Tensor networks in machine learning

Richik Sengupta, Soumik Adhikary, Ivan Oseledets and Jacob Biamonte

A tensor network is a type of decomposition used to express and approximate large arrays of data. A given dataset, quantum state, or higher-dimensional multilinear map is factored and approximated by a composition of smaller multilinear maps. This is reminiscent to how a Boolean function might be decomposed into a gate array: this represents a special case of tensor decomposition, in which the tensor entries are replaced by 0, 1 *and the factorisation becomes exact. The associated techniques are called tensor network methods: the subject developed independently in several distinct fields of study, which have more recently become interrelated through the language of tensor networks. The tantamount questions in the field relate to expressability of tensor networks and the reduction of computational overheads. A merger of tensor networks with machine learning is natural. On the one hand, machine learning can aid in determining a factorisation of a tensor network approximating a data set. On the other hand, a given tensor network structure can be viewed as a machine learning model. Herein the tensor network parameters are adjusted to learn or classify a data-set. In this survey we review the basics of tensor networks and explain the ongoing effort to develop the theory of tensor networks in machine learning.*

1 Introduction

Tensors networks are ubiquitous in most areas of modern science including data science [11], condensed matter physics [19], string theory [22] and quantum computer science. The manners in which tensors are employed/treated exhibit significant overlap across many of these areas. In data science, tensors are used to represent large datasets. In condensed matter physics and in quantum computer science, tensors are used to represent states of quantum systems.

Manipulating large tensors comes at a high computational cost [28]. This observation has inspired techniques for tensor decompositions that would reduce computational complexity while preserving the original data that they represents. Such techniques are now known as tensor network methods.

Tensor networks have risen to prominence in the last fifteen years, with several European schools playing leading roles in their modern development, as a means to describe and approximate quantum states (see the review [14]). The topic however dates back much further, to work of Penrose [32] and in retrospect, even arose as special cases in work of Cayley [10]. Tensor networks have a rich modern history in mathematical physics [32], in category theory [44], in computer science, algebraic logic and related disciplines [1]. Such techniques are now becoming more common in data science and machine learning (see the reviews [12, 13]).

1.1 Basic material about multilinear maps

It might be stated that the objective of linear algebra is to classify linear operators up to isomorphism and study the simplest representative in each equivalence class. This motivates the prevalence of decompositions such as SVD, LU and the Jordan normal form. A special case of linear operators are linear maps from a vector space V to an arbitrary field like $\mathbb{C}$ or $\mathbb{R}$. These linear maps form the dual space (vector space of covectors) $V^\star$ to our vector space V.

A natural generalisation of linear maps is provided by the multilinear maps, i.e., maps that are linear in each argument when the values of other arguments are fixed. For a given pair of non-negative integers p and q, a type-(p,q) tensor T is defined as a multilinear map

$$T\colon \underbrace{V^* \times \cdots \times V^*}_{p\text{ copies}} \times \underbrace{V \times \cdots \times V}_{q\text{ copies}} \to \mathbb{K}, \tag{1}$$

where $\mathbb{K}$ is an arbitrary field. The tensor T is said to be of order (valence) $p+q$. Note that some authors refer to this as rank $p+q$, but we will never do that.

It is often more convenient to view tensors as elements of a vector space known as the tensor product space. Thus, the above (p,q)-tensor T in this alternative interpretation can be defined as an element

$$T \in \underbrace{V \otimes \cdots \otimes V}_{p\text{ copies}} \otimes \underbrace{V^* \otimes \cdots \otimes V^*}_{q\text{ copies}}.$$

Moreover, the universality property of tensor products of vector spaces states that any multilinear map can be replaced by a unique linear map acting from the tensor product of vector spaces to the base field.

If we assume the axiom of choice, every vector space admits a Hamel basis. If $\mathbf{e}_i$ is such a basis in V, then the components of a tensor T are therefore the coefficients of T with respect to the basis $\mathbf{e}_i$ and its dual basis $\boldsymbol{\varepsilon}^j$ (basis of the dual space V^*), that is

$$T = T^{i_1\ldots i_p}_{j_1\ldots j_q}\mathbf{e}_{i_1}\otimes\cdots\otimes\mathbf{e}_{i_p}\otimes\boldsymbol{\varepsilon}^{j_1}\otimes\cdots\otimes\boldsymbol{\varepsilon}^{j_q}. \tag{2}$$

Adopting Einstein's summation convention, summation over repeated indices is implied in (2).

Returning to (1), given p covectors $(c^1,\ldots,c^p)$ and q vectors $(v_1,\ldots,v_q)$, the value of the tensor is evaluated as

$$T^{i_1\ldots i_p}_{j_1\ldots j_q}c^1(\mathbf{e}_{i_1})\times\cdots\times c^p(\mathbf{e}_{i_p})\times\boldsymbol{\varepsilon}^{j_1}(v_1)\times\cdots\times\boldsymbol{\varepsilon}^{j_q}(v_q)\in\mathbb{K}, \tag{3}$$

where $c^k(\mathbf{e}_{i_k})$ and $\boldsymbol{\varepsilon}^{j_l}(v_l)$ are numbers obtained by evaluating covectors (functionals) at the corresponding vectors. In quantum computation, the basis vectors $\mathbf{e}_{i_k}$ are denoted by $|i_k\rangle$ and the basis covectors $\boldsymbol{\varepsilon}^{j_l}$ are denoted by $\langle j_l|$. Using Dirac's notation, tensor products are written in compact form as

$$\mathbf{e}_{i_m}\otimes\mathbf{e}_{i_l} := |i_m i_l\rangle,\quad \boldsymbol{\varepsilon}^{j_m}\otimes\boldsymbol{\varepsilon}^{j_l} := \langle j_m j_l|,\quad \mathbf{e}_{i_m}\otimes\boldsymbol{\varepsilon}^{j_l} := |i_m\rangle\langle j_l|,$$

and the tensor T takes the form

$$T = T^{i_1\ldots i_p}_{j_1\ldots j_q}|i_1\ldots i_p\rangle\langle j_1\ldots j_p|.$$

Similarly, given a tuple of p covectors $(c^1,\ \ldots,c^p)$ and q vectors $(v_1,\ldots,v_q)$ we write them as $\langle c_1\ldots c_p|$ and $|v_1\ldots v_q\rangle$, as elements of the corresponding tensor product space(s).

In this notation the evaluation (3) takes the form

$$T^{i_1\ldots i_p}_{j_1\ldots j_q}\langle c_1\ldots c_p|i_1\ldots i_p\rangle\langle j_1\ldots j_p|v_1\ldots v_q\rangle.$$

Since in quantum computation the vector spaces under consideration as well as their duals are Hilbert spaces, Riesz's representation theorem for linear functionals implies that the evaluation $\langle *|*\rangle$ above can be seen as an inner product.

Finally, a tensor can be identified with the array of coefficients $T^{i_1\ldots i_p}_{j_1\ldots j_q}$ in a specific basis decomposition. This approach is not basis independent, but is useful in applications. Henceforth in this review we will fix the standard basis, which will establish a canonical isomorphism between the vector space and its dual. In the simplest case $p=q=1$, for example, this gives us the following equivalences:

$$T_{ij}\cong T_{ji}\cong T^{ij}\cong T^{ji}\cong T^i_j\cong T^j_i.$$

In the more general case this leads to the equivalence between the components $T^{i_1\ldots i_p}_{j_1\ldots j_q}$ and $T_{j_1\ldots j_q i_1\ldots i_p}$. Given a valence-$m$ tensor T, the total number of tensors in the equivalence class formed by raising, lowering and exchanging indices has cardinality $(1+m)!$ (see [4]). Recognising this arbitrariness, Penrose considered a graphical depiction of tensors [32], stating that "it now ceases to be important to maintain a distinction between upper and lower indices." This convention is widely adopted in the contemporary literature.

1.2 Tensor trains aka matrix product states

Consider a tensor T with components $T_{j_1j_2\ldots j_n}$ with $j_k=1,2,\ldots,d$. Hence, T has d^n components, a number that can exceed the total number of electrons in the universe when d is as small as 2 and $n\approx 300$. Clearly, storing the components of such a large tensor in a computer memory and subsequently manipulating it can become impossible. The good news is that for most practical purposes, a tensor typically contains a large amount of redundant information. This enables factoring of T into a sequence of "smaller" tensors.

Tensor trains (see [31]) and matrix product states (MPS) [30, 33] arose in data science and in condensed matter physics, where it was shown that any tensor T with components $T_{j_1j_2\cdots j_n}$ admits a decomposition of the form

$$T_{j_1j_2\ldots j_n} = \boldsymbol{\alpha}^\dagger A^{(1)}_{j_1}A^{(2)}_{j_2}\cdots A^{(n)}_{j_n}\,\boldsymbol{\beta} \tag{4}$$

where $A^{(k)}_{j_k}$ is an $(r_{k-1}\times r_k)$-dimensional matrix, and $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ as r_0- and r_n-dimensional vectors, respectively.

Likewise, an n-qubit state $|\psi\rangle\in(\mathbb{CP}^2)^{\otimes n}$, written in the computational basis as $|\psi\rangle=\sum_{j_1j_2\cdots j_n}T_{j_1j_2\ldots j_n}|j_1j_2\ldots j_n\rangle$, $j_k\in\{0,1\}$, can equivalently be expressed as

$$|\psi\rangle = \sum_{j_1\ldots j_n}\langle\alpha|A^{(1)}_{j_1}A^{(2)}_{j_2}\cdots A^{(n)}_{j_n}|\beta\rangle|j_1j_2\ldots j_n\rangle. \tag{5}$$

Here $A^{(k)}_{j_k}$ is a $r_{k-1}\times r_k$ dimensional matrix and $|\alpha\rangle$, $|\beta\rangle$ are r_0 and r_n dimensional vectors, respectively. Note that here we are adhering to the braket notation, as is customary in quantum mechanics. The representation of $|\psi\rangle$ in (5) is called the matrix product state representation with an open boundary condition (OBC-MPS). See Figure 1 for a graphical representation.

Yet another useful MPS decomposition that a state might admit is the MPS with periodic boundary condition (PBC-MPS) [30]. The PBC-MPS representation of an n-qubit state

$$|\psi\rangle = \sum_{j_1j_2\ldots j_n}T_{j_1j_2\ldots j_n}|j_1j_2\ldots j_n\rangle,\quad j_k=0,1,$$

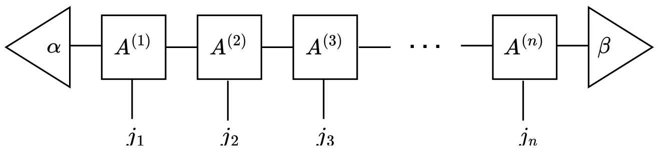


Figure 1. Graphical representation of tensor trains (open boundary condition—matrix product state representation). See (4), (5). Each index in the tensor $T_{j_1\ldots j_n}$ is represented in the diagram by an open wire pointing downwards. We call these wires physical bonds. The horizontal wires represent extra indices which are summed over. Such internal wires are known as virtual bonds.

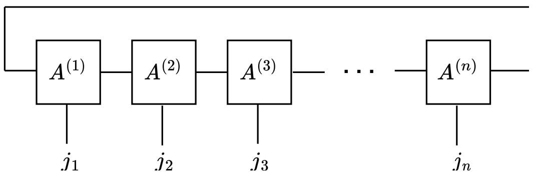


Figure 2. Graphical representation of the matrix product state in (6).

is given by

$$|\psi\rangle = \sum_{j_1 j_2 \dots j_n} \mathrm{Tr}(A^{(1)}_{j_1} A^{(2)}_{j_2} \cdots A^{(n)}_{j_n}) \, |j_1 j_2 \dots j_n\rangle, \tag{6}$$

where $A^{(k)}_{j_k}$ is an $r \times r$ matrix. The graphical representation of a PBC-MPS is shown in Figure 2.

An n-qubit state $|\psi\rangle = \sum_{j_1 \dots j_n} T_{j_1 \dots j_n} |j_1 \dots j_n\rangle$ has 2^n independent coefficients $T_{j_1 \dots j_n}$. The MPS representation of $|\psi\rangle$, on the other hand, is less data intensive. If $A^{(k)}_{j_k}$ is an $r \times r$ matrix for all k, the size of the representation becomes $2nr^2$, which is linear in n for a constant r. The point of the method is to choose r such that a good and compact approximation of $|\psi\rangle$ is obtained. The number r is often also referred to as the virtual bond dimension. Data compression becomes even more effective if the MPS is site independent, that is, if $A^{(k)}_{j_k} = A_{j_k}$ for all k. It has been shown that a site-independent representation of a PBC-MPS always exists if the state is translation invariant [33]. Note that MPS is invariant under the transformation $A_j \to P A_j P^{-1}$ for any invertible P; this follows from the cyclicity of the trace operator. Therefore, it is often customary to impose an additional constraint here, viz. $\sum_j A_j A_j^\dagger = \mathbf{1}$, in order to fix the gauge freedom [14] (see also the connections to algebraic invariant theory [5]).

2 Machine learning: classical to quantum

2.1 Classical machine learning

At the core of machine learning is the task of data classification. In this task, we are typically provided with a labelled dataset $\mathcal{S} = \{(\mathbf{x}_j, \mathbf{y}_j)\}_{j=1}^M$, where the vectors $\mathbf{x}_j \in \mathbb{R}^N$ are the input data (e.g., animal images) and the vectors $\mathbf{y}_j \in \mathbb{R}^d$ are the corresponding labels (e.g., animal types). The objective is to find a suitable machine learning model F with tunable parameters $\boldsymbol{\theta} \in \mathbb{R}^k$ that generates the correct label for a given input $\mathbf{x} \in \mathbb{R}^N$. Note that our model can be looked upon as a family of functions parameterised by $\boldsymbol{\theta}$: F takes a data vector as an input and outputs a predicted label; for an input datum $\mathbf{x}_j$, the predicted label is $F(\mathbf{x}_j, \boldsymbol{\theta})$. To ensure that our model generates the correct labels, it needs to be trained; in order to accomplish this, a training set $\mathcal{T} \subset \mathcal{S}$ is chosen, the elements of which serve as input data to train F. Training requires a cost function,

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{(\mathbf{x}_j, \mathbf{y}_j) \in \mathcal{T}} D(F(\mathbf{x}_j, \boldsymbol{\theta}), \mathbf{y}_j), \tag{7}$$

where $D(\cdot, \cdot)$ measures the mismatch between the real label and the estimated label. Typical choices for D include, e.g., the negative log-likelihood function, mean squared errors (MSE), etc. [37]. By minimising (7) with respect to $\boldsymbol{\theta}$, one obtains the value $\boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$, which completes the training. After the model F has been trained, we can evaluate its performance by feeding it inputs from $\mathcal{S} \setminus \mathcal{T}$ (often referred to as the validation set) and checking for classification accuracy.

For a more formal description, let us assume that the dataset $\mathcal{S}$ is sampled from a joint probability distribution with a density function $p(\mathbf{x}, \mathbf{y})$. The role of a classifier model is to approximate the conditional distribution $p(\mathbf{y}|\mathbf{x})$. The knowledge of $p(\mathbf{x}, \mathbf{y})$ allows us, in principle, to establish theoretical bounds on the performance of the classifier. Consider the generalisation error (also called risk), defined as $\mathcal{G}(\boldsymbol{\theta}) = \mathbb{E}_{p(\mathbf{x},\mathbf{y})}(D(F(\mathbf{x}, \boldsymbol{\theta}), \mathbf{y}))$. A learning algorithm is said to generalise if $\lim_{M' \to \infty}(\mathcal{L}(\boldsymbol{\theta})/M') = \mathcal{G}(\boldsymbol{\theta})$; here M' is the cardinality of the training set. However, since in general we do not have access to $p(\mathbf{x}, \mathbf{y})$ we can only attempt to provide necessary conditions to bound the difference of the generalisation error and the empirical error by checking certain stability conditions to ensure that our learning model is not too sensitive to noise in the data [8]. For example, we can try to ensure that our learning model is not affected if one of the data points is left out during training. The technique of regularisation prevents overfitting.

Several different types of machine learning models F exist, which range from fairly elementary models, such as perceptrons, to highly involved ones, such as deep neural networks (DNNs) [24, 38]. The choice of F is heavily dependent on the classification task, the type of the dataset, and the desired training properties. Consider a dataset $\mathcal{S}$ with two classes (a binary dataset) that is linearly separable. That is, (i) $\mathbf{y}_j \in \{-1, 1\}$ and (ii) one can construct a hyperplane that separates the input data belonging to the different classes. Finding this hyperplane, aka the decision boundary, is therefore sufficient for data classification in $\mathcal{S}$. This task can be accomplished with a simplistic machine learning model—the perceptron—which is in fact a single McCulloch–Pitts neuron [26]. The algorithm starts with the candidate solution for the hyperplane $\mathbf{w}^\top \cdot \mathbf{x} + b = 0$, where $\mathbf{w}, b$ are tunable parameters and play the role of $\boldsymbol{\theta}$. It is known from the perceptron convergence algorithm that one can always find a set of parameters $\mathbf{w} = \mathbf{w}^\star, b = b^\star$, such that for every $\mathbf{x}_j$, if $\mathbf{y}_j = -1$, then $\mathbf{w}^{\star\top} \cdot \mathbf{x}_j + b^\star \le 0$, while if $\mathbf{y}_j = 1$, then $\mathbf{w}^{\star\top} \cdot \mathbf{x}_j + b^\star > 0$.

Most datasets of practical importance, however, are not linearly separable and consequently cannot be classified by the perceptron model alone. Assuming that $\mathcal{S}$ is a binary dataset which is not linearly separable, we consider a map $\Lambda\colon \mathbb{R}^N \to \Gamma$, $\dim(\Gamma) > N$, with the proviso that Λ is nonlinear in the components of its inputs [6]. In machine learning Λ is called a feature map and the vector space Γ is known as a feature space. Thus Λ nonlinearly maps each input datum $\mathbf{x}_j$ to a vector $\Lambda(\mathbf{x}_j)$ in the feature space. The significance of this step follows from Cover's theorem on separability of patterns [16], which suggest that the transformed dataset $S' = \{(\Lambda(\mathbf{x}_j), \mathbf{y}_j)\}_{j=1}^{M}$ is more likely to be linearly separable. For a good choice of Λ, the data classification step now becomes straightforward, as it is sufficient to fit a hyperplane to separate the two classes in the feature space. Indeed, the sought-for hyperplane can be constructed, using the perceptron model, provided the feature map Λ is explicitly known. Actually, a hyperplane can still be constructed even when Λ is not explicitly known. A particularly elegant way to accomplish this is via the support vector machine (SVM) [15], by employing the so-called kernel trick [45].

Consider again the binary dataset $S' = \{(\Lambda(\mathbf{x}_j), \mathbf{y}_j)\}_{j=1}^{M}$. The aim is to construct a hyperplane that separates the samples belonging to the two classes. In addition, we would like to maximise the margin of separation. Formally, we search for a set of parameters $\mathbf{w} = \mathbf{w}^\star, b = b^\star$ such that for every $\mathbf{x}_j$, if $\mathbf{y}_j = -1$, then $\mathbf{w}^{\star\top} \cdot \Lambda(\mathbf{x}_j) + b^\star \le -1$, while if $\mathbf{y}_j = 1$, then $\mathbf{w}^{\star\top} \cdot \Lambda(\mathbf{x}_j) + b^\star \ge 1$. As in the perceptron model, the SVM algorithm starts with the candidate solution $\mathbf{w}^\top \cdot \mathbf{x} + b$ and the parameters $\mathbf{w}, b$ are tuned based on the training data. An interesting aspect of the SVM approach, however, is the dependence of the algorithm on a special subset of the training data called support vectors, namely, the ones that satisfy the relation $\mathbf{w}^{\star\top} \cdot \Lambda(\mathbf{x}_j) + b^\star = \pm 1$. We assume that there are S such vectors $\Lambda(\mathbf{x}_j^{(s)})$, $j = 1, \dots, S$. With some algebra, which we omit here, it can be shown that the decision boundary is given by the hyperplane

$$\sum_{j=1}^{S} a_j^\star \mathbf{y}_j [\Lambda(\mathbf{x}_j^{(s)})^\top \Lambda(\mathbf{x})] + b^\star = 0, \tag{8}$$

where

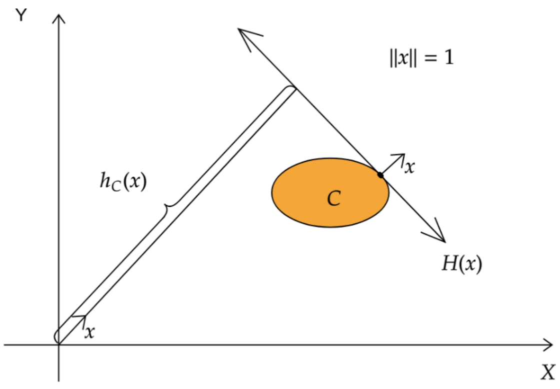
$$b^\star = \frac{1}{S} \sum_{j=1}^{S} \Big(\mathbf{y}_j - \sum_{k=1}^{S} a_j^\star \mathbf{y}_j [\Lambda(\mathbf{x}_j^{(s)})^\top \Lambda(\mathbf{x}_k^{(s)})] \Big), \tag{9}$$

$$a^\star = \arg\max_{a} \Big(\sum_j a_j - \frac{1}{2} \sum_j \sum_k a_j a_k \mathbf{y}_j \mathbf{y}_k \Lambda(\mathbf{x}_j)^\top \Lambda(\mathbf{x}_k) \Big) \tag{10}$$

with the additional conditions $\sum_j a_j \mathbf{y}_j = 0$ and $a_j \ge 0$. All summations in (10) are over the entire training set. We make a key observation here: from (8), (9) and (10) we see that the expression of the decision boundary has no explicit dependence on the feature vectors $\Lambda(\mathbf{x}_j)$. Instead, the dependence is solely on the inner products of the form $\Lambda(\mathbf{x}_j)^\top \Lambda(\mathbf{x}_k)$. This allows us to use the kernel trick.

Support functions in optimisation

If $C \subset \mathbb{R}^n$ is a nonempty closed convex set, the support function $h_C \colon \mathbb{R}^n \to \mathbb{R}$ of C is given by $h_C(x) = \sup\{\langle x|c\rangle : c \in C, x \in \mathbb{R}^n\}$. The hyperplane $H(x) = \{y \in \mathbb{R}^n : \langle y|x\rangle = h_C(x)\}$ is called a supporting hyperplane of C with outer unit normal vector x. The function $h_C(x)$ outputs the signed distance of $H(x)$ from the origin. The data points that lie on $H(x)$ are called support vectors in the machine learning literature.



Graphical representation of the support function and supporting hyperplane.

Kernel functions

The concept of the kernel function originates in the theory of Reproducing Kernel Hilbert Spaces (RKHS). Consider a Hilbert space H consisting of real-valued functions defined on an arbitrary set X. We can define an evaluation functional $E_x(f) = f(x)$ that maps each function to a real number. If for every $x \in X$ the linear functional E_x is bounded on H, we call H an RKHS. Applying Riesz's theorem mentioned in the introduction, we have the representation $E_x(f) = \langle f|K_x\rangle$, where $K_x \in H$. It follows that $E_y(K_x) = \langle K_x|K_y\rangle = K_x(y) = K(x,y)$.

The function $K(x,y) \colon X \times X \to \mathbb{R}$ is called the reproducing kernel of the space H. Briefly, in an RKHS the evaluation of a function at a point can be replaced by taking an inner product with a function determined by the kernel in the associated function space.

Now given a feature map $\Phi \colon X \to \Gamma$, where Γ is a Hilbert space, we define a normed space $H_\Phi = \{f \colon X \to \mathbb{C} : \exists g \in \Gamma, f(x) = \langle g|\Phi(x)\rangle \; \forall x \in X\}$ with the norm $\|f\|_\Phi = \inf\{\|g\|_\Gamma : g \in G, f(x) = \langle g|\Phi(x)\rangle \; \forall x \in X\}$. One can show that H_Φ is a RKHS with the kernel $K(x,y) = \langle \Phi(x)|\Phi(y)\rangle$.

Formally, a kernel $k(\mathbf{x}_j, \mathbf{x}_k)$ is defined as a function that computes the inner product of the images of $\mathbf{x}_j, \mathbf{x}_k$ in the feature space, i.e., $k(\mathbf{x}_j, \mathbf{x}_k) = \Lambda(\mathbf{x}_j)^\top \Lambda(\mathbf{x}_k)$. Accordingly, all the inner products in (8), (9) and (10) can be replaced by the corresponding kernels. We can then use the kernel trick, that is, assign an analytic expression for the kernel in (8), (9) and (10), provided that the expression satisfies all the required conditions for it to be an actual kernel [39]. Indeed, every choice of a kernel can be implicitly associated to a feature map Λ. However, in the current approach we do not need to know the explicit form of the feature map. In fact, this is the advantage of the kernel trick, as calculating $\Lambda(\mathbf{x})$ is less efficient than using the kernels directly.

Most modern applications in machine learning, however, involve deep neural networks (DNNs) [43]. A DNN can also be regarded as a collection of perceptrons arranged in a definite fashion. The architecture of a DNN is best understood and visualised in the form of a graph. Specifically, a DNN is composed of an input layer, an output layer, and several intermediate hidden layers. Each layer consists of several nodes. Each of these nodes represents a neuron. Edges are allowed to exist between nodes belonging to adjacent layers only: nodes in layer j share edges with nodes in layer $j+1$ and nodes in layer $j-1$. For the sake of simplicity, we consider the case where all the nodes in the j-th layer are connected to all the nodes in the $(j \pm 1)$-th layers—a fully connected neural network.

A DNN takes a data vector $\mathbf{x}$ as an input (at the input layer). This input is subsequently manipulated by the neurons in the next layer (the first hidden layer) to output a transformed vector $\mathbf{x}^{(1)}$, and this process is repeated till the last layer (output layer) is reached. Consider the k-th neuron in the j-th layer; for convenience, we denote this neuron by (k,j). It receives an input vector $\mathbf{x}^{(j-1)}$ whose components are the outputs of the neurons in the $(j-1)$-th layer, and then transforms $\mathbf{x}^{(j-1)}$ by the rule

$$\mathbf{x}^{(j-1)} \to \Psi((\mathbf{w}_k^{(j-1)})^\top \cdot \mathbf{x}^{(j-1)} + b_k^{(j)}) = \mathbf{x}_k^{(j)}, \qquad (11)$$

where $\mathbf{w}_k^{(j-1)}$ are weights associated with the edges that connect the neuron (k,j) to the neurons in the previous layer, $b_k^{(j)}$ is the bias corresponding to the neuron (k,j), and $\Psi(\cdot)$ is a differentiable nonlinear function known as the activation function. This is the fundamental mathematical operation that propagates data in a DNN, layer by layer, in the forward direction (input layer to output layer), through a series of complex transformations. The training component of the algorithm is however accomplished through the back-propagation step [35]: a cost function is calculated by comparing the signal at the output layer (the model predicted label for the data $\mathbf{x}$) and the desired signal (the actual label for the data $\mathbf{x}$), based on which the weights and the biases are adjusted so that the cost function is minimised. Apart from supervised learning, DNNs are routinely used for unsupervised learning, including generative learning. In generative learning, given access to a training dataset, a machine learning model learns its underlying probability distribution for future sample generation. To formulate this mathematically, consider a dataset $\mathcal{S} = \{\mathbf{x}_j\}$, whose entries $\mathbf{x}_j \in \mathbb{R}^N$ are independent and identically distributed vectors and are sampled according to a distribution $q(\mathbf{x})$. The purpose of a generative model is to approximate $q(\mathbf{x})$, given the access to training data from the dataset $\mathcal{S}$. To achieve this, a machine learning model (with tunable parameters $\boldsymbol{\theta}$) is trained so that the model generated distribution, $p(\mathbf{x}, \boldsymbol{\theta})$, mimics the true distribution. The standard practice in generative learning is to minimise the negative log-likelihood with respect to the model parameters, which is tantamount to minimising the Kullback–Leibler divergence $D_{\mathrm{KL}}(q(\mathbf{x})||p(\mathbf{x}, \boldsymbol{\theta}))$ between the two distributions.

2.2 Variational algorithms and quantum machine learning (QML)

Variational quantum computing has emerged as the preeminent model for quantum computation. The model merges ideas from machine learning to better utilise modern quantum hardware.

Mathematically, the problem in variational quantum computing can be formulated as follows: given (i) a variational quantum circuit (aka ansatz) $U(\boldsymbol{\theta}) \in \mathrm{U}_{\mathbb{C}}(2^n)$ which produces an n-qubit variational state $|\psi(\boldsymbol{\theta})\rangle = U(\boldsymbol{\theta})|0\rangle^{\otimes n}$; $\boldsymbol{\theta} \in [0, 2\pi)^{\times p}$, (ii) an objective function $\mathcal{H} \in \mathrm{herm}_{\mathbb{C}}(2^n)$, and (iii) the expectation $\langle\psi(\boldsymbol{\theta})|\mathcal{H}|\psi(\boldsymbol{\theta})\rangle$, find

$$\boldsymbol{\theta}^\star \in \underset{\boldsymbol{\theta} \in [0,2\pi)^{\times p}}{\arg\min}\ \langle\psi(\boldsymbol{\theta})|\mathcal{H}|\psi(\boldsymbol{\theta})\rangle.$$

Then $|\psi(\boldsymbol{\theta}^\star)\rangle$ will approximate the ground state (eigenvector corresponding to the lowest eigenvalue) of the Hamiltonian $\mathcal{H}$. The operator $\mathcal{H}$, often called the problem Hamiltonian, can suitably treat several classes of problems so that the solution to a problem is encoded in the ground state of $\mathcal{H}$. The variational model of quantum computation was shown to be universal in [3].

Quantum machine learning, both discriminative and generative, emerged as an important application of variational algorithms with suitable modifications to the aforementioned scheme. Indeed, by their very design, variational algorithms are well suited to implement machine learning tasks on a quantum device. The earlier developments in QML came mainly in the form of classification tasks [18, 27].

Classification of a classical dataset $\mathcal{S} = \{(\mathbf{x}_j, \mathbf{y}_j))\}_{j=1}^M$ on quantum hardware typically involves four steps. First, the input vector $\mathbf{x}_j$ is embedded into an n-qubit state $|\psi(\mathbf{x}_j)\rangle$. The effect of data encoding schemes on the expressive power of a quantum machine learning model was studied in [42]. What is the most effective data embedding scheme? Although there are several interesting candidates [25, 34], this question remains largely unanswered. In the second step, a parameterised ansatz $U(\boldsymbol{\theta})$ is applied to $|\psi(\mathbf{x}_j)\rangle$ to output $|\psi(\mathbf{x}_j, \boldsymbol{\theta})\rangle$. A number of different ansatzes are in use today, including the hardware efficient ansatz, the checkerboard ansatz, the tree tensor network ansatz, etc., which are chosen according to the application and implementation specifications under consideration. The third step in the process is where data is read out of $|\psi(\mathbf{x}_j, \boldsymbol{\theta})\rangle$: expectation values of certain chosen observables (Hermitian operators) are calculated with respect to $|\psi(\mathbf{x}_j, \boldsymbol{\theta})\rangle$ to generate a predicted label $F(\mathbf{x}_j, \boldsymbol{\theta})$. The measured operators are typically the Pauli strings, which form a basis in $\mathrm{herm}_{\mathbb{C}}(2^n)$. In the final step, a cost function is constructed as in (7) and minimised by tuning $\boldsymbol{\theta}$. This approach was used in several studies to produce successful classifications in practical datasets (see, e.g., [40]).

An interesting variation of the approach described above was shown in [21, 41] to implement data classification based on the kernel trick. In this method the Hilbert space is treated as a feature space and the data embedding step, $\mathbf{x}_j \to |\psi(\mathbf{x}_j)\rangle$, as a feature map. A quantum circuit is used directly to compute the inner product $\langle \psi(\mathbf{x}_j) | \psi(\mathbf{x}_k) \rangle$, using, e.g., the swap test, which is then employed for data classification by means of classical algorithms such as SVMs.

Quantum machine learning has also been used to classify genuine quantum data. Some prominent examples of such applications include: classifying phases of matter [47], quantum channel discrimination [23], and entanglement classification [20]. Other machine learning problems with quantum mechanical origins that have been solved by variational algorithms include quantum data compression [36] and denoising of quantum data [7]. Both of these applications use a quantum autoencoder. A quantum autoencoder, much like its classical counterparts, consists of two parts: an encoder and a decoder. The encoder removes the redundant information from the input data to produce a minimal low-dimensional representation. This process is known as feature extraction. To ensure that the minimal representation produced by the encoder is efficient, a decoder is used which takes the output of the encoder and tries to reconstruct the original data. Thus, in an autoencoder, both the encoder and the decoder are trained in tandem to ensure that the input at the encoder and the output at the decoder closely match each other. While in the classical case the encoders and the decoders are chosen to be neural networks, in the quantum version of an autoencoder neural networks are replaced by variational circuits.

Considerable advances were made on the front of quantum generative learning as well. In [2] it was shown that generative modelling can be used to prepare quantum states by training shallow quantum circuits. The central idea is to obtain the model generated probability distribution $p(\boldsymbol{\theta})$ by performing repeated measurements on a variational state $|\psi(\boldsymbol{\theta})\rangle$. The state $|\psi(\boldsymbol{\theta})\rangle$ is prepared on a short-depth circuit with a fixed ansatz and parameterised with the vector $\boldsymbol{\theta}$. The target distribution q is also constructed in much the same manner, by performing repeated measurements on the target state. The measurement basis (preferably informationally complete positive operator-valued measures), as expected, is kept to be the same in both cases. The training objective therefore is to ensure that $p(\boldsymbol{\theta})$ mimics q so that the variational circuit learns to prepare the target state. The same task in an alternate version can be looked upon as a machine-learning assisted quantum state tomography [9].

3 Tensor networks in machine learning

3.1 *Tensor networks in classical machine learning*

Recently tensor network methods have found several applications in machine learning. Here we discuss some of these applications with a focus on supervised learning models. We return to our labelled dataset $\mathcal{S} = \{(\mathbf{x}_j, \mathbf{y}_j)\}_{j=1}^{M}$, where $\mathbf{x}_j \in \mathbb{R}^N$. As mentioned earlier, there are several machine learning models F to choose from to perform a classification on the dataset $\mathcal{S}$.

However, in more abstract terms, a classifier F can be expressed as a function of the form

$$F_{\mathrm{W}}(\mathbf{x}) = \sum_{j_1, j_2, \ldots, j_N \in \{0,1\}} W_{j_1 j_2 \ldots j_N} x_1^{j_1} x_2^{j_2} \cdots x_N^{j_N}, \tag{12}$$

in the polynomial basis [29]. Here $\mathbf{x} \in \mathbb{R}^N$ is an input datum and $x_k \in \mathbb{R}$ is the k-th component of $\mathbf{x}$. The tensor $W_{j_1 j_2 \ldots j_N}$ is what we call as the weight tensor, which encodes the tunable parameters in F. Going back to the case of binary classification, that is, $\mathbf{y}_j \in \{1, -1\}$, $F(\mathbf{x})$ can be regarded as a surface in $\mathbb{R}^{N+1}$ that can be tuned (trained) so that it acts as a decision boundary between the two classes of input data. Indeed, the training can be accomplished by the minimisation

$$\min_{\mathrm{W}} \Big(\sum_{j=1}^{M} |\mathrm{sgn}(F_{\mathrm{W}}(\mathbf{x}_j)) - \mathbf{y}_j|^2 \Big),$$

where $\mathrm{sgn}(F_{\mathrm{W}}(\mathbf{x}_j))$ is the predicted label. However, in practice we run into a bottleneck when we compute (12), since this involves 2^N components of the weight tensor. One way to circumvent this bottleneck is to express the weight tensor as a MPS. Following the observation in Section 1.2, for a suitable choice of the virtual bond dimension r, the MPS representation of the weight tensor W would involve only $O(\mathrm{poly}\, N)$ components, thus making the computation of (12) less resource intensive [46]. Here it is worth noting that we could alternatively have opted for any other basis in the decomposition of the function F, depending on the optimisation problem at hand.

Yet another application of tensor networks in machine learning can be seen in DNNs. Consider the transformation in (11). For most practically relevant DNNs, this transformation is highly resource intensive. This is due to the fact that the vectors $\mathbf{x}^{(j-1)}$ are typically very large and hence computing the inner products $(\mathbf{w}_k^{(j-1)})^\top \cdot \mathbf{x}^{(j-1)}$ is difficult. This computation can be made efficient by using MPS. In order to do this, the vectors $\mathbf{w}_k^{(j-1)}, \mathbf{x}^{(j-1)}$ are first reshaped, converting them into tensors and then expressing them as an MPS. When we express a vector as a MPS we need to keep track of much fewer components compared to the original representation. This makes the computation of (11) tractable.

3.2 *The parent Hamiltonian problem*

Consider the quantum state preparation problem using a variational algorithm. Given a variational circuit $U(\boldsymbol{\theta})$ and an n-qubit target state $|t\rangle$, tune $\boldsymbol{\theta} \to \boldsymbol{\theta}^\star$ such that $U(\boldsymbol{\theta}^\star)|0\rangle^{\otimes n}$ approximates $|t\rangle$. To accomplish this task one needs to construct a Hamiltonian $\mathcal{H} \in \mathrm{herm}_{\mathbb{C}}(2^n)$ with $|t\rangle$ as its unique ground state, which will serve as the objective function of the algorithm. Constructing such a Hamiltonian for a given target state is known as the parent Hamiltonian problem. The simplest recipe is to set $\mathcal{H} = \mathbf{1} - |t\rangle\langle t|$. This construction is however not always useful, because expressing $\mathcal{H}$ in the basis of Pauli strings—the basis of measurement—may require an exponential number of terms. Thus estimating the expectation of $\mathcal{H}$ in polynomial time becomes unfeasible.

Ideally, we want the sought-for Hamiltonian to enjoy the following properties:

1. The Hamiltonian is non-negative.
2. The Hamiltonian has a non-degenerate (unique) ground state $|t\rangle$.
3. The Hamiltonian is gapped. An n-qubit Hamiltonian $\mathcal{H}(n) \geq 0$ is said to be gapped if
$$\lim_{n\to\infty}[\dim\ker\mathcal{H}(n)] = 1. \tag{13}$$
Validity of (13) ensures that $\mathcal{H}(n)$ is gapped for all finite n.
4. The Hamiltonian is local. An n-qubit Hamiltonian $\mathcal{H}(n)$ is said to be local if it can be expressed as
$$\mathcal{H}(n) = \sum_{j\in 2^V} h(j),$$
where V is the set of n symbols (qubits) and $h(j) = \bigotimes_{k\in j} P_k \in \mathrm{herm}_{\mathbb{C}}(2^n)$, where $P \in \mathrm{herm}_{\mathbb{C}}(2)$. The Hamiltonian $\mathcal{H}(n)$ is said to be k-local if none of the $h(j)$'s operates on more than k symbols (qubits) nontrivially; here a trivial operation refers to the case when P_k is the identity for some index k.
5. The Hamiltonian must have $O(\operatorname{poly} n)$ terms when expressed in the Pauli basis. The number of terms in a Hamiltonian when expressed in the Pauli basis is also known as the cardinality of the Hamiltonian, $\|\mathcal{H}\|_{\mathrm{card}}$ (see [3]).

Hamitonians with such properties can indeed be constructed if $|t\rangle$ admits a matrix product state, albeit with the additional requirement that $|t\rangle$ must satisfy the injectivity condition. For the parent Hamiltonian construction consider the following setting. Let $|t\rangle$ be an n-qubit state written as a translation-invariant and site-independent MPS with periodic boundary conditions:
$$|t\rangle = \sum_{j_1\ldots j_n} \mathrm{Tr}(A_{j_1}\cdots A_{j_n})|j_1\ldots j_n\rangle,$$
where $A_{j_k} \in \mathrm{Mat}_{\mathbb{C}}(r)$. For the sake of brevity we will call these matrices Kraus operators.[1] Consider the map
$$\Gamma_L : X \to |\psi^{(L)}\rangle_X = \sum_{j_1\ldots j_L} \mathrm{Tr}(XA_{j_1}\cdots A_{j_L})|j_1\ldots j_L\rangle,$$
where $X \in \mathrm{Mat}_{\mathbb{C}}(r)$. We say that the state $|t\rangle$ is injective with injectivity length L if the map Γ_L is injective. Several corollaries follow from this definition. A particularly useful one connects the notion of injectivity to the rank of reduced density matrices. It asserts that for an L-qubit reduced density matrix, $\rho^{(L)}$, of $|t\rangle$, we have $\mathrm{rank}(\rho^L) = r^2$ if injectivity holds. It has been shown that in the large-n limit, $\rho^{(L)}$ is given by
$$\rho^{(L)} = \mathrm{Tr}_{n-L}(|t\rangle\langle t|) = \sum_{\alpha,\beta=1}^{r} \Lambda_\alpha |\psi_{\alpha\beta}^{(L)}\rangle\langle\psi_{\alpha\beta}^{(L)}|, \tag{14}$$
with $|\psi_{\alpha\beta}^{(L)}\rangle = \sum_{j_1\ldots j_L}\langle\alpha|A_{j_1}\cdots A_{j_L}|\beta\rangle|j_1\ldots j_L\rangle$, $|\alpha\rangle, |\beta\rangle \in \mathbb{C}^r$ and $\Lambda_\alpha \in \mathbb{R}_+$. Alternatively, this would mean that $\{|\psi_{\alpha\beta}^{(L)}\rangle\}_{\alpha\beta}$ is a linearly independent set.

The form of the reduced density matrix in (14) is particularly telling and allows us to construct the parent Hamiltonian of $|t\rangle$: $\mathcal{H} \geq 0$. We formally write our parent Hamiltonian as
$$\mathcal{H} = \sum_{j=1}^{n} h_j^{(L)}, \tag{15}$$
where $h_j^{(L)}$ operates nontrivially over L-qubits from j to $L+j$ and obeys the condition $\ker h_j^{(L)} = \mathrm{span}\,\{|\psi_{\alpha\beta}^{(L)}\rangle\}_{\alpha\beta}$. The latter condition combined with (14) ensures that $\mathrm{Tr}(h_j^{(L)}\rho_j^{(L)}) = 0$ for all j, which in turn implies that $|t\rangle \in \ker\mathcal{H}$. In fact, $|t\rangle = \ker\mathcal{H}$, provided $|t\rangle$ is injective, and so condition 1 for $\mathcal{H}$ is satisfied. Conditions 3 and 4 are satisfied naturally due to the form of $\mathcal{H}$ in (15). In addition, $\mathcal{H}$ can also seen to be frustration free, that is, $\langle t|\mathcal{H}|t\rangle = 0 \Rightarrow \langle t|h_j^{(L)}|t\rangle = 0$ for all j. Finally, it was shown in [17] that if $|t\rangle$ is injective, then $\mathcal{H}$ is gapped.

4 Conclusion

The importance of matrix product states in physics is due to the ease with which one could calculate and verify important quantities or properties, such as two-point functions, thermal properties, and more. This is also true in machine learning applications. For example, images of size 256×256 can be viewed as rank-one tensor networks on $\mathbb{R}^{256}$. Departing from this linear (train) structure results in tensors with potentially much greater expressability at the cost of many desirable properties being lost.

References

[1] J. Baez and M. Stay, Physics, topology, logic and computation: A Rosetta Stone. In *New Structures for Physics*, Lecture Notes in Phys. 813, Springer, Heidelberg, 95–172 (2011)

[2] M. Benedetti, D. Garcia-Pintos, O. Perdomo, V. Leyton-Ortega, Y. Nam and A. Perdomo-Ortiz, A generative modeling approach for benchmarking and training shallow quantum circuits. *npj Quantum Inf.* **5**, 1–9 (2019)

[3] J. Biamonte, Universal variational quantum computation. *Phys. Rev. A* **103**, L030401 (2021)

[4] J. Biamonte and V. Bergholm, Tensor networks in a nutshell, preprint, arXiv:1708.00006 (2017)

[5] J. Biamonte, V. Bergholm and M. Lanzagorta, Tensor network methods for invariant theory. *J. Phys. A* **46**, 475301 (2013)

[1] There is indeed there a connection between the matrices A_{j_k} and completely positive trace-preserving (CPTP) maps from which the A_{j_k} derive their name. For the purpose of this paper, however, we will skip the detailed explanation.

[6] C. M. Bishop, *Pattern recognition and machine learning*. Information Science and Statistics, Springer, New York (2006)

[7] D. Bondarenko and P. Feldmann, Quantum autoencoders to denoise quantum data. *Phys. Rev. Lett.* **124**, 130502 (2020)

[8] O. Bousquet and A. Elisseeff, Stability and generalization. *J. Mach. Learn. Res.* **2**, 499–526 (2002)

[9] J. Carrasquilla, G. Torlai, R. G. Melko and L. Aolita, Reconstructing quantum states with generative models. *Nat. Mach. Intell.* **1**, 155–161 (2019)

[10] A. Cayley, On the theory of groups as depending on the symbolic equation $\theta^n = 1$. *Philos. Mag.* **7**, 40–47 (1854)

[11] A. Cichocki, Tensor networks for big data analytics and large-scale optimization problems, preprint, arXiv:1407.3124 (2014)

[12] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao and D. P. Mandic, Tensor networks for dimensionality reduction and large-scale optimization. I: Low-rank tensor decompositions. *Found. Trends Mach. Learn.* **9**, 249–429 (2016)

[13] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama and D. P. Mandic, Tensor networks for dimensionality reduction and large-scale optimization. II: Applications and future perspectives. *Found. Trends Mach. Learn.* **9**, 431–673 (2017)

[14] J. I. Cirac, D. Pérez-García, N. Schuch and F. Verstraete, Matrix product states and projected entangled pair states: Concepts, symmetries, theorems. *Rev. Modern Phys.* **93**, 045003 (2021)

[15] C. Cortes and V. Vapnik, Support-vector networks. *Mach. Learn.* **20**, 273–297 (1995)

[16] T. M. Cover, Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Trans. Electron. Comput.* **14**, 326–334 (1965)

[17] M. Fannes, B. Nachtergaele and R. F. Werner, Finitely correlated states on quantum spin chains. *Comm. Math. Phys.* **144**, 443–490 (1992)

[18] E. Farhi and H. Neven, Classification with quantum neural networks on near term processors, preprint, arXiv:1802.06002 (2018)

[19] C. Fernández-González, N. Schuch, M. M. Wolf, J. I. Cirac and D. Pérez-García, Frustration free gapless Hamiltonians for matrix product states. *Comm. Math. Phys.* **333**, 299–333 (2015)

[20] E. Grant, M. Benedetti, S. Cao, A. Hallam, J. Lockhart, V. Stojevic, A. G. Green and S. Severini, Hierarchical quantum classifiers. *npj Quantum Inf.* **4**, 1–8 (2018)

[21] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow and J. M. Gambetta, Supervised learning with quantum-enhanced feature spaces. *Nature* **567**, 209–212 (2019)

[22] A. Jahn and J. Eisert, Holographic tensor network models and quantum error correction: A topical review. *Quantum Sci. Technol.* **6**, 033002 (2021)

[23] A. Kardashin, A. Pervishko, D. Yudin, J. Biamonte et al., Quantum machine learning channel discrimination, preprint, arXiv: 2206.09933 (2022)

[24] Y. LeCun, Y. Bengio and G. Hinton, Deep learning. *Nature* **521**, 436–444 (2015)

[25] S. Lloyd, M. Schuld, A. Ijaz, J. Izaac and N. Killoran, Quantum embeddings for machine learning, preprint, arXiv:2001.03622 (2020)

[26] W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **5**, 115–133 (1943)

[27] K. Mitarai, M. Negoro, M. Kitagawa and K. Fujii, Quantum circuit learning. *Phys. Rev. A* **98**, 032309 (2018)

[28] A. Novikov, D. Podoprikhin, A. Osokin and D. P. Vetrov, Tensorizing neural networks. *Adv. Neural Inf. Process. Syst.* **28** (2015)

[29] A. Novikov, M. Trofimov and I. Oseledets, Exponential machines, preprint, arXiv:1605.03795 (2016)

[30] R. Orús, Advances on tensor network theory: Symmetries, fermions, entanglement, and holography. *Eur. Phys. J. B* **87**, 280 (2014)

[31] I. V. Oseledets, Tensor-train decomposition. *SIAM J. Sci. Comput.* **33**, 2295–2317 (2011)

[32] R. Penrose, Applications of negative dimensional tensors. In *Combinatorial Mathematics and its Applications (Proc. Conf., Oxford, 1969)*, Academic Press, London, 221–244 (1971)

[33] D. Perez-Garcia, F. Verstraete, M. M. Wolf and J. I. Cirac, Matrix product state representations. *Quantum Inf. Comput.* **7**, 401–430 (2007)

[34] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster and J. I. Latorre, Data re-uploading for a universal quantum classifier. *Quantum* **4**, 226 (2020)

[35] R. Rojas, The backpropagation algorithm. In *Neural Networks*, Springer, 149–182 (1996)

[36] J. Romero, J. P. Olson and A. Aspuru-Guzik, Quantum autoencoders for efficient compression of quantum data. *Quantum Sci. Technol.* **2**, 045001 (2017)

[37] L. Rosasco, E. De Vito, A. Caponnetto, M. Piana and A. Verri, Are loss functions all the same? *Neural Comput.* **16**, 1063–1076 (2004)

[38] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **65**, 386 (1958)

[39] B. Schölkopf, A. J. Smola, F. Bach et al., *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press (2002)

[40] M. Schuld, A. Bocharov, K. M. Svore and N. Wiebe, Circuit-centric quantum classifiers. *Phys. Rev. A* **101**, 032308 (2020)

[41] M. Schuld and N. Killoran, Quantum machine learning in feature Hilbert spaces. *Phys. Rev. Lett.* **122**, 040504 (2019)

[42] M. Schuld, R. Sweke and J. J. Meyer, Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Phys. Rev. A* **103**, 032430 (2021)

[43] H. Schulz and S. Behnke, Deep learning. *KI-Künstliche Intelligenz* **26**, 357–363 (2012)

[44] P. Selinger, A survey of graphical languages for monoidal categories. In *New Structures for Physics*, Lecture Notes in Phys. 813, Springer, Heidelberg, 289–355 (2011)

[45] J. Shawe-Taylor, N. Cristianini et al., *Kernel methods for pattern analysis*. Cambridge University Press, Cambridge (2004)

[46] E. Stoudenmire and D. J. Schwab, Supervised learning with tensor networks. *Adv. Neural Inf. Process. Syst.* **29** (2016)

[47] A. V. Uvarov, A. S. Kardashin and J. D. Biamonte, Machine learning phase transitions with a quantum processor. *Phys. Rev. A* **102**, 012415 (2020)

Richik Sengupta is a research scientist at Skolkovo Institute of Science and Technology.

r.sengupta@skoltech.ru

Soumik Adhikary is a research scientist at Skolkovo Institute of Science and Technology.

s.adhikari@skoltech.ru

Ivan Oseledets is full professor, director of the Center for Artificial Intelligence Technology, head of the Laboratory of Computational Intelligence at Skolkovo Institute of Science and Technology.

i.oseledets@skoltech.ru

Jacob Biamonte is full professor, head of the Laboratory of Quantum Algorithms for Machine Learning and Optimisation at Skolkovo Institute of Science and Technology.

j.biamonte@skoltech.ru