RESEARCH ARTICLE

A matrix product state model for simultaneous classification and generation

Alex Mossi[1] · Bojan Žunkovič[2] · Kyriakos Flouris[3]

Abstract

Quantum machine learning (QML) is a rapidly expanding field that merges the principles of quantum computing with the techniques of machine learning. One of the powerful mathematical frameworks in this domain is tensor networks. These networks are used to approximate high-order tensors by contracting tensors with lower ranks. Initially developed for simulating quantum systems, tensor networks have become integral to quantum computing and, by extension, to QML. Drawing inspiration from these quantum methods, specifically the Matrix Product States (MPS), we apply them in a classical machine learning setting. Their ability to efficiently represent and manipulate complex, high-dimensional data makes them effective in a supervised learning framework. Here, we present an MPS model, in which the MPS functions as both a classifier and a generator. The dual functionality of this novel MPS model permits a strategy that enhances the traditional training of supervised MPS models. This framework is inspired by generative adversarial networks and is geared towards generating more realistic samples by reducing outliers. In addition, our contributions offer insights into the mechanics of tensor network methods for generation tasks. Specifically, we discuss alternative embedding functions and a new sampling method from non-normalized MPSs.

Keywords Tensor networks · MPS · GAN · Noise robustness · Outlier reduction · Quantum embeddings

1 Introduction

In recent years, quantum technologies have undergone rapid and substantial advancements, holding the promise of revolutionizing various scientific and industrial domains (Ramakrishnan et al. 2023). These advancements are closely intertwined with quantum machine learning (QML) (Suzuki et al. 2024; Hdaib et al. 2024) and tensor networks (TNs) emerging as a vital mathematical tool. Central to the concept of TNs is their ability to approximate high-dimensional tensors via memory-efficient multi-dimensional arrays (Biamonte and Bergholm 2017). Originally devised as tools for the approximation and simulation of quantum systems within the domains of many-body quantum physics (Dalzell and Brandão 2019) and condensed matter physics (Orús 2019), TNs now encompass a diverse array of fields, including data compression (Bengua et al. 2017), privacy (Pozas-Kerstjens et al. 2023), and high-dimensional PDEs (Weinan et al. 2021). Of particular interest is their application in machine learning for both supervised (Miles and Schwab 2016) and unsupervised tasks (Han et al. 2018; Cheng et al. 2021; Vieijra et al. 2022).

Prominent tensor networks in contemporary research include Projected Entangled Pair States (PEPS) (Vieijra et al. 2022), Matrix Product State (MPS) (Miles and Schwab 2016; Han et al. 2018), Locally Purified State (LPS) (Glasser et al. 2019), Multiscale Entanglement Renormalization Ansatz (MERA) (Vidal 2007), tree tensor networks (TTNs) (Cheng et al. 2019), and isometric tensor networks (Zaletel and Pollmann 2020). Of note, MPS, characterized by rank-3 tensors and sequential one-dimensional contractions, has garnered considerable attention due to its relative simplicity and versatility (Orús 2014).

✉ Alex Mossi
　alex.mossi@alumni.ethz.ch

　Bojan Žunkovič
　bojan.zunkovic@uni-lj.si

　Kyriakos Flouris
　kflouris@ethz.ch

[1]　D-MATH, ETH Zürich, Zürich, Switzerland

[2]　FRI, University of Ljubljana, Ljubljana, Slovenia

[3]　D-ITET, ETH Zürich, Zürich, Switzerland

MPSs were originally used to describe and simulate the quantum states of one-dimensional systems since they can faithfully represent quantum states featuring limited entanglement (Perez-Garcia et al. 2006; Verstraete and Cirac 2006; Hastings 2007). Such systems are notoriously challenging owing to the curse of dimensionality, which arises from the exponential growth of Hilbert spaces in such contexts (Bridgeman and Chubb 2017). Since then, MPSs have been adapted to address a wide spectrum of datasets, including two-dimensional systems (Bruognolo et al. 2017), rendering them suitable for image processing in machine learning applications. This has led to their successful application in tasks such as image classification using datasets like MNIST (LeCun et al. 2010) and Fashion MNIST (Xiao et al. 2017). While early training methods for MPSs in machine learning relied on density matrix renormalization group (DMRG) techniques (Miles and Schwab 2016), contemporary machine learning libraries provide more accessible approaches (Efthymiou et al. 2019) based on automatic differentiation (Liao et al. 2019; Francuz et al. 2023).

MPS models are typically employed for conventional classification (Miles and Schwab 2016) or as generators in unsupervised contexts (Han et al. 2018). However, the distinctive structure of MPS, combined with the characteristics of the embedding functions employed in this work, allows the use of a single model for both classification and generation tasks (Bishop 2006; Flouris and Konukoglu 2023). This enables the use of a GAN-style method to improve its generative performance without affecting its classification accuracy.

We propose a novel approach for training supervised MPSs in a GAN-style setting. The MPS serves as both a classifier and a generator, resulting in improved generative performance and a reduction in the number of outliers generated, while maintaining robust classification accuracy. To this end, we present several contributions that allow the realization of such a model while providing insights into the mechanics of tensor network methods for generation. Thus, we are broadening the utility of tensor network methods for machine learning tasks.

The first part of this work establishes the theoretical framework underlying MPS models, as described in Sects. 2.1 and 2.2. First, Matrix Product States are introduced, with a discussion of their canonical forms and why these forms are not strictly required in machine learning applications. Second, this section provides an overview of embedding functions, which are essential tools for transforming input data into representations compatible with MPS structures. Third, alternative embedding functions are explored, and techniques that simplify the computation of marginalized probability density functions (PDFs) over single variables, $p(x_i)$, while avoiding the evaluation of high-dimensional integrals, are introduced. Fourth, an exact sampling procedure for nonnormalized MPS is discussed, which removes the need for iterative sampling methods like Markov chain Monte Carlo (MCMC).

The second part of this work focuses on practical applications of MPS models in machine learning, as outlined in Sect. 2.3. The dual capability of MPS for simultaneous classification and generation tasks is emphasized, with particular attention to the GAN-style framework that serves as the foundation of this study. Key implementation challenges, such as managing exploding and vanishing values during tensor contractions, are addressed through specific techniques designed to stabilize training. Additionally, the use of MPS as a generator is examined, highlighting its ability to provide a latent space representation of the input data, which enables meaningful insights into its structure. The impact of perturbations within the embedded space on classification performance is also analyzed, with comparisons made across different embedding functions.

Finally, Sect. 3 presents a detailed evaluation of our approach, including experimental results that validate the performance of GAN-style training and analyze the properties of the embedding functions. Section 4 then summarizes the findings and discusses potential directions for future research.

2 Methods

2.1 Matrix product state and embedding functions

A Matrix Product State can be formally defined as a collection of $n - 2$ rank-3 tensors $\{A_i^{\alpha_{i-1}, \alpha_i, d_i}\}_{i \in \{2, \ldots n-1\}}$ and two rank-2 tensors $\{A_1^{\alpha_1, d_1}, A_n^{\alpha_{n-1}, d_n}\}$, called sites, that can be contracted sequentially, resulting in the decomposition of a rank-n tensor W, as described in the following equation:

$$
W^{d_1, d_2, \ldots, d_n} = \\
\sum_{\{\alpha_i\}} A_1^{\alpha_1, d_1} A_2^{\alpha_1 \alpha_2, d_2} \cdots A_j^{\alpha_{j-1} \alpha_j, d_j} \cdots A_N^{\alpha_{N-1}, d_N}. \tag{1}
$$

The physical dimension d corresponds to the dimension of indices d_i, and its role will be discussed in Sect. 2.1.2. The bond indices α_i determine the expressivity of the MPS, with the bond dimension D representing their maximum allowed size. Larger D enables modeling of more complex correlations but increases computational cost, see Appendix.

A visual representation of an MPS decomposition in Eq. 1 is presented in Fig. 1, using Penrose graphical notation for
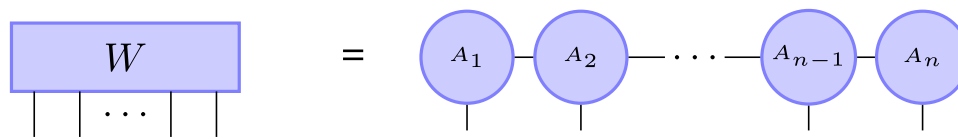
Fig. 1 Penrose diagram of the MPS decomposition in Eq. 1. Horizontal lines, bond indices α_i, connect adjacent tensors A_i, while vertical lines, physical indices d_j, represent input features via $\phi(x_i)$. Boundary tensors A_1, A_n are rank-2, two open indices, and internal tensors are rank-3 three open indices

tensors (Penrose 1971). Each tensor A_i, represented by the blue circles in Fig. 1, has horizontal indices α_i, whose size corresponds to the bond dimension D, and vertical indices d_i, whose size corresponds to the physical dimension d. The boundary tensors A_1 and A_n are rank-2, while intermediate tensors $A_2, \ldots, A_n$ are rank-3.

Throughout this work, we use different Penrose tensor network representations of the MPS depending on the context. In some diagrams, such as Fig. 1, we explicitly show the tensors A_i as separate entities, highlighting the individual components of the MPS. However, in later TN diagrams, such as in Eq. 2, we omit the explicit labeling of the sites A_i for clarity, as the focus shifts toward the overall network structure rather than the individual tensors. For completeness, a detailed description of all equations involving Penrose tensor notation is provided in Appendix, where each expression is explicitly rewritten in standard indexed notation.

Additionally, the TN representation changes when the MPS is contracted with an input. In the initial formulation, vertical edges represent the physical indices d_j, which correspond to the input space. These are referred to as open indices, as they are not initially contracted with other tensors. When the MPS is contracted with the embedded input $\Phi(\mathbf{x})$, these vertical edges become connected to the one-dimensional tensor $\Phi(\mathbf{x})$, effectively integrating the input into the network.

An MPS can be used to approximate any rank-n tensor W using a DMRG-based method (Schollwöck 2011; White 1992, 1993) to find its optimal decomposition. In quantum physics, MPS approximates the wavefunction $\psi(\mathbf{x})$ of a system, allowing the calculation of the probability density as the squared magnitude: $P(\mathbf{x}) = |\psi(\mathbf{x})|^2$. Similarly, in our machine learning framework, the MPS is used to approximate the PDF of the data:

$$p(x_1, \ldots, x_n) =$$

$$\tag{2}$$

The use of the squared value $|W(\mathbf{x})|^2$ aligns with the generative interpretation of MPS, inspired by its origins in quantum mechanics, where probabilities are derived from the squared magnitudes of wavefunctions (Miles and Schwab 2016). This formulation ensures that the MPS framework remains both expressive and interpretable for probabilistic modeling.

We employ an embedding function, $\Phi(\mathbf{x})$, also referred to as a feature map or feature embedding in the tensor network literature (Žunkovič and Ilievski 2024). The embedding function transforms input vectors of length N, with support $\text{supp}(\Phi) = [0, 1]^N$, into a format suitable for MPS theory. Since an MPS operates linearly on its input, through a sequence of tensor contractions and matrix multiplications, the embedding function introduces the necessary non-linearity into the system.

The embedding function is defined as a tensor contraction of local feature maps $\phi(x_i) \in \mathbb{R}^d$, which can alternatively be referred to as local embedding functions or local embedding features. Specifically:

$$\Phi(\mathbf{x}) = \phi(x_1) \otimes \cdots \otimes \phi(x_n), \tag{3}$$

and the specific forms of the feature maps are elaborated upon in Sect. 2.1.2, where the necessary conditions are explained.

There are three crucial hyperparameters used to define the structure of an MPS. The first one is the number of sites, corresponding to the number of bodies in the quantum physics scenario, and is determined by the input size in our model settings and will be denoted by N. The second is the bond dimension, D, which governs the size of indices connecting two different sites. The third parameter is the physical dimension, d, and is equal to the dimension of the vector $\phi(x_i)$, itself determined by our choice of embedding functions. Our implementation of an MPS is stored in a single tensor with dimensions (N, D, D, d). It should be noted that the boundary sites are also included in the tensor, although they possess fewer indices compared to the intermediate sites. This setup is called open boundary condition, used for non-periodic quantum systems (Perez-Garcia et al. 2006). For periodic quantum systems, where translational invariance is present, a different approach is employed where the first and last sites are rank-3 tensors connected to each other. However, this approach is usually not employed in the case of

MPS applications in machine learning scenarios (Miles and Schwab 2016; Han et al. 2018).

2.1.1 Canonical form of an MPS

It is evident that an MPS does not possess a unique form. In fact, if we introduce any sequence of invertible matrices $\{X_i\}_{i\in\{1,\dots N-1\}}$ of size $d \times d$ between any two sites along with their inverses, and setting $X_0 = X_N = I_d$, we can transform the tensors that comprise the MPS as follows:

$$B_i = X_{i-1}^{-1} A_i X_i. \tag{4}$$

This implies, ignoring the nonrelevant indices,

$$W = \sum_{\{\alpha\}} A_1^{\alpha_1} A_2^{\alpha_1\alpha_2} \cdots A_N^{\alpha_{N-1}} = \tag{5}$$

$$= \sum_{\{\alpha\}} A_1^{\alpha_1} X_1 X_1^{-1} A_2^{\alpha_1\alpha_2} X_2 \cdots X_{n-1}^{-1} A_N^{\alpha_{N-1}} = \tag{6}$$

$$= \sum_{\{\alpha\}} B_1^{\alpha_1} B_2^{\alpha_1\alpha_2} \cdots B_j^{\alpha_j\alpha_{j+1}} \cdots B_N^{\alpha_{N-1}}, \tag{7}$$

proving the non-uniqueness of the representation of an MPS. These degrees of freedom are termed gauge degrees of freedom. The non-unique representation of MPSs is exploitable and can help devise better privacy-preserving machine learning algorithms (Pozas-Kerstjens 2023). Canonical forms are also crucial for an efficient time-dependent variational principle for MPS (Haegeman et al. 2016).

To establish a canonical form for the MPS, various methods can be employed. One method uses DMRG-based and singular value decomposition techniques (Schollwöck 2011). Another alternative (Mueller Group 2020) employs the QR decomposition of the matrices that compose the MPS to achieve an orthogonal decomposition of the matrices. Those canonical forms of MPSs are needed to describe a so-called sweeping algorithm used for optimization (Chan et al. 2016). However, canonical forms can often be difficult to obtain for arbitrary MPSs (Oseledets 2011), since some of the calculations needed (for example, calculating the optimal rank d) are NP-hard (Håstad 1989) and/or an ill-posed problem (De Silva and Lim 2008). The sweeping algorithm and its necessary calculation of a canonical form will not be required as we can effectively leverage a more accessible approach via PyTorch automatic differentiation (Paszke et al. 2017), discussed in Sect. 2.2.1. Specifically, we optimize the parameters of the model by minimizing the cross-entropy loss function, an approach commonly used for tensor networks applied in machine learning scenarios (Efthymiou et al. 2019).

2.1.2 Embedding function

If the objective is data classification, the embedding function should primarily introduce non-linearity and transform the data such that our MPS can effectively achieve linear separation in the high-dimensional space where the data is embedded. Conversely, if the model is being used for data sampling, there is a greater emphasis on the selection of the embedding function. To employ the methodology outlined in Sect. 2.2.2 and perform simultaneous classification and generation tasks, an essential prerequisite for the local embedding function $\phi(x_i) \in \mathbb{R}^d$ is as follows:

$$\int_{x_i \in X} \phi_j(x_i)\phi_k(x_i)dx_i = \delta_{j,k}, \tag{8}$$

where $X = [0, 1]$, assumed to be the support of the input data, with limited exceptions noted herein. Equation 8 allows for a greatly simplified computation of the marginal probability over single variables, avoiding high-dimensional integrals as shown in Sect. 2.1.3. In certain cases, the embedding functions may also be defined with

$$\int_{x_i \in X} \phi_j(x_i)\phi_k(x_i)dx_i = c \cdot \delta_{j,k}, \quad c > 0. \tag{9}$$

Such a condition yields a non-normalized PDF. Nevertheless, we demonstrate that our method can accommodate any resulting inconsistencies (Sects. 2.2.2, 2.2.3).

The marginal probability over a single variable will have the form of

$$P(x_i) = \phi(x_i) \cdot V_i \cdot \phi(x_i), \tag{10}$$

with V_i being the symmetric and semi-positive definite reduced density matrix (Sect. 2.1.3). This implies that the PDF and its complexity will depend on our choice of $\phi(x_i)$ and the physical dimension of the model.

In the case of handling simple low-frequency data, such as high-contrast or binary images, a common embedding function (Cheng et al. 2021; Miles and Schwab 2016) is given by the following:

$$\phi(x_i) = [\sin(\frac{\pi}{2}x_i), \cos(\frac{\pi}{2}x_i)]. \tag{11}$$

This adheres to the constraint specified in Eq. 8 over the interval $[-1, 1]$. However, this function exhibits a limitation in that it possesses a low physical dimension of only 2. Consequently, for certain datasets, it may struggle to introduce the requisite complexity, thereby potentially impeding the model's performance.

A generalized replacement, outlined in Miles and Schwab (2016), involves the use of the embedding function:

$$\phi_j(x_i) = \sqrt{\binom{d-1}{j-1}} \cos(x_i)^{d-j} \sin(x_i)^j. \tag{12}$$

This function belongs to the class of functions referred to as spin coherent states (Miles and Schwab 2016). However, Eq. 12 does not satisfy the condition specified in Eq. 8 and is therefore not a candidate embedding function for generation.

An alternative proposal is the Fourier embedding (Žunkovič 2023):

$$\phi_j(x_i) = \cos(j\pi x_i), \text{ for } j \in \{0, ...d-1\}, \tag{13}$$

for $\text{Supp}(\Phi) = [0,1]^n$. This feature map satisfies Eq. 8 and can model more general PDFs, due to the fact that we can use arbitrarily high values of d. In this case, the PDF over a single variable will be modeled by the following equation:

$$\sum_{j=0}^{d-1} \sum_{k=0}^{d-1} a_{jk} \cos(\frac{\pi}{2} j x_i) \cos(\frac{\pi}{2} k x_i). \tag{14}$$

Our alternative proposal for an embedding function for high-dimensional physical spaces is to leverage polynomials, and due to Eq. 8, a natural choice is the use of Legendre polynomials. These are a set of polynomials $\{P_1(x_i), P_2(x_i), ...\}$ such that $P_j(x_i)$ is a polynomial of degree j and any 2 polynomials satisfy Eq. 8, obtained recursively by

$$P_0(x_i) = 1, \tag{15}$$
$$P_1(x_i) = x_i, \tag{16}$$
$$P_j(x_i) = \frac{(2j-1)x_i P_{j-1} - (j-1)P_{n-2}(x_i)}{j}, \quad \forall j \geq 2. \tag{17}$$

Unliked Fourier embeddings, $\text{Supp}(P) = [-1,1]^N$ instead of the previous interval $[0,1]^N$, so P will require rescaling in a pre-processing step. This embedding leads to a PDF over a single variable which is modelled by a non-negative polynomial of degree d^2 over $[-1,1]$.
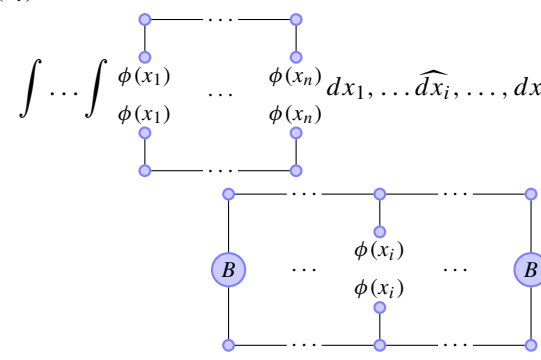
Fourier and Legendre embeddings can be used with arbitrarily large physical dimensions, thus suitable for modelling multi-modal probability distribution functions during the generative phase of Sect. 2.2.2. The generative results of these embedding functions are discussed in Sect. 3.

2.1.3 Computing the reduced density matrix

Given any local embedding function $\phi(x_i)$, we define the following matrix:
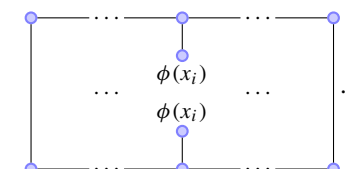
$$B := \int \begin{matrix} \phi(x_i) \\ \phi(x_i) \end{matrix} \, dx_i. \tag{18}$$

This is in contrast to Ferris and Vidal (2012), where the reduced density matrix is described by unitary tensor networks. Recalling that the PDF is approximated by an MPS by Eq. 2, the marginalized PDF can be simplified over a single variable $p(x_i)$ without necessitating the evaluation of multi-dimensional integrals:

$$p(x_i) = $$

$$\tag{19}$$

We assume that the selection of $\phi(x)$ satisfies Eq. 8 such that $B = \mathbb{I}_d$, leading to

$$p(x_i) = $$

$$\tag{20}$$

The marginalized distribution over a single variable is given by Eq. 10.

In the case where we want to calculate a conditional probability given some other variables' value, we can substitute the value of the conditioning variables. This conditional probability density over a single variable is given by the following:

$$p(x_i|\{x_j\}_{j\in\mathcal{I}}) = \phi(x_i) \cdot V_{i,\{x_j\}_{j\in\mathcal{I}}} \cdot \phi(x_i). \tag{21}$$

In order to compute the density matrix $V_{i,\{x_j\}_{j\in\mathcal{I}}}$, for $\{x_j | j \in \mathcal{I}\}$, we perform a contraction operation involving two copies

of the MPS. At each site, there are three possible scenarios for the tensor contraction in the physical dimension:

1) For $j = i$, we leave these two indices open.
2) For $j \in \mathcal{I}$, we compute the embedding $\phi(x_j)$ and contract the physical indices at site j of the two copies of the MPS with $\phi(x_j)$.
3) For $j \notin \mathcal{I}$, we directly contract the open indices of the two copies of the MPS, as described visually in Eq. 20.

The outcome of this function is a semi-positive definite symmetric matrix, a real matrix M is semi-positive definite if and only if there exists a matrix B such that $M = B^\mathsf{T} B$. Without loss of generality and supposing $\mathcal{I} = \{1, \cdots, i-1\}$, B is defined as

$$
B_i := \quad\begin{array}{c}\includegraphics{eq22}\end{array}. \tag{22}
$$

After reformatting $B_i^{d_i, d_{i+1}, \ldots d_n}$ into matrix form B_i' of shape $d^{n-i-1} \times d$, where we let the first index of the matrix correspond to the physical index of A_i and we regroup the remaining physical indices of $\{A_{i+1}, ..., A_n\}$ as the second index of the matrix, the tensor contraction $\sum_{j_{i+1}, ..., j_n} B_{j_1, ..., j_{i-1}} B_{j_1, ..., j_{i-1}}$ will correspond to the matrix multiplication $B_i'^\mathsf{T} B_i'$. We can rewrite $V_i = B_i' B_i'^\mathsf{T}$, proving its semi-positive definiteness.

Throughout this section, we did not assume to have a normalized MPS since that will not be the case during the training procedure. This can cause the computed PDFs to be unnormalized, but this will not affect our methods of classification and sampling, as we will see later in Sect. 2.2.2.

2.2 Classification and generation

2.2.1 Classification with MPS

In this section, we will discuss two methods that are commonly employed to perform classification using MPSs (Miles and Schwab 2016). The first approach involves creating an ensemble of MPSs, where each MPS corresponds to a distinct label class within the data. This ensemble model generates an output vector of length C, where each component of the vector arises from the contraction of a different MPS with the input. In Fig. 2, we show these contraction steps for a single component of the ensemble.

The second approach utilizes a single modified MPS, where we introduce an additional tensor placed in the middle of the tensor network. This tensor, with dimensions (C, D, D), plays a crucial role in producing the desired vector of length C. The network structure of this approach is
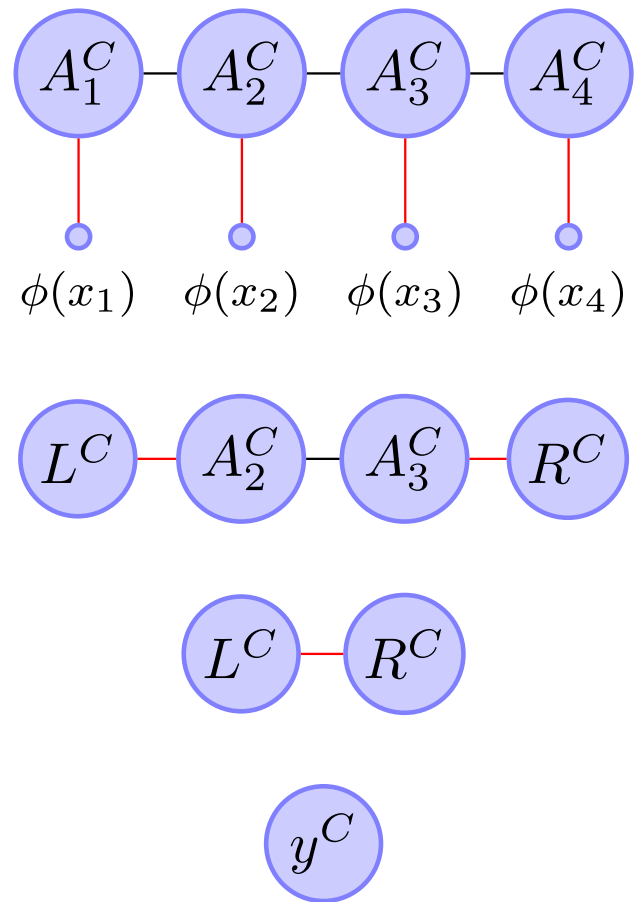

Fig. 2 This illustrates an example of how a single MPS of the ensemble, corresponding to the class C, and input are contracted in the forward pass, considering the case $N = 4$. The red lines indicate the indices that are being contracted in each step. By squaring the value of the final scalar y^C, we get a non-normalized probability, i.e., $p(c = i | \mathbf{x}) = \frac{1}{Z} \cdot y_i^2$, with $Z = \sum_c y_c^2$ being a normalization constant depending on the outputs of the ensemble of MPSs

described by the following equation:

$$
\begin{aligned}
W_c^{d_1, d_2, ..., d_n} = \\
\sum_\alpha A_1^{\alpha_1 d_1} A_2^{\alpha_1 \alpha_2 d_2} \cdots C_c^{\alpha_{n/2} \alpha_{n/2+1}} \cdots A_N^{\alpha_{N-1} d_N},
\end{aligned} \tag{23}
$$

and is also illustrated graphically in Fig. 3. The contraction of the model during the forward pass with this additional tensor is similar to the ones of the single MPS and can be visualized in Fig. 4.

An ensemble of MPSs is fundamentally equivalent to the single MPS with the additional central tensors. Each matrix in the single MPS must be block-diagonal with bond dimension $d * C$ and blocks of size d, where each of those blocks corresponds to a different MPS from the ensemble. The primary advantage of using a single model with an additional tensor is the capability to store and compute everything with just one MPS. This contrasts with the ensemble approach, which
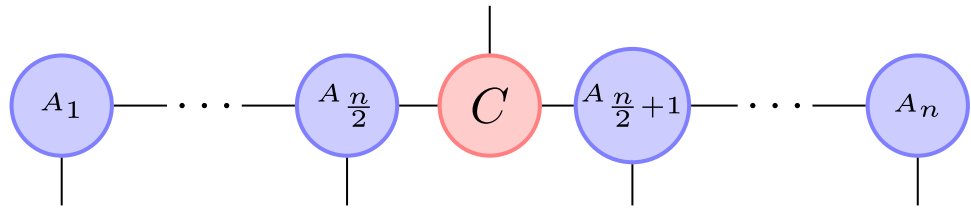
requires multiple MPSs. For instance, the MPS with a central tensor can be effective for low-dimensional inputs with a low bond dimension. However, in cases involving higher-dimensional data with a higher bond dimension, using an ensemble of MPSs can lead to an easier path to optimizing the MPS model, which is desirable in our case instead of the single model with a central tensor used in multiple other studies (Cheng et al. 2021; Miles and Schwab 2016).

In both scenarios, the final result after contraction is a vector of length C, where the squared entries of C are proportional to the probability for the input to belong to each given class, i.e., $p(c = i|\mathbf{x}) = \frac{1}{Z} \cdot y_i^2$, with $Z = \sum_c y_c^2$ being a normalization constant and y being the output of the MPSs.

In contrast to the typical classification settings, our model's output does not conform to the equation $\sum_{i=1}^{C} Y_i = 1$, where Y_i is the probability of the class i. This is because our model does not directly compute $p(c = i|\mathbf{x})$. Instead, our classification is determined (in the case of an ensemble of MPSs) by comparing which of the MPSs results in a higher squared value after the contraction with the input. We avoid the commonly used softmax activation function because we aim to generate new samples, and the usual probabilistic constraint set using a softmax activation function would alter the capabilities of the MPS for generating purposes, described later in Sect. 2.2.2.

We adopt the following initialization scheme for our Matrix Product State (MPS):

$$\text{mps}[i, :, :, j] = \frac{I_D}{\sqrt{d}}, \quad \forall j \in \{0, \dots, d-1\}, i \in \{1, \dots, N\}.$$
$$(24)$$

This ensures that each sequential operation on the preceding matrix, i.e., the multiplication of each matrix with the vectors R and L outlined in Algorithm 1, behaves as an identity matrix.
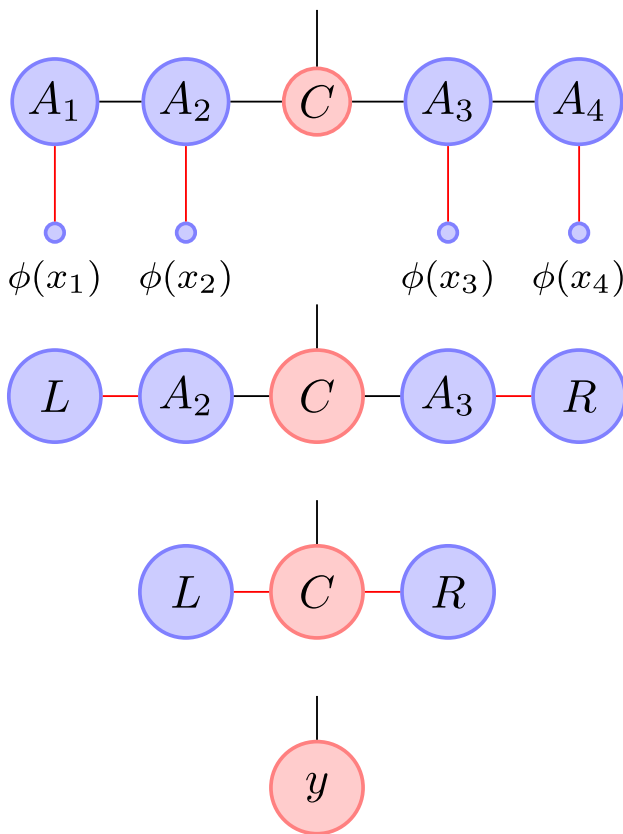

Fig. 4 This illustrates an example of how MPS and inputs are contracted in the forward pass of the classification, considering the case $N = 4$. The red lines indicate the indices that are being contracted in each step

---

Algorithm 1 MPS Classification

1: **procedure** CLASSIFICATION(mps, C, input)
2:     $mps_{nlr} \leftarrow \sum_{n,e} mps_{nlre} input_{ne}$
3:     $L \leftarrow mps[0, 0]$
4:     $R \leftarrow mps[-1, :, 0]$
5:     **for** i **in** range(1, N//2) **do**
6:         $L \leftarrow \sum_r L_r mps[i]_{rl}$
7:         $R \leftarrow \sum_l R_l mps[-i]_{rl}$
8:     **end for**
9:     $y \leftarrow \sum_{r,l} L_r C_{crl} R_l$
10:     return y^2
11: **end procedure**

In Algorithm 1, we use the following index conventions:

- n is the site index, referring to a specific site in the MPS representation. It corresponds to the blue circles in the tensor network diagram and has size N, which is equal to the input dimension.
- l and r are bond indices that connect different sites. These correspond to the horizontal lines in the TN diagram,

representing the outgoing bonds between adjacent sites. They have size D, which is the bond dimension.

- e is the embedding index, which connects each site to the embedded input $\phi(x)$. This index encodes the data fed into the MPS and has size d, which is the physical dimension.
- c is the class index. A separate MPS is assigned to each class c, as represented in the final summation.

These index definitions clarify how the tensor contractions are performed during the forward pass in the MPS classification procedure, using the Einstein summation notation.

Although past research (Hrinchuk et al. 2019) has explored non-trivial initialization configurations, we opt for this simpler and more intuitive approach which encourages stability during training. The rank-3 tensors present at each network site assume the form $A_s = \frac{I_d}{\sqrt{d}} + \hat{A}^s$, where only $\hat{A}^s$ is trainable. Each entry of $\hat{A}^s$ is initialized from a normal distribution $\hat{A}^s \sim \mathcal{N}(0, \sigma^2)$, where σ requires manual adjustment. The introduction of noise serves to disrupt symmetries and prevent convergence to local minima during the initial stages. Adding weight decay to the model parameters will only affect the matrices $\{\hat{A}^s\}_s$ and will have no effect on the initial settings except to remove the noise, leading to a more stable system during training while using weight decay.

During training, we minimize the cross-entropy loss in our classification settings with the ensemble of MPSs, which is equivalent to minimizing the Kullback–Leibler (KL) divergence between a single label model and the data coming from that class, as described in Liu et al. (2023).

With this method, we avoid using the DMRG-based method during the optimization of the MPS in Miles and Schwab (2016). Our approach carries the drawback that it could result in a non-normalized MPS; however, our method addresses this and effectively samples from such non-normalized distributions.

2.2.2 Generation with MPS

While previous works have employed Metropolis-Hastings and similar Markov chain Monte Carlo (MCMC) methods for sampling from MPS models (Han et al. 2018; Bonnevie and Schmidt 2021; Ferris 2015), we perform an exact sampling approach. An iterative method is used to sample coordinate after coordinate from a one-dimensional non-normalized PDF. A noise vector v is used as the input of our generative model, where each component v_i of the noise vector will correspond to the quantile of our sample $\hat{x}_i$, i.e.,

$$p(x_i \leq \hat{x}_i \mid x_1, \ldots, x_{i-1}) = v_i. \tag{25}$$

This approach eliminates the need for MCMC methods, which rely on constructing a Markov chain to sample from

complex distributions. Traditional MCMC algorithms, such as Metropolis-Hastings, require long iterative stochastic sampling processes to ensure convergence and representative sampling. In contrast, our method is deterministic and directly samples from the desired distribution. Without relying on the random fluctuations of MCMC, this approach avoids issues like autocorrelation, burn-in periods, and convergence diagnostics.

Furthermore, the exact nature of our sampling approach prevents biases that arise in MCMC due to its random walk behavior, which can lead to inefficiencies and inaccuracies. By generating samples directly, our method provides a more efficient and reliable alternative for sampling from the MPS.

Once the ensemble of MPSs has been trained for classification, its unique structure allows us to generate new samples from it. This is facilitated by the fact that any probability distribution over multiple variables can be expressed as follows, using the chain rule for probabilities:

$$p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_i | \{x_j\}_{j<i}), \tag{26}$$

and due to the structure of the tensor network and the choice of embedding function, the conditional distributions $p(x_i | \{x_j\}_{j<i})$ can be computed using the reduced density matrix. The sampling procedure is outlined in the following pseudocode for Algorithm 2:

Algorithm 2 MPS Sampling

1: **procedure** SAMPLE(mps, v)
2:     $samples \leftarrow$ zeros(N)
3:     **for** i **in** range(N) **do**
4:         $V = $ density_matrix(i, mps, samples)
5:         pdf(x) $= \phi(x) \cdot V \cdot \phi(x)$
6:         cdf(x) $= \int_{-1}^{x} $pdf($y$) dy
7:         $samples[i] = cdf^{-1}(v[i])$
8:     **end for**
9:   **return** $samples$
10: **end procedure**

Figure 5 shows the reduced density matrix produced at step i by Algorithm 2.

The challenge of the non-normalized MPSs is addressed by normalizing the cumulative distribution functions before sampling from them. An alternative approach could be to calculate the norm of the MPS before each sampling step and divide each site by the n-th root of the norm, which could be computed using the contractions visualized in Fig. 6, but would require more contraction calculations during training and therefore be less efficient.

The computation of the cumulative probability density and sampling phase must be differentiable in order to apply the methods described in Sect. 2.3 to facilitate gradient descent

Fig. 5 Visual description of the tensor contractions that produce the matrix $V_{i,\{x_j\}_{j<i}}$, used to calculate the conditional probability $p(x_i|x_1, ..., x_{i-1}) = \phi(x_i) \cdot V_{i,\{x_j\}_{j<i}} \cdot \phi(x_i)$ during the i-th iteration of our sampling algorithm
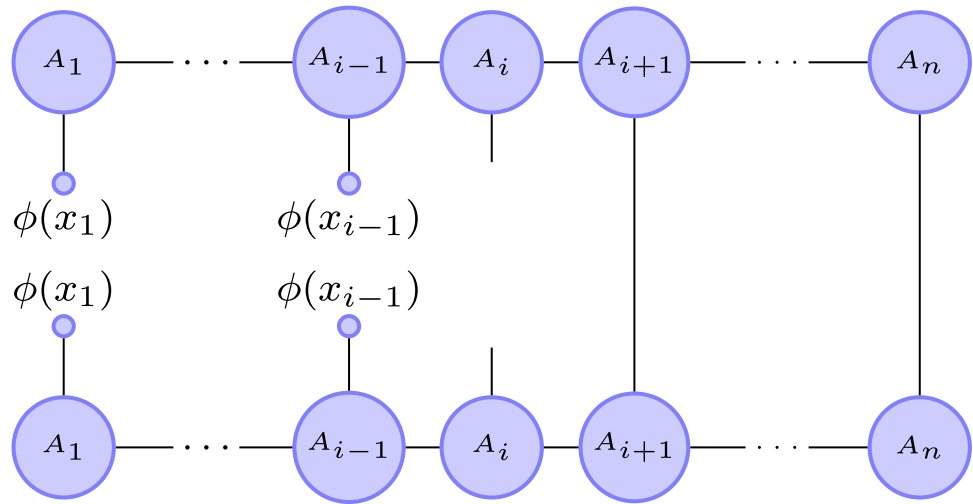
during training. If we denote the cumulative distribution function as,

$$f_{V_{i,\{x_j\}_{j<i}}}(x) := \int_{-1}^{x} \phi(y) \cdot V_{i,\{x_j\}_{j<i}} \cdot \phi(y)\, dy, \qquad (27)$$

we need to compute:

$$sample(v, V_{i,\{x_j\}_{j<i}}) = f_{V_{i,\{x_j\}_{j<i}}}^{-1}(v) \qquad (28)$$

to obtain a sample given the quantile v and the reduced density matrix $V_{i,\{x_j\}_{j<i}}$.

Depending on the choice of $\phi(x)$, there may not exist an explicit solution for $f^{-1}(x)$. Hence, the integral is approximated as a finite sum by dividing the input's support into 1000 bins:

$$I(x_k) = \int_{-1}^{x_k} \phi(x) \cdot V \cdot \phi(x)\, dx =$$

$$\sum_{i=0}^{k} \phi(x_i) \cdot V \cdot \phi(x_i)\Delta x, \qquad (29)$$
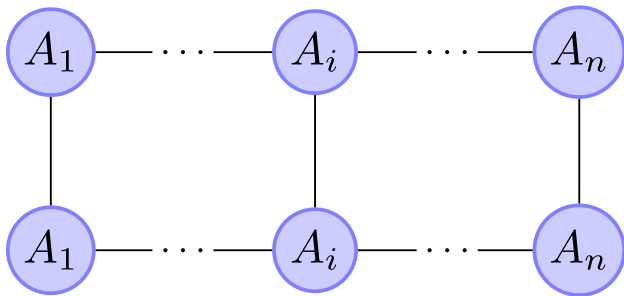

Fig. 6 How to compute the norm of an MPS, using Penrose graphical notation

where the bin width is denoted as $\Delta x = \frac{1}{1000}$. We define a set of points as $x_k = k\Delta x$ for $k = 0, 1, 2, \ldots, 1000$ in the case of an embedding function with $\text{Supp}(\Phi) = [0, 1]$. Then, for a given v within the cumulative density function range, we can approximate the inverse function as follows, opting for linear interpolation between the two closest points:

$$f^{-1}(v) \approx x_k + \frac{v - I(x_k)}{I(x_{k+1}) - I(x_k)} \cdot \Delta x, \qquad (30)$$

where $x_k \leq f^{-1}(v) \leq x_{k+1}$. Using this approximation, we can effectively compute the inverse function of the CDF and use it for sampling in Algorithm 2. We generate the vector entries iteratively by sampling points and calculating the conditional reduced density matrix for the next coordinate. For further details on the impact of binning resolution on accuracy and computational efficiency, refer to Appendix.

2.2.3 Avoiding vanishing/exploding values in the contractions

If we want to initialize our MPS as a stable system, because of the sequential nature of the MPS contraction phase, each of the contractions must be identity operators on the previous tensor; if we initialize each site of the MPS as stated in Eq. 24, the embedding function must then follow the property:

$$\forall x : \sum_{k} emb(x)_k = \sqrt{d}. \qquad (31)$$

However, this condition, in addition to Eq. 8, would limit the choice for the embedding functions excluding those described in Sect. 2.1.2, so we decided to opt for an alternative approach.

Given that our model can be represented as a linear model, with the exception of the embedding phase of our data, any

subsequent scalar multiplication will not have any influence on the classification and final sampling outcomes. This holds true both during the forward pass of the MPS, as well as in instances involving contraction for the derivation of the density matrix $V_{i,\{x_j\}_{j<i}}$.

To solve the problem of exploding and vanishing values, at each site of the MPS, we divide the vectors of the contraction, i.e., the variables denoted by L and R in Algorithm 1, by the value of their largest component after each step. In this way, we will always obtain vectors with a norm contained in the interval $[1, \sqrt{d}]$, avoiding both vanishing and exploding values.

A similar technique is implemented during the computation of the reduced density matrix. Specifically, the matrix is divided by the absolute value of its maximum entry, since the non-normalized PDF defined by Eq. 10 will not be affected by any positive scalar multiplication, allowing us to use the sampling technique in Sect. 2.2.2 even after rescaling the matrices.

2.3 MPS for simultaneous classification and generation

As discussed in Sect. 2.2.2, an MPS trained for classification can be used as a generator to emulate the data that was fed during training. However, this procedure would produce a lot of outliers that does not accurately reproduce the original data (Žunkovič 2023). A solution proposed in Žunkovič (2023) is to accept generated samples only if the MPS itself outputs a high enough probability that the sample is in the correct class; otherwise, if the output is lower than a predetermined threshold, the sample is rejected. This removes most outliers in a post-processing step but relies on a manually tuned threshold.

We opt to address this challenge directly from the network architecture to diminish the generation of outliers by using the MPS as a generator in a GAN-style setting and letting it compete with a discriminator. There are few regularization techniques for MPS models, for example, weight decay techniques are often used Miles and Schwab (2016), but dropout can not be used easily as in fully connected networks. This GAN-style setting can also be seen as a regularization method of the MPS training. As seen in Sect. 3, our GAN-style training produces more realistic samples, without changing the classification accuracy of the model. Here is an overview of our, MPS-GAN, training process:

1) Pre-training the MPS: Before the adversarial training, the MPS can be pre-trained using a classification-oriented approach to obtain realistic samples. We empirically observe that this step helps the MPS learn the underlying patterns. Note that it is an intensive training, up to acceptable classification accuracy for the model, since

we will use the score of this pre-trained model as a baseline for the adversarial-trained model. This is necessary for the classification accuracy of the final generator to be superior to the pretrained MPS.

2) Pre-training the discriminator: The discriminator, which can be a fully connected neural network or a specialized network (such as convolutional neural networks in the case of images), is pre-trained on samples generated by the previously trained MPS and the original dataset.

3) Adversarial training: The adversarial training iteratively optimizes both the MPS generator and the discriminator.

- Generator optimization: Generate samples using the MPS, and then optimize it to minimize the discriminator's ability to distinguish between real and generated samples.
- Discriminator optimization: The discriminator is optimized using the standard GAN objective, where it maximizes the probability of assigning the correct label to real samples and the incorrect label to generated samples.

4) Classification accuracy check: the classification accuracy is checked every epoch to ensure it remains above the initial classification accuracy threshold. If ever it falls below this threshold, the model is retrained in the classical classification setting. This step ensures that the generator maintains its classification capabilities.

An ensemble of discriminators for the different labels is used to reduce the number of outliers among the sample points, while also helping to distinguish between classes. It also prevents samples from being wrongly identified as real samples if they are unlabeled and assigned to the false class.

3 Results and discussion

The results section will primarily concentrate on the generative performance of our method, as our primary objective is to enhance the network's generative capabilities. During the following experiments, we will search for the best MPS model, according to its classification accuracy,

We demonstrate the generation capabilities of the GAN-style MPS and present an analysis of the latent space. The parameters are set as $D \in \{4, 10, 30\}$, $d \in \{4, 10, 30\}$, with an initial standard deviation of $\sigma = 0.1$ and an initial learning rate of $lr = 0.01$. Additionally, learning rate decay and early stopping procedures are applied.

The bond dimension D plays a crucial role in determining the expressive power of MPS models, as it controls both the number of parameters and the ability to model correlations between variables (Navascues and Vertesi 2018). To assess the impact of D on classification performance, an experiment

Table 1 Comparison of FID-like score (lower is better) on generated samples for Fourier and Legendre embedding functions, pre- and post-GAN-style training

Methods	Datasets		
	2D Spiral	2 Moon	Iris
Legendre, pre-GAN	4.63e−3	1.20e−2	6.36e−2
Fourier, pre-GAN	3.25e−3	8.79e−4	1.02e−1
Legendre, post-GAN	4.57e−3	5.93e−3	4.04e−2
Fourier, post-GAN	1.52e−3	3.02e−4	1.96e−2

was conducted in which the model was trained across a range of bond dimensions. The results, detailed in Appendix, indicate that beyond a threshold of $D \approx 4$, classification accuracy does not significantly improve. This suggests that, for low-dimensional datasets, large bond dimensions are not necessary for capturing meaningful correlations. Higher values of D help model long-range dependencies, especially in images (Han et al. 2018), but these are absent in our low-dimensional data.

3.1 GAN-style training

In this section, the generative performances of the MPS models are analyzed pre- and post-GAN-style training, with comparisons between Fourier and Legendre embedding functions. The results are based on simulated datasets, the dual 2D Spiral, the 2 Moon, and the Iris datasets, and more details of the simulated data can be found in the Appendix.

An FID-like score is used to compare the generated samples before and after training the MPS in a GAN-style setting. For a training dataset $x_1, \ldots, x_n$ and a set of generated samples $x_1^{(g)}, \ldots, x_m^{(g)}$, we denote the respective means as $\hat{\mu}$ and $\hat{\mu}^{(g)}$, and the covariance matrices as $\hat{\Sigma}$ and $\hat{\Sigma}^{(g)}$. The FID-like metric is defined as

$$\|\hat{\mu} - \hat{\mu}^{(g)}\|^2 + \text{Tr}(\hat{\Sigma} + \hat{\Sigma}^{(g)} - 2(\hat{\Sigma}\hat{\Sigma}^{(g)})^{1/2}). \tag{32}$$

This metric quantifies the dissimilarity between the real and generated data based on their means and covariance matrices, where a lower value indicates greater similarity between the two datasets.

The results for both Legendre and Fourier embeddings pre- and post-GAN-style training are shown in Table 1. The Fourier embedding after the GAN-style training generated the lowest FID-like score across all datasets.

Figures 7, 8, and 9 also visually show how GAN-style training affects the result of the samples generated by the ensemble of MPSs.

The number of outliers generated by the model decreases after the GAN-style training, as shown in Table 2 which compares the number of outliers before and after the GAN-style training and for different choices for the embedding functions. We evaluate the percentage of outlier samples, where a sample point is considered an outlier if the average of its k-nearest neighbors in the training data is higher than the maximum k-distance of the original dataset.

The values reported in Table 2 confirm the hypothesis suggested visually by Figs. 7, 8, and 9, where the number of outliers decreases thanks to the GAN-style training. Additionally, the performances of the models that use Fourier embedding functions have lower FID-like scores across the board compared to those resulting from the Legendre embeddings.

3.2 Latent space analysis

Within our experimental framework, the latent space dimensionality of our generative model, which follows GAN principles, corresponds to the dataset's dimensionality. The latent space is equal to $[0, 1]^n$, and as described in Sect. 2.2.2, the noise vector $v \in [0, 1]^n$ part of our latent space satisfies the condition:

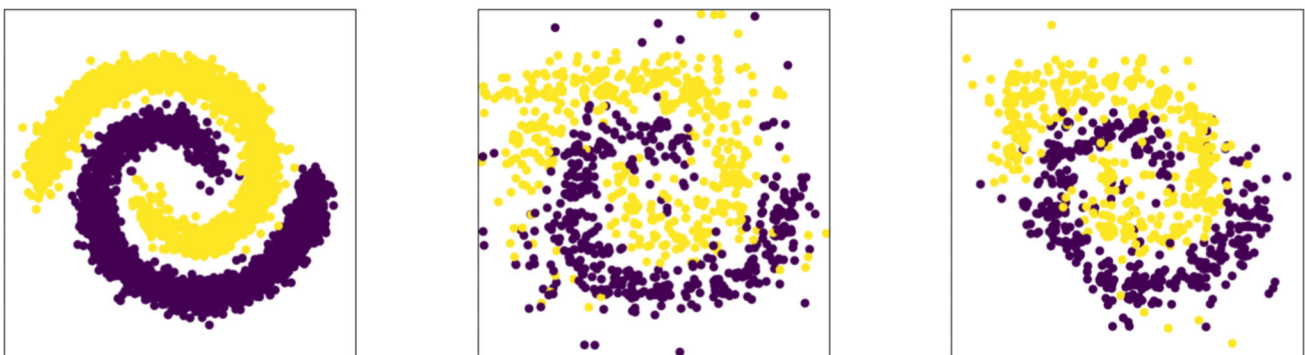$$p(x_i \leq \hat{x}_i \mid x_1, \ldots, x_{i-1}) = v_i.$$

Fig. 7 Visualization of the 2D Spiral dataset and MPS-generated samples. Left: original dataset. Middle: Samples generated by a classically trained MPS, optimized using cross-entropy loss. Right: Samples generated by an adversarially trained MPS. Fourier embedding was used
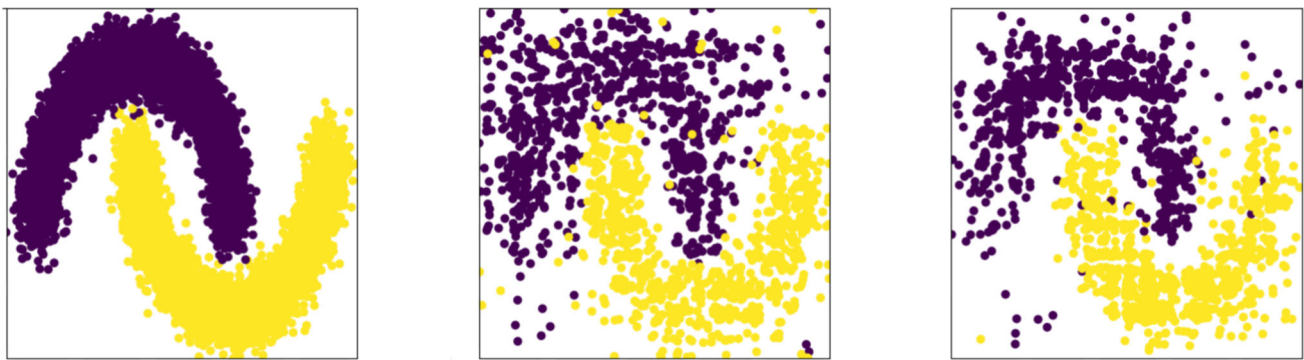
Fig. 8 Comparison of generated samples on the Two Moons dataset. Left: The original dataset. Middle: Samples generated by a classically trained MPS, optimized using cross-entropy loss. Right: Samples generated by an adversarially trained MPS. Fourier embedding was used

For an ideally trained model, every point within this latent space corresponds to a realistic sample. Additionally, the set of points in the latent space that do not correspond to realistic samples should have a measure of 0, ensuring that the vast majority of the latent space is occupied by realistic data points. Consequently, conventional latent space analyses, such as clustering, become inapplicable, as the latent space does not exhibit well-defined clusters but instead reflects the smooth, continuous distribution of realistic data points.

In scenarios where the probability density function (PDF) is strictly positive and is modeled using the Matrix Product State (MPS) framework, the latent space can be mathematically represented as the hypercube $[0, 1]^n$. While the latent space's dimensionality is fixed at $\mathbb{R}^n$, independent of the model's parameters, its structure may be influenced by factors such as the bond dimension. However, as shown in the experiments in Appendix, the bond dimension has minimal impact in our setting due to the low dimensionality of our datasets. To better understand its influence in more complex scenarios, further experiments with larger datasets would be

necessary. Moreover, there exists a one-to-one mapping, i.e., a bijection, between this latent space and the physical data space. As a result, any topological analysis within the latent space yields trivial findings, since the latent space directly corresponds to the physical space of the data.

However, latent space analysis can be performed thanks to the utilization of the MPS architecture in our experimental setup which is itself an ensemble of models, with each model dedicated to a distinct class. This architecture enables us to perform interpolation and comparison exclusively between intra-class samples. A visual representation of this process is presented in Fig. 10, where two instances of linear interpolation within the latent space are depicted, one for each class, employing different Fourier and Legendre embedding functions.

The sampled trajectories for Legendre-embedded data form a more regular shape, while those for Fourier embeddings appear more chaotic. This structural distinction aligns with their contrasting classification behaviors under input perturbations (see Sect. 3.3).
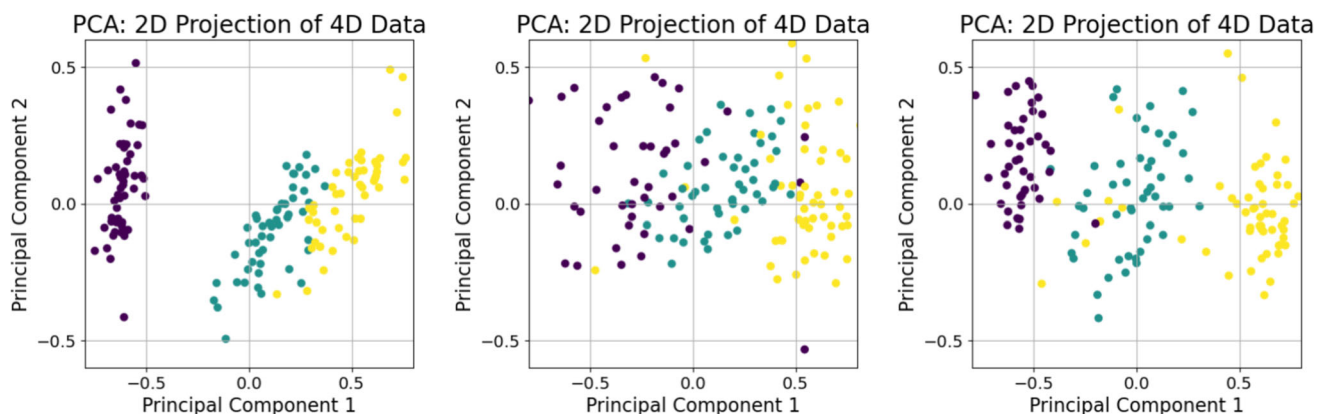

Fig. 9 2D PCA visualization of generated samples from the Iris dataset. Left: PCA projection of the original dataset. Middle: PCA projection of samples generated by a classically trained MPS model, optimized using cross-entropy loss. Right: PCA projection of samples generated by an adversarially trained MPS model. Fourier embedding was applied to the data

Table 2 Comparison of outlier percentages of generated samples for Fourier and Legendre embedding functions, before and after GAN-style training

Methods	Datasets		
	2D Spiral	2 Moon	Iris
Legendre, pre-GAN	9.03e−2	1.00e−1	3.60e−1
Fourier, pre-GAN	8.90e−2	1.13e−1	4.62e−1
Legendre, post-GAN	7.63e−2	1.00e−1	2.20e−1
Fourier, post-GAN	6.32e−2	3.40e−2	2.36e−1

Fig. 11 Validation accuracy as a function of σ, where σ represents the standard deviation of the centered normal distribution from which the noise added to the embedded inputs is sampled. In this experiment, the model is trained on clean data and evaluated on noisy inputs, with hyperparameters set to $d = 20$ and $D = 50$. A comparison between Fourier and Legendre embeddings is provided

- Legendre embeddings produce smooth sample trajectories in latent space, reflecting their stable polynomial basis. This regularity leads to well-defined decision boundaries in noiseless conditions, enabling higher initial classification accuracy.
- Fourier embeddings, with their oscillatory basis, generate irregular trajectories. While this introduces instability in noise-free settings, the frequency-rich representation improves adaptability to input variations, as shown in Fig. 11.

In both cases, most samples remain within their original class (Fig. 10), demonstrating the MPS's ability to preserve semantic consistency. However, exceptions occur in low-probability regions where trajectories intersect the "false" class. These intersections result from the MPS's non-zero probability of sampling across classes—a consequence of its quantum-inspired structure, which avoids strict orthogonality. While undesirable for purity, this property ensures full support over the data manifold, crucial for robust generation under perturbations (see Sect. 3.3).

3.3 Robustness to perturbations

We assumed that our model has an input space on $[0, 1]^N$ and that the embedding function used in this work transforms the data onto an n-dimensional manifold in a $d \cdot n$ dimensional

space. In this section, we consider how adding noise to this manifold can change the performance of the MPS model, depending on the choice of the embedding function. After obtaining a trained MPS, the classification accuracy of the model is observed for increasing values of σ, which determines the amount of noise ϵ that is added to the inputs of the MPS after having embedded the data using $\Phi(x)$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Figure 11 shows the accuracy performance of two models, one with Fourier and the other with Legendre embedding function, as noise is added to the embedded inputs.

As seen in Fig. 11, at $\sigma = 0$, the Fourier embedding achieves lower initial classification accuracy compared to the Legendre embedding. A precise theoretical justification remains an open question; however, a potential explanation may be related to the regression properties of these embeddings.

Legendre embeddings, being based on orthogonal polynomials, appear to be empirically better suited to the dataset,

Fig. 10 In the background, the original dataset, and in green and blue, the trajectories of samples for, (left) Fourier and (right) Legendre embedding functions, corresponding to the yellow and violet class respectively. The trajectories are obtained by linearly interpolating two points of the latent space of the MPS, highlighted in red
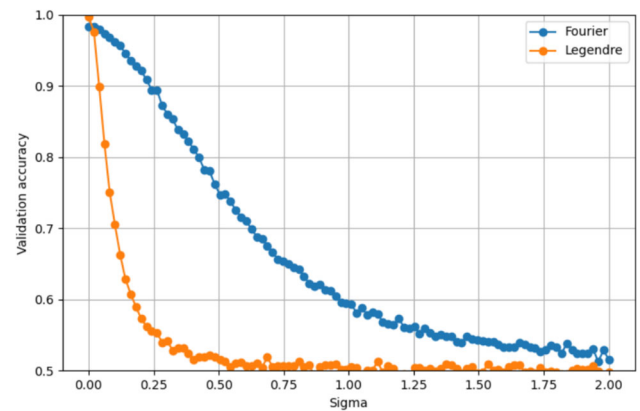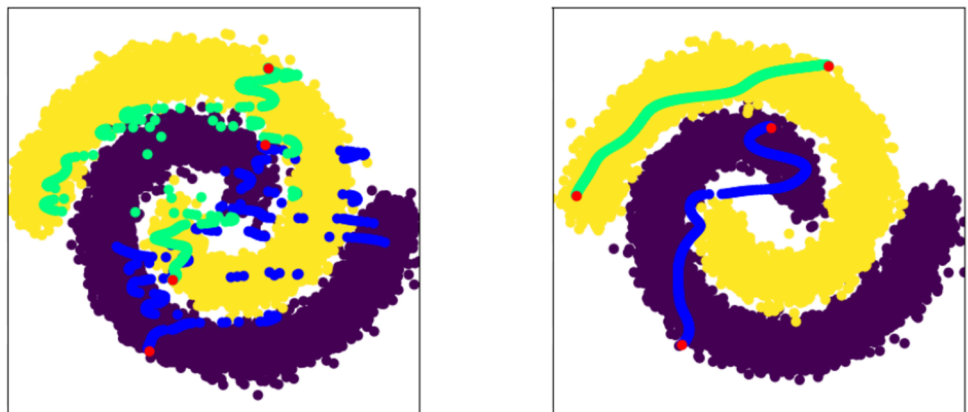
leading to higher validation accuracy. This suggests that the dataset's underlying patterns can be more naturally captured by polynomial basis functions.

Fourier embeddings introduce oscillatory basis functions, which can lead to high-frequency artifacts in the learned function when trained on limited data, a phenomenon sometimes associated with interpolation instability or Runge's phenomenon in polynomial interpolation (Trefethen 2013). This effect can cause poor generalization in classification tasks when noise is absent ($\sigma = 0$). In contrast, Legendre embeddings, based on orthogonal polynomials, tend to produce smoother approximations with reduced oscillations, potentially leading to more stable classification boundaries.

However, as noise increases ($\sigma > 0$), Fourier embeddings demonstrate higher robustness, likely due to their ability to capture frequency-based features efficiently. This aligns with prior findings in function approximation theory, where Fourier-based methods excel in encoding structured data but may struggle with instability in noiseless conditions (Tancik et al. 2020).

An additional experiment, comparing the classification accuracy of the MPS as σ increases, is shown in Fig. 12. In this experiment, noise is introduced during the training phase, whereas in the previous experiment, noise was only added during inference with a fixed trained model.

The results of this experiment align with the current interpretation. While the Legendre embedding achieves better classification performance at $\sigma = 0$, we observe that as noise is introduced during training the Legendre embedding also gains robustness by learning to fit the noise. Conversely, the Fourier embedding retains its previous robustness and performance remains consistent even at $\sigma = 1$, as it is inherently adapted to handle noise without requiring specialized learning.

4 Conclusion and future work

In this study, we explore the MPS and its applications in the field of machine learning. We demonstrate the fundamental structure of MPS and its utility in machine learning for tasks related to classification and generative modeling. We expand upon the various canonical forms of MPS and justify a more accessible training approach for its applications in machine learning, specifically avoiding the necessity for the sweeping algorithm.

To facilitate the discussion, we identified and examined the essential criteria for an embedding function that enables accurate sampling from a non-normalized model trained for classification. We demonstrate the computation of the reduced density matrix and establish its positive semi-definiteness. Furthermore, we introduce and contrast the widely adopted Fourier embedding with our novel proposition of using Legendre polynomials. Our empirical findings indicate that the former yields superior generative results.

Building upon principles from GANs, we leverage dual roles of the MPS, allowing it to serve both as a generator and a classifier simultaneously. This enhances training and generative performance without compromising the model's classification accuracy. Notably, this approach results in a reduction of the number of outliers generated during the sampling process, yielding samples with more favorable FID-like scores when compared to classical training techniques for MPSs.

Within the framework of this procedure, we introduce a latent space representation for the MPS model when using it as a generator and subject it to analysis. We investigate the impact of introducing perturbations after data embedding on the classification accuracy for both Fourier and Legendre embeddings, where the former demonstrated greater resilience in the presence of increasing noise.

In further research, alternative embedding functions can be explored. For instance, the use of the Welsh basis, a non-continuous set of orthonormal functions, warrants exploration. Additionally, investigating non-orthogonal bases is promising, leveraging the research developed in this work on the role of the embedding function in calculating the reduced density matrix. Furthermore, it is interesting to investigate how perturbations may impact the classification accuracy, particularly when various embedding functions are employed.

While our experiments focus on low-dimensional datasets, future work should explore the impact of bond dimension D on higher-dimensional datasets. In such settings, larger bond dimensions may be necessary to capture long-range corre-
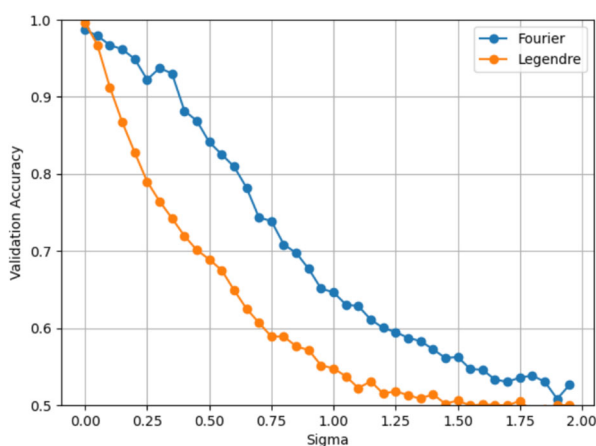

Fig. 12 Validation accuracy as a function of σ, where σ represents the standard deviation of the centered normal distribution from which the noise added to the embedded inputs is sampled. In this experiment, models are trained on noisy data and evaluated on clean inputs, with hyperparameters set to $d = 20$ and $D = 50$. A comparison between Fourier and Legendre embeddings is provided

lations and complex dependencies, which are not prominent in our current datasets. Investigating the interplay between bond dimension, dataset dimensionality, and model expressivity would provide valuable insights into the scalability and applicability of MPS-based models to more complex machine learning tasks.

Furthermore, the MPS holds promise for integration with other neural network paradigms, potentially as a latent space representation within a variational autoencoder or a normalizing flow in manifold learning frameworks (Flouris and Konukoglu 2023), where the density can be directly estimated mitigating the need of approximate inference.

Appendix. Penrose tensor notation in detail

In this section, we provide additional details on the equations used throughout the work. We explicitly present the formulas that were previously expressed only using Penrose tensor notation.

- Probability distribution representation, Eq. 2:

$$
p(x_1, \ldots, x_n) =
$$

$$
\left[\sum_{\{d_i\}} W^{d_1,d_2,\ldots,d_{n-1},d_n} \, \Phi(\mathbf{x})^{d_1,d_2,\ldots,d_{n-1},d_n} \right]^2 =
$$

- Definition of B, Eq. 18:

$$
B := \int \phi(x_i) \otimes \phi(x_i) dx_i = \int \begin{matrix} \phi(x_i) \\ \phi(x_i) \end{matrix} dx_i.
$$

- Marginal probability computation, Eq. 19:

$$
p(x_i) = \int \ldots \int p(x_1, \ldots, x_n) \, dx_1 \ldots \widehat{dx_i} \ldots dx_n
$$

$$
= \int \ldots \int \begin{matrix} \phi(x_1) & & \phi(x_n) \\ \phi(x_1) & & \phi(x_n) \end{matrix} dx_1 \ldots \widehat{dx_i} \ldots dx_n
$$
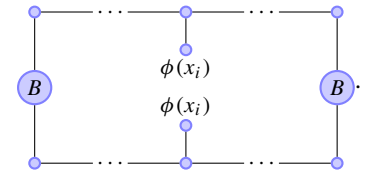
- Marginal probability computation in case of $B = I_d$, Eq. 20

$$
p(x_i) = \sum_{d_1,\ldots,d_i,d_i',\ldots,d_n} W^{d_1,\ldots,d_i,\ldots,d_n} \phi(x_i)^{d_i}
$$

$$
W^{d_1,\ldots,d_i',\ldots,d_n} \phi(x_i)^{d_i'} =
$$
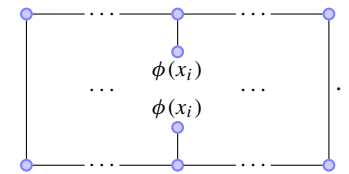
- Definition of B_i, Eq. 22:

$$
B_i := \sum_{d_1,\ldots,d_{i-1}} W^{d_1,\ldots,d_{i-1},d_i,\ldots,d_n}
$$

$$
\phi(x_1)_1^d \otimes \cdots \otimes \phi(x_{i-1})^{d_{i-1}} =
$$

$$
\sum_{d_1,\ldots,d_{i-1}} A_1^{\alpha_1,d_1} \phi(x_1)^{d_1}
$$

$$
\cdots
$$

$$
A_{i-i}^{\alpha_{i-2}\alpha_{i-1},d_{i-1}} \phi(x_{i-1})^{d_{i-1}}
$$

$$
A_i^{\alpha_{i-1}\alpha_i,d_i}
$$

$$
A_{i+1}^{\alpha_i\alpha_{i+1},d_{i+1}}
$$

$$
\cdots
$$

$$
A_N^{\alpha_{N-1},d_N} =
$$

Effect of bond dimension on classification performance

To analyze the effect of the bond dimension D on classification accuracy, we conducted an experiment in which we varied D while keeping all other hyperparameters fixed.

The results presented in Fig. 13 indicate that classification accuracy initially improves with D but saturates beyond $D \approx 4$. This suggests that for this dataset, the expressive power provided by higher bond dimensions does not yield further improvements.

A possible explanation is that this dataset has low intrinsic dimensionality and lacks long-range correlations, which
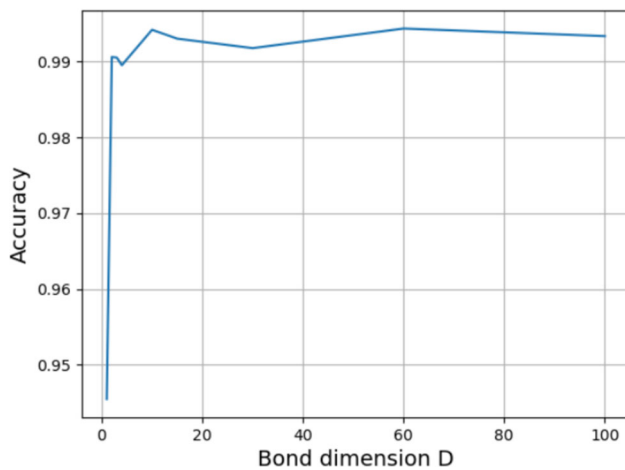
Fig. 13 Classification accuracy as a function of bond dimension D, for Fourier embedding on the 2D spiral dataset

Fig. 15 Computation time for a sample as a function of the number of bins

are typically modeled by increasing D. In problems with strong correlations between distant variables, a larger bond dimension would be necessary to capture these dependencies. However, in this case, the dataset structure does not demand high expressivity, making smaller bond dimensions sufficient.

Impact of binning on accuracy and computational cost

The number of bins used to discretize the input space directly impacts both approximation accuracy and computational efficiency. A finer discretization, i.e., increasing the number of bins, leads to a more precise numerical approximation, reducing the squared error. However, it also increases computational cost due to the higher resolution required for numerical integration and sampling.

Fig. 14 Squared error as a function of the number of bins

To analyze the impact of binning on both approximation accuracy and computational efficiency, we conducted an experiment where we sampled from the identity matrix I_d with a fixed physical dimension $d = 10$ using the Fourier embedding. The quantile value was set to $\nu = 0.5$, which allowed us to know the theoretical expected result for the sample $x_1 = 0.5$ and compute the squared loss against the true value.

Figure 14 illustrates how the squared error decreases exponentially with the number of bins, demonstrating that beyond 10^3 bins, further improvements become negligible. Conversely, Fig. 15 shows that computational time initially remains low but begins increasing significantly beyond 10^5 bins. This reflects the increased computational burden required to process finer binning resolutions.

Thus, after analyzing the complete data, presented in Table 3, we selected 1000 bins as the optimal balance, where the squared error is sufficiently low and computational time remains efficient. This choice ensures numerical stability while avoiding unnecessary overhead.

Table 3 Effect of the number of bins on squared error and computation time

Number of bins	Squared error	Computation time (s)
10^1	3.09×10^{-3}	4.68×10^{-4}
10^2	2.55×10^{-5}	2.26×10^{-4}
10^3	2.51×10^{-7}	2.29×10^{-4}
10^4	2.50×10^{-9}	1.17×10^{-3}
10^5	2.51×10^{-11}	4.86×10^{-3}
10^6	2.57×10^{-13}	5.87×10^{-2}
10^7	3.55×10^{-15}	6.75×10^{-1}

Dataset generation

In this appendix, we provide details on the datasets used in our experiments.

A. 2D Spiral dataset

The spiral dataset was generated by sampling $N = 8000$ points along two spirals, with angles drawn from a uniform distribution and perturbed with Gaussian noise. The data was then normalized to fit within $[0, 1]^2$.

The dataset was generated using the following Python script:

```python
import numpy as np

N = 8000
theta = np.sqrt(np.random.rand(N)) * 2 np.pi

# First spiral
r_a = 2 * theta + np.pi
x_a = np.array([np.cos(theta) * r_a,\\
    np.sin(theta) * r_a]).T
x_a += np.random.randn(N, 2)

# Second spiral
r_b = -2 * theta - np.pi
x_b = np.array([np.cos(theta) * r_b, \\
    np.sin(theta) * r_b]).T
x_b += np.random.randn(N, 2)

# Normalize and stack
x_a, x_b = x_a / 20., x_b / 20.
x = np.vstack((x_a, x_b))
x = (x - x.min()) / (x.max() - x.min())
y = np.vstack((np.zeros((N,1)), \\
    np.ones((N,1))))
```

B. Two moons dataset

The Two Moons dataset was generated using the `sklearn.datasets.make_moons` function with added noise.

```python
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=2000, noise=0.1)
```

C. Iris dataset

The Iris dataset was obtained using `sklearn.datasets.load_iris`. Features were normalized before training.

```python
from sklearn.datasets import load_iris

data = load_iris()
X, y = data.data, data.target

# Normalize features
X = (X - X.min()) / (X.max() - X.min())
```

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

References

Bengua JA et al (2017) Matrix product state for higher-order tensor compression and classification. IEEE Trans Signal Process 65(15):4019–4030

Biamonte J, Bergholm V (2017) Tensor networks in a nutshell. arXiv:1708.00006

Bishop CM (2006) Pattern recognition and machine learning (information science and statistics). Springer-Verlag, Berlin, Heidelberg

Bonnevie R, Schmidt MN (2021) Matrix product states for inference in discrete probabilistic models. J Mach Learn Res 22(1):8396–8443

Bridgeman JC, Chubb CT (2017) Hand-waving and interpretive dance: an introductory course on tensor networks. J Phys A: Math Theor 50(22):223001

Bruognolo B, et al (2017) Matrix product state techniques for two-dimensional systems at finite temperature. arXiv:1705.05578

Chan GK, et al (2016) Matrix product operators, matrix product states, and ab initio density matrix renormalization group algorithms. J Chem Phys 145(1)

Cheng S et al (2019) Tree tensor networks for generative modeling. Phys Rev B 99(15):155131

Cheng S, Wang L, Zhang P (2021) Supervised learning with projected entangled pair states. Phys Rev B 103(12):125117

Dalzell AM, Brandão FG (2019) Locally accurate MPS approximations for ground states of one-dimensional gapped local Hamiltonians. Quantum 3:187

De Silva V, Lim L-H (2008) Tensor rank and the ill-posedness of the best low-rank approximation problem. SIAM J Matrix Anal Appl 30(3):1084–1127

Weinan E, Han J, Jentzen A (2021) Algorithms for solving high dimensional PDEs: from nonlinear Monte Carlo to machine learning. Nonlinearity 35(1):278–310. https://doi.org/10.1088/1361-6544/ac337f

Efthymiou S, Hidary J, Leichenauer S (2019) Tensornetwork for machine learning. arXiv:1906.06329

Ferris AJ (2015) Unbiased Monte Carlo for the age of tensor networks. arXiv:1507.00767

Ferris AJ, Vidal G (2012) Perfect sampling with unitary tensor networks. Phys Rev B 85(16):165146

Flouris K, Konukoglu E (2023) Canonical normalizing flows for manifold learning. Oh A, Naumann T, Globerson A, Saenko K, Hardt M, and Levine (Eds.) S, Advances in neural information processing systems (Vol. 36, pp 27294–27314). Curran Associates, Inc. https://proceedings.neurips.cc/paperspsfiles/paper/2023/file/572a6f16ec44f794fb3e0f8a310acbc6-Paper-Conference.pdf

Francuz A, Schuch N, Vanhecke B (2023) Stable and efficient differentiation of tensor network algorithms

Glasser I, et al (2019) Expressive power of tensor-network factorizations for probabilistic modeling

Haegeman J, Lubich C, Oseledets I, Vandereycken B, Verstraete F (2016) Unifying time evolution and optimization with matrix product states. Phys Rev B 94(16):165116

Han Z-Y et al (2018) Unsupervised generative modeling using matrix product states. Phys Rev X 8(3):031012

Hastings MB (2007) An area law for one-dimensional quantum systems. J Stat Mech: Theor Exper P08024

Hdaib M, Rajasegarar S, Pan L (2024) Quantum deep learning-based anomaly detection for enhanced network security. Quant Mach Intell 6(1):26. https://doi.org/10.1007/s42484-024-00163-2

Hrinchuk O, Khrulkov V, Mirvakhabova L, Orlova E, Oseledets I (2019) Tensorized embedding layers for efficient model compression. arXiv:1901.10787

Håstad J (1989) Tensor rank is NP-complete

LeCun Y, Cortes C, Burges C (2010) MNIST handwritten digit database. ATT Labs 2. http://yann.lecun.com/exdb/mnist. Accessed 2 Apr 2025

Liao H-J, Liu J-G, Wang L, Xiang T (2019) Differentiable programming tensor networks. Phys Rev X 9:031041. https://doi.org/10.1103/PhysRevX.9.031041

Liu J et al (2023) Tensor networks for unsupervised machine learning. Phys Rev E 107(1):L012103

Mueller Group (2020) Matrix Product States: Canonical Forms. https://muellergroup.lassp.cornell.edu/bt2020chap4.pdf Accessed 2 Apr 2025

Miles E, Schwab (2016) Supervised learning with quantum-inspired tensor networks. 4799

Navascues M, Vertesi T (2018) Bond dimension witnesses and the structure of homogeneous matrix product states. Quantum 2:50

Orús R (2014) A practical introduction to tensor networks: Matrix Product States and projected entangled pair states. Ann Phys 349:117

Orús R (2019) Tensor networks for complex quantum systems. Nat Rev Phys 1(9):538–550

Oseledets IV (2011) Tensor-train decomposition. SIAM. J Sci Comput 33(5):2295–2317

Paszke A, et al (2017) Automatic differentiation in Pytorch

Penrose R (1971) Applications of negative dimensional tensors. pp 221–244

Perez-Garcia D, et al (2006) Matrix product state representations. arXiv:quant-ph/0608197

Pozas-Kerstjens A (2023) Privacy-preserving machine learning with tensor networks. Bullet Am Phys Soc

Pozas-Kerstjens A, Hernández-Santana S, Monturiol JRP, López MC, Scarpa G, González-Guillén CE, Pérez-García D (2023) Privacy-preserving machine learning with tensor networks

Ramakrishnan RK, Ravichandran AB, Mishra A, Kaushalram A, Hegde G, Talabattula S, Rohde PP (2023) Integrated photonic platforms for quantum technology: a review. ISSS J Micro Smart Syst 12(2):83–104. https://doi.org/10.1007/s41683-023-00115-1

Schollwöck U (2011) The density-matrix renormalization group in the age of matrix product states. Ann Phys 326(1):96–192

Suzuki T, Hasebe T, Miyazaki T (2024) Quantum support vector machines for classification and regression on a trapped-ion quantum computer. Quant Mach Intell 6(1):31. https://doi.org/10.1007/s42484-024-00165-0

Tancik M, Srinivasan PP, Mildenhall B, Fridovich-Keil S, Raghavan N, Singhal U, Ng R (2020) Fourier features let networks learn high frequency functions in low dimensional domains. Adv Neural Inf Process Syst 33:7537–7547

Trefethen LN (2013) Approximation theory and approximation practice. SIAM

Žunkovič B (2023) Positive unlabeled learning with tensor networks. Neurocomputing 126556

Verstraete F, Cirac JI (2006) Matrix product states represent ground states faithfully. Phys Rev B 73(9):094423

Vidal G (2007) Entanglement renormalization. Phys Rev Lett 99(22):220405

Vieijra T, Vanderstraeten L, Verstraete F (2022) Generative modeling with projected entangled-pair states. arXiv:2202.08177

White SR (1992) Density matrix formulation for quantum renormalization groups. Phys Rev Lett 69(19):2863

White SR (1993) Density-matrix algorithms for quantum renormalization groups. Phys Rev B 48(14):10345

Xiao H, Rasul K, Vollgraf R (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747

Zaletel MP, Pollmann F (2020) Isometric tensor network states in two dimensions. Phys Rev Lett 124:037201. https://doi.org/10.1103/PhysRevLett.124.037201

Žunkovič B, Ilievski E (2024) Grokking phase transitions in learning local rules with gradient descent. J Mach Learn Res 25(199):1–52