

# Circuit-centric quantum classifiers

Maria Schuld,<sup>1,2,3</sup> Alex Bocharov,<sup>3</sup> Krysta Svore,<sup>3</sup> and Nathan Wiebe<sup>3</sup>

<sup>1</sup>*Quantum Research Group, School of Chemistry and Physics,  
University of KwaZulu-Natal, Durban 4000, South Africa*

<sup>2</sup>*National Institute for Theoretical Physics, KwaZulu-Natal, Durban 4000, South Africa*

<sup>3</sup>*Quantum Architectures and Computation Group,  
Station Q, Microsoft Research, Redmond, WA (USA)*

(Dated: April 3, 2018)

The current generation of quantum computing technologies call for quantum algorithms that require a limited number of qubits and quantum gates, and which are robust against errors. A suitable design approach are variational circuits where the parameters of gates are learnt, an approach that is particularly fruitful for applications in machine learning. In this paper, we propose a low-depth variational quantum algorithm for supervised learning. The input feature vectors are encoded into the amplitudes of a quantum system, and a quantum circuit of parametrised single and two-qubit gates together with a single-qubit measurement is used to classify the inputs. This circuit architecture ensures that the number of learnable parameters is poly-logarithmic in the input dimension. We propose a quantum-classical training scheme where the analytical gradients of the model can be estimated by running several slightly adapted versions of the variational circuit. We show with simulations that the circuit-centric quantum classifier performs well on standard classical benchmark datasets while requiring dramatically fewer parameters than other methods. We also evaluate sensitivity of the classification to state preparation and parameter noise, introduce a quantum version of dropout regularisation and provide a graphical representation of quantum gates as highly symmetric linear layers of a neural network.

## I. INTRODUCTION

Quantum computing - information processing with devices that are based on the principles of quantum theory – is currently undergoing a transition from a purely academic discipline to an industrial technology. So called “non-fault-tolerant”, “small-scale” or “near-term” quantum devices are being developed on a variety of hardware platforms, and offer for the first time a testbed for quantum algorithms. However, allowing for only of the order of 1,000 – 10,000 elementary operations on 50 – 100 qubits [1] and without the costly feature of error correction, these early devices are not yet suitable to implement the algorithms that made quantum computing famous. A new generation of quantum routines that use only very limited resources and are robust against errors has therefore been created in recent years [2–4]. While many of those small-scale algorithms have the sole purpose of demonstrating the power of quantum computing over classical information processing [5], an important goal is to find quantum solutions to useful applications.

One increasingly popular candidate application for near-term quantum computing is machine learning [6]. Machine learning is data-driven decision making in which a computer fits a mathematical model to data (*training*) and uses the model to derive decisions (*inference*). Numerous quantum algorithms for machine learning have been proposed in the past years [7, 8]. A prominent strategy [9–12] is to encode data into the amplitudes of a quantum state (here referred to as *amplitude encoding*), and use quantum circuits to

manipulate these amplitudes. Quantum algorithms that are only polynomial in the number  $n$  of qubits can perform computations on  $2^n$  amplitudes. If these  $2^n$  amplitudes are used to encode the data, one can therefore process data inputs in polylogarithmic time. However, most of the existing literature on amplitude encoded quantum machine learning translates known machine learning models into non-trivial quantum subroutines that lead to resource-intensive algorithms which cannot be implemented on small-scale devices. Furthermore, quantum versions of training algorithms are limited to specific, mostly convex optimisation problems. Hybrid approaches called “variational algorithms” [2, 13, 14] are much more suited to near term quantum computing and are rapidly getting popularity in the quantum research community in recent months. A general picture of variational circuits for machine learning is introduced in [15]. The emphasis of low-depth circuits for quantum machine learning has been made in [16], where the importance of entanglement as a resource has been analysed for the low-depth architectures in the context of Boltzmann machines. A very recent preprint that comes closest to the designs presented here is Farhi and Neven [17]. The latter focusses mostly on classification of discrete and discretized data that is encoded into qubits rather than amplitudes, which requires an exponentially larger number of qubits for a given input dimension. The circuit architectures proposed in the work are of more general nature compared to our focus on a slim parameter count through the systematic use of entanglement.

This paper presents a quantum framework for supervised

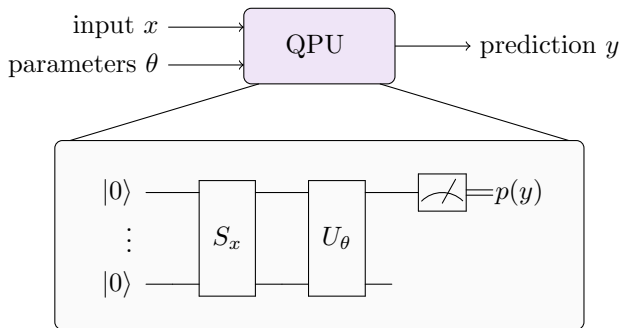


FIG. 1. Idea of the circuit-centric quantum classifier. Inference with the model  $f(x, \theta) = y$  is executed by a quantum device (the quantum processing unit or QPU) which consists of a *state preparation circuit*  $S_x$  encoding the input  $x$  into the amplitudes of a quantum system, a *model circuit*  $U_\theta$ , and a single qubit measurement. The measurement retrieves the probability of the model predicting 0 or 1, from which in turn the binary prediction can be inferred. The classification circuit parameters  $\theta$  are learnable and can be trained by a variational scheme.

learning that makes use of the advantages of amplitude encoding, but is based on a variational approach and therefore particularly designed for small-scale quantum devices (see Fig. 1). To achieve low algorithmic depth we propose a *circuit-centric* design which understands a generic strongly entangling quantum circuit  $U_\theta$  as the core of the machine learning model  $f(x; \theta) = y$ , where  $x$  is an input,  $\theta$  a set of parameters and  $y$  is the prediction or output of the model. We call this circuit the *model circuit*. The model circuit consists of parametrised single and controlled single qubit gates, with learnable (classical) parameters. The number of parametrised gates in the family of model circuits we propose grows only polynomially with the number of qubits, which means that our quantum machine learning algorithm has a number of parameters that is overall poly-logarithmic in the input dimension.

The model circuit acts on a quantum state that represents the input  $x$  via amplitude encoding. To prepare such a quantum state, a static *state preparation circuit*  $S_x$  has to be applied to the initial ground state. After applying the state preparation as well as the model circuit, the prediction is retrieved from the measurement of a single qubit. If the data is sufficiently low-dimensional or its structure allows for efficient approximate preparation, this yields a compact circuit that can be understood as a black box routine that executes the inference step of the machine learning algorithm on a small-scale quantum computer.

We propose a hybrid quantum-classical gradient descent training algorithm. On the analytical side we show how the exact gradients of the circuit can be retrieved from running slight variations of the inference algorithm (and for now assuming perfect precision in the prediction)

a small, constant number of times and adding up the results, a strategy we call *classical linear combination of unitaries*. The parameter updates are then calculated on a classical computer. Keeping the model parameters as a classical quantity allows us not only to implement a large number of iterations without worrying about growing coherence times, but also to store and reuse learnt parameters at will. Using single-batch gradient descent only requires the state preparation circuit  $S_x$  to encode one input at a time. In addition to that, we can easily improve the gradient descent scheme by standard methods such as an adaptive learning rate, regularisation and momenta.

We analyse the resulting *circuit-centric quantum classifier* theoretically as well as via simulations to judge its performance compared to other models. We show that mathematically, a quantum circuit closely resembles a neural network architecture with unitary layers, and discuss ways to include dropout and nonlinearities. The unitarity of the “pseudo-layers” is a favourable property from a machine learning point of view [19, 20], since it maintains the length of an input vector throughout the layers and therefore circumvents notorious problems of vanishing or exploding gradients. Unitary weight matrices have also been shown to make the convergence time of gradient descent independent of the circuit depth [21] - an important guarantee to avoid the growing complexity of training deep architectures. Possibly the most important feature of the model is that it uses a number of parameters that is logarithmic in the data size, which is a huge saving to a neural network where the first layer already has weights at least linear in the dimension of the input.

In the remainder of the paper we will introduce the circuit-centric quantum classifier in Section II, along with design considerations for the circuit architecture in Section III, as well as the training scheme in Section IV. We analyse its performance in Section V and show that compared with out-of-the-box methods it performs reasonable well. We finally propose a number of ways to extend the work in Section VI.

## II. THE CIRCUIT-CENTRIC QUANTUM CLASSIFIER

The task our model intends to solve is that of *supervised pattern recognition*, and is a standard problem in machine learning with applications in image recognition, fraud detection, medical diagnosis and many other areas. To formalise the problem, let  $\mathcal{X}$  be a set of inputs and  $\mathcal{Y}$  a set of outputs. Given a dataset  $\mathcal{D} = \{(x^1, y^1), \dots, (x^M, y^M)\}$  of pairs of so called *training inputs*  $x^m \in \mathcal{X}$  and *target outputs*  $y^m \in \mathcal{Y}$  for  $m = 1, \dots, M$ , our goal is to predict the output  $y \in \mathcal{Y}$  of a new input  $x \in \mathcal{X}$ . For simplicity we will assume in

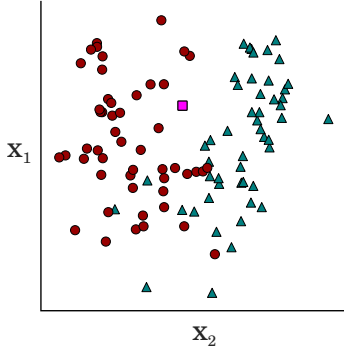


FIG. 2. Supervised binary classification for 2-dimensional inputs. Given the red circle and blue triangle data points belonging to two different classes, guess the class of the new input (pink square).

the following that  $\mathcal{X} = \mathbb{R}^N$  and  $\mathcal{Y} = \{0, 1\}$ , which is a binary classification task on a  $N$ -dimensional real input space (see Fig. 2). Most machine learning algorithms solve this task in two steps: They first *train* a model  $f(x, \theta)$  with the data by adjusting a set of parameters  $\theta$ , and then use the trained model to *infer* the prediction  $y$ .

The main idea of the circuit-centric design is to turn a generic quantum circuit of single and 2-qubit quantum gates into a model for classification. One can divide the full inference algorithm into four steps. As shown in Fig. 3, these four steps can be described using the language of quantum circuits, but also as a formal mathematical model, and finally, using the idea of graphical representation for neural networks, as a graphical model.

From a quantum circuit point of view we use the state preparation circuit  $S_x$  to encode the data into the state of a  $n$  qubit quantum system, which effectively maps an input  $x \in \mathbb{R}^N$  to the  $2^n$ -dimensional amplitude vector  $\varphi(x)$  that describes the initial quantum state  $|\varphi(x)\rangle$ . Second, the model circuit  $U_\theta$  is applied to the quantum state. Third, the prediction is read out from the final state  $|\varphi'\rangle = U_\theta|\varphi(x)\rangle$ . For this purpose we measure the first of the  $n$  qubits. Repeated applications of the overall circuit and measurements resolve the probability of measuring the qubit in state 1. Lastly, the result is postprocessed by adding a learnable bias parameter  $b$  and mapping the result through a step function to the output  $y \in \{0, 1\}$ .

From a purely mathematical point of view, this procedure (that is, if we could perfectly resolve the probability of the first qubit by measurements) formally defines a classifier

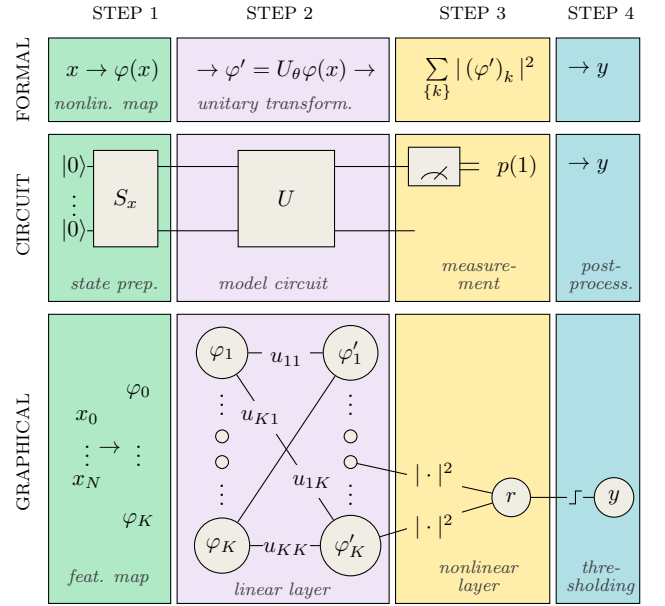


FIG. 3. Inference with the circuit-centric quantum classifier consists of four steps, here displayed in four colours, and can be viewed from three different perspectives, i.e. from a formal mathematical framework, a quantum circuit framework and a graphical neural network framework. In the first step, the feature map from the input space to the feature space  $\mathbb{R}^N \rightarrow \mathbb{R}^K$  is executed for an input by a state preparation scheme. The quantum circuit applies a unitary transformation to the feature vector which can be understood as one linear layer (or, when decomposed into gates, several linear layers) of a neural network. The measurement statistics of the first qubit are interpreted as the continuous output of the classifier and effectively implement a weightless nonlinear layer in which every component of the last half of all units is mapped by an absolute square and summed up. The postprocessing stage binarises the result with a thresholding function via classical computing.

that takes decisions according to

$$f(x; \theta, b) = \begin{cases} 1 & \text{if } \sum_{k=2^{n-1}+1}^{2^n} |(U_\theta \varphi(x))_k|^2 + b > 0.5, \\ 0 & \text{else.} \end{cases} \quad (1)$$

Here  $\varphi : \mathbb{R}^N \rightarrow \mathbb{C}^{2^n}$  is a map that describes the procedure of information encoding via the state preparation routine ( $n$  is an integer such that  $2^n \geq N$ ),  $U_\theta$  is the parametrised unitary matrix describing the model circuit, and  $(U_\theta \varphi(x))_k$  is the  $k$ th entry of the result after we applied this matrix to  $\varphi(x)$ . The sum over the second half of the resulting vector corresponds to the single qubit measurement resulting in state 1. Postprocessing adds the bias  $b$  and thresholds to compute a binary prediction.

Lastly, if we formulate the four steps in the language of neural networks and their graphical representation, state preparation corresponds to a feature map on the input

space, while the unitary circuit resembles a neural network of several parametrised linear layers. This is followed by two nonlinear layers, one simulating the read-out via measurement (adding the squares of some units from the previous layer) and one that maps the output to the final binary decision. We will go through the four different steps in more detail and discuss our specific design decisions for the model.

### A. State preparation

There are various strategies to encode input vectors into the  $n$ -qubit system of a quantum computer. In the most general terms, state preparation implements a feature map  $\varphi : \mathbb{R}^N \rightarrow \mathbb{C}^{2^n}$  where  $n$  is the total number of qubits used to represent the features. In the following we focus on *amplitude encoding*, where an input vector  $x \in \mathbb{R}^N$  – possibly with some further preprocessing to bring it into a suitable form – is directly associated with the amplitudes of the  $2^n$ -dimensional ‘ket’ vector of the quantum system written in the computational basis. **This option can be extended by preparing a set of copies of the initial quantum state, which effectively implements a tensor product of copies of the input, mapping it to much higher dimensional spaces.**

To directly associate an amplitude vector in computational basis with a data input, we require that  $N$  is a power of 2 (so that we can use all  $2^n$  amplitudes of a  $n$ -qubit system), and that the input is normalised to unit length,  $x^T x = 1$ . If  $N$  is no power of 2, we can ‘pad’ the original input with a suitable number of zero features. (For example  $x = (x_1, x_2, x_3)^T$  would be extended to  $x' = (x_1, x_2, x_3, 0)^T$ ). Normalisation can pose a bigger challenge. Although many datasets carry proximity relations between vectors in their angles and not their length, some data sets can become significantly distorted by normalisation. A possible solution is to embed the data in a higher dimensional space. Practically, this can be achieved by adding non-zero padding terms before normalization. Let  $N$  be the dimensionality of the original feature space, and let  $c_1, \dots, c_D$  be the padding terms that may in general depend on the informative features  $x_1$  to  $x_N$ . The preprocessing necessary for amplitude encoding maps

$$(x_1, \dots, x_N)^T \rightarrow \chi (x_1, \dots, x_N, c_1, \dots, c_D)^T, \quad (2)$$

with

$$\chi = \frac{1}{\sqrt{\sum_j x_j^2 + \sum_k |c_k|^2}},$$

on the original data.

For the designs investigated in this paper it is convenient to choose the padding width  $D$  such that  $N' = N + D$

is some exact power of 2, and to choose  $\{c_1, \dots, c_D\}$  as a set of non-informative constants. This choice has in fact two desirable side-effects of feature normalisation (2): first, it creates an ‘ancillary’ space of dimension  $D$ , which in the language of neural networks is analogous to having more “nodes” in the first hidden layer than in the input layer; second, **the constants create a state vector that is not homogeneous with respect to the vector of the original features** (the importance of this will appear shortly in the discussion of the tensorial maps).


Preparing a quantum state whose amplitude vector in the computational basis is equivalent to the pre-processed input  $\varphi(x)$  can always be done with a circuit that is linear in the number of features in the input vector, for example with the routines in Refs [22–24]. When more structure in the data can be exploited, preparation routines with polylogarithmic dependence on the number of features might be applicable [25, 26]. **A largely uninvestigated option is also approximate state preparation of feature vectors, which may reduce the resources needed for the circuit  $S_x$  at the expense of an error in the inputs.**

To map input data into vastly higher dimensional spaces we can apply a *tensorial feature map* by preparing  $d$  copies of the state [27]. If  $|\psi\rangle$  is the ‘ket’ vector produced by amplitude encoding, this prepares

$$|\psi\rangle \rightarrow \underbrace{|\psi\rangle \otimes \dots \otimes |\psi\rangle}_{d \text{ times}}.$$

For amplitude encoding with  $N = 2$  and  $d = 2$ , and without any of the preprocessing described above, this would map a feature vector  $(x_1, x_2)^T$  to

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \otimes \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 \\ x_1 x_2 \\ x_2 x_1 \\ x_2^2 \end{pmatrix},$$

**and can give rise to interesting nonlinearities that may facilitate the classification procedure in the following steps** (see also [28]). 

### B. The model circuit

Given an encoded feature vector  $\varphi(x)$  which is now a ‘ket’ vector in the Hilbert space of a  $n$  qubit system, the model circuit maps this ket vector to another ket vector  $\varphi' = U_\theta \varphi(x)$  by a unitary operation  $U_\theta$  which is parametrised by a set of variables  $\theta$ .

### 1. Decomposition into (controlled) single qubit gates

As described before, we decompose  $U$  into

$$U = U_L \dots U_\ell \dots U_1, \quad (3)$$

where each  $U_\ell$  is either a single qubit or a two-qubit quantum gate. As a reminder, a single qubit gate  $G_k$  acting on the  $k$ th of  $n$  qubits can be expressed as

$$U_l = \mathbb{I}_0 \otimes \dots \otimes G_k \otimes \dots \otimes \mathbb{I}_{n-1}. \quad (4)$$

If the circuit depth  $L$  is in  $\Omega(4^n)$ , this decomposition allows us to represent general unitary transformations. Remember that unitary operators are linear transformations that preserve the length of a vector, a fact that holds a number of advantages for the classifier as we will discuss later.

We further restrict the type of 2-qubit gate to simplify our “elementary parametrised gate set”. A 2-qubit unitary gate is called *imprimitive* if it can map a 2-qubit product state into a non-product state. A common case of an imprimitive two-qubit gate is a singly-controlled single-qubit gate  $C(G)$  that in standard computational basis can be written as

$$C_a(G_b) |x\rangle|y\rangle = |x\rangle \otimes G^x|y\rangle, \quad (5)$$

where  $G$  is a single-qubit gate other than a global phase factor on the qubit  $b$  and the state  $x$  of qubit  $a$  is either 0 or 1 ( $G^0$  is the identity). For example,  $G$  could be a NOT gate, in which case the  $C(G)$  is simply the frequently used CNOT gate. It is known ([29]), that single-qubit gates together with any set of imprimitive 2-qubit gates provide for quantum universality:

**Observation 1.** *Circuits of the form (3) composed out of single-qubit gates and at least one type of imprimitive 2-qubit gates generate the entire unitary group  $U(2^n)$  in a topological sense. That is, for any  $\varepsilon > 0$  and any unitary  $V \in U(2^n)$  there is a circuit of the the form (3) the value of which is  $\varepsilon$ -close to  $V$ .*

To make the single qubit gates trainable we need to formulate them in terms of parameters that can be learnt. The way the parametrisation is defined can have a significant impact on training, since it defines the shape of the cost function. A single qubit gate  $G$  is a  $2 \times 2$  unitary, which can always be written [30] as

$$G(\alpha, \beta, \gamma, \phi) = e^{i\phi} \begin{pmatrix} e^{i\beta} \cos \alpha & e^{i\gamma} \sin \alpha \\ -e^{-i\gamma} \sin \alpha & e^{-i\beta} \cos \alpha \end{pmatrix} \quad (6)$$

and is fully defined by four parameters  $\{\alpha, \beta, \gamma, \phi\}$ . For quantum gates – where we cannot physically measure overall phase factors – we may neglect the prefactor  $e^{i\phi}$  and only consider three learnable parameters per gate. The advantage in using angles (instead of, for example, a parametrisation with Pauli matrices) is that

training does not need an additional condition on the model parameters. A disadvantage might unfavourable convergence properties of trigonometric functions close to their optima.

Note that there may be much more efficient “elementary parametrised gatesets” for a specific hardware, since some single qubit gates might naturally be parametrised in the device (i.e. where the parameter corresponds to the intensity of a laser pulse). For the agnostic case we consider here, every parametrised gate has to be decomposed into the constant elementary gate set of the physical device, which adds an efficient overhead per gate that depends on the fidelity with which we seek to approximate it (see Section IV E).

We treat the circuit *architecture*, i.e. which qubit a certain gate acts on and where to place the controls, as fixed here and will discuss design choices in Section III. Of course, strategies to learn the circuit architecture are also worth investigating, but we expect this to be a non-trivial problem due to the vast impact that each gate choice in the architecture bears for the final state (see [31] and the discussion on “quantum chaos” in random circuits).

### C. Read out and postprocessing

After executing the quantum circuit  $U_\theta \varphi(x)$  in Step 2, the measurement of the first qubit (Step 3) results in state 1 with probability[32]

$$p(q_0 = 1, x; \theta) = \sum_{k=2^{n-1}+1}^{2^n} |(U_\theta \varphi(x))_k|^2.$$

To resolve these statistics we have to run the entire circuit  $S$  times and measure the first qubit. We estimate  $p(q_0 = 1)$  from these samples  $s_1, \dots, s_S$ . This is a Bernoulli parameter estimation problem which we discuss in Section IV E.

The classical postprocessing (Step 4) consists of adding a learnable bias term  $b$  to produce the continuous output of the model,

$$\pi(x; \theta, b) = p(q_0 = 1, x, \theta) + b. \quad (7)$$

Thresholding the value finally yields the binary output that is the overall prediction of the model:

$$f(x; \theta) = \begin{cases} 1 & \text{if } \pi(x; \theta) > 0.5 \\ 0 & \text{else} \end{cases}.$$

In Dirac notation the measurement result can be written as the expectation value of a  $\sigma_z$  operator acting on the first qubit, measured after applying  $U$  to the initial state



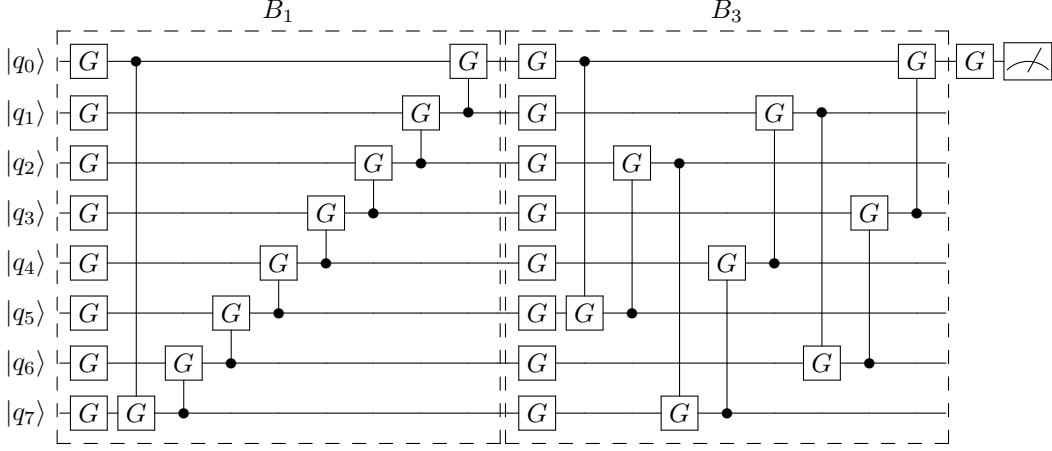


FIG. 4. Generic model circuit architecture for 8 qubits. The circuit consists of two ‘code blocks’  $B_1$  and  $B_3$  with a range of controls of  $r = 1$  and  $r = 3$  respectively. The circuit consists of 17 trainable single-qubit gates  $G = G(\alpha, \beta, \gamma)$ , as well as 16 trainable controlled single qubit gates  $C(G)$ , which have in turn to be decomposed into the elementary constant gate set used by the quantum computer on which to implement it. If the optimisation methods are used to reduce the controlled gates to a single parameter, we have  $3 \cdot 33 + 1 = 100$  parameters to learn in total for this model circuit. These 100 parameters are used to classify inputs of  $2^8 = 256$  dimensions, which shows that the circuit-centric classifier is a much more compact model than a conventional feed-forward neural network.

$|\varphi(x)\rangle$ . In absence of non-linear activation, the expectation value of the  $\sigma_z$  operator on the subspace of the first qubit is given by

$$\mathbb{E}(\sigma_z) = \langle \varphi(x) | U^\dagger (\sigma_z \otimes \mathbb{I} \otimes \dots \otimes \mathbb{I}) U | \varphi(x) \rangle,$$

and we can retrieve the continuous output via

$$\pi(x; \theta) = \left( \frac{\mathbb{E}(\sigma_z)}{2} + \frac{1}{2} \right) + b. \quad (8)$$

### III. CIRCUIT ARCHITECTURES

Our initial goal was to build a classifier that at its core has a low-depth quantum circuit. With the circuit decomposed into  $L$  single or controlled single qubit gates, we therefore want to constrain  $L$  to be polynomial in  $n$  which will allow us to do inference with a number of elementary quantum operations that grows only polylogarithmically in the dimension of the data set. However, this obviously comes at a price. The vectors of the form  $U_\theta|0\dots 0\rangle$  exhaust only a small subset of the Hilbert space of  $n$  qubits. In other words, the set of amplitude vectors  $\varphi' = U_\theta\varphi(x)$  that the circuit can ‘reach’ is limited. In machine learning terms, this limits the flexibility of the classifier. **Much like in classical machine learning, the challenge of finding a generic circuit architecture is therefore to engineer circuits (3) of polynomial depth that still create powerful classifiers for a subclass of datasets.**

#### A. Strongly entangling circuits

A natural approach to the problem of circuit design is to consider circuits that prepare strongly entangled quantum states. For one, such circuits can reach ‘wide corners of the Hilbert space’ with  $U_\theta|0, \dots, 0\rangle$ . Reversibly argued, they have a better chance to project input data state  $|\varphi(x)\rangle$  with the class label  $y$  onto the subspace  $|y\rangle \otimes |\eta\rangle$ ,  $\eta \in \mathbb{C}^{2^{n-1}}$ , which corresponds to a decision of  $p(q_0) = 0, 1$  in our classifier (for a zero bias). **Moreover, from a theoretical point of view a classifier has to capture both short and long-range correlations in the input data, and there is mounting evidence [19, 33] that shallow circuits may be suitable for the purpose when they are strongly entangling.**

More specifically, we compose the circuit (3) out of several *code blocks*  $B$  (see dotted boxes in the example in Figure 4). A code block consists of a layer of single qubit gates  $G = G(\alpha, \beta, \gamma)$  applied to each of the  $n$  qubits, followed by a layer of  $n/\text{gcd}(n, r)$  controlled gates, where  $r$  is the ‘range’ of the control and  $\text{gcd}(n, r)$  is the greatest common denominator of  $n$  and  $r$ . For  $j \in [1..n/\text{gcd}(n, r)]$  the  $j$ th 2-qubit gate  $C_{c_j}(G_{t_j})$  of a block has qubit number  $t_j = (jr - r) \bmod n$  as the target, qubit number  $c_j = jr \bmod n$  as control. A full block has the following composition,

$$B = \prod_{k=0}^{n-1} C_{c_k}(G_{t_k}) \prod_{j=0}^{n-1} G_j. \quad (9)$$

We observe that such code block is capable of entangling/unentangling all the qubits with numbers that

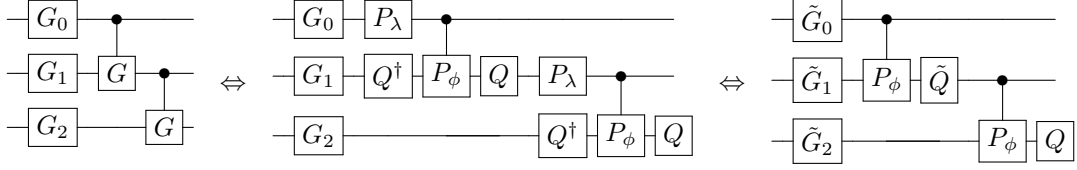


FIG. 5. Illustration of first step of the proof from Observation 2 for an example of the first 5 gates of a codeblock of 3 qubits with range  $r = 1$ . Decomposing the controlled rotations and merging single qubit gates reduces the number of parameters needed to represent the model circuit architecture. For simplification the gates are displayed without indices or parameters.

are a multiple of  $\gcd(n, r)$ . In particular, assuming  $r$  is relatively prime with  $n$ , all  $n$  qubits can be entangled/unentangled.

As an example that demonstrates the entangling power of the circuit, select a block with  $n = 4$ ,  $r = 1$ . Let all controlled gates be CNOTs and let all single qubit gates be identities, except from  $G_0 = G_2 = H$ , which are Hadamard gates. Applying the circuit to the basis product state  $|0000\rangle$  we get the state

$$|\psi\rangle = \frac{1}{2}(|00\rangle|00\rangle + |01\rangle|11\rangle + |10\rangle|01\rangle + |11\rangle|10\rangle).$$

If  $A$  is the subsystem consisting of qubits 0,1 and  $B$  the subsystem of qubits 2,3, then the marginal density matrix, corresponding to the state  $|\psi\rangle$  and the partitioning  $A \otimes B$ , is completely mixed. Therefore the state  $|\psi\rangle$  strongly entangles the two subsystems.

### B. Optimising the architecture

The definition of the code block as per Equation (9) is fairly redundant. It turns out that the parameter space of the circuit (9) can for practical purposes be reduced to roughly  $5n$  parameters. For this we need to introduce a controlled phase gate  $C_j(P_k(\phi))$ ,  $\phi \in \mathbb{R}$  that applies the phase shift  $e^{i\phi}$  to a standard basis vector if and only if both the  $j$ -th and  $k$ -th qubits are in state  $|1\rangle$ . (Note the symmetry of the definition, which means that it does not matter which of the qubits is the control and which is the target.)

**Observation 2.** A circuit block of the form (9) can, up to global phase, be uniquely rewritten as

$$B = \prod_{k=0}^{n-1} R_k^X C_{ck}(P_{t_k}) \prod_{j=0}^{n-1} G_j. \quad (10)$$

where  $\forall j, G_j \in SU(2)$  are single qubit gates with the usual three parameters (and, moreover,  $G_j$  is an axial rotation for  $j > 0$ ),  $P$  is a single-parameter phase gate, and  $R^X$  is a single-parameter  $X$ -rotation.

*Proof.* The proof is based on transformations of the  $C_a(G_b)$  gates and subsequent normalizations of the

single-qubit unitaries. Let us diagonalise the single-qubit unitary  $G = Q D Q^\dagger$ , where  $Q$  is some other single-qubit unitary and  $D = \text{diag}(e^{i\lambda}, e^{i(\lambda+\phi)})$  with  $\lambda, \phi \in \mathbb{R}$  is the diagonal matrix of the eigenvalues. Then  $C_a(G_b) = Q_b C_a(P_b(\phi)) Q_b^\dagger P_a(\lambda)$  (see Figure 5). We further merge the  $Q_b^\dagger$  with the corresponding  $G_b$  of the code block (9). In case  $a = 0$  the  $P_a(\lambda)$  can be commuted to the beginning of the layer and merged with  $G_0$ . In case  $a \neq 0$  the  $P_a(\lambda)$  can be commuted through the end of the layer and either merged into the next layer or, if we are looking at the last layer in the circuit, traced out. At this point the action of the circuit (9) is equivalent to that of  $(\prod_{k=0}^{n-1} \tilde{Q}_{t_k} C_{ck}(P_{t_k})) \prod_{j=0}^{n-1} \tilde{G}_j$  where  $\tilde{G}_j$  and  $\tilde{Q}_{t_k}$  are updated single qubit gates from the merging operation. Note that the single qubit gates in the following layer are also updated.

In the second round of the we split all the single-qubit gates  $\tilde{Q}$  up to global phase into product of three rotations  $\tilde{Q} \sim R_Z(\mu_1) R_X(\mu_2) R_Z(\mu_3)$ . We conclude the proof by noting that each of the diagonal operators  $R_Z(\mu_1)_{t_j}, R_Z(\mu_3)_{t_j}$  can be commuted through all controlled phase gates to either the end of the layer or to the beginning (in which case it can be merged with one of the  $\tilde{G}_j$  gates).  $\square$

To summarize, with the possible exception of the last layer in the classifier, a layer is described (up to a global phase) by at most  $5n$  parameters, at most  $n$  for all controlled phase gates  $C(P)$ , at most  $n$  for all  $x$ -rotations  $R^X$  and at most  $3n$  for all fully parametrised single qubit gates  $G$ .

### C. Graphical representation of gates

As a product of elementary gates, the model circuit  $U_x$  can be understood as a sequence of linear layers of a neural network with the same number of units in each “hidden layer”. This perspective facilitates the comparison of the circuit-centric quantum classifier with widely studied neural network models, and visualises the connectivity power of (controlled) single qubit gates. The position of the qubit (as well as the control) determine the architecture of each layer, i.e. which units are

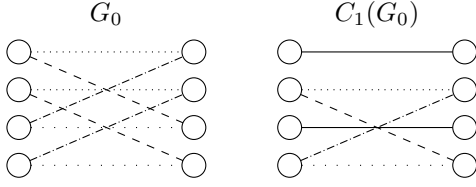


FIG. 6. Graphical representation of quantum gates. Left: A single qubit gate applied to the first qubit of a 2-qubit register. Right: A single qubit gate and a controlled single qubit gate applied to a two-qubit register. A solid line corresponds to a unit weight, while the other lines stand for a variable weight parameter. The same line styles indicate the same weights.

connected and which “weights” are tied in a “gate-layer”.

To show an example, consider a Hilbert space of dimension  $2^n$  with  $n = 2$  qubits  $|q_0 q_1\rangle$ . A single qubit unitary  $G$  applied to  $q_0$  would have the following matrix representation

$$G_0 = \begin{pmatrix} e^{i\beta} \cos \alpha & 0 & e^{i\gamma} \sin \alpha & 0 \\ 0 & e^{i\beta} \cos \alpha & 0 & e^{i\gamma} \sin \alpha \\ -e^{-i\gamma} \sin \alpha & 0 & e^{-i\beta} \cos \alpha & 0 \\ 0 & -e^{-i\gamma} \sin \alpha & 0 & e^{-i\beta} \cos \alpha \end{pmatrix},$$

while the same unitary but controlled by qubit  $q_1$ ,  $C_1(G_0)$  has matrix representation

$$C_1(G_0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\beta} \cos \alpha & 0 & e^{i\gamma} \sin \alpha \\ 0 & 0 & 1 & 0 \\ 0 & -e^{-i\gamma} \sin \alpha & 0 & e^{-i\beta} \cos \alpha \end{pmatrix}$$

At the same time, these two gates can be understood as layers with connections displayed in Figure 6.

It becomes obvious that a single qubit gate connects two sets of two variables with the same weights, in other words, it ties the parameters of these connections. The control removes half of the ties and replaces them with identities. A quantum circuit can therefore be understood as an analog of a neural network architecture with highly symmetric, unitary linear layers, and controls break some of the symmetry. Note that although we speak of linear layers here, the weights (i.e., the entries of the weight matrix representing a gate) have a nonlinear dependency on the model parameters  $\theta$ , a circumstance that plays a role for the convergence of the hybrid training method.

#### IV. TRAINING

We consider a stochastic gradient descent method for training. The parameters that define every single qubit gate of the quantum circuit are at every stage of the quan-

tum algorithm classical values. However, we are computing the model function on a quantum device, and have therefore no ‘classical’ access to its gradients. This means that the training procedure has to be a hybrid scheme that combines classical processing to update the parameters, and quantum information processing to extract the gradients. We will show how to use the quantum circuit to extract estimates of the analytical gradients, as opposed to other proposals for variational algorithms based on derivative-free or finite-difference gradients (see [34]). A related approach, but for a different gate representation, has been proposed during the time of writing in Ref. [17].

##### A. Cost function

We choose a standard least-squares objective to evaluate the cost of a parameter configuration  $\theta$  and a bias  $b$  given a training set,  $\mathcal{D} = \{(x^1, y^1), \dots, (x^M, y^M)\}$ ,

$$C(\theta, b; \mathcal{D}) = \frac{1}{2} \sum_{m=1}^M |\pi(x^m; \theta, b) - y^m|^2,$$

where  $\pi$  is the continuous output of the model defined in Equation (7). Note that we can easily add a regularisation term (i.e., an  $L_1$  or  $L_2$  regulariser) to this objective, since it does not require any additional quantum information processing. For the sake of simplicity we do not consider regularisation in this paper.

Gradient descent updates each parameter  $\mu$  from the set of circuit parameters  $\theta$  via

$$\mu^{(t)} = \mu^{(t-1)} - \eta \frac{\partial C(\theta, b; \mathcal{D})}{\partial \theta},$$

and similarly for the bias,

$$b^{(t)} = b^{(t-1)} - \eta \frac{\partial C(\theta, b; \mathcal{D})}{\partial b}.$$

The learning rate  $\eta$  can be adapted during training and we can also add momenta to the updates, which can significantly decrease the convergence time.

In *stochastic* gradient descent, we do not consider the entire training set  $\mathcal{D}$  in every iteration, but only a subset or *batch*  $\mathcal{B} \subset \mathcal{D}$  [35]. The derivatives in the parameter updates are therefore taken with respect to  $C(\theta, b; \mathcal{B})$  instead of  $C(\theta, b; \mathcal{D})$ . In principle, quantum computing allows us to encode a batch of  $B$  training inputs into a quantum state in superposition and feed it into the classifier, which can be used to extract gradients for the updates from the quantum device. However, guided by the design principle of a low-depth circuit, this would extend the state preparation routine to be in  $\mathcal{O}(BN)$  for general cases, where  $N$  is the size of each input in the batch (which becomes even worse for more sophisticated



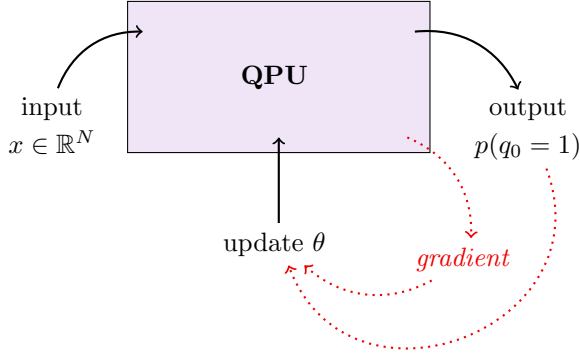


FIG. 7. Idea of the hybrid training method. The quantum processing unit (QPU) is used to compute outputs and gradients of the model in order to update the parameters for each step of the gradient descent training.

feature maps in Step 1). We therefore consider single-batch gradient descent here (i.e.,  $B = 1$ ), where only one randomly sampled training input is considered in each iteration. Single-batch stochastic gradient descent can have favourable convergence properties, for example in cases where there is a lot of data available [36].

### B. Hybrid gradient descent scheme

The derivative of the objective function with respect to a model parameter  $\nu = b, \mu$  (where  $\mu \in \theta$  is a circuit parameter) for a single data sample  $\{(x^m, y^m)\}$  is calculated as

$$\frac{\partial C}{\partial \nu} = (\pi(x^m; \nu) - y^m) \partial_\nu \pi(x^m; \nu).$$

Note that  $\pi(x^m; \nu)$  is a real-valued function and the  $y^m$  and the parameters are also real-valued. Hence  $\frac{\partial C}{\partial \nu} \in \mathbb{R}$ .

While  $\pi(x^m; \nu)$  is a simple prediction we can get from the quantum device, and  $y^m$  is a target from the classical training set, we have to look closer at how to compute the gradient  $\partial_\nu \pi$ . For  $\nu = b$  this is in fact trivial, since

$$\partial_b \pi(x^m; b) = 1.$$

In case of  $\nu = \mu$ , the gradient forces us to compute derivatives of the unitary operator. In the following we will calculate the gradients in vector as well as in Dirac notation and show how a trick allows us to estimate these gradients using a slight variation of the model circuit  $S_x$ .

The derivative of the continuous output of the model with

respect to the circuit parameter  $\mu$  is formally given by

$$\begin{aligned} \partial_\mu \pi(x^m; \mu) &= \partial_\mu p(q_0 = 1; x^m, \theta) \\ &= \partial_\mu \sum_{k=2^{n-1}+1}^{2^n} (U_\theta \varphi(x))_k^\dagger (U_\theta \varphi(x))_k \\ &= 2\text{Re} \left\{ \sum_{k=2^{n-1}+1}^{2^n} (\partial_\mu U_\theta \varphi(x))_k^\dagger (U_\theta \varphi(x))_k \right\}. \end{aligned}$$

The last expression contains the ‘derivative of the circuit’,  $\partial_\mu U_\theta$ , which is given by

$$\partial_\mu U_\theta = U_L \dots (\partial_\mu U_i) \dots U_1,$$

where we assume for simplicity that only the parametrised gate  $U_i$  depends on parameter  $\mu$ . If the parameters of different unitary matrices are tied then the derivative can simply be found by applying the product rule.

In Dirac notation, we have expressed the probability of measuring the first qubit in state 1 through the expectation value of a  $\sigma_z$  operator acting on the same qubit,  $p(q_0 = 1; x^m, \theta) = \frac{1}{2}(\langle U_\theta \varphi(x) | \sigma_z | U_\theta \varphi(x) \rangle + 1)$  (see Equation 8). We can use this expression to write the gradient in Dirac notation,

$$\partial_\mu \pi(x^m; \theta, b) = \text{Re}\{(\partial_\mu U_\theta \varphi(x^m) | \sigma_z | U_\theta \varphi(x^m))\}. \quad (11)$$

This notation reveals the challenge in computing the gradients using the quantum device. The gradient of a unitary is not necessarily a unitary, which means that  $|\partial_\mu U_\theta \varphi(x^m)\rangle$  is not a quantum state that can arise from a quantum evolution. How can we still estimate gradients using the quantum device?

### C. Classical linear combinations of unitaries

It turns out that in our architecture we can always represent  $\partial_\mu U_\theta$  as a linear combination of unitaries. Linear combination of unitaries is a known technique in quantum mechanics [37], where the sum is implemented in a coherent fashion. In our case where we allow for classical postprocessing, we do not have to apply unitaries in superposition, but can simply run the quantum circuit several times and collect the output. This is what we will call *classical linear combinations of unitaries* here.

Consider the derivative of  $U_i$  for the single-qubit gate defined in Equation (4),

$$\partial_\mu U_i = \mathbb{I} \otimes \dots \otimes \partial_\mu G(\alpha, \beta, \gamma) \otimes \dots \otimes \mathbb{I},$$

where  $G(\alpha, \beta, \gamma)$  is given in the parametrisation introduced in Equation (6) and discounting the global phase. The derivatives of the single qubit gate  $G(\alpha, \beta, \gamma)$  with

respect to the parameters  $\mu = \alpha, \beta, \gamma$  are as follows:

$$\partial_\alpha G = G(\alpha + \frac{\pi}{2}, \beta, \gamma) \quad (12)$$

$$\partial_\beta G = \frac{1}{2}G(\alpha, \beta + \frac{\pi}{2}, 0) + \frac{1}{2}G(\alpha, \beta + \frac{\pi}{2}, \pi) \quad (13)$$

$$\partial_\gamma G = \frac{1}{2}G(\alpha, 0, \gamma + \frac{\pi}{2}) + \frac{1}{2}G(\alpha, \pi, \gamma + \frac{\pi}{2}) \quad (14)$$

One can see that while the derivative with respect to  $\alpha$  requires us to implement the same gate but with the first parameter shifted by  $\frac{\pi}{2}$ , the derivative with respect to  $\mu = \beta$  [ $\mu = \gamma$ ] is a linear combination of single qubit gates where the original parameter  $\beta$  [ $\gamma$ ] is shifted by  $\frac{\pi}{2}$ , while  $\gamma$  [ $\beta$ ] is replaced by 0 or  $\pi$ .

Differentiating a controlled single qubit gate is not that immediate, but fortunately we have

$$\partial_\mu C(G) = \frac{1}{2} \left( C(\partial_\mu G) - C(-\partial_\mu G) \right),$$

which means that the derivative of the controlled single qubit gate is half of the difference between a controlled derivative gate and the controlled negative version of that gate. In our design, when  $\mu = \alpha$ , each of the two controlled gates is unitary, while  $\mu = \beta, \gamma$  requires us to use the linear combinations in (13) and (14).

If we plug the gate derivatives back into the expressions for the gradient in Equation (11), we see that the gradients, irrespective of the gate or parameter, can be computed as ‘classical’ linear combinations of the form

$$\partial_\mu \pi(x^m; \theta, b) = \sum_{j=1}^J a_j \operatorname{Re} \{ \langle U_{\theta[j]} \varphi(x^m) | \sigma_z | U_\theta \varphi(x^m) \rangle \},$$

where  $\theta[j]$  is a modified vector of parameters corresponding to a term appearing in Equations (12 - 14), and  $a_j$  is the corresponding coefficient also stemming from the Equations (12 - 14). If there is no parameter tying between the constituent gates, for example, then  $J$  is either 2 or 4 depending on whether the gate containing parameter  $\mu$  is a one- or two-qubit gate. For each circuit, the eventual derivative has to be estimated by repeated measurements, and we will discuss the number of repetitions in the following section.

The last thing to show is that we can compute the terms  $\operatorname{Re} \{ \langle U_{\theta[j]} \varphi(x^m) | \sigma_z | U_\theta \varphi(x^m) \rangle \}$  with the quantum device, so that classical multiplication and summation can deliver estimates of the desired gradients.

**Observation 3.** *Given two unitary quantum circuits  $A$  and  $B$  that act on a  $n$  qubit register to prepare the two quantum states  $|A\rangle, |B\rangle$ , and which can be applied conditioned on the state of an ancilla qubit, we can use the quantum device to sample from the probability distribu-*

$$\text{tion } p = \frac{1}{2} + \frac{1}{2} \operatorname{Re} \langle A|B \rangle$$

*Proof.* The proof of this observation follows from the exact same reasoning that underlies the Hadamard test. For concreteness, we specify the algorithm below. We use the circuits  $A, B$  to prepare the two states  $|A\rangle, |B\rangle$  conditioned on an ancilla,

$$\frac{1}{\sqrt{2}} (|0\rangle|A\rangle + |1\rangle|B\rangle).$$

Applying a Hadamard on the ancilla yields

$$\frac{1}{2} (|0\rangle(|A\rangle + |B\rangle) + |1\rangle(|A\rangle - |B\rangle)),$$

where the probability of the ancilla to be in state 0 is given by

$$p(a=0) = \frac{1}{2} + \frac{1}{2} \operatorname{Re} \langle A|B \rangle.$$

□

To use this interference routine we have to add an extra qubit and implement  $U_\theta$  and  $U_{\theta[j]}$  conditioned on the state of the ancilla. Since these two circuits coincide in all except from one gate, we do in fact only need to apply the differing gate in conditional mode. This turns a single qubit gate into a singly controlled single qubit gate, and a controlled gate into a double controlled gate. The desired value  $\operatorname{Re} \langle A|B \rangle$  can be derived by resolving  $p(a=0)$  through measurements and computing

$$\operatorname{Re} \langle A|B \rangle = 2p(a=0) - 1.$$

## D. Dropout

Despite the relatively small parameter space, our circuit-centric architecture is not immune to overfitting. Benchmarking on smaller data sets reveals cases where the training data is fit perfectly (zero misclassifications) by a model with exponentially few parameters, but the same model has significant generalization errors on the test holdout.

The approach that often helps is a simple *dropout regularization* that is both quantum-inspired and quantum ready (in the sense that it is easy in both classical simulation and quantum execution). The essence of the approach is to randomly select and measure one of the qubits, and set it aside for a certain number  $N_{\text{dropout}}$  of parameter update epochs. After that, the qubit is re-added to the circuit and another qubit (or, perhaps, no qubit) is randomly dropped. This strategy works by “smoothing” the model fit and it generally inflates the training error, but often deflates the generalization error.

The effect of such dropout regularization is similar, in spirit, to dropout regularization in a traditional neural

network when the dropout probability  $p = 0.5$  is used. Indeed, freezing a randomly chosen qubit for a certain number of epochs prevents a half of the amplitudes in the amplitude encoding from affecting the stochastic gradient during these epochs. In the graphical representation of the circuit-centric classifier this is analogous to removing a half of the nodes from a hidden layer for a certain number of epochs.

### E. Performance analysis

In order to use the circuit-centric quantum classifier with near-term quantum devices, we need to motivate that it only requires a small number of qubits, a low circuit depth as well as a high error tolerance. After introducing the details of the algorithms for inference and training, we want to discuss these three points in more detail.

#### 1. Circuit depth and width

The number of qubits needed for the circuit-centric quantum classifier (if we use amplitude encoding as explained in Section II A) is given by  $d \lceil \log_2 N \rceil$  where  $N$  is the dimension of the inputs and  $d$  is the number of copies we consider for a tensorial feature map. For example, if  $d = 1$ , we can process a dataset of 1000-dimensional inputs with  $n = 10$  qubits. With about 50 qubit we can use a tensorial feature map of  $d = 5$  (i.e., prepare 5 copies of the state) and map the data into a  $2^{50}$  dimensional feature space. For the inner products subroutine in the hybrid training scheme, we need one extra ancilla qubit. The algorithm is therefore very compact as much as circuit width is concerned, a feature stemming from the amplitude encoding strategy.

The bottleneck of the circuit depth is the state preparation routine  $S_x$ . Comparably, implementing the model circuit costs a negligible amount of resources. Using an architecture with  $K$  codeblocks of ranges  $(r_1, \dots, r_K)$  and  $n$  qubits, we need

$$Kn + \sum_{k=1}^K n/\gcd(n, r_k)$$

parametrised (controlled) single qubit gates to implement  $U_\theta$ , which is polynomial in the number of qubits. Each of these gates has to be decomposed into the elementary constant gate set used in the physical implementation of the quantum computer. Every parametrised single qubit gate can be efficiently translated into circuit  $\tilde{G}$  of at most  $\mathcal{O}(\log \frac{1}{\delta})$  constant elementary gates from a given gate set such as ‘‘Clifford-plus-T’’ to a fidelity of at least  $(1 - \delta)$  (cf. [38–41]). Methods such as automated optimization [42] may reduce the costs further.

General state preparation can in the worst case require  $c_{\text{cn}} 2^n$  CNOT gates as well as  $c_{\text{sgl}} 2^n$  single qubit gates. For current algorithms  $c_{\text{sgl}}$  and  $c_{\text{cn}}$  is equal to or slightly larger than 1 [22–24, 43, 44]. This means that for the example of  $N = 1000$  from above, we would indeed require  $2 \cdot 2^n = 2048$  gates only to prepare the states. Issues of fidelity arise, since without error correction we cannot guarantee to prepare a close enough approximation of  $x$ . Our simulations show that adding 5% noise to the inputs does not change the classification result significantly, which suggests that the classifier is rather robust against input noise. Still, until error correction becomes a reality, it is therefore advisable to focus on lower-dimensional datasets. Two interesting exceptions have to be mentioned. First, if an algorithm is known that efficiently allows us to approximate the (preprocessed) inputs with a product state,  $x \approx a_1 \otimes \dots \otimes a_K$  the resources reduce to the number of gates required to prepare the  $a_1, \dots, a_K$  in amplitude encoding [25]. Second, as other authors in quantum machine learning research, we point out that if the data is given by a shallow and robust digital quantum simulation routine performed on the same register of qubits, our classifier can be used to train with ‘quantum data’, or inputs that are ‘true’ wavefunctions.

#### 2. Number of repetitions for output estimation

The continuous output of the circuit-centric quantum classifier was based on the probability of measuring the first qubit in state 1. To resolve this number, we have to repeat the entire algorithm multiple times. Each measurement samples from the Bernoulli distribution  $p(q_0 = 1) = \nu$ , and we want to estimate  $\nu$  from the  $S$  samples  $q_1^1, \dots, q_1^S$ . The number of samples needed to estimate  $\nu$  at error  $\epsilon$  with probability  $> 2/3$  scales as  $O(\text{Var}(\sigma_z)/\epsilon^2)$ , where  $\text{Var}(\sigma_z)$  is the variance of the sigma-z operator that we measure with respect to the final quantum state [12, 34]. If amplitude estimation is used then the number of repetitions of circuit centric classifier falls into  $O(1/\epsilon)$  at a price of increasing the circuit depth by a factor of  $O(1/\epsilon)$ .

#### 3. Parameter noise

An important feature of the circuit-centric classifier is its robustness to noise in the inputs and parameters. Suppose  $\delta > 0$  is some small value and we allow parameter permutations (resp. input permutations) such that for each constituent gate  $G$  the permuted gate  $G'$  is  $\delta$ -close to  $G$ :  $\|G - G'\| < \delta$ . (Or for encoded input  $\phi$  a perturbed input  $\phi'$  is  $\delta$ -close to  $\phi$ .) we allow certain imprecisions in some or all parameter values and that such imprecisions are bounded below some constant  $\delta$ . Since all the constituent operations are unitary, the impact of the parameter imprecisions is never amplified across the circuit at the defect imposed by the imperfect

Noise level	RI Mean	RI St.Dev.	RI max	RI min
0.1%	1%	1.47%	3.5%	0%
1%	7%	6.67%	17%	0%
10%	60.2%	55.8%	192.3%	0%

TABLE I. Relative impact (RI) of uncorrelated parameter noise on the classification test error over SEMEION and MNIST256 data, using the generic 8-qubit model circuit displayed in Figure 4

circuit on the final state before the measurement is bounded by  $4L\delta$  in the worst theoretical case, where  $L$  is the number of elementary parametrised gates which have at most 4 parameters. In practice the propagated error should be much smaller than this bound.

The same analysis applies to imperfections in the quantum gates execution (other than parameter drift). There is no amplification of defect across the circuit and the imperfection of the final state is bounded by the sum of imperfections of individual gates. Finally, the ket encoding of the input data does not have to be perfect either. A possible imperfection or approximation during the state preparation will not be amplified by the classification circuit, and the drift of the pre-measurement state will be never be greater than the drift of the initial state. Another widely advertised advantage of variational quantum algorithms is that they can learn to counterbalance systematic errors in the device architecture – for example when one gate always over-rotates the state by the same value.

In our simulation experiments we have systematically evaluated the effects on the quality of the classification of 0.1%, 1% and 10% random perturbations in (a) the circuit parameters and (b) the input data vectors. As expected due to the unitariness, the effect of input noise is not amplified by the classifier circuit and thus had proportionate impact on the percentage of misclassifications. Somewhat more surprisingly, random perturbations of the trained circuit parameters almost never had the worst case estimated impact on the classification error. The 0.1% uncorrelated parameter noise in the majority of cases had no impact on the classification results.

Here we limit the noise impact discussion to the context of the “SEMEION” and “MNIST256” data sets of the benchmark sets displayed in Table II. Our observations are easier to calibrate in this context since both datasets are encoded with 8 qubits and the same model circuit architecture with 33 gates at depth 19 (as shown in Figure 4) is used in the classifier.

The 0.1% parameter noise had no impact on classifi-

cation in about 60% of our test runs. The maximum relative drift of the test error has been 3.5% (in one of the remaining runs), the mean drift has been 1% with the standard deviation of approximately 1.47%. The 1% parameter noise had a more pronounced, albeit fairly robust impact, which was non-trivial in about 90% of our test runs. The maximum relative change in the test error rate has been 17%, the mean relative change has been 7% with the standard deviation of approximately 6.67%. This statistics is summarized in Table I. Finally the 10% parameter noise lead to significant loss of classification robustness (although still smaller than the worst case analysis suggests). The maximum relative change in the test error rate has been 192.3%, the mean has been 60.2% with the standard deviation of 55.8%. Curiously, the minimum change has been zero in one of the “SEMEION”-based runs. This suggests that 10% perturbation of parameters has no stable amplification pattern, and the model should be best re-trained after such perturbation.

The practical takeaway from these observations is that the circuit-centric classifiers may work on small quantum computers even in the absence of strong quantum error correction.

## V. SIMULATIONS AND BENCHMARKING

To demonstrate that the circuit-centric quantum classifier works well in comparison with classical machine learning methods we present some simulations. The circuit-centric classifier was implemented on a classical computer using the F# programming language.

### 1. Datasets

We select six standard benchmarking datasets (see Table II). The CANCER, SONAR, WINE and SEMEION data sets are taken from the well-known UCI repository of benchmark datasets for machine learning [45]. The MNIST data set is the official NIST Modified handwritten digit recognition data set. While CANCER and SONAR are binary classification exercises, the other data sets call for multi-class classification. Although the circuit-centric quantum classifier could be operated as a multi-class classifier, we limit ourselves to the case of binary classification discussed above and cast the multi-label tasks as a set of “one-versus-all” binary discrimination subtasks. For example, in the case of digit recognition the  $i$ th subtask would be to distinguish the digit “ $i$ ” from all other digits. The results are an average taken over all subtasks.

The data is preprocessed for the needs of the quantum algorithm. The MNIST data vectors were course-grained into real-valued data vectors of dimension 256. With the

ID	domain	# features	# classes	# samples	preprocessing
CANCER	decision	32	2	569	none
SONAR	decision	60	2	208	padding with noninformative features
WINE	decision	13	3	178	padding with noninformative features
SEMEION	OCR <sup>a</sup>	256	10	1593	padding with noninformative features
MNIST256	OCR	256	10	2766	coarse-graining and deskewing

<sup>a</sup> Optical Character Recognition

TABLE II. Benchmark datasets and preprocessing.

ID	model	fixed hyperparameters	variable hyperparameters
QC	Circuit-centric quantum classifier	entangling circuit architecture	dropout rate, number of blocks, range
PERC	perceptron	-	regularisation type
MLPlin	neural network	dim. of hidden layer = $N$	regularisation strength, optimiser, initial learning rate
MLPshal	neural network	dim. of hidden layer = $\lceil \log_2 N \rceil$	regularisation strength, optimiser, initial learning rate
MLPdeep	neural network	dim. of hidden layers = $\lceil \log_2 N \rceil$	regularisation strength, optimiser, initial learning rate
SVMpoly1	support vector machine	polynomial kernel with $d = 1$ , regularisation strength of slack variables = 1, offset $c = 1$	-
SVMpoly2	support vector machine	polynomial kernel with $d = 2$ , regularisation strength of slack variables = 1, offset $c = 1$	-

TABLE III. Benchmark models explained in the text and possible choices for hyperparameters.

exception of this data set, in all other cases the data vectors have been padded non-informatively so that their dimension after padding is the nearest power of 2. After padding, each input vector was renormalized to unit norm. Thus each of the  $N$ -dimensional original data vectors would require  $n = \lceil \log_2(N) \rceil$  qubits to encode in amplitude encoding. In our simulations we do not use feature maps which would multiply the input dimensions. However, we expect that the circuit-centric classifier will gain a lot of power from this strategy, which can be tested on real quantum devices without the same amount of overhead.

## 2. Benchmark models

For the circuit-centric quantum classifier (QC) we use the data-agnostic entangling circuit of  $n$  qubits, which has been explained in Section III. We use one, two or three entangling layers, which means that the circuit contains anywhere from  $n + 1$  to  $6n + 1$  single-qubit and two-qubit gates. Therefore the number of real trainable

parameters varies from  $3n + 2$  to  $18n + 2$ . For each dataset we selected the circuit architecture with the lowest training error while reducing overfitting.

Defining a fair, systematic comparison is not straight forward since there are many different models and training algorithms we could compare with, each being further specified by hyperparameters. Without the use of feature maps, our classifier has only limited flexibility, and benchmarking against state-of-the-art models such as convolutional neural networks will therefore not provide much insight. Instead, we choose to compare our model to six classical benchmark models (see Table III) that are selected for their mathematical structure which is related to the circuit-centric classifier.

Section IIIC showed an interesting parallel to neural networks, which is why we take neural networks as one benchmark model family. From this family we choose 3 different architectures shown in Figure 8. The MLPlin model has a linear hidden layer of the same dimension  $N$  as the input layer and resembles the



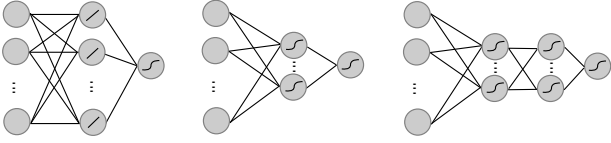


FIG. 8. The three architectures of the benchmark artificial neural network models. Left is the MLPlin model, which has a hidden linear layer of the size of the input  $N$  and a logistic output layer. The MLPshal model in the middle has a hidden layer of size  $\lceil \log_2 N \rceil$  with nonlinear activations and a logistic output layer. The MLPdeep model on the right adds a second nonlinear hidden layer.

architecture of our QC model displayed in Figure 3. The MLPshal and MLPdeep models have hidden layers of size  $\lceil \log_2 N \rceil$ . The motivation is to compare the QC with an architecture that - after the input layer of size  $N$  - gives rise to a polylogarithmic number of weights.

We use a support vector machine as a second benchmark model. Support vector machines are similar to the circuit-centric classifier since we can also think of them as a linear classifier in a feature space that is defined by a *kernel*  $\kappa$  [46]. We mentioned in Section II A that amplitude encoding can be supplemented by a feature map which is very similar to that of a support vector machine with a polynomial kernel,

$$\kappa(x, x') = (x^T x' + c)^d.$$

The offset  $c$  can be loosely compared to the effect of padding. The degree  $d$  of the kernel is not one-to-one comparable to the degree of the polynomial feature map in amplitude encoding, since our QC model effectively contains an ‘extra’ nonlinearity which derives from the measurement process (and is therefore not precisely a linear model in feature space). This can be seen in Figure 9 where we compare the decision boundaries of an SVM with polynomial degree  $d = 1, 2$  with the QC model and a feature map of degree  $d = 1, 2$ . To counterbalance the potential advantage of the QC, we consider two support vector machines, one with  $d = 1$  (SVMpoly1) and one with  $d = 2$  (SVMpoly2). The QC model does not use any feature maps. Finally, we add a perceptron (PERC) to the list of benchmark models to get an impression about the linear separability of the datasets.

Since one of our goals is to build a model with a small parameter space, we compare the number of trainable parameters for each model in Table IV. For the MLP and PERC models these parameters are the weights between units, and their number is defined by the dimension of inputs as well as the architecture of the network. For the SVM models we consider the number of input data points, since in their dual formulation these models start with assigning a weight to each input, after which they

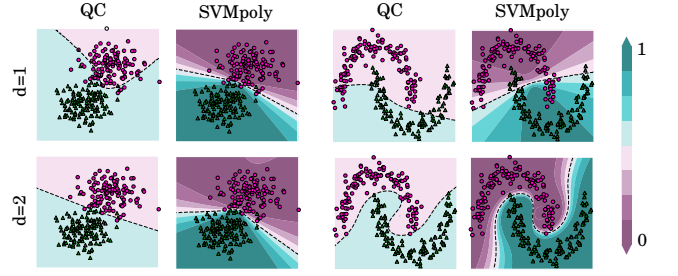


FIG. 9. Comparison of the decision boundary for the circuit-centric quantum classifier (QC) and a support vector machine with polynomial kernel (SVMpoly). The 2-dimensional data gets embedded into a 4-dimensional feature space (we padded with 2 non-informative features), where it is classified by the two models. The colour scale indicates the probability that the model predicts class “green circles”. For the QC model, the parameter  $d$  corresponds to the degree of the polynomial feature map in amplitude encoding (see Section II A). For the SVMpoly,  $d$  is the degree of the polynomial kernel. One can see that for  $d = 1$  the QC model is slightly more flexible than the SVMpoly.

ID	QC	PERC	MLPlin	MLPshal	MLPdeep	SVM
CANCER	79	32	1056	165	190	208
SONAR	60	60	1952	305	330	569
WINE	28	13	272	51	60	178
SEMEION	100	256	65792	2056	2120	1593
MNIST256	124	256	65792	2056	2120	800

TABLE IV. Number of parameters each model has to train for the different benchmark datasets. The circuit-centric quantum classifier QC has a logarithmic growth in the number of parameters with the input dimension  $N$  and data set size  $M$ , while all other models show a linear or polynomial parameter growth in either  $N$  or  $M$ .

reduce the training set to a few support vectors used for inference.

### 3. Results

For every benchmarking test (except from the QC model on SEMEION and MNIST256) we use 5-fold cross-validation with 10 repetitions each. This means that the results are an average of 50 repetitions of training the model and calculating the training and test error. Due to the significant cost of quantum circuit simulations, for the QC experiments on the SEMEION and MNIST256 datasets only one repetition of the 5-fold cross-validation was carried out. The results are summarized in Table V.

As we can read from the non-zero training error of the PERC model, none of the datasets is linearly separable.

	CANCER	SONAR	WINE*	SEMEION*	MNIST256*
QC	0.022/ <b>0.058</b>	0.000/0.195	0.000/0.028	0.031/0.031	0.031/0.033
PERC	0.128/0.137	0.283/0.315	0.067/0.134	0.022/0.038	0.065/0.066
MLPlin	0.060/0.075	0.117/0.263	0.001/ <b>0.039</b>	0.001/0.025	0.038/0.041
MLPshal	0.064/0.077	0.010/ <b>0.174</b>	0.029/0.063	0.002/ <b>0.024</b>	0.011/ <b>0.018</b>
MLPdeep	0.056/0.076	0.001/ <b>0.174</b>	0.010/0.063	0.001/0.026	0.014/0.021
SVMpoly1	0.373/0.367	0.452/0.477	0.430/0.466	0.101/0.100	0.092/0.092
SVMpoly2	0.169/0.169	0.334/0.383	0.090/0.099	0.100/0.101	0.091/0.092
Average	0.125/0.136	0.171/0.283	0.090/0.137	0.037/0.049	0.040/0.043

TABLE V. Results of the benchmarking experiments. The cells are of the format “training error/validation error”. The variance between the 50 repetition for each experiment was of the order of 0.01 – 0.001 for the training and test error. \*For multilabel classification problems with  $d$  labels, the average of all  $d$  one-versus-all problems train and test errors were taken.

One finds that the QC model performs significantly better than the SVMpoly1 and SVMpoly2 models across all data sets. In further simulations we verified that for support vector machines with polynomial kernel, degrees of  $d = 6$  to  $d = 8$  return the best results on the datasets, which are also better than those of the MLP models.

Although showing slightly worse test errors than the MLPshal and MLPdeep (and with mixed success compared to the MLPlin), the QC performs comparable with the MLP models that train a lot more parameters. For the SONAR and WINE dataset we find that the QC model overfits the training data. This is even worse without using the dropout qubit regularization technique explained above. The observation is interesting, since the QC model is ‘slimmer’ than the MLP and SVM models in terms of its parameter count. An open question is therefore how to introduce other means of regularisation.

## VI. CONCLUSION

We have developed a machine learning design which is both quantum inspired and implementable by near-term quantum technology. The key building block of this design is a unitary model circuit with relatively few trainable parameters that assumes amplitude encoding of the data vectors and uses systematically the entangling properties of quantum circuits as a resource for capturing correlations in the data. After state preparation, the prediction of the model is computed by applying only a small number of one- and two-qubit gates quantum gates. At the same time, simulating these gates on a classical computer requires a of elementary operations that scales with the number of features. We are aware of prior designs of unitary neural nets available in literature (cf. [19, 20] and related work). In these designs the number of learnable parameters is proportional to the number of data features, whereas the size of our model

circuit therefore scales with the number of qubits and thus allows, qubit-wise, for exponentially fewer learnable parameters than what traditional methods would use. We have also shown in some preliminary simulations that the design can indeed achieve results close to off-the-shelf methods that have comparable limitations but considerably more tunable parameters.

The quantum classifiers based on model circuits that we have explored so far belong to a class of weakly nonlinear classifiers. The main source of nonlinearity in our classifiers stems from the concluding measurement and thresholding on the probabilities of the measurement outcomes. These probabilities are roughly quadratic in the amplitudes of the final post-circuit state. Since the overall effect of the model circuit on the amplitudes is linear reversible, one concludes that the models we have experimented with can capture (amplitude-wise) quadratic separation of classes in the original feature space.

There is a potential for tracing class separation boundaries of higher polynomial degrees by encoding several copies of a classical data vector in disjoint quantum registers and then having the model circuit work on the tensor power of the data vector. Of course, this setup requires vastly more computational resources to simulate the quantum circuit, and we leave such experiments for the future. We furthermore expect that the most beneficial application of our quantum classifiers is to quantum data. One can conceive entangling a quantum system with a classifier circuit and use the latter to discriminate between various states of the quantum system.

This work contributes to the growing literature on variational circuits for machine learning applications in proposing a specific circuit architecture and parametrisation, a dropout regularisation technique, a hybrid train-

ing method, as well as a graphical interpretation of quantum operations in the language of neural networks. However, an overwhelming number of questions is still largely unexplored. For example, we noticed that our slim circuit design still suffers from overfitting. Also, full-fledged numerical benchmarks on larger datasets are needed to systematically analyse the effect of logarithmically few parameters with growing input spaces. The power of feature maps to introduce nonlinearities is another open question. Further numerical as well as theoretical studies are therefore crucial to understand the convergence properties and representational power of circuit-centric models for classification.

## ACKNOWLEDGMENTS

The Authors are grateful to Jeongwan Haah for insightful remarks and wish to thank Martin Roetteler for numerous useful discussions of the ongoing research and early drafts of this paper. MS wishes to thank the entire QuArC group at Microsoft Research, Redmond, whose collective help during her research internship had been generous, professional and very welcoming.

- 
- [1] Masoud Mohseni, Peter Read, Hartmut Neven, Sergio Boixo, Vasil Denchev, Ryan Babbush, Austin Fowler, Vadim Smelyanskiy, and John Martinis. Commercialize quantum technologies in five years. *Nature*, 543(7644):171, 2017.
  - [2] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.
  - [3] Michael J Bremner, Richard Jozsa, and Dan J Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, page 20100301. The Royal Society, 2010.
  - [4] Joonsuk Huh, Gian Giacomo Guerreschi, Borja Peropadre, Jarrod R McClean, and Alán Aspuru-Guzik. Boson sampling for molecular vibronic spectra. *Nature Photonics*, 9(9):615–620, 2015.
  - [5] Aram W Harrow and Ashley Montanaro. Quantum computational supremacy. *Nature*, 549(7671):203, 2017.
  - [6] Alejandro Perdomo-Ortiz, Marcello Benedetti, John Realpe-Gómez, and Rupak Biswas. Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers. *arXiv preprint arXiv:1708.09757*, 2017.
  - [7] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, 2015.
  - [8] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195, 2017.
  - [9] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical Review Letters*, 113:130503, Sep 2014.
  - [10] Iordanis Kerenedis and Anupam Prakash. Quantum recommendation systems. *arXiv preprint arXiv:1603.08675*, 2016.
  - [11] Nathan Wiebe, Daniel Braun, and Seth Lloyd. Quantum algorithm for data fitting. *Physical Review Letters*, 109(5):050505, 2012.
  - [12] Maria Schuld, Mark Fingerhuth, and Francesco Petruccione. Implementing a distance-based classifier with a quantum interference circuit. *EPL (Europhysics Letters)*, 119(6):60002, 2017.
  - [13] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.
  - [14] Jonathan Romero, Jonathan P Olson, and Alan Aspuru-Guzik. Quantum autoencoders for efficient compression of quantum data. *Quantum Science and Technology*, 2(4):045001, 2017.
  - [15] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *arXiv preprint arXiv:1803.00745*, 2018.
  - [16] G. Verdon, M. Broughton, and J. Biamonte. A quantum algorithm to train neural networks using low-depth circuits. *arXiv preprint arXiv:1712.05304*, 2017.
  - [17] E. Farhi and H. Neven. Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002*, 2018.
  - [18] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L Obrien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5, 2014.
  - [19] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *Journal of Machine Learning Research*, 48, 2016.
  - [20] Li Jing, Yichen Shen, Tena Dubček, John Peurifoy, Scott Skirlo, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnn. *arXiv preprint arXiv:1612.05231*, 2016.
  - [21] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.
  - [22] Mikko Möttönen, Juha J Vartiainen, Ville Bergholm, and Martti M Salomaa. Quantum circuits for general multiqubit gates. *Physical Review Letters*, 93(13):130502, 2004.
  - [23] Emanuel Knill. Approximation by quantum circuits. *arXiv preprint quant-ph/9508006*, 1995.
  - [24] Martin Plesch and Časlav Brukner. Quantum-state preparation with universal gate decompositions. *Physical Review A*, 83(3):032302, 2011.
  - [25] Lov Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions. *arXiv preprint arXiv:0208112v1*, 2002.

- [26] Andrei N Soklakov and Rüdiger Schack. Efficient state preparation for a register of quantum bits. *Physical Review A*, 73(1):012307, 2006.
- [27] These can be exact copies when direct classical access to data is available. Otherwise these could be approximate clones of the data point generated at sufficient fidelity.
- [28] Edwin Stoudenmire and David J Schwab. Supervised learning with tensor networks. In *Advances In Neural Information Processing Systems*, pages 4799–4807, 2016.
- [29] J.-L. Brylinski and R. Brylinski. Universal quantum gates. pages 101–116, 2002.
- [30] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457, 1995.
- [31] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, John M Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *arXiv preprint arXiv:1608.00263*, 2016.
- [32] According to the laws of quantum mechanics we have to sum over the absolute square values of the amplitudes that correspond to basis states where the first qubit is in state 1. Using the standard computational basis, this is exactly the ‘second half’ of the amplitude vector, ranging from entry  $2^{n-1} + 1$  to  $2^n$ .
- [33] Yoav Levine, David Yakira, Nadav Cohen, and Amnon Shashua. Deep learning and quantum entanglement: Fundamental connections with implications to network design. *arXiv preprint arXiv:1704.01552*, 2017.
- [34] Gian Giacomo Guerreschi and Mikhail Smelyanskiy. Practical optimization for hybrid quantum-classical algorithms. *arXiv preprint arXiv:1701.01450*, 2017.
- [35] Stochastic gradient descent originally referred to the case  $B = 1$ , but is often used as a synonym for *minibatch* gradient descent as opposed to *batch* gradient descent using the full dataset.
- [36] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT’2010*, pages 177–186. Springer, 2010.
- [37] Andrew M Childs and Nathan Wiebe. Hamiltonian simulation using linear combinations of unitary operations. *Quantum Information and Computation*, 12:901–924, 2012.
- [38] A. Bocharov, Y. Gurevich, and K.M. Svore. Efficient decomposition of single-qubit gates into v basis circuits. *Physical Reviews, A*, 88:012313, 2013.
- [39] V. Kliuchnikov, A. Bocharov, and K.M. Svore. Asymptotically optimal topological quantum compiling. *Physical Review Letters*, 112:140504, 2014.
- [40] P. Selinger. Efficient clifford+t approximation of single-qubit operators. *Quantum Information and Computation*, 15:159–180, 2015.
- [41] V. Kliuchnikov, A. Bocharov, M. Roetteler, and J. Yard. A framework for approximating qubit unitaries. *arXiv preprint arXiv:1510.03888*, 2016.
- [42] Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. *arXiv preprint arXiv:1710.07345*, 2017.
- [43] Juha J Vartiainen, Mikko Möttönen, and Martti M Salomaa. Efficient decomposition of quantum gates. *Physical review letters*, 92(17):177902, 2004.
- [44] Raban Iten, Roger Colbeck, Ivan Kukuljan, Jonathan Home, and Matthias Christandl. Quantum circuits for isometries. *Physical Review A*, 93(3):032318, 2016.
- [45] The most recently accessed location of the UCI database is <https://archive.ics.uci.edu/ml/datasets.html> (more specifically, the CANCER data set is retrieved from <https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29>, the SEMEION from <https://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit>, the SONAR from <https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar%2C+Mines+vs.+Rocks%29> and the WINE from <https://archive.ics.uci.edu/ml/datasets/Wine>).
- [46] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press, 2002.