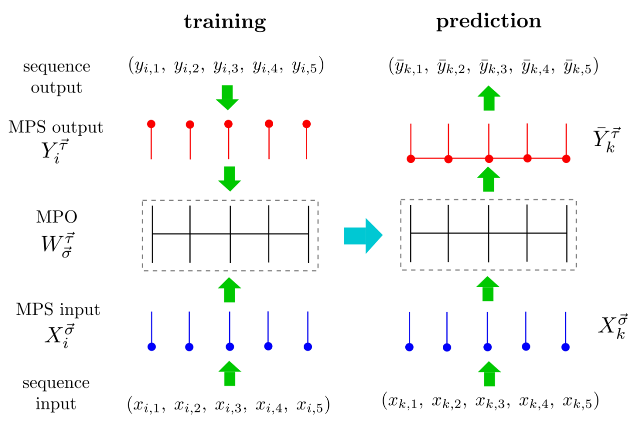
training
prediction
sequence output
$(y_{i,1},\ y_{i,2},\ y_{i,3},\ y_{i,4},\ y_{i,5})$
$(\bar{y}_{k,1},\ \bar{y}_{k,2},\ \bar{y}_{k,3},\ \bar{y}_{k,4},\ \bar{y}_{k,5})$
MPS output $Y_i^{\vec{\tau}}$
$\bar{Y}_k^{\vec{\tau}}$
MPO $W_{\vec{\sigma}}^{\vec{\tau}}$
MPS input $X_i^{\vec{\sigma}}$
$X_k^{\vec{\sigma}}$
sequence input
$(x_{i,1},\ x_{i,2},\ x_{i,3},\ x_{i,4},\ x_{i,5})$
$(x_{k,1},\ x_{k,2},\ x_{k,3},\ x_{k,4},\ x_{k,5})$

FIG. 1: Training phase: a pair of vectors, $\vec{x}_i$ and $\vec{y}_i$, is converted to a pair of matrix product states (MPS), $X_i^{\vec{\sigma}}$ and $Y_i^{\vec{\tau}}$, and used to train a matrix product operator (MPO) $W_{\vec{\sigma}}^{\vec{\tau}}$. Prediction phase: an input vector $\vec{x}_k$ is converted to an MPS $X_k^{\vec{\sigma}}$ and then multiplied by the trained MPO $W_{\vec{\sigma}}^{\vec{\tau}}$. The resulting MPS $\bar{Y}_k^{\vec{\tau}}$ is then converted to a vector $\vec{\bar{y}}_k$.

The paper is organized as follows: in Sec.II we present the main parts of the algorithm, in Sec.III we will apply our algorithm to sequence to sequence prediction, both in discrete (cellular automata) and continuous (coupled maps) problems, and to a classification task. Last, in Sec.IV we draw our conclusions. In the Appendices we provide further technical details and explanations for the more interested readers.

II. METHOD

In the following section we describe how we implement the training (or learning phase) of our model and how to test the accuracy of its predictions (testing phase). The MPO algorithm will be able to generate a sequence of L numbers given another one of same length. Stated differently, we consider mapping from a vector space $\mathcal{X}$, with vectors $\vec{x}_i$, to another vector space $\mathcal{Y}$, with vectors $\vec{y}_i$, which is given by

$$\vec{y}_i = f(\vec{x}_i), \tag{1}$$

where f does not need to be linear. The MPO algorithm would try to provide an accurate approximation $\vec{\bar{y}}_i$ to the exact vector $\vec{y}_i$.

We will focus on the case for which each vector $\vec{x}_i, \vec{y}_i$ have the same length L and corresponding elements $x_{i,l}$ and $y_{i,l}$. The algorithm will be explained in detail in the following, however, to help the reader we summarize it in Fig.(1). In the training phase we take a set of M input and output sequences $\vec{x}_i$ and $\vec{y}_i$, where $1 \leq i \leq M$. Each sequence $\vec{x}_i$ and $\vec{y}_i$ is converted to a matrix product state (MPS) – that is a product of tensors. All these MPS are used to generate/train a mapping from MPS to MPS which is known as matrix product operator (MPO).

In contrast, in the testing/prediction phase, see right portion of Fig.1, we take an input sequence $(x_{k,1}, x_{k,2}, \dots, x_{k,L})$ (different from those used in the training), convert it to an MPS, multiply it to the MPO trained earlier and this will result in an output MPS which is then converted to a sequence $(\bar{y}_{k,1}, \bar{y}_{k,2}, \dots, \bar{y}_{k,L})$. To compute the accuracy of the algorithm, we compare the predicted output sequence with the exact one.

A. Training

The algorithm aims to provide, given a sequence, another sequence from an exponentially large number of possible ones. The output will thus be a function of the possible output sequences, and it will be used to choose the output sequence. One key ingredient in the proposed algorithm is to describe inputs and outputs, or their probability distribution, as MPS [36–39]. For example, the probability of a given sequence of integer numbers $\vec{\sigma} = (\sigma_1, \sigma_2, \dots, \sigma_L)$, i.e. $P(\vec{\sigma})$, can be written as

$$P(\vec{\sigma}) = \sum_{a_0,\dots,a_L} M^{\sigma_1}_{a_0,a_1} M^{\sigma_2}_{a_1,a_2} \dots M^{\sigma_L}_{a_{L-1},a_L}, \tag{2}$$

where σ_l describe the local degrees of freedom, whose total number is d, and the a_l are auxiliary degrees of freedom needed to take into account correlations between different elements of the sequence. The larger the number of the auxiliary local degrees of freedom a_l, also known as "bond dimension" D, the more accurate is the probability distribution. A single sequence $\vec{\sigma}$ can be written as an MPS with $D = 1$, also known as a product state. Since we rewrite inputs and outputs as MPS the natural object to train, which will then be able to map an input MPS to an output MPS, is an MPO $W_{\vec{\sigma}}^{\vec{\tau}}$ where $\vec{\tau}$ is a sequence of output integer numbers. The MPOs can be parameterized by a product of 4-dimensional tensors

$$W_{\vec{\sigma}}^{\vec{\tau}} = \sum_{b_0,b_1,\dots,b_L} W^{\sigma_1,\tau_1}_{b_0,b_1} W^{\sigma_2,\tau_2}_{b_1,b_2} \dots W^{\sigma_L,\tau_L}_{b_{L-1},b_L} \tag{3}$$

Here, the indices b_l indicate an auxiliary dimension which we refer to as the MPO's bond dimension D_W.

In the next sections we are going to describe how to convert a sequence (also of real numbers) to an MPS and then how to train an MPO.

1. *From sequence to MPS*

The first necessary step to train an MPO is to convert a sequence (input or output) to an MPS. The input and output sequences may belong to different spaces, for example an input sequence could be made of words, letters, real numbers or integers. Each of the above can be represented as a vector of finite size d for the input, and d'

for the output (which can potentially be different). Each sequence is thus readily mapped to a list of vectors. For clarity of explanation, in the following we will consider the cases in which $x_{i,l}$ is an integer number or a real number between 0 and 1. A completely analogous discussion can be done for the output sequences with elements $y_{i,l}$. If $x_{i,l}$ is an integer number which spans over d possible values, it is then mapped to a vector with all 0 entries except for one of them, corresponding to a particular σ_l, which is set equal to 1 (this representation is known as one hot vector). This means that considering $x_{i,l} = 0$, 1 or 2, then $d = 3$ and, for example, if $x_{i,l} = 1$ we get $\vec{\sigma}_l = (0, 1, 0)$. If $x_{i,l}$ is instead a real number between 0 and 1, it can be mapped, for example, to a vector with dimension 2 with elements $\left(\sqrt{1 - x_{i,l}^2}\,, x_{i,l}\right)$.

In this way, each input sequence $\vec{x}_i$ can be mapped into a product MPS of physical dimension d.

$$\vec{x}_i \to X_i^{\vec{\sigma}} = \sum_{a_0,\dots,a_L} X^{\sigma_1}_{i,a_0,a_1} X^{\sigma_2}_{i,a_1,a_2} \dots X^{\sigma_L}_{i,a_{L-1},a_L} \tag{4}$$

where all the auxiliary indices $a_l = 1$ (bond dimension $D = 1$), and the i index in the tensors $X^{\sigma_l}_{i,a_l,a_{l+1}}$ differentiate the $i-$th sequence from others. The index σ_l signals which element of the d dimensional vector is being considered. As an example, for the case in which $x_{i,l}$ is a real number between 0 and 1, each matrix product state $X^{\sigma_l}_{i,a_l,a_{l+1}}$ is given by

$$X^0_{i,1,1} = \sqrt{1 - x_{i,l}^2} \;\text{ and }\; X^1_{i,1,1} = x_{i,l}. \tag{5}$$

Note that a similar mapping was done in [26].

Analogously, each output $\vec{y}_i$ can be mapped into a tensor product MPS

$$\vec{y}_i \to Y_i^{\vec{\tau}} = \sum_{c_0,\dots,c_L} Y^{\tau_1}_{i,c_0,c_1} Y^{\tau_2}_{i,c_1,c_2} \dots Y^{\tau_L}_{i,c_{L-1},c_L} \tag{6}$$

where the auxiliary indices $c_l = 1$, while the index τ_l signals which element of the d' dimensional local output vector is being considered.

As discussed earlier, multiplying an MPS by an MPO results in another MPS

$$\bar{Y}_i^{\vec{\tau}} = W^{\vec{\tau}}_{\vec{\sigma}} X_i^{\vec{\sigma}} = \sum_{\bar{c}_0,\dots,\bar{c}_L} \bar{Y}^{\tau_1}_{i,\bar{c}_0,\bar{c}_1} \dots \bar{Y}^{\tau_L}_{i,\bar{c}_{L-1},\bar{c}_L}, \tag{7}$$

where $\bar{c}_l$ is a new auxiliary index given by $\bar{c}_l = (a_l, b_l)$ on each site l, and where we have used

$$\bar{Y}^{\tau_l}_{i,\bar{c}_{l-1},\bar{c}_l} = \sum_{\sigma_l} W^{\sigma_l,\tau_l}_{b_{l-1},b_l} X^{\sigma_l}_{i,a_{l-1},a_l}. \tag{8}$$

2. *Iterative minimization of the cost function*

With the above notations, we can define a cost function $C(W^{\vec{\tau}}_{\vec{\sigma}})$, which takes into account the distance between all the predicted and the exact output sequences. The cost function we use is

$$\begin{aligned} C(W^{\vec{\tau}}_{\vec{\sigma}}) = \sum_{i=1}^{N} \left(\bar{Y}_i^{\vec{\tau}\dagger} - Y_i^{\vec{\tau}\dagger}\right)\left(\bar{Y}_i^{\vec{\tau}} - Y_i^{\vec{\tau}}\right) \\ + \alpha \operatorname{tr}\left(W^{\vec{\tau}\dagger}_{\vec{\sigma}} W^{\vec{\tau}}_{\vec{\sigma}}\right), \end{aligned} \tag{9}$$

where the last term with the coefficient α regularizes the MPO.

To minimize the cost function, we use an iterative approach, similar to the MPS method used for ground state search for one-dimensional quantum many body problems [35]. The central idea is to turn a global minimization problem into many local minimization problems via an iterative procedure. To minimize the cost function we compute its derivative versus the local tensor $W^{\sigma_l,\tau_l}_{b_{l-1},b_l}$ of $W^{\vec{\tau}}_{\vec{\sigma}}$, and set it to zero

$$\frac{\partial C(\hat{W})}{\partial W^{\sigma_l,\tau_l}_{b_{l-1},b_l}} = 0. \tag{10}$$

The minimization procedure is done from site $l = 1$ to site $l = L$ and back, and this procedure is commonly referred to, in many body quantum physics, as sweep. Since the cost function is quadratic, a linear solver can be used to locally compute the optimal tensor $W^{\sigma_l,\tau_l}_{b_{l-1},b_l}$. The sweeps are repeated until a maximum number k_{max} or until the cost function converges to a previously determined precision ϵ_t. The algorithm can be run with different bond dimensions D_W (the size of the auxiliary space) and with different training samples number M in order to reach more accurate sequence predictions.

B. Predictions

Once the MPO has been trained, it is possible to use it to make predictions. In order to do so, first the input sequence $\vec{x}_k$ is converted into an input MPS $X_k^{\vec{\sigma}}$. This MPS is then multiplied with the trained MPO $W^{\vec{\tau}}_{\vec{\sigma}}$, and this results in the output matrix product state $\bar{Y}_k^{\vec{\tau}}$, see also the right side of Fig.1. It is then necessary to convert the output MPS into an output sequence, and this is done in two steps. First we approximate the output MPS with an MPS of bond dimension $D = 1$ (this can be done iteratively by minimizing the distance between the output MPS and the approximated one [35]). Then, the MPS with bond dimension $D = 1$ is converted to a sequence by reversing the way in which, during training, a sequence $\vec{x}_i$ (or $\vec{y}_i$) was converted into a $D = 1$ matrix product state $X_i^{\vec{\sigma}}$ (or $Y_i^{\vec{\tau}}$). For example, for $\bar{y}_{k,l}$ real between 0 and 1, then we set $\bar{y}_{k,l} = \bar{Y}^1_{k,1,1}$; when $\bar{y}_{k,l}$ is integer, we set $\bar{y}_{k,l} = \tau_l$ for the τ_l for which $\bar{Y}^{\tau_l}_{k,1,1}$ is largest.

III. APPLICATIONS

We are now going study how this sequence to sequence learning model can be used to predict the evolution of different systems. In general one can consider systems which are extended in space, evolve in time, and take different values at each different location and time. We will first concentrate on *cellular automata*, for which time, space and the possible values are all discrete. We will then consider *coupled maps*, for which, while time and space are discretized, the possible values that a function has at a certain time and location are continuous.

A. Cellular automata

Cellular automata are models in which time and space are discretized and when at each location and time, a function takes discrete values. The values of this function at later times are determined by the values at previous time in a neighbourhood of the site considered. Such models have been used in many fields from mathematics to physics, including computer science and biology [55]. In the models that we study here, we consider sequences of 0s and 1s. At first, to each input sequence we associate an output sequence using the following algorithm. A 0 at position l is converted to a 1, or vice versa, only if the digit at position $l+j$ is a 0 [56]. This is a nonlinear problem because the evolution depends on the state of the system, moreover it is a long-range system because the evolution of a digit at site l depends on the digit at a distance j. In the classification of elementary cellular automata, and for $j=1$, this corresponds to rule 153. For $j>1$ we will refer to this rule as long-range rule 153.

Steady states of cellular automata can be written exactly using matrix product states [57], however here we are interested in training an MPO that describes the evolution rule of a cellular automata. Such cellular automata rule can be exactly mapped to MPOs [58]. Hence this is an ideal example to test the functioning of the MPO learning model. To show how to compute the exact MPO for cellular automata we take the, readily generalizable, example of $j=1$ (see the Appendix B for the MPOs for rules 18 and 30). In this case, the exact evolution is given by an MPO with bond dimension $D_W=2$. For sites l different from 1 and L, the MPOs are

$$\begin{aligned} W^{0,0}_{b_{l-1},b_l} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad W^{0,1}_{b_{l-1},b_l} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \\ W^{1,1}_{b_{l-1},b_l} &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad W^{1,0}_{b_{l-1},b_l} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \end{aligned} \tag{11}$$

while for the first and last site they are respectively given by

$$\begin{aligned} W^{0,0}_{b_0,b_1} &= [0,\ 1], \quad W^{0,1}_{b_0,b_1} = [1,\ 0], \\ W^{1,1}_{b_0,b_1} &= [0,\ 1], \quad W^{1,0}_{b_0,b_1} = [1,\ 0], \end{aligned} \tag{12}$$

and

$$\begin{aligned} W^{0,0}_{b_{L-1},b_L} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad W^{0,1}_{b_{L-1},b_L} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \\ W^{1,1}_{b_{L-1},b_L} &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad W^{1,0}_{b_{L-1},b_L} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \end{aligned} \tag{13}$$

It is straightforward to check that given an input sequence and an output sequence, the MPO made by the product of the tensors above gives the scalar 1 only if the output sequence $\vec{\tau}$ is the correct evolution of the input sequence $\vec{\sigma}$, otherwise it gives 0. It should be noted though, that there are many MPOs which fulfil this requirement, as gauge transformation can be applied to the tensors in a way that their product is unchanged.

In the following we test the MPO based code using an interaction range $j=3$ for which, since the required MPO bond dimension increases as 2^j, should be described exactly by MPOs with bond dimension $D_W=8$ or above. required.

To train the MPO we use a set of M input random sequences and their corresponding output. An example of input and output sequences is depicted in Fig.2(t1,t2). Within the output sequences we use, with a probability ϵ_r, a wrong sequence also drawn randomly as the input. After having trained the MPO, we consider another set of N initial conditions and each one of them we evolve it for T iterations. Two examples of the exact time evolution of sequences are depicted each in one of the two panels (e1) and (e2) of Fig.2. The results for the evolution predicted with the MPO algorithm are depicted in Fig.2(p1-p2), and show perfect agreement with the exact evolutions. In this case we had considered a system with $L=40$ sites and evolved the initial conditions for $N=40$ steps, while choosing $M=5000$, $D_W=8$, $\alpha=0.001$, $k_{max}=20$, $\epsilon_t=10^{-5}$ and $\epsilon_r=0$ to train the MPO. For this case, which is the long-range rule 153 with $j=3$, the evolution is periodic with a relatively short period. It is however interesting to note that, even for evolutions with much longer periods, as for example rule 18 with fixed boundary conditions for a system with $L=60$ sites, the evolution can be perfectly predicted while no periodic evolution has yet manifested. In fact, in Fig.2(a) we show an example of exact prediction of the evolution when using MPO bond dimension $D_W=4$ (which is large enough, as shown in Appendix B, to provide an exact evolution of rule 18), and a number of training input/output pairs $M=50000$.

We now study the accuracy of the algorithm as a function of the various parameters. We consider again the long-range rule 153 with $j=3$ and consider $N=100$ initial conditions. In Fig.3(a) we show the fraction of wrong predictions versus time averaged over N initial conditions and for different MPO bond dimensions D_W. The fraction of wrong predictions is given by

$$\varepsilon = \sum_{k,l} |y_{k,l}(t) - \bar{y}_{k,l}(t)|/(L\ N) \tag{14}$$

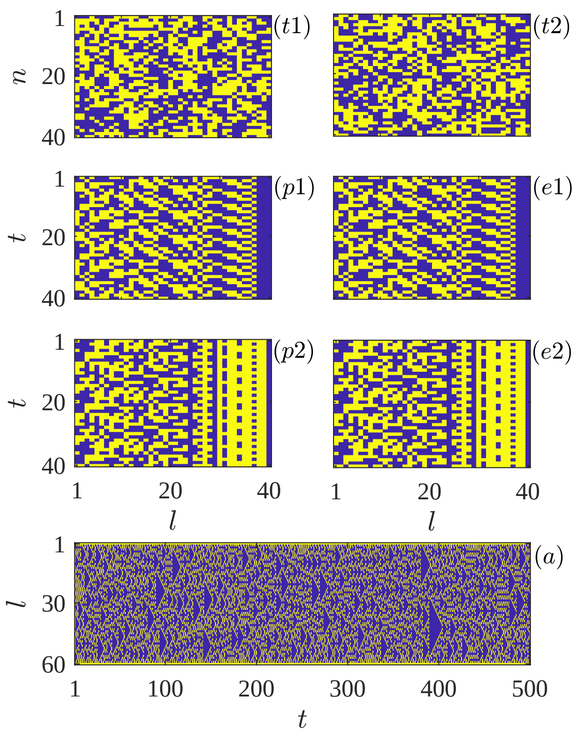

FIG. 2: Evolution of the long-range rule 153 cellular automata with different random initial conditions from the trained MPO ($p1$, $p2$) and the respective exact evolutions ($e1$, $e2$). Panels (t1-t2) show respectively a sample of 40 input and output training data. The system is chosen to have $L = 40$, interaction distance $j = 3$. The training parameters are training sample size $M = 5000$, bond dimension of the MPO $D_W = 8$, error rate in the training $\epsilon_r = 0$, regularizing coefficient $\alpha = 0.001$, maximum number of sweeps $k_{max} = 20$ and convergence tollerance $\epsilon_t = 10^{-5}$. In panel (a) we show the predicted evolution (which perfectly matches the exact one) for rule 18 with fix boundary conditions. In this case $L = 60$, the MPO bond dimension $D_W = 4$, and we have used $M = 50000$ training pairs of inputs and outputs. Other parameters are $\alpha = 0.001$, $k_{max} = 20$ and $\epsilon_t = 10^{-5}$.

where $y_{k,l}(t)$ and $\bar{y}_{k,l}(t)$ are respectively the exact and the predicted output for initial condition i, position l, and time step t, where $k \in [1, N]$, $l \in [1, L]$ and $t \in [1, T]$. We observe that the error ε decreases as the bond dimension increases, and, as expected, for MPO bond dimension $D_W \geq 8$ the error goes down to 0 at any time, clearly indicating that the evolution is predicted exactly. As we introduce noise in the training samples the accuracy of the prediction decreases, however, for this particular problem, the algorithm still predicts the correct evolution for error rates up to 20% in the training set provided that the sample size is large enough. This is shown in Fig.3(b) where the continuous blue line is computed for $M = 3000$ training data, and $\epsilon_r = 0.2$, while the red-dashed line is for $M = 7000$ and $\epsilon_r = 0.2$. In the second case the

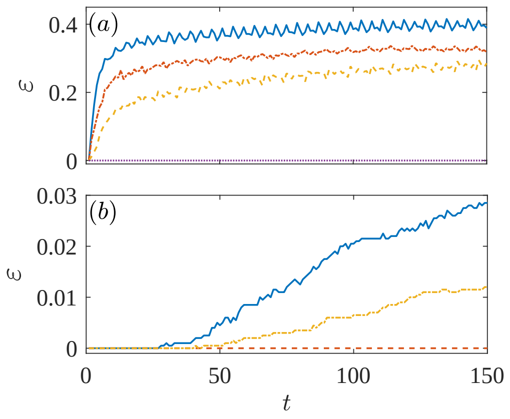

FIG. 3: Average error ε between exact and predicted evolution, computed using Eq.(14), versus time for different MPO bond dimensions D_W (a), or for different percentage of errors in training data and training sample size ϵ_r (b). In (a) $M = 7000$ while $D_W = 5$, 6, 7 or 8 respectively for the blue continuous line, red dot-dashed line, yellow dashed line and purple dotted line. In (b) $D_W = 8$ while $M = 3000$ and $\epsilon_r = 0.2$ for the blue continuous line, $M = 7000$ and $\epsilon_r = 0.2$ for the red dashed line, and $M = 7000$ and $\epsilon_r = 0.4$ for the yellow dot-dashed line. Other parameters are $N = 100$, $k_{max} = 20$, $\epsilon_t = 10^{-5}$ and $\alpha = 0.001$.

prediction is still perfect. In Fig.3(b) we also show that a training data size of $M = 7000$, if the error in the training data is even larger, for instance $\epsilon = 0.4$ as in the yellow dot-dashed line, the error ε increases in time.

1. *Comparison to conditional random fields*

The evolution of cellular automata can also be studied with conditional random fields (CRF) [48]. Such method aims to best approximate a conditional probability distribution based on a set of chosen features in the training set (see Appendix C for a more detailed description of this method). Such feature can be non-local, thus allowing to take into account long distance correlations (or contextuality), and for this reason the method is used, for example, for natural language processing. We find that while the evolution of elementary automata can also be exactly predicted using CRF for the sequence to sequence prediction, there are important differences between the CRF method and the MPO algorithm. For CRF to perform well, one needs to choose the correct features. For example, for the case of rule 153, one needs to use bi-gram features based on the value of the local element of the sequence and the element just to its right. For the long-range rule 153, one should use bi-gram features based on the local element and the one at the particular distance j to obtain exact evolution. For more general elementary rules, like rule 18 or rule 30, one needs to use a tri-gram with features based on the local element and the ones to

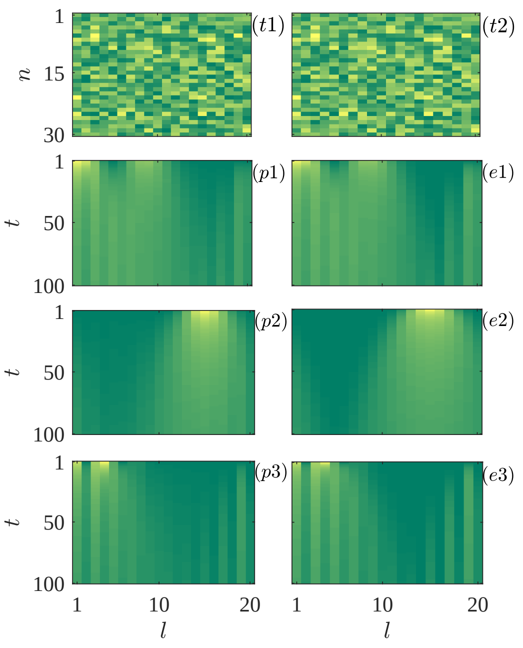

FIG. 4: Portion of the training data for input (t1) and output (t2). Each of the n rows is an input (t1) or the corresponding output (t2) sequence. The input sequence is chosen from uniformly distributed random numbers between 0 and 1 such that their sum is 1. The output sequence is computed using Eq.(15). Panels (p1-p3) and (e1-e3) represent the evolution in time of an initial sequence: the left column panels (p1-p3) show the predicted evolution from the MPO algorithm, while the right colum panels (e1-e3) show the exact evolution. The pairs (p1,e1), (p2,e2) and (p3,e3) correspond each to a different initial condition. Other parameters are $D_W = 20$, $M = 20000$, $k_{max} = 20$, $\varepsilon_t = 10^{-5}$ and $\alpha = 0.001$.

its right and left. However, in general, it is difficult (or not possible), to have a priori knowledge of the features needed for the algorithm to work well.

The MPO based algorithm instead has the advantage that, provided the auxiliary dimension D_W is large enough, it can figure which are the most relevant features, whether the element to the right, or to the left, or both, or at a certain, large, distance. This flexibility is what allows MPS and MPO based methods to be so successful in the field of many body quantum system, and in this case, to give perfect prediction with no apriori knowledge of the problem.

B. Coupled nonlinear maps

After having studied cellular automata, we now consider a coupled nonlinear map. In particular we study a nonlinear diffusive evolution with next-nearest neighbor coupling of a probability distribution, for which the sequences are made of real numbers between 0 and 1 whose sum is equal to 1. In this case, the mapping described in Eq.(4) allows us to convert a certain probability distribution to an MPS of bond dimension $D = 1$. In this case, in juxtaposition with the previous study on cellular automata, we use periodic boundary conditions. The evolution is given by

$$\begin{aligned} P_{l,t+1} = & P_{l,t} \\ & + g_1/2\left[(P_{l-1,t})^{m_1} + (P_{l+1,t})^{m_1} - 2\,(P_{l,t})^{m_1}\right] \\ & + g_2/2\left[(P_{l-2,t})^{m_2} + (P_{l+2,t})^{m_2} - 2\,(P_{l,t})^{m_2}\right] \quad (15) \end{aligned}$$

where t is a natural number indicating the time step. The other parameters of the evolution are g_1 and g_2, the diffusion rates to, respectively, the nearest and next nearest sites, and m_1, m_2, the exponents which make the diffusion nonlinear. For the evolution we use initial conditions for which $P_{l,t} \in [0,1]$ up to the times studied.

Similarly to the case of cellular automata, we train the MPO by choosing random input sequences and their corresponding output sequences. In Fig.4 we show, similarly to Fig.2, samples of input and output training data, panels ($t1$) and ($t2$), and the comparison between the predicted evolutions, panels ($p1 - p3$), and exact ones, panels ($e1 - e3$), for three different initial conditions (chosen between one hundred which are used for further calculations). The initial conditions we use are functions of the position with different number of peaks, each of different variance. This choice is very different from the training data (for which the input sequences were chosen randomly), and also it allows one to have a qualitative understanding of the dynamics. We have in fact chosen the initial conditions as $P_{l,t=1} = [1+\cos[2\pi l\lambda/L)]\exp\{-[(l-l_0)^2]/(2v^2)\}/\Gamma$ where λ is an integer number uniformly chosen between 1 and 5, the position l_0, chosen uniformly between all the site, is the location of a gaussian envelope of standard deviation v (a real number picked uniformly between 1 and 5) and Γ is a normalization which ensures that the sum of all values at all the sites gives 1. The evolution of the three different initial conditions in Fig.2(p1-p3,e1-e3) shows a peculiar diffusive dynamics.

To gain more insight in the prediction we compute the average position $\langle l\rangle = \sum_l P_{l,t} l$, variance $\Delta_l = \left(\sum_l l^2\, P_{l,t}\right) - \langle l\rangle$ and total probability $P_T = \sum_l P_{l,t}$. These quantities are depicted in Fig.5, respectively in panels (a), (b) and (c). In each panel the dotted black lines represent the exact result, the blue continuous line the prediction for a bond dimension $D_W = 5$ and the red dashed line for $D_W = 20$. We observe that the accuracy significantly increases as the bond dimension changes from $D_W = 5$ to $D_W = 20$.

While in Fig.4,5 we observe that the dynamics is well predicted by the MPO algorithm, to be more quantitative, we compute the average error rate as a function of time for different bond dimensions D_W, panel (a), and for different number of training data M. The results are represented in the two panels of Fig.6. In Fig.6(a), the bond dimension varies between $D_W = 5$ (continuous blue

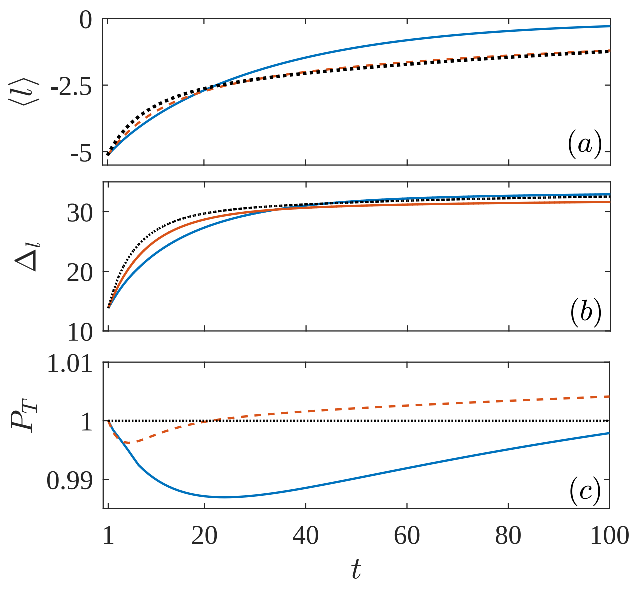

FIG. 5: (a) Average position $\langle l\rangle$, (b) standard deviation Δ_l and (c) total probability P_T versus time for $D_W = 5$, blue continuous line, and $D_W = 20$, red dashed line. The exact solution is represented by the black dotted line. Other parameters are $N = 100$, $M = 20000$, $k_{max} = 20$, $\varepsilon_t = 10^{-5}$ and $\alpha = 0.001$.

line) to $D_W = 10$ (dashed red line) and $D_W = 20$ (dot-dashed yellow line), while the size of the training set is kept to $M = 20000$. In Fig.6(b) instead, we keep the same bond dimension $D_W = 20$ and we vary the training data set size between $M = 5000$ (continuous blue line), $M = 10000$ (dashed red line) and $M = 20000$ (dot-dashed yellow line). We observe that the error decreases when both D_W and M increase, and ε can be as low as 5/1000.

1. *Comparison to bidirectional LSTM neural networks*

Long short-term memory cells are very useful building blocks for recurrent neural networks [49, 50]. Thanks to their ability of remembering and forgetting information passed to them, they allow to grasp relevant long-range correlations (see Appendix D for a more detailed description of this method). They are thus used for natural language processing, to classify, to predict time series and more. Here we consider a bidirectional network [51], in which information is passed from left to right and from right to left, with LSTM cells. Such a single layer network, despite using as many parameters as the MPO algorithm, is not able to give accurate results for the evolution. This is shown in Fig.6(c), where the error ε for the single layer network, blue continuous line, is plotted against time. We then tried a double layer network (which has about twice as many parameters) [52, 53], and the error was significantly decreased, red-dotted line in

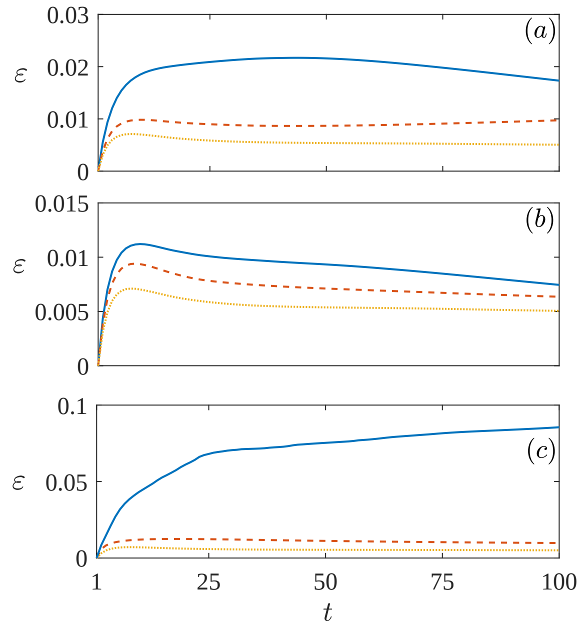

FIG. 6: Average error ε between exact and predicted evolution, computed using Eq.(14), versus time for different MPO bond dimensions D_W (a), or for different number of training data M (b). In (a) $M = 20000$ while $D_W = 5$, 10 or 20 respectively for the blue continuous line, red dashed line and yellow dotted line. In (b) $D_W = 20$ while $M = 5000$, 10000 or 20000 respectively for the blue continuous line, red dashed line and yellow dotted line. Other parameters are $N = 100$, $k_{max} = 20$, $\epsilon_t = 10^{-5}$ and $\alpha = 0.001$. (c) Comparison with bidirectional LSTM neural networks. Blue continuous curve is the error for a single layer bidirectional network while red-dashed curve is for a double layer bidirectional network. The yellow dotted line is the error for the MPO algorithm with $D_W = 20$. All three algorithms have been trained with $M = 20000$ samples and $N = 100$.

Fig.6(c), but still larger than the one obtained with the MPO algorithm. This shows that, for this task, the MPO algorithm is more accurate than our implementation of recurrent neural networks using LSTM cells.

C. Classification

It has been previously shown that algorithms based on MPS can be useful for classification tasks [26, 28, 29]. Here we show how the MPO model can also be used for such purpose. In order to explain how to do so effectively, we consider a well known example which is that of categorizing pictures of handwritten digits from the MNIST data set [54]. In this problem, handwritten digits from 0 to 9 are converted in gray scale pictures with 28×28 pixels, and the data set is composed of 60000 training images and 10000 testing images, each associated to one of the ten possible digits. In order to threat this problem we convert each gray scale figure to a sequence of real numbers between 0 and 1 by evaluating how dark each pixel is. Note that this portion of the treatment of the problem is completely analogous to [26]. From these input sequences we generate input MPSs which will be used for the training of the MPO. The output is instead composed of only ten possible digits, while the MPO algorithm can return one of the d^L possible sequences, where $d = 2$. We then write each possible output as a one hot vector of size L with the care that the element of the vector which is equal to 1 is in positions of the vector which span, more or less evenly, between the first and the last site [59, 60]. By doing so, we have ten possible different vector output, which we can readily write as bond dimension $D = 1$ MPS, as in the previous example for cellular automata. We shall stress that in this problem the input sequence is composed of real numbers while the output is made of integer numbers. This exemplifies that the different nature of the input and output sequences is not an obstacle to the functioning of the algorithm.

We then use the MNIST training data set (60000 images) and the validation set (10000 images). We find that by using a bond dimension $D = 10$ we obtain 93.9% accuracy on the training data set and 94.0% on the validation set. For $D = 20$ the results improve to 97.6% accuracy for the training data set and 97.2% accuracy in the testing. In both cases we have used regularizing coefficient $\alpha = 0.001$ and maximum number of sweeps $k_{max} = 10$. These results are comparable to [26].

IV. CONCLUSIONS

We have presented an algorithm based on matrix product operators, MPOs, for sequence to sequence prediction. In this work we have presented the main algorithm and, to show its versatility, effectiveness and robustness, we have applied it to various examples ranging from evolution of cellular automata, coupled maps and also for classification.

Two important aspects of using MPOs, and hence MPS for learning should be stressed now. First, the code can accept input and output sequences which are probabilistic. In this case, the input and output probabilities are each converted to matrix product states with a bond dimension typically larger than 1. These input and output matrix product states can then be readily used to train the MPO. The other noteworthy aspect, which should be further investigated, is that the algorithm returns, not a mere output sequence, but a matrix product state which is then converted to an output sequence, and such matrix product state contains information on the building up of correlations during the evolution.

Our comparison of the MPO algorithm with CRF shows that the MPO algorithm does not need to know the relevant features which are important to describe the conditional probabilities, but that it is able to find them or approximate them at best, within the limititation of the size of the matrices used. For the tasks studied, the predictions of the MPO algorithm where much more accurate than our implementations of a bidirectional LSTM neural network. One current defect of the MPO algorithm is that the training phase is much slower, in terms of computational time, than CRF and LSTM neural networks. This can however be significantly boosted by different methods, including the use of GPUs and/or by implementing a translational invariant solution.

In the future we aim to extend and apply such algorithm to problems related to natural language processing, as these problems are inherently one dimensional, a situation for which matrix product states and operators algorithms perform at their best.

Various improvements are required such as allowing for the study of sequences of unknown lengths or for which the length of the input is different from that of the output. Fundamental questions regarding the effectiveness of training an MPO depending on the underlying dynamics also need to be addressed. These will be the subjects of follow-up works.

Acknowledgments

We acknowledge support from the SUTD-MIT International Design Centre. D.P. acknowledges fruitful discussions with F.C. Binder, N. Lim and S. Lin.

[1] M. Schuld, I. Sinayskiy, and F. Petruccione, Contemporary Physics **56**, 172 (2014).
[2] J. Adcock, E. Allen, M. Day, S. Frick, J. Hinchliff, M. Johnson, S. Morley-Short, S. Pallister, A. Price, and S. Stanisic, arXiv:1512.02900 (2015).
[3] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, Nature **549**, 195 (2017).
[4] S. V. Kalinin, B. G. Sumpter, and R. K. Archibald, Nat. Mater. **14**, 973 (2015).
[5] S.S. Schoenholz, E.D. Cubuk, D.M. Sussman, E. Kaxiras, and A.J. Liu, Nature Phys **12**, 469 (2016).
[6] L. Wang, Phys. Rev. B **94**, 195105 (2016).
[7] J. Carrasquilla, and R.G. Melko, Nature Physics **13**, 431 (2017).
[8] P. Zhang, H. Shen, and H. Zhai, Phys. Rev. Lett. **120**, 066401 (2018).
[9] E. P. van Nieuwenburg, Y.-H. Liu, and S. D. Huber, Nat. Phys. **13**, 435 (2017).
[10] P. Broecker, J. Carrasquilla, R. G. Melko, and S. Trebst, Scientific Reports **7**, 8823 (2017).
[11] K. Chng, J. Carrasquilla, R. G. Melko, and E. Khatami, Phys. Rev. X **7**, 031038 (2017).
[12] Y. Zhang, and E.-A. Kim, Phys. Rev. Lett. 118, 216401 (2017).

[13] L.F. Arsenault, A. Lopez-Bezanilla, O.A. von Lilienfeld, and A.J. Millis, Phys. Rev. B **90**, 155136 (2014).
[14] L.-F. Arsenault, O. A. von Lilienfeld, and A. J. Millis, arXiv:1506.08858 (2015).
[15] G. Torlai and R. G. Melko, Phys. Rev. B 94, 165134 (2016).
[16] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko, arXiv:1601.02036.
[17] J. Liu, Y. Qi, Z. Y. Meng, and L. Fu, Phys. Rev. B **95**, 041101 (2017).
[18] L. Huang and L. Wang, Phys. Rev. B **95**, 035105 (2017).
[19] K.-I. Aoki and T. Kobayashi, Mod. Phys. Lett. B, 1650401 (2016).
[20] G. Carleo, and M. Troyer, Science **355**, 602 (2017).
[21] Y. Nomura, A.S. Darmawan, Y. Yamaji, and M. Imada, Phys. Rev. B **96**, 205152 (2017).
[22] S. Czischek, M. Gr̈ttner, and T. Gasenzer, arxiv:1803.08321.
[23] C. Beny, arXiv:1301.3124.
[24] P. Mehta and D. J. Schwab, arXiv:1410.3831.
[25] H.W. Lin, M. Tegmark, and D. Rolnick, Journal of Statistical Physics **168**, 1223 (2017).
[26] E.M. Stoudenmire and D.J. Schwab, Advances In Neural Information Processing Systems **29**, 4799 (2016).
[27] Z.-Y. Han, J. Wang, H. Fan, L. Wang, and P. Zhang, arxiv:1709.01662.
[28] E.M. Stoudenmire, arXiv:1801.00315 (2017).
[29] A. Novikov, M. Trofimov, and I. Oseledets, arxiv:1605.03795, ICLR (2017).
[30] C. Yang, F.C. Binder, V. Narasimhachar, M. Gu, arxiv:1803.08220.
[31] V. Pestun, J. Terilla, and Y. Vlassopoulos, arxiv:1711.01416.
[32] D.L. Deng, X. Li, and S.D. Sarma, Physical Review X **7**, 021021 (2017).
[33] J. Chen, S. Cheng, H. Xie, L. Wang, and T. Xiang, Phys. Rev. B **97**, 085104 (2018).
[34] A. Cichocki, arxiv:1403.2048.
[35] U. Schollwöck, Annals of Physics **326**, 96 (2011).
[36] B. Derrida, M.R. Evans, V. Hakim and V. Pasquier, J. Phys. A: Math. Gen. **26**, 1493 (1993).
[37] K. Krebs and S. Sandow, J. Phys. A: Math. Gen. **30**, 3165 (1997).
[38] T.H. Johnson, S.R. Clark, and D. Jaksch, Phys. Rev. E **82**, 036702 (2010).
[39] T.H. Johnson, T.J. Elliott, S.R. Clark, and D. Jaksch, Phys. Rev. Lett. **114**, 090602 (2015).
[40] S.R. White, Phys. Rev. Lett. **69**, 2863 (1992).
[41] U. Schollwöck, Rev. Mod. Phys. **77**, 259 (2005).
[42] I.P. McCulloch, J. Stat. Mech., P10014 (2007).
[43] S.R. White and A.E. Feiguin, Phys. Rev. Lett. **93**, 076401 (2004).
[44] G. Vidal, Phys. Rev. Lett. **93**, 040502 (2004).
[45] A.J. Daley, C. Kollath, U. Schollwöck, and G. Vidal, J. Stat. Mech.: Theor. Exp. P04005 (2004).
[46] F. Verstraete, J.J. Garcia-Ripoll, and J.I. Cirac, Phys. Rev. Lett. **93**, 207204 (2004).
[47] A.J. Daley, Adv. Phys. **63**, 77 (2014).
[48] J. Lafferty, A. McCallum, and F. Pereira, Proc. 18th International Conf. on Machine Learning. Morgan Kaufmann. 282 (2001).
[49] S. Hochreiter and J. Schmidhuber, Neural Computation **9**, 1735 (1997).
[50] F.A. Gers, J. Schmidhuber, and F. Cummins, Neural Computation **12**, 2451 (2000).
[51] A. Graves and J. Schmidhuber, IEEE International Joint Conference on Neural Networks Proceedings, 2047 (2005).
[52] H. Sak, A. Senior, and F.Beaufays, ISCA, 338 (2014).
[53] Y. Zhang, G. Chen, D. Yu, K. Yaco, and S. Khudanpur, J. Glass, ICASSP, 5755 (2016)
[54] J.C. Christopher, Y.L. Burges, C. Cortes, MNIST handwritten digit database, http://yann.lecun.com/exdb/mnist/
[55] S. Wolfram, Rev. Mod. Phys. **55**, 601 (1983).
[56] If $l + j > L$ then the value does not change.
[57] T. Prosen, and B. Buca, J. Phys. A: Math. Theor. **50**, 395002 (2017).
[58] Note also that matrix product operators for unitary evolutions are quantum cellular automata [61].
[59] In our calculations the sites with the 1 where sites $l = 1+78\delta$ where δ is one of the possible digits $0,\ 1, \dots, 9$. The choice of 78 is due to the size of the images used which have $28 \times 28 = 784$ pixels, which are then divided by 10 which is the number of possible digits in the classification.
[60] More precise results could be obtained by optimizing the location of the ones.
[61] J.I. Cirac, D. Perez-Garcia, N. Schuch, F. Verstraete, J. Stat. Mech. 083105 (2017).

Appendix A: Iterative minimization

Here we describe in more detail how we use Eq.(10) to compute the optimal $W^{\sigma_l,\tau_l}_{b_{l-1},b_l}$. As commonly done in many MPS algorithms for quantum mechanical systems, it is convenient to group the product of the tensors in various parts as depicted in Fig.A1. From the simple derivative in Eq.(10), the tensor $W^{\sigma_l,\tau_l}_{b_{l-1},b_l}$ will be the result of the linear equation

$$\sum_{b_{l-1},b_l,\sigma_l} \left(F^{\sigma_l,\sigma'_l}_{b_{l-1},b_l,b'_{l-1},b'_l} + \alpha C_{b_{l-1},b'_{l-1}} D_{b_l,b'_l} \right) W^{\sigma_l,\tau_l}_{b_{l-1},b_l} = U^{\sigma'_l,\tau_l}_{b'_{l-1},b'_l}. \quad \text{(A1)}$$

This equation is described graphically in Fig.A1(a) where the elementary tensors $W^{\sigma_l,\tau_l}_{b_{l-1},b_l}$, $X^{\sigma_l}_{i,a_{l-1},a_l}$ and $Y^{\tau_l}_{i,c_{l-1},c_l}$ are described in Fig.A1(b) and where the dotted line is used to indicate the trace operation. The tensor $F^{\sigma_l,\sigma'_l}_{b_{l-1},b_l,b'_{l-1},b'_l}$ is composed of the product of tensors to the left and to the right of the site of interest

$$F^{\sigma_l,\sigma'_l}_{b_{l-1},b_l,b'_{l-1},b'_l} = \sum_{\substack{i,a_{l-1},\\ c_{l-1},a_l,c_l}} A_{i,a_{l-1},b_{l-1},b'_{l-1},c_{l-1}} \times B_{i,a_l,b_l,b'_l,c_l} X^{\sigma_l}_{i,a_{l-1},a_l} X^{\sigma'_l}_{i,c_{l-1},c_l} \quad \text{(A2)}$$

with

$$A_{i,a_{l-1},b_{l-1},b'_{l-1},a'_{l-1}} = \sum_{\substack{a_k,b_k,b'_k,a'_k,\\ \sigma_k,\tau_k,\sigma'_k}} \prod_{k<l} \left(W^{\sigma_k,\tau_k}_{b_{k-1},b_k} W^{\sigma'_k,\tau_k}_{b'_{k-1},b'_k} X^{\sigma_k}_{i,a_{k-1},a_k} X^{\sigma'_k}_{i,a'_{k-1},a'_k} \right) \quad \text{(A3)}$$

and

$$B_{i,a_l,b_l,b'_l,a'_l} = \sum_{\substack{a_k,b_k,b'_k,a'_k,\\ \sigma_k,\tau_k,\sigma'_k}} \prod_{k>l} \left(W^{\sigma_k,\tau_k}_{b_{k-1},b_k} W^{\sigma'_k,\tau_k}_{b'_{k-1},b'_k} X^{\sigma_k}_{i,a_{k-1},a_k} X^{\sigma'_k}_{i,a'_{k-1},a'_k} \right) \quad \text{(A4)}$$

and where $A_{i,a_0,b_0,b'_0,a'_0} = A_{i,a_L,b_L,b'_L,a'_L} = 1$. These tensors are depicted in Fig.A1(c). Note also that here and in the following, the sum over the tensor indices are done only if the same index label appears in two different tensors. From the graphical representation of our equations in Fig.A1, this would mean that the sum is done over the indices for lines joining different tensors. For Eq.(A1) we also need to compute the tensors C, D and U. Tensor U is given by

$$U^{\sigma'_l,\tau_l}_{b'_{l-1},b'_l} = \sum_{\substack{i,a'_{l-1},c_{l-1},\\ a'_l,c_l}} L_{i,a'_{l-1},b'_{l-1},c_{l-1}} R_{i,a'_l,b'_l,c_l} X^{\sigma'_l}_{i,a'_{l-1},a'_l} Y^{\tau_l}_{i,c_{l-1},c_l} \quad \text{(A5)}$$

where

$$L_{i,a'_{l-1},b'_{l-1},c_{l-1}} = \sum_{\substack{a'_k,b'_k,c_k\\ \sigma_k,\tau_k}} \prod_{k<l} \left(W^{\sigma_k,\tau_k}_{b'_{k-1},b'_k} X^{\sigma_k}_{i,a'_{k-1},a'_k} Y^{\tau_k}_{i,c_{k-1},c_k} \right) \quad \text{(A6)}$$

and

$$R_{i,a'_l,b'_l,c_l} = \sum_{\substack{a'_k,b'_k,c_k\\ \sigma_k,\tau_k}} \prod_{k>l} \left(W^{\sigma_k,\tau_k}_{b'_{k-1},b'_k} X^{\sigma_k}_{i,a'_{k-1},a'_k} Y^{\tau_k}_{i,c_{k-1},c_k} \right). \quad \text{(A7)}$$

The U tensor is depicted in the right-hand side of the equation in Fig.A1(a). C and D, are instead given by

$$C_{b_{l-1},b'_{l-1}} = \sum_{\substack{b_k,b'_k,\\ \sigma_k,\tau_k}} \prod_{k<l} \left(W^{\sigma_k,\tau_k}_{b_{k-1},b_k} W^{\sigma_k,\tau_k}_{b'_{k-1},b'_k} \right) \quad \text{(A8)}$$

and

$$D_{b_l,b'_l} = \sum_{\substack{b_k,b'_k,\\ \sigma_k,\tau_k}} \prod_{k>l} \left(W^{\sigma_k,\tau_k}_{b_{k-1},b_k} W^{\sigma_k,\tau_k}_{b'_{k-1},b'_k} \right). \quad \text{(A9)}$$

Computing the products of the tensors in such groupings allows to speed up the calculations of the multiplications of tensors by growing iteratively these block of tensors. Note also that the number of tensors required for these calculations scales linearly with the number of training data N. The memory requirements can be mitigated by computing the products when needed.

Appendix B: Example of matrix product operator rewriting of cellular automata evolution

The cellular automata considered in the main text, rule 153, is such that the value at one site at the next iteration only depends on a site to its right and it is independent from those to the left. For this reason the evolution can be exactly described by a matrix product operator of bond dimension $D = 2$. However, in general, the 256 rules of cellular automata by Wolfram [55], can be exactly described by an MPO of bond dimension $D = 4$. We here describe two examples, rules 18 and 30. Rule 18 is also used in the main text as, also for fixed boundary conditions, can produce evolutions with long periods. Rule 30 instead is chaotic, but not within fix boundary conditions considered here, and hence, with such boundary conditions the evolution becomes regular in a short time (periodic boundary conditions can be also considered with MPOs but in this case the bond dimension D_W could be the square of the fix boundaries condition case).

Since for both cases, rule 18 and rule 30, the maximum needed bond dimension is $D = 4$, there will be 5 types of tensors. Those on the first or last site, those on the

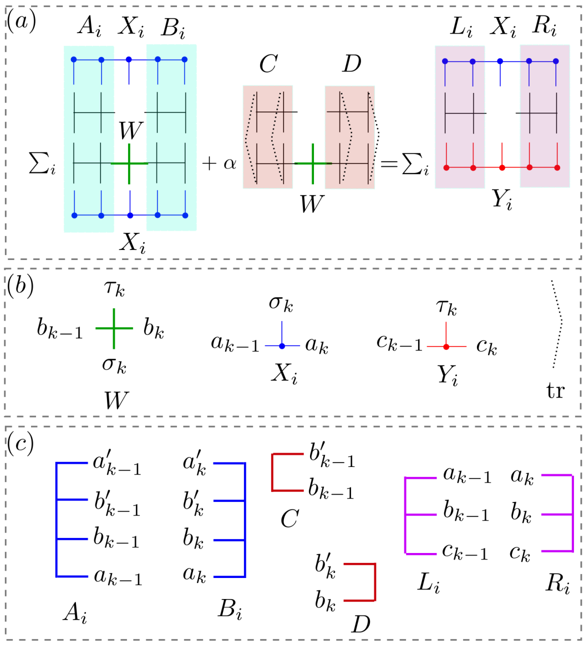

FIG. A1: (a) Graphical representation of the linear equation (A1) in which every tensor is a structure with a different number of 'legs' depending on the number of tensor indices (for lighter notation we have removed the indices l, σ_l and τ_l). The vertical lines correspond to physical indices (like σ_l or τ_l) while the horizontal lines to auxiliary indices (like a_l, b_l or c_l). Joined legs are summed over. In panel (b) we show the elementary tensors $W^{\sigma_l,\tau_l}_{b_{l-1},b_l}$, $X^{\sigma_l}_{i,a_{l-1},a_l}$ and $Y^{\tau_l}_{i,c_{l-1},c_l}$ while in panel (c) we show the graphical representation of intermediate tensors which are computed to evaluate more efficiently Eq.(A1).

second and second last site and the other tensors in the middle of the chain. For rule 30 we have, for the first site

$$W^{0,0}_{b_0,b_1}=[1\;0]\,,\quad W^{0,1}_{b_0,b_1}=[0\;0]\,,$$
$$W^{1,0}_{b_0,b_1}=[0\;0]\,,\quad W^{1,1}_{b_0,b_1}=[0\;1]\,,\tag{B1}$$

for the second site,

$$W^{0,0}_{b_1,b_2}=\begin{bmatrix}1&0&0&0\\0&1&0&0\end{bmatrix},\quad W^{0,1}_{b_1,b_2}=\begin{bmatrix}0&1&0&0\\1&0&0&0\end{bmatrix},$$
$$W^{1,0}_{b_1,b_2}=\begin{bmatrix}0&0&0&0\\0&0&1&1\end{bmatrix},\quad W^{1,1}_{b_1,b_2}=\begin{bmatrix}0&0&1&1\\0&0&0&0\end{bmatrix},\tag{B2}$$

for the intermediate sites

$$W^{0,0}_{b_{l-1},b_l}=\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&1&0&0\\0&0&0&0\end{bmatrix},\quad W^{0,1}_{b_{l-1},b_l}=\begin{bmatrix}0&1&0&0\\0&0&0&0\\1&0&0&0\\0&0&0&0\end{bmatrix},$$
$$W^{1,1}_{b_{l-1},b_l}=\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&1&1\end{bmatrix},\quad W^{1,0}_{b_{l-1},b_l}=\begin{bmatrix}0&0&0&0\\0&0&1&1\\0&0&0&0\\0&0&0&0\end{bmatrix},\tag{B3}$$

for the before last site

$$W^{0,0}_{b_{L-2},b_{L-1}}=\begin{bmatrix}1&0\\0&0\\0&1\\0&0\end{bmatrix},\quad W^{0,1}_{b_{L-2},b_{L-1}}=\begin{bmatrix}0&1\\0&0\\1&0\\0&0\end{bmatrix},$$
$$W^{1,1}_{b_{L-2},b_{L-1}}=\begin{bmatrix}0&0\\0&0\\0&0\\1&1\end{bmatrix},\quad W^{1,0}_{b_{L-2},b_{L-1}}=\begin{bmatrix}0&0\\1&1\\0&0\\0&0\end{bmatrix},\tag{B4}$$

and for the last site

$$W^{0,0}_{b_{L-1},b_L}=\begin{bmatrix}1\\0\end{bmatrix},\quad W^{0,1}_{b_{L-1},b_L}=\begin{bmatrix}0\\0\end{bmatrix},$$
$$W^{1,0}_{b_{L-1},b_L}=\begin{bmatrix}0\\0\end{bmatrix},\quad W^{1,1}_{b_{L-1},b_L}=\begin{bmatrix}0\\1\end{bmatrix}.\tag{B5}$$

For rule 18 we have, for the first site

$$W^{0,0}_{b_0,b_1}=[1\;0]\,,\quad W^{0,1}_{b_0,b_1}=[0\;0]\,,$$
$$W^{1,0}_{b_0,b_1}=[0\;0]\,,\quad W^{1,1}_{b_0,b_1}=[1\;1]\,,\tag{B6}$$

for the second site,

$$W^{0,0}_{b_1,b_2}=\begin{bmatrix}1&0&0&0\\0&1&0&0\end{bmatrix},\quad W^{0,1}_{b_1,b_2}=\begin{bmatrix}0&1&0&0\\1&0&0&0\end{bmatrix},$$
$$W^{1,0}_{b_1,b_2}=\begin{bmatrix}0&0&1&1\\0&0&1&1\end{bmatrix},\quad W^{1,1}_{b_1,b_2}=\begin{bmatrix}0&0&0&0\\0&0&0&0\end{bmatrix},\tag{B7}$$

for the intermediate sites

$$W^{0,0}_{b_{l-1},b_l}=\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&1&0&0\\0&0&0&0\end{bmatrix},\quad W^{0,1}_{b_{l-1},b_l}=\begin{bmatrix}0&1&0&0\\0&0&0&0\\1&0&0&0\\0&0&0&0\end{bmatrix},$$
$$W^{1,1}_{b_{l-1},b_l}=\begin{bmatrix}0&0&0&0\\0&0&1&1\\0&0&0&0\\0&0&1&1\end{bmatrix},\quad W^{1,0}_{b_{l-1},b_l}=\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix},\tag{B8}$$

for the before last site

$$W^{0,0}_{b_{L-2},b_{L-1}}=\begin{bmatrix}1&0\\0&0\\0&1\\0&0\end{bmatrix},\quad W^{0,1}_{b_{L-2},b_{L-1}}=\begin{bmatrix}0&1\\0&0\\1&0\\0&0\end{bmatrix},$$
$$W^{1,1}_{b_{L-2},b_{L-1}}=\begin{bmatrix}0&0\\1&1\\0&0\\1&1\end{bmatrix},\quad W^{1,0}_{b_{L-2},b_{L-1}}=\begin{bmatrix}0&0\\0&0\\0&0\\0&0\end{bmatrix},\tag{B9}$$

and for the last site

$$W^{0,0}_{b_{L-1},b_L}=\begin{bmatrix}1\\0\end{bmatrix},\quad W^{0,1}_{b_{L-1},b_L}=\begin{bmatrix}0\\0\end{bmatrix},$$
$$W^{1,0}_{b_{L-1},b_L}=\begin{bmatrix}0\\0\end{bmatrix},\quad W^{1,1}_{b_{L-1},b_L}=\begin{bmatrix}0\\1\end{bmatrix}.\tag{B10}$$

Appendix C: Conditional random fields

Conditional random fields (CRF) [48] is a popular model for structured prediction and has been widely used in natural language processing tasks such as part-of-speech (POS) tagging and named entity recognition [A1].

Given an input sequence $\vec{x} = (x_1, x_2, \cdots, x_L)$, where L is the number of elements, the probability of predicting a possible output sequence $\vec{y} = (y_1, y_2, \cdots, y_L)$ is defined as

$$p(\vec{y}_i|\vec{x}_i) = \frac{\exp(\vec{w}^{\mathbf{T}} \vec{f}(\vec{x}_i, \vec{y}_i))}{Z(\vec{x})} \tag{C1}$$

$$Z(\vec{x}_i) = \sum_{\vec{y}_k} \exp(\vec{w}^{\mathbf{T}} \vec{f}(\vec{x}_i, \vec{y}_k)) \tag{C2}$$

where $\vec{f}(\vec{x}_i, \vec{y}_i)$ is the feature vector (carefully chosen by the user depending on the problem), $\vec{w}$ is the weight vector consisting of parameters for the features, and $Z(\vec{x}_i)$ is the partition function used for normalization. $\mathbf{T}$ stands for transpose of a matrix/vector. The number of parameters is the number of features, which is the size of the feature vector. We aim to minimize the negative log-likelihood with L_2 regularization

$$\mathcal{L}(\vec{w}) = -\sum_i \log p(\vec{y}_i|\vec{x}_i) + \lambda \vec{w}^{\mathbf{T}} \vec{w} \tag{C3}$$

where $(\vec{x}_i, \vec{y}_i)$ is the i-th training instance and λ is the L_2 regularization coefficient. Since the objective function is convex, we can make use of the L-BFGS [A2] algorithm to optimize it. The gradient with respect to each parameter w_k is calculated by setting

$$\frac{\partial \mathcal{L}}{\partial w_k} = 0 \tag{C4}$$

Appendix D: Bidirectional long short-term memory network(LSTM)

Recurrent neural networks (RNNs) are a family of neural networks for sequence-to-sequence task. Different from CRF, it takes a sequence of vectors $\mathbf{X}_i = (\vec{\mathrm{x}}_{i,1}, \vec{\mathrm{x}}_{i,2}, \cdots, \vec{\mathrm{x}}_{i,L})$ as input and return a sequence of hidden output vectors $\mathbf{H} = \left(\vec{\mathrm{h}}_{i,1}, \vec{\mathrm{h}}_{i,2}, \cdots, \vec{\mathrm{h}}_{i,L}\right)$. In the case discussed in Sec. II.B of the main paper, a sequence $\vec{x}_i$ becomes a sequence of vectors $\mathbf{X}_i = \left(\left(x_{i,1}, \sqrt{1 - x_{i,1}^2}\right), \cdots, \left(x_{i,L}, \sqrt{1 - x_{i,L}^2}\right)\right)$. Theoretically, traditional RNNs can learn long dependencies as the output at each position depends on information from previous positions, although an effective implementation may be difficult [A3]. Long short-term memory networks [49] are designed to provide an effective solution by using a memory-cell and they have been shown to capture long-range dependencies. Several gates are used to control the proportion of the information sent to the memory

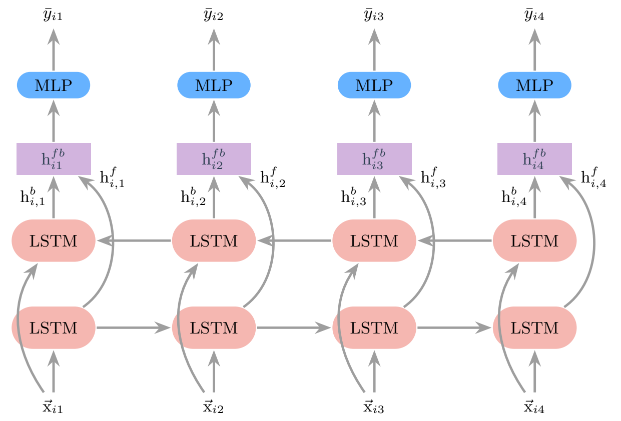

FIG. A2: Bidirectional LSTM neural network architecture.

cell and proportion of information to forget. Specifically, for a one directional LSTM cell we use the implementation

$$\vec{\mathrm{i}}_{i,l} = \sigma\left(\mathbf{W}_{\mathrm{xi}}\, \vec{\mathrm{x}}_{i,l} + \vec{\mathrm{b}}_{\mathrm{xi}} + \mathbf{W}_{\mathrm{hi}}\, \vec{\mathrm{h}}_{i,l-1} + \vec{\mathrm{b}}_{\mathrm{hi}}\right) \tag{D1}$$

$$\vec{\mathrm{f}}_{i,l} = \sigma\big(\mathbf{W}_{\mathrm{xf}}\, \vec{\mathrm{x}}_{i,l} + \vec{\mathrm{b}}_{\mathrm{xf}} + \mathbf{W}_{\mathrm{hf}}\, \vec{\mathrm{h}}_{i,l-1} + \vec{\mathrm{b}}_{\mathrm{hf}}\big) \tag{D2}$$

$$\vec{\mathrm{g}}_{i,l} = \tanh\big(\mathbf{W}_{\mathrm{xg}}\, \vec{\mathrm{x}}_{i,l} + \vec{\mathrm{b}}_{\mathrm{xg}} + \mathbf{W}_{\mathrm{hg}}\, \vec{\mathrm{h}}_{i,l-1} + \vec{\mathrm{b}}_{\mathrm{hg}}\big) \tag{D3}$$

$$\vec{\mathrm{o}}_{i,l} = \sigma\big(\mathbf{W}_{\mathrm{xo}}\, \vec{\mathrm{x}}_{i,l} + \vec{\mathrm{b}}_{\mathrm{xo}} + \mathbf{W}_{\mathrm{ho}}\, \vec{\mathrm{h}}_{i,l-1} + \vec{\mathrm{b}}_{\mathrm{ho}}\big) \tag{D4}$$

$$\vec{\mathrm{c}}_{i,l} = \vec{\mathrm{f}}_{i,l} \odot \vec{\mathrm{c}}_{i,l-1} + \vec{\mathrm{i}}_{i,l} \odot \vec{\mathrm{g}}_{i,l} \tag{D5}$$

$$\vec{\mathrm{h}}_{i,l} = \vec{\mathrm{o}}_{i,l} \odot \tanh(\vec{\mathrm{c}}_{i,l}) \tag{D6}$$

where σ is the element-wise sigmoid function and $\odot$ is the element-wise product. Each $\mathbf{W}$ matrix represents the parameters of the LSTM. $\vec{\mathrm{i}}_{i,l}, \vec{\mathrm{f}}_{i,l}, \vec{\mathrm{g}}_{i,l}$ and $\vec{\mathrm{o}}_{i,l}$ are, respectively, the input, forget, cell and output gates at position l.

However, in order to capture better the correlations both to the right and to the left of a certain position l, we use a bidirectional LSTM [51], with forward and backward LSTM pairs of cells as shown in Fig.A2.

The representation of a specific position l is given by the concatenation of the forward and backward LSTM, respectively each providing hidden output vectors $\vec{h}_{i,l}^{f}$ and $\vec{h}_{i,l}^{b}$, to give the overall forward-backward hidden output vector $\vec{h}_{i,l}^{fb} = [\vec{h}_{i,l}^{f}; \vec{h}_{i,l}^{b}]$.

In order to obtain a scalar at each time step, we use a final non-linear layer (multi-layer percerptron MLP) to map the hidden vector to a scalar value at each position l

$$\bar{y}_{i,l} = \sigma\big(\vec{w}_{\mathrm{hy}}^{\mathbf{T}}\; \vec{\mathrm{h}}_{i,l}^{fb} + b_{\mathrm{hy}}\big). \tag{D7}$$

Once the sequence of $\bar{y}_{i,l}$ is generated, the optimization of the model parameters is achieved by minimizing the error between predicted $\vec{\bar{y}}_i$ and exact sequences $\vec{y}_i$.

A two layers bidirectional LSTM neural network [52, 53], as the one used in the main text, is represented in Fig.A3.

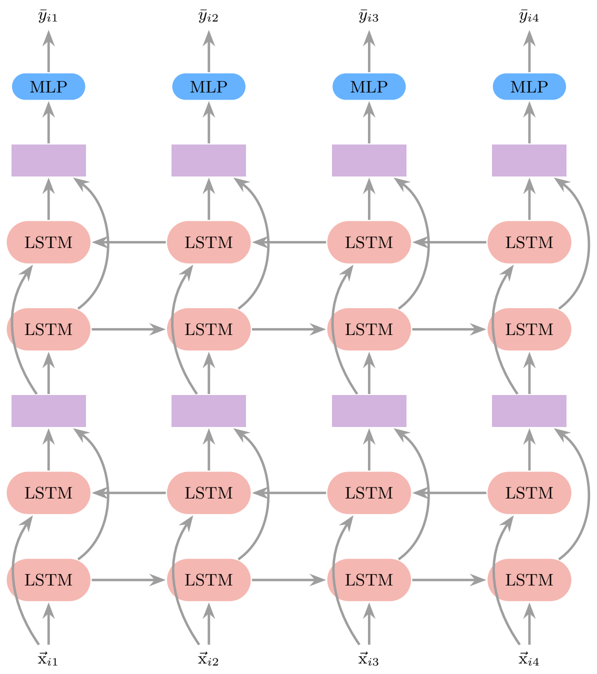

FIG. A3: Two layers bidirectional LSTM neural network architecture.

[A1] X. Ma and E. Hovy, Proceedings of ACL, 1064 (2016).
[A2] D. Liu and J. Nocedal, Mathematical programming **45**, 503 (1989).
[A3] Y. Bengio, P. Simard and P. Frasconi, IEEE transactions on neural networks **5**, 157 (1994).