Quantum Machine Intelligence (2019) 1:41–51
https://doi.org/10.1007/s42484-019-00004-7

RESEARCH ARTICLE

Bayesian deep learning on a quantum computer

Zhikuan Zhao[1,2,3] · Alejandro Pozas-Kerstjens[4] · Patrick Rebentrost[3] · Peter Wittek[5,6,7,8]

Received: 23 November 2018 / Accepted: 13 March 2019 / Published online: 15 May 2019

Abstract

Bayesian methods in machine learning, such as Gaussian processes, have great advantages compared to other techniques. In particular, they provide estimates of the uncertainty associated with a prediction. Extending the Bayesian approach to deep architectures has remained a major challenge. Recent results connected deep feedforward neural networks with Gaussian processes, allowing training without backpropagation. This connection enables us to leverage a quantum algorithm designed for Gaussian processes and develop a new algorithm for Bayesian deep learning on quantum computers. The properties of the kernel matrix in the Gaussian process ensure the efficient execution of the core component of the protocol, quantum matrix inversion, providing at least a polynomial speedup over classical algorithms. Furthermore, we demonstrate the execution of the algorithm on contemporary quantum computers and analyze its robustness with respect to realistic noise models.

Keywords Bayesian methods · Quantum computing · Quantum algorithms · Quantum-enhanced AI · Experimental quantum computing

1 Introduction

The Bayesian approach to machine learning provides a clear advantage over traditional techniques, namely, it provides information about the uncertainty in their predictions. But not only that, they have further advantages, including automated ways of learning structure and avoiding overfitting, a principled foundation (Ghahramani 2015), and robustness to adversarial attacks (Bradshaw et al. 2017; Grosse et al. 2017). The Bayesian framework has been making advances in various deep architectures (Blundell et al. 2015; Gal and Ghahramani 2016). Some recent advances made a connection between a quintessentially Bayesian model, Gaussian processes (GPs) (Rasmussen and Williams 2006), and deep feedforward neural networks (Lee et al. 2018; Matthews et al. 2018).

Parallel to these developments, quantum technologies have been making advances in machine learning. A new breed of quantum neural networks is aimed at current and near-future quantum computers (Verdon et al. 2017 2018; Torrontegui and Garcia-Ripoll 2018; Khoshaman et al. 2018; Farhi and Neven 2018), which is in stark contrast with attempts in the past (Schuld et al. 2014). Some constraints must be observed that are unusual in classical machine learning algorithms. In particular, the protocol must be coherent, that is, we require from a quantum machine learning algorithm that it is described by a unitary map that maps input nodes to output nodes. While the common wisdom is that a nonlinear activation is a necessary component in neural networks, a linear, unitary mapping between the inputs and outputs actually reduces the vanishing gradient problem (Arjovsky et al. 2015; Hyland and Rätsch 2017). Training a hierarchical representation in a unitary fashion is also possible on

✉ Alejandro Pozas-Kerstjens
alejandro.pozas@icfo.es

1 Department of Computer Science, ETH Zurich, Universitätstrasse 6, 8092 Zürich, Switzerland

2 Singapore University of Technology and Design, 8 Somapah Road, Singapore, 487372 Singapore

3 Centre for Quantum Technologies, National University of Singapore, 3 Science Drive 2, Singapore, 117543 Singapore

4 ICFO-Institut de Ciencies Fotoniques, The Barcelona Institute of Science and Technology, 08860 Castelldefels, Barcelona, Spain

5 Rotman School of Management, University of Toronto, M5S 3E6 Toronto, Canada

6 Creative Destruction Laboratory, M5S 3E6 Toronto, Canada

7 Vector Institute for Artificial Intelligence, M5G 1M1 Toronto, Canada

8 Perimeter Institute for Theoretical Physics, N2L 2Y5 Waterloo, Canada

classical computers (Liu et al. 2017; Stoudenmire 2018). So while this constraint is unusual, it is not entirely unheard of in classical machine learning, and it is the most common setting in quantum-enhanced machine learning (Biamonte et al. 2017). Furthermore, the description of quantum mechanics uses complex numbers and some promising results in machine learning show advantages of using these over real numbers (Trabelsi et al. 2017).

In this paper, we exploit the connection between deep learning and Gaussian processes and rely on a quantum-enhanced protocol for the latter (Zhao et al. 2015) to develop new algorithms that perform quantum Bayesian training of deep neural networks. We implement the core of the algorithm on both the Rigetti Forest (Smith et al. 2016) and the IBM QISKit (Cross et al. 2017) software stacks, and analyze how noise affects the success of the calculations on both quantum simulators. To run on real quantum processing units, we implement a simplified, shallow-circuit version of the protocol, and compare the outcome with the simulations. The source code is available under an open source license.[1]

2 Background

The algorithm that we present makes use of two previous results, which we now briefly review: the connection between deep neural networks and Gaussian processes (Section 2.1), and the quantum Gaussian process protocol (Section 2.2).

2.1 Gaussian processes and deep learning

The correspondence between Gaussian processes and a neural network with a single hidden layer is well-known (Neal 1994). Let $z(x) \in \mathbb{R}^{d_{out}}$ denote the output with input $x \in \mathbb{R}^{d_{in}}$, with $z_i(x)$ denoting the ith component of the output layer. If the weight and bias parameters are taken to be i.i.d, each z_i will be a sum of i.i.d terms. If the hidden layer has an infinite width, the Central Limit Theorem implies that z_i follows a Gaussian distribution. Now let us consider a set of k input data points, with corresponding outputs $\{z_i(x^{[1]}), z_i(x^{[2]}), \ldots z_i(x^{[k]})\}$. Any finite collection of the set will follow a joint multivariate Gaussian distribution. Therefore, z_i corresponds to a Guassian process, $z_i \sim \mathcal{GP}(\mu, K)$. Conventionally, the parameters are chosen to have zero mean, so the mean of the GP, μ, is equal to 0. The covariance matrix K is given by $K(x, x') = \mathbb{E}[z_i(x)z_i(x')]$.

The Bayesian training of the neural network then corresponds to computing the posterior distribution of the given GP model, that is, calculating the mean and variance of the predictive distribution from inverting the covariance matrix. Choosing the GP prior amounts to the selection of the covariance function and tuning the corresponding hyperparameters. These include the information of the neural network model class, depth, nonlinearity, and weight and bias initializations.

This argument is generalized to a deep neural network architecture in a recursive manner (Lee et al. 2018; Matthews et al. 2018). Let z_i^l denote the ith component of the output of the lth layer. By induction, it follows that $z_i^l \sim \mathcal{GP}(0, K^l)$. The covariance matrix on the lth layer is given by $K^l(x, x') = \mathbb{E}[z_i^l(x)z_i^l(x')]$. To explicitly compute $K^l(x, x')$, we need to specify the variance on the weight and bias parameters, σ_w^2 and σ_b^2, as well as the nonlinearity ϕ. In a single-line recursive formula, this reads as:

$$K^l(x, x') = \sigma_b^2 + \sigma_w^2 \mathbb{E}[\phi(z_i^{l-1}(x))\phi(z_i^{l-1}(x'))], \tag{1}$$

where $z_i^{l-1} \sim \mathcal{GP}(0, K^{l-1})$. The base case of the induction is given by $K^0(x, x') = \sigma_b^2 + \sigma_w^2 \left(\frac{x.x'}{d_{in}}\right)$.

Remarkably, numerical experiments suggest that the infinite-width neural network trained with Gaussian priors outperforms finite deep neural networks trained with stochastic gradient descent in many cases (Lee et al. 2018; Matthews et al. 2018).

2.2 Quantum Gaussian process algorithm

A quantum algorithm for Gaussian process regression was introduced in Zhao et al. (2015). Given a supervised learning problem with a training dataset with input points $\{x_i\}_{i=0}^{n-1}$ and corresponding output points $\{y_i\}_{i=0}^{n-1}$, the quantum GP algorithm leverages the quantum linear system subroutine introduced in Harrow et al. (2009), and computes a GP model's mean predictor,

$$\bar{f}_* = k_*^T (K + \sigma_n^2 I)^{-1} y \tag{2}$$

and variance predictor,

$$\mathbb{V}[f_*] = k(x_*, x_*) - k_*^T (K + \sigma_n^2 I)^{-1} k_*. \tag{3}$$

Here, $(K + \sigma_n^2 I)$ denotes the model's covariance matrix with Gaussian noise entries of variance σ_n^2, and k_* denotes the row in the covariance matrix that corresponds to the target point for prediction. The scalar $k(x_*, x_*)$ is the covariance function of the target point with itself, and takes only a constant time to compute.

Assuming a black box access to the matrix elements of K, the quantum GP algorithm simulates $(K + \sigma_n^2 I)$ as a Hamiltonian acting on an input state, $|b\rangle$, performs quantum phase estimation (Kitaev 1995) to extract estimates of the eigenvalues of $(K + \sigma_n^2 I)$, and stores them in a quantum register as a weighted superposition. While in superposition, the stored eigenvalues are inverted and used to construct a controlled rotation on an ancillary system. Conditioned on a final measurement result on the ancillary system, the algorithm probabilistically completes a computation

[1] https://gitlab.com/apozas/bayesian-dl-quantum/

for $(K + \sigma_n^2 I)^{-1}|b\rangle$. Depending on whether the aim is computing the mean predictor or the variance predictor, one chooses $|b\rangle = |y\rangle$ or $|b\rangle = |k_*\rangle$, which encodes the classical vectors y or k_* respectively. Finally, applying a quantum inner product routine, such as those described in (Tacchino et al. 2018; Schuld and Killoran 2018), allows for a good estimation of the quantities $k_*^T(K + \sigma_n^2 I)^{-1}y$ and $k_*^T(K + \sigma_n^2 I)^{-1}k_*$, which leads to the goal of a GP regression model computation.

The quantum GP algorithm runs in $\tilde{\mathcal{O}}(\log(n))$ time when K is sparse and well-conditioned. A caveat here is that the quantum algorithm only runs in logarithmic time for sparse covariance matrices, and this could restrict the form of the nonlinear function or other parameters in the network architecture. The simulation of sparse Hamiltonians is more efficient when using quantum computers (Lloyd 1996; Childs 2010; Berry and Childs 2012). This can be addressed by tapering the covariance function using a compactly supported function (Furrer et al. 2006); a similar methodology is also known in kernel methods (Wittek and Tan 2011). Furthermore, one could apply the methods in Wossnig et al. (2018) to construct a $\mathcal{O}(\sqrt{n})$ time algorithm for Gaussian processes. This should ensure at least a polynomial quantum speedup for general constructions. Subsequent to the quantum GP algorithm, a corresponding quantum method for enhancing the training and model selection of GPs was introduced in Zhao et al. (2018).

3 Quantum Bayesian training of neural networks

Now, we leverage the previous two results to develop a way of conducting Bayesian training of deep neural networks using a Gaussian prior.

According to the connection described in Section 2.1, Bayesian training of a deep neural network of L layers requires sampling the values of the neurons in the final layer from the Gaussian process $\mathcal{GP}(0, K^L)$, where K^L can be computed in a recursive manner beginning from K^0 following Eq. 1. If we had classical access to the elements of the covariance matrix K^0, one possibility could be to classically compute K^L and then resort to the simulation of the Hamiltonian evolution generated by K^L to obtain the mean predictor $\bar{f}_*$ and variance predictor $\mathbb{V}[f_*]$ needed in the quantum Gaussian process algorithm of Section 2.2 (Zhao et al. 2015). This procedure would require simulating the Hamiltonian evolution from a classical encoding of K^L, which may hinder the speedup expected from the algorithm in this case.

The algorithm we propose makes use of the following observation: for the quantum Gaussian process algorithm, there is no need to have a complete knowledge of the covariance matrix. In reality, one just needs to know the time evolution operator under the covariance matrix encoded as a Hamiltonian. We propose a way of constructing such time evolution operator given access to a quantum encoding of the base case covariance matrix K^0, either in the form of oracular access or encoded as a density matrix of a qubit system (we discuss both possibilities later in this section). Once the time evolution operator is simulated, our algorithm, as the quantum Gaussian process algorithm, needs sampling from only one Gaussian process, that corresponds to the last layer in the network.

A requirement of the algorithm is, as in the classical case, a functional expression of the covariance matrix in the last layer in terms of the base case K^0. For general nonlinear activation functions, this can only be done with numerical integration, which seems quite unreachable to implement coherently with contemporary quantum computers. A complete quantum protocol would require a large number of qubits and at least polynomial-size quantum circuits, which remains out of reach with current technology. However, different works showed activation functions which yield kernels and recursion relations that can be analytically calculated or approximated (Cho and Saul 2009; Daniely et al. 2016). A particularly useful special case amounts to using only the ReLU nonlinear activation on every layer. The ReLU activation function is $\phi(x) = \max(0, x)$, and has been crucial in addressing issues such as the vanishing gradient problem in deep learning (Glorot et al. 2011). For this case, the lth layer covariance matrix has an analytical formula (Lee et al. 2018):

$$K^l(x, x') = \sigma_b^2 + \frac{\sigma_w^2}{2\pi}\sqrt{K^{l-1}(x', x')K^{l-1}(x, x)} \times \left(\arcsin(\theta_{x,x'}^{l-1}) - (\pi - \theta_{x,x'}^{l-1})\arccos(\theta_{x,x'}^{l-1})\right), \quad (4)$$

where

$$\theta_{x,x'}^l = \arccos\left(\frac{K^l(x, x')}{\sqrt{K^l(x, x)K^l(x', x')}}\right).$$

The nonlinear functions featured in Eq. 4 can be approximated by polynomial series with some convergence conditions. The factor $K^l(x, x)K^l(x', x')$ represents outer products between the two identical vectors of diagonal entries in K^l. As such, the computation of Eq. 4 can be decomposed into such outer product operations combined with element-wise matrix multiplication. In Sections 3.2 and 3.3, we provide a construction for simulating the evolution under the Hamiltonians generated by these operations on the matrix elements of a quantum state.

For the remaining discussion, we briefly introduce the mathematical formalism of quantum computing. In particular, a ket $|x\rangle$ denotes a column vector $x \in \mathbb{C}^d$ for some dimension d, with norm 1. Its complex conjugate is a bra $\langle x|$.

A ket represents a pure quantum state. A quantum computer essentially transforms quantum states into quantum states, and the result of the quantum computation is a quantum state with some desired properties. The density matrix of a pure state is the outer product of ket and the corresponding bra, and it is a positive semidefinite matrix with trace 1. For pure states, the density matrix is an equivalent way of describing a quantum state. In addition, the density matrix allows to describe mixed quantum states, i.e., statistical ensembles of pure states. For the algorithm proposed here, it is needed that K^0 is given as a real symmetric, positive semi-definite matrix, normalized by its trace in order to qualify as a quantum state (Rebentrost et al. 2014). All but the last property are satisfied by the definition of covariance matrix, and the last one can be achieved with an appropriate rescaling, equivalent to an appropriate choice of the kernel function. For more details on quantum computations, we refer the reader to Nielsen and Chuang (2000).

As introduced above, the quantum algorithms used in the present work can admit two data-input models. First, we can assume efficient computability or oracular access to the matrix elements of the covariance matrix K^0. In this model, the quantum simulation methods of Berry and Childs (2012) and Berry et al. (2015) can be used in the quantum GP algorithm, as long as the assumptions of these methods are satisfied. Second, we can assume that the covariance matrix is presented as the quantum density matrix of a qubit system. Multiple copies of such a density matrix allow the use of a method inspired by the quantum principal component analysis algorithm (Rebentrost et al. 2014). We discuss the first method for the single-layer case and the second method for the multiple-layer case.

3.1 Single-layer case

Assume that we are given oracle access to the matrix elements of the base case:

$$O_{K^0}|j,k\rangle|z\rangle \rightarrow |j,k\rangle|z \oplus K^0_{jk}\rangle,$$

where the matrix elements are written in the notation $K^0_{jk} = K^0(x_j, x_k)$. The desired kernel function of Eq. 4 can be implemented by oracle queries using ancillary labeling registers with $|j,\, j\rangle$, $|k,\, k\rangle$, and $|j,\, k\rangle$, as well as an additional register which stores the value of a classical computation step. This procedure can be described as follows:

$$O_{K^0}|j,\, j\rangle|k,\, k\rangle|j,\, k\rangle|0\rangle \rightarrow |j,\, j\rangle|k,\, k\rangle|j,\, k\rangle|0 \oplus K^1_{jk}\rangle. \quad (5)$$

With the oracle access to the elements of K^0, the first, and final, layer covariance matrix K^1 can be classically computed and simulated as a Hamiltonian used in the quantum GP algorithm.

3.2 Multi-layer case

In the case of multi-layer network architectures, we describe a method to simulate the lth-layer kernel matrix as a Hamiltonian. Our approach is inspired by the quantum principal component analysis algorithm (Rebentrost et al. 2014) where the density matrix ρ of a quantum state is treated as a Hamiltonian and used to construct the desired controlled unitary $e^{it\rho}$ acting on a target quantum state for a time period t. This is an unusual concept for classical machine learning and classical algorithms: a high-dimensional vector becomes an operator on itself to reveal its own eigenstructure. A thorough description of this density matrix-based Hamiltonian simulation procedure is presented in Kimmel et al. (2017). Here, we will first give an overall description of the quantum method, while the detailed analysis is presented later in the paper.

In order to apply density matrix-based Hamiltonian simulation using the lth-layer kernel, we need to incorporate methods to compute certain element-wise matrix operations between two density matrices. It is convenient to define the following:

$$S_1 = \sum_{j,k} |j\rangle\langle k| \otimes |j\rangle\langle k| \otimes |k\rangle\langle j|,$$

$$S_2 = \sum_{j,k} |j\rangle\langle j| \otimes |k\rangle\langle k| \otimes |k\rangle\langle j|.$$

With an augmented density matrix exponentiation scheme, S_1 computes exponential of the Hadamard product of two density matrices, while S_2 computes the exponential of the outer product between the diagonal entries of two density matrices. Specifically, we have:

$$\begin{aligned}&\mathrm{tr}_{1,2}\{e^{-\mathrm{i}S_1\delta}(\rho_1 \otimes \rho_2 \otimes \sigma)e^{\mathrm{i}S_1\delta}\}\\ &\quad = \exp[-\mathrm{i}(\rho_1 \odot \rho_2)\delta]\,\sigma\,\exp[\mathrm{i}(\rho_1 \odot \rho_2)\delta] + \mathcal{O}(\delta^2), \quad (6)\end{aligned}$$

where $\rho_1 \odot \rho_2$ denotes the Hadamard product between ρ_1 and ρ_2, and $\mathrm{tr}_{1,2}$ denotes tracing out the first and second subsystems, respectively. The factor δ represents a small evolution time with the operator in the exponents. We also have:

$$\begin{aligned}&\mathrm{tr}_{1,2}\{e^{-\mathrm{i}S_2\delta}(\rho_1 \otimes \rho_2 \otimes \sigma)e^{\mathrm{i}S_2\delta}\}\\ &\quad = \exp[-\mathrm{i}(\rho_1 \oslash \rho_2)\delta]\sigma\,\exp[\mathrm{i}(\rho_1 \oslash \rho_2)\delta] + \mathcal{O}(\delta^2), \quad (7)\end{aligned}$$

where $\rho_1 \oslash \rho_2$ denotes taking the outer product between the diagonal entries of ρ_1 and ρ_2. The derivations of Eqs. 6 and 7 are presented later in Section 3.3. Both S_1 and S_2 are sparse and thus efficiently simulable as a Hamiltonian with methods based on quantum walks (Berry and Childs 2012; Berry et al. 2015). A similar method of using a modified version of the SWAP operator combined with density matrix exponentiation scheme was used in Rebentrost et al. (2018) for a quantum singular value decomposition algorithm.

In order to approximately compute the nonlinear function of Eq. 4, we make use of a polynomial series in $K^0(x, x')$. Note that due to the structure of Eq. 4, the products involved in this polynomial series are the Hadamard products denoted by $\odot$, and the diagonal outer products denoted by $\oslash$. We will denote the polynomial in K^0 to the order $N(l)$ which approximates the lth-layer kernel function as $P^N_{(\odot,\oslash)}(K^0)$.

We note that by using a generalized $\tilde{S}$ operator which combines the components in S_1 and S_2, one can implement a total N number of $\odot$ and $\oslash$ operations in arbitrary orders. In Section 3.3, we will show this simply amounts to summing over the tensor product of the projectors $|j\rangle\langle j|$, $|j\rangle\langle k|$, and $|k\rangle\langle k|$. Similar polynomial series simulation problems were addressed in Kimmel et al. (2017) and Rebentrost et al. (2016), but the type of product considered was standard matrix multiplication instead of element-wise operations.

The quantum technique described above combined with using the series expansions of the nonlinear functions in Eq. 4 gives us a way to approximate $e^{\mathrm{i}tK^l}\sigma e^{-\mathrm{i}tK^l}$, where σ is an arbitrary input state. Hence, given multiple copies of a density matrix which encodes the initial layer covariance matrix, K^0, the unitary operator, $\exp(-\mathrm{i}tK^l)$ can be constructed to act on an arbitrary input state, as required by applying the quantum GP algorithm described in Section 2.2. Note that there is a subtle but crucial difference between the single-layer and the multi-layer case: while in the training of single-layer networks one needs a quantum random access memory to perform the oracle queries of the matrix elements of K^0, in the multi-layer case we substitute this requirement by having access to multiple copies of the quantum state encoding K^0. This requirement is much more feasible given the current technology since the desired state preparation can be encoded in a quantum circuit and run as many times as needed.

3.3 Coherent element-wise operations

In this section, we give a more formal description of the quatum method for approximately computing the polynomial $P^N_{(\odot,\oslash)}(K^0)$. The main results needed are well summarized by the following Lemmas 1 and 2, and Theorem 1.

Lemma 1 *Given $\mathcal{O}(t^2/\epsilon)$ copies of d-dimensional qubit density matrices, ρ_1 and ρ_2, let $\rho_1 \odot \rho_2$ denote the Hadamard product between ρ_1 and ρ_2. There exists a quantum algorithm to implement the unitary $e^{-\mathrm{i}\rho_1\odot\rho_2 t}$ on a d-dimensional qubit input state σ, for a time t to accuracy ϵ in operator norm.*

Proof The usual SWAP matrix employed in quantum principal component analysis (Rebentrost et al. 2014) is given by $S = \sum_{j,k} |j\rangle\langle k| \otimes |k\rangle\langle j|$. Here, we take the modified SWAP operator $S_1 = \sum_{j,k} |j\rangle\langle k|\otimes|j\rangle\langle k|\otimes|k\rangle\langle j|$. With an arbitrary input state σ, the operation

$$\mathrm{tr}_{1,2}\{e^{-\mathrm{i}S_1\delta}(\rho_1\otimes\rho_2\otimes\sigma)e^{\mathrm{i}S_1\delta}\} \tag{8}$$

can be efficiently performed with a small parameter δ. The symbol $\mathrm{tr}_{1,2}$ represents the trace over the subspaces of ρ_1 and ρ_2. Expanding (8) to $\mathcal{O}(\delta^2)$ leads to:

$$\begin{aligned}&\mathrm{tr}_{1,2}\{e^{-\mathrm{i}S_1\delta}(\rho_1\otimes\rho_2\otimes\sigma)e^{\mathrm{i}S_1\delta}\}\\ &\quad= 1-\mathrm{i}\,\mathrm{tr}_{1,2}\{S_1(\rho_1\otimes\rho_2\otimes\sigma)\}\delta\\ &\quad+\mathrm{i}\,\mathrm{tr}_{1,2}\{(\rho_1\otimes\rho_2\otimes\sigma)S_1\}\delta\\ &\quad+\mathcal{O}(\delta^2).\end{aligned} \tag{9}$$

Examining the first element linear in the parameter δ reveals:

$$\begin{aligned}&\mathrm{tr}_{1,2}\{S_1(\rho_1\otimes\rho_2\otimes\sigma)\}\\ &\quad=\mathrm{tr}_{1,2}\left\{\sum_{j,k}|j\rangle\langle k|\otimes|j\rangle\langle k|\otimes|k\rangle\langle j|(\rho_1\otimes\rho_2\otimes\sigma)\right\}\\ &\quad=\sum_{n,m,j,k}\langle n|j\rangle\langle k|\rho_1|n\rangle\langle m|j\rangle\langle k|\rho_2|m\rangle|k\rangle\langle j|\sigma\\ &\quad=\sum_{j,k}\langle k|\rho_1|j\rangle\langle k|\rho_2|j\rangle|k\rangle\langle j|\sigma\\ &\quad=(\rho_1\odot\rho_2)\,\sigma.\end{aligned} \tag{10}$$

In the same manner, we have:

$$\mathrm{tr}_{1,2}\{(\rho_1\otimes\rho_2\otimes\sigma)S_1\}=\sigma(\rho_1\odot\rho_2). \tag{11}$$

Thus, in summary, we have shown that:

$$\begin{aligned}&\mathrm{tr}_{1,2}\{e^{-\mathrm{i}S_1\delta}(\rho_1\otimes\rho_2\otimes\sigma)e^{\mathrm{i}S_1\delta}\}\\ &\quad=\sigma-\mathrm{i}[(\rho_1\odot\rho_2),\sigma]\delta+\mathcal{O}(\delta^2).\end{aligned} \tag{12}$$

The above is equivalent to applying the unitary $\exp[-\mathrm{i}(\rho_1\odot\rho_2)\delta]$ to σ up to $\mathcal{O}(\delta)$:

$$\begin{aligned}&\exp[-\mathrm{i}(\rho_1\odot\rho_2)\delta]\sigma\exp[\mathrm{i}(\rho_1\odot\rho_2)\delta]\\ &\quad=[\mathbb{1}-\mathrm{i}(\rho_1\odot\rho_2)\delta+O(\delta^2)]\,\sigma\,[\mathbb{1}+\mathrm{i}(\rho_1\odot\rho_2)\delta+O(\delta^2)]\\ &\quad=\sigma-\mathrm{i}[(\rho_1\odot\rho_2),\sigma]\delta+\mathcal{O}(\delta^2).\end{aligned} \tag{13}$$

The above completes the derivation of Eq. 6. Note that if the small time parameter is taken to be $\delta=\epsilon/t$, and the above procedure is implemented $\mathcal{O}(t^2/\epsilon)$ times, the overall effect amounts to implementing the desired operation, $e^{-\mathrm{i}\rho t}\sigma e^{\mathrm{i}\rho t}$ up to an error $\mathcal{O}(\delta^2 t^2/\epsilon) = \mathcal{O}(\epsilon)$, while consuming $\mathcal{O}(t^2/\epsilon)$ copies of ρ_1 and ρ_2. This concludes the proof of Lemma 1. □

Lemma 2 *Given $\mathcal{O}(t^2/\epsilon)$ copies of d-dimensional qubit density matrices, ρ_1 and ρ_2, let $\rho_1 \oslash \rho_2$ denote the outer product between the diagonal entries of ρ_1 and ρ_2. There exists a quantum algorithm to implement the unitary*

$e^{-\mathrm{i}\rho_1 \oslash \rho_2 t}$ *on a d-dimensional qubit input state,* σ *for a time* t *to accuracy* ϵ *in operator norm.*

Proof By simply re-indexing the S_1 operator, one obtains $S_2 = \sum_{j,k} |j\rangle\langle j| \otimes |k\rangle\langle k| \otimes |k\rangle\langle j|$. Analogously with the proof of Lemma 1, we have:

$$\begin{aligned}&\mathrm{tr}_{1,2}\{e^{-\mathrm{i}S_2\delta}(\rho_1 \otimes \rho_2 \otimes \sigma)e^{\mathrm{i}S_1\delta}\}\\ &\quad = \sigma - \mathrm{i}[(\rho_1 \oslash \rho_2), \sigma]\delta + O(\delta^2). \end{aligned} \tag{14}$$

The above can be compared with:

$$\begin{aligned}&\exp[-\mathrm{i}(\rho_1 \oslash \rho_2)\delta]\,\sigma\,\exp[\mathrm{i}(\rho_1 \oslash \rho_2)\delta]\\ &\quad = \sigma - \mathrm{i}[(\rho_1 \oslash \rho_2), \sigma]\delta + O(\delta^2). \end{aligned} \tag{15}$$

The equivalence up to the linear term in δ confirms the validity of Eq. 7. Similarly with Lemma 1, with a $\mathcal{O}(t^2/\epsilon)$ repetition consuming $\mathcal{O}(t^2/\epsilon)$ copies of ρ_1 and ρ_2, the desired $e^{-\mathrm{i}\rho t}\sigma e^{\mathrm{i}\rho t}$ can be implemented up to error ϵ. □

Given the density matrix $\rho = K^0$ which encodes the base case covariance matrix, we approximate the nonlinear kernel function at lth layer with the order N polynomial, $P^N_{(\odot,\oslash)}(\rho) = \sum_r^N c_r \rho^{(\odot,\oslash)r}$. Here, the label $(\odot, \oslash)$ indicates that we work in the setting where the types of product operation involved for taking the r^{th} power of ρ are arbitrary combinations of Hadamard and diagonal outer products. Now, we are in the position of presenting the main theorem required to implement the kernel function at the lth layer.

Theorem 1 *Given* $\mathcal{O}(N^2t^2/\epsilon)$ *copies of the d-dimensional qubit density matrix* ρ*, and the order-N polynomial of Hadamard and diagonal outer products,*

$$P^N_{(\odot,\oslash)}(\rho) = \sum_r^N c_r \rho^{(\odot,\oslash)r},$$

there exists a quantum algorithm to implement the unitary $e^{-\mathrm{i}P^N_{(\odot,\oslash)}(\rho)t}$ *on a d-dimensional qubit input state* σ *for a time* t *to accuracy* ϵ *in operator norm.*

Proof We first address how to implement the unitary $e^{-\mathrm{i}\rho^{(\odot,\oslash)r}t}$. Intuitively, this can be achieved by constructing a generalized $\tilde{S}$ operator with tensor product components of $|j\rangle\langle j|$, $|j\rangle\langle k|$, $|k\rangle\langle k|$, and $|k\rangle\langle j|$, corresponding to the contributing elements in the matrices in each term. We give a recursive procedure to determine $\tilde{S}$:

In the case of $r = 2$, we have already shown in Lemma 1 and Lemma 2 that the desired operation can be achieved using S_1 and S_2 corresponding to the $\odot$ and $\oslash$ cases respectively. Thus, we can write the base case of the recursive procedure as:

$$\tilde{S}^{(r=2)} = \sum_{j,k} T^{(2)}(j,k) \otimes |k\rangle\langle j|,$$

where $T^{(2)}(j,k)$ denotes the possible combinations of tensor products, $|j\rangle\langle k| \otimes |j\rangle\langle k|$ or $|j\rangle\langle j| \otimes |k\rangle\langle k|$. Now, considering the $r = 3$ case, the additional factor of ρ will come in two possible cases. If it comes as a $\odot$ product, the updated operator $\tilde{S}^{(r=3)}_{\odot}$ is simply given by:

$$\tilde{S}^{(r=3)}_{\odot} = \sum_{j,k} T^{(2)}(j,k) \otimes |j\rangle\langle k| \otimes |k\rangle\langle j|.$$

If the additional ρ comes in as a $\oslash$ product, the updated operator $\tilde{S}^{(r=3)}_{\oslash}$ is instead given by:

$$\tilde{S}^{(r=3)}_{\oslash} = \sum_{j,k} |j\rangle\langle j| \otimes |j\rangle\langle j| \otimes |k\rangle\langle k| \otimes |k\rangle\langle j|.$$

This can be seen by observing that the contributing elements to a $\oslash$ product are exclusively diagonal, which we use $|j\rangle\langle j|$ to pick up. Any off-diagonal information about the previous element-wise product operations is irrelevant. In general, if we have the r^{th} order $\tilde{S}$ operator given by:

$$\tilde{S}^{(r)} = \sum_{j,k} T^{(r)}(j,k) \otimes |k\rangle\langle j|,$$

the operators $\tilde{S}^{(r+1)}_{\odot}$ and $\tilde{S}^{(r+1)}_{\oslash}$ can be generated as follows:

$$\begin{aligned}\tilde{S}^{(r+1)}_{\odot} &= \sum_{j,k} T^{(r)}(j,k) \otimes |j\rangle\langle k| \otimes |k\rangle\langle j|,\\ \tilde{S}^{(r+1)}_{\oslash} &= \sum_{j,k} (|j\rangle\langle j|)^{\otimes r} \otimes |k\rangle\langle k| \otimes |k\rangle\langle j|. \end{aligned} \tag{16}$$

We have shown a recursive procedure to construct $\tilde{S}^{(r)}$ up to $r = N$ such that:

$$\begin{aligned}&\mathrm{tr}_{1\ldots r}\{e^{-\mathrm{i}\tilde{S}^{(r)}\delta}(\rho^{\otimes r} \otimes \sigma)e^{\mathrm{i}\tilde{S}^{(r)}\delta}\}\\ &\quad = \exp[-\mathrm{i}\rho^{(\odot,\oslash)r}\delta]\,\sigma\,\exp[\mathrm{i}\rho^{(\odot,\oslash)r}\delta] + \mathcal{O}(\delta^2), \end{aligned} \tag{17}$$

for a small evolution δ. Analogously with Lemma 1 and Lemma 2, with a $\mathcal{O}(t^2/\epsilon)$ repetition consuming $\mathcal{O}(rt^2/\epsilon)$ copies of ρ, the desired

$$\exp[-\mathrm{i}\rho^{(\odot,\oslash)r}t]\,\sigma\,\exp[\mathrm{i}\rho^{(\odot,\oslash)r}t]$$

can be implemented up to an ϵ error.

Finally, one makes use of the Lie product formula for summing the terms in the polynomial (Suzuki 1992; Childs et al. 2003; Wiebe et al. 2010):

$$e^{\mathrm{i}\delta(A+B)+\mathcal{O}(\delta^2/m)} = (e^{\mathrm{i}\delta A/m}e^{\mathrm{i}\delta B/m})^m, \tag{18}$$

where A and B are taken to different terms in $P^N_{(\odot,\oslash)}(\rho) = \sum_r^N c_r \rho^{(\odot,\oslash)r}$, and the factors c_r simply amount to multiplying the $S^{(r)}$ matrices with the respective coefficients. The parameter m can be chosen to further suppress the error by repeating the entire procedure. However, for the purpose of implementing:

$$e^{-\mathrm{i}P^N_{(\odot,\oslash)}(\rho)t}\sigma e^{\mathrm{i}P^N_{(\odot,\oslash)}(\rho)t}$$

to our desired accuracy ϵ, $\mathcal{O}(N^2t^2/\epsilon)$ copies of ρ are required. The quadratic dependency in the order of the polynomial, N^2,

stems from implementing the unitary $\exp[-i\rho^{(\odot,\oslash)r}t]$ up to $r=N$, each consuming $\mathcal{O}(Nt^2/\epsilon)$ copies as argued before. □

4 Experiments

The central part of the algorithm described in Section 3 is the intricate quantum protocol of matrix inversion for computing the predictors in Eqs. 2, 3. This protocol (Harrow et al. 2009) is probabilistic, meaning that it only succeeds conditioned on obtaining specific results after measuring specific qubits in the protocol. Therefore, it is not assured that the protocol will succeed in a particular run, and it has to be repeatedly performed until it succeeds in obtaining the correct solution. Moreover, computations on real quantum computers are subject to imprecisions in the gates applied to the qubits, readout errors, and losses of coherence in the state of the system.

Therefore, when thinking about a realistic application of the quantum Bayesian algorithm, the important questions to ask are how experimentally feasible it is, and how far we are from running it on real quantum computers. With this goal in mind, we have performed two sets of experiments: on the one hand, we have run simulations of the quantum matrix inversion protocol on two different quantum virtual machines with various noise models that affect real quantum computers, and analyzed their impact on the output—the final quantum state after the protocol—of the algorithm. On the other hand, we have run scaled-down versions of the protocol on two real, state-of-the-art quantum processing units to gauge how far we are from implementations of practical relevance.

We have implemented the complete quantum matrix inversion protocol in the Rigetti Forest API using PyQuil and Grove (Smith et al. 2016). This implementation can perform approximate eigenvalue inversion on a Hermitian matrix of arbitrary size. The PyQuil framework has advanced gate decomposition features and provides a way to perform arbitrary unitary operations on a multi-qubit quantum state. Furthermore, Rigetti's classical simulator of quantum circuits (referred to as a *quantum virtual machine*) provides a variety of noise models that can affect computations in real quantum architectures, allowing a detailed analysis of how noise affects accuracy and computational overhead.

In addition, we have implemented reduced, 2×2 matrix inversion problems in both PyQuil—to be run in Rigetti's Quantum Processing Unit—and in IBM's QISKit software stack (Cross et al. 2017)—to be run in IBM's Quantum Experience computers. QISKit also provides a noisy classical simulator, of which we also make use to contrast the performance of the quantum matrix inversion algorithm run in the real QPUs against simulations with realistic noise models.

The quantum processing units employed in the experiments are IBM's 16-qubit Rueschlikon (IBMQX5) (Wang et al. 2018) and Rigetti's 8-qubit 8Q-Agave. While the number of available physical qubits is in both cases higher than the number of qubits required for the implementation (a total of six for the 2×2 reduced version), the depth of the circuit is much higher for larger matrices, and the current noise levels in the QPUs would not allow obtaining meaningful results when inverting larger examples.

4.1 Simulations of algorithm success on a quantum virtual machine

In this section, we report the results of the simulations conducted in Rigetti's quantum virtual machine. We have conducted two sets of experiments to analyze the sensitivity of the protocol to different noise types that appear in real quantum computers. In the first, we restrict ourselves to the simplest possible scenario of inverting the 2×2 matrix $A = \frac{1}{2}\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$ with the problem-specific circuit in Cao et al. (2012). This circuit is much shallower than the full protocol detailed in Cao et al. (2013), making it more realistic to implement on current and near-future quantum computers due to its reduced depth. The second case is the complete implementation of the full quantum matrix inversion protocol (Harrow et al. 2009; Cao et al. 2013). This version requires a large number of ancilla qubits to perform the calculations, in particular for the computation of the reciprocals of the eigenvalues. We choose to simulate the inversion of a 4×4 matrix with four bits of precision, which is the largest example that could fit on the largest Rigetti QPU.

We have studied the impact of two noise models, both being instances of parametric depolarizing noise. The first one, known as *gate noise*, applies a Pauli X operator—which swaps the states $|0\rangle$ and $|1\rangle$ of the qubit it acts upon—with a certain probability on each qubit after *every* gate application. The probability of application of the operator indicates the noise level. The second type of noise that we study is known as *measurement noise*. In this case, a Pauli X operator is applied with certain probability only on every qubit that is measured, before the measurement takes place. Therefore, it can also be understood as a readout error that, with a certain probability, instead of recording the result of a measurement, y, it records $NOT(y)$.

The circuits we implement have a much larger number of gates ($\sim$20 for the 2×2 reduced version, increasing for the increasing size of the matrix being inverted) than measurements (just one, that which certifies the success of the eigenvalue inversion). This is the reason why in all the experiments we run we observe that the gate noise has a stronger impact on the results than the measurement noise.

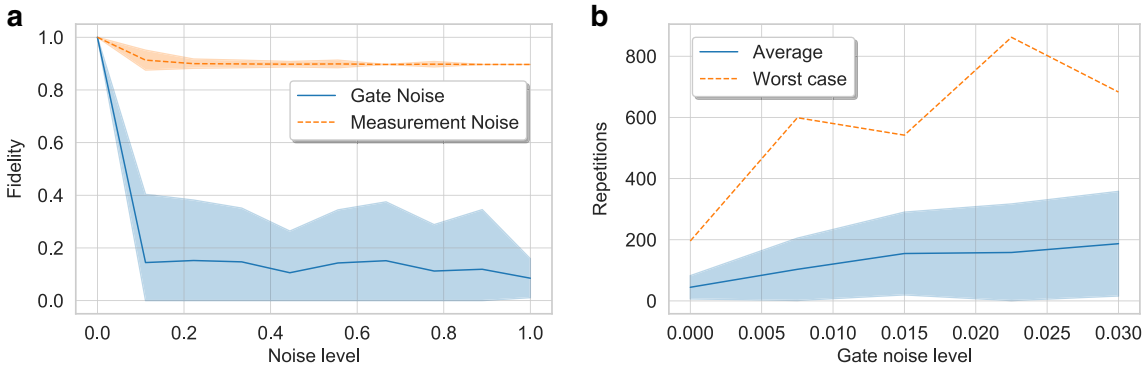

Fig. 1 Simulated gate and measurement noise on a specialized circuit for inverting the 2×2 matrix A, run in Rigetti's quantum virtual machine. **a** The fidelity shows the overlap with the expected correct state after the computation. A fidelity of zero means that the output state (and hence the result of the computation) is completely orthogonal to the correct solution, while a fidelity of one means that the output state coincides with the expected one. **b** The number of repetitions expresses the average of how many times the probabilistic program is executed before it succeeds. We define a successful run with two conditions: the qubit in which the controlled rotation is performed is in the state $|1\rangle$ after its measurement, and the final state of the qubits has a fidelity greater than 0.9 with the expected outcome of an ideal run of the protocol. The dashed line represents the maximum number of runs observed in the simulations before having a successful one

In Fig. 1, we show the results for the inversion of the 2×2 matrix A. We analyze the two critical factors of the protocol, namely how different are the expected result of the protocol and the output from the simulator when we know that the inversion has succeeded, and how many times it is needed to run the protocol in order to obtain a successful run. As expected, the measurement noise has a much smaller impact on the result than the gate noise, which for reasonably low noise levels already renders the output state (and hence the result of the inversion) with low overlap with the expected result.

The number of repetitions needed for the algorithm to succeed, understood as the average number of times the algorithm must be run in order to obtain the outcome associated to the state $|1\rangle$ when measuring the qubit to which the conditional rotation is applied, is a fragile quantity that, on its own, does not provide meaningful insights when dealing with noise. In the case of measurement noise, an error in the measurement either discards a successful run of the algorithm or accepts as successful a failed run, deeming further computations useless. In the case of gate noise, even in the case the measurement succeeds and therefore the state of the flag qubit is $|1\rangle$, the remaining computations on the other qubits may lead to a final state that deviates from the expected result.

In order to obtain a good estimation of the number of runs needed to detect a final state that encodes the desired solution, in Fig. 1b, we show the number of repetitions of the algorithm needed in order to have a successful run according to the flag qubit (i.e., that its state after the measurement is $|1\rangle$; Harrow et al. 2009), in which the overlap of the final state and the desired state is higher than a specific value. We measure such an overlap with the fidelity, given by $\mathcal{F} = |\langle\psi_{\text{real}}|\psi_{\text{ideal}}\rangle|^2$, where $|\psi_{\text{real}}\rangle$ and $|\psi_{\text{ideal}}\rangle$ determine the state of the qubits after a noisy simulation and a noiseless successful run, respectively.

Given that the protocol is probabilistic, the number of repetitions needed to have a successful run is dependent

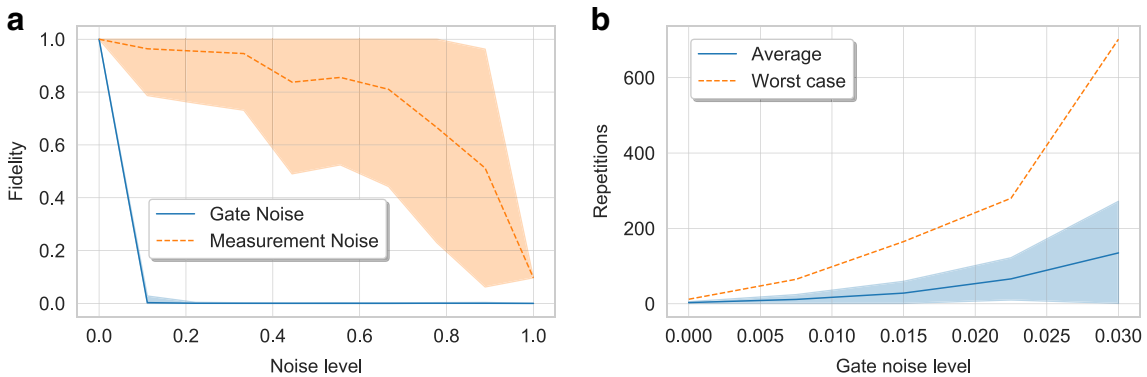

Fig. 2 Simulated gate and measurement noise on the generic circuit for inverting a matrix. The matrix in the benchmark was 4×4, and the eigenvalues were represented by four bits of precision. Together with the ancilla qubits in the calculations, this is the largest system that can be simulated with less than 19 qubits, which is the size of Rigetti's largest QPU (19Q-Acorn)

on the actual matrix to be inverted even in the case of a noiseless run, as can be observed by comparing Figs. 1b and 2b, and grows fast with the gate noise level in the qubits. It is important to track not only the average behavior of the protocol (in solid blue in Fig. 1b), but also worst-case scenarios (in dashed orange) where the protocol must be run up to more than five times than average in order to have a successful execution. Nevertheless, worst-case performance scales with the noise level in a similar way as the average performance.

In Fig. 2, we perform the same studies for the implementation of the general algorithm inverting a random 4×4 matrix. It is immediately apparent that increasing the circuit depth makes the protocol more sensitive to noise, and the fidelity drops to zero with lower variance in the case of the gate noise. However, the noise level for which the fidelity of the output of the circuit with the expected state drops abruptly is approximately equal in both the 2×2 and 4×4 cases, and it would be interesting to see whether it remains constant for larger problems. We still observe better robustness to measurement noise, but the impact of this kind of noise in the resulting state is stronger than in the problem-specific algorithm of Fig. 1. The number of repetitions for a successful run now has a nonlinear behavior with the level of gate noise in the simulation, although the ratio of the worst-case scenario to the average is the same as in the case of inverting the 2×2 matrix.

4.2 Evaluation on quantum processing units

In this section, we implement the restricted 2×2-matrix inversion algorithm in two real quantum computing architectures. The reason of choosing the restricted algorithm is that current quantum computers have a small number of qubits, limited qubit-qubit connectivity, and most importantly, short coherence times, which implies that only shallow quantum circuits can be implemented. The restricted algorithm can be implemented with a much simpler circuit than the general one, resulting in about 20 gates for the full protocol (Cao et al. 2012).

In the case of runs on real QPUs, one does not have direct access to the whole output state of the circuit, but only to samples of measurements on it. This makes it difficult to compute the fidelity with the expected state, and instead, we perform a *swap test* (Gottesman and Chuang 2001). The test runs as follows: the expected result of the algorithm is encoded in auxiliary qubits, and after operations between the output and the expected result, a flag qubit indicates whether both states are equal, in which case the state of the flag qubit is $|0\rangle$, or not, in which case the state is $|1\rangle$. The figure of merit is now the probability of success in the test $P(success) = P(0)$, which can then be related to the fidelity by the expression $\mathcal{F} = |2P(success) - 1|$. Note that this success probability is different from the probability that the eigenvalue inversion subroutine succeeds, which is the quantity that has already been studied in Figs. 1b and 2b.

We have implemented the protocol to be run in both Rigetti's 8Q-Agave and IBM's IBMQX5 quantum processing units. The IBM QISKit software (Cross et al. 2017) also provides a classical simulator to run noisy experiments, and we use these to benchmark the performance of the runs on the real chips. The results of the experiments can be found in Fig. 3.

As in the case of the simulations in Rigetti's software stack, the measurement noise produces a smaller impact on the protocol than the gate noise. Note that the qubit that encodes the success of the swap test is also subject to readout error when simulating measurement noise.

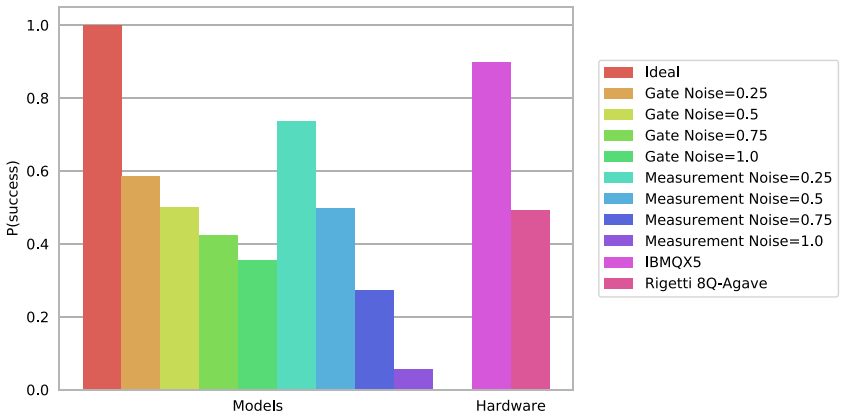

Fig. 3 Probability of success of the swap test (i.e., the probability of the circuit result being the desired) after success in the eigenvalue inversion subroutine, for different classical noisy simulations, and executions on the IBM and Rigetti quantum processing units (rightmost bars). The noise models involve faulty gate operations—gate noise—and faulty readout errors—measurement noise—with different probabilities of failure. The algorithm is run 8192 times for each instance, after which $P(success)$ is computed

Therefore, for large measurement noise levels, the fact that $P(success) = P(0)$ is very low means that the actual state of the flag qubit is $|0\rangle$ (i.e., the protocol has succeeded, and the output state is the desired one), but due to the noise, the result that is recorded after measuring is 1.

Gate noise has a stronger impact in the final state. This kind of error, unlike the measurement noise, does affect the computations in the circuit, so lower success probabilities now represent a real discrepancy between the output and desired states. In this case, the success probabilities lie in the range of [0.35, 0.6], which translates into fidelities in the range of [0, 0.3].

Turning to executions in the real QPUs, the probability of protocol success is higher in IBMQX5. This is mostly due to its improved coherence time,[2,3] that allows keeping the state in the circuit better isolated from external perturbations during computation. The probability of protocol success is 89%, which translates into a fidelity with the expected state of 0.78. This is a very encouraging result, despite the size of the matrix inverted. In contrast, the fidelity when the protocol is run in 8Q-Agave is close to 0, which means that all the information about the computation is lost during the process. This is mainly due to the circuit depth being too large to maintain the quantum state isolated enough from the environment.

5 Conclusions

As quantum computers become available and continue improving in scale and noise tolerance, it is an exciting question to ask whether they can make a qualitative difference in machine learning applications. Seminal works that explored this question focused on idealized, fully noise-tolerant, large-scale quantum computers, and implemented simple machine learning algorithms like support vector machines and nearest-neighbor clustering. However, an important fact is that, for at least the next decade, quantum computers will remain limited in scale and noise tolerance, and we must factor this in when we construct quantum-enhanced algorithms. Furthermore, simple machine learning methods are already efficiently executed on classical hardware, so there is no need for the use of quantum algorithms in this case.

In this work, we studied a complex, Bayesian approach to deep architectures that is difficult to perform on digital hardware. We developed a quantum algorithm for learning Gaussian processes that can be applied layer by layer for training arbitrarily deep neural networks. Furthermore, our protocol is a classical-quantum hybrid that largely removes the currently unrealistic technological requirements, such as a quantum random access memory. The algorithm makes use of the quantum matrix inversion protocol which, albeit intricate, its mathematical assumptions are fulfilled by the kernel matrices originating from Gaussian processes. In order to analyze the feasibility of a real use of the algorithm, we implemented its core routine, the quantum matrix inversion protocol, to be run in both quantum simulators and real state-of-the-art quantum processors. We observe that the accuracy of the protocol sharply drops with noise, but even with current, small quantum computers, high success rates can be achieved.

Although promising, these experimental results do not completely prove that the full protocol will be efficiently implementable in near-term quantum technologies. Full implementation in architectures with limited coherence time and sparse connectivity, as well as a fully coherent variant of the training algorithm (which would have important applications in quantum simulations and quantum control), are interesting avenues for future research.

Not only are commercial quantum computers proliferating, but also the tools to program them, and, even more importantly, the collection of high-level algorithmic primitives (Coles et al. 2018). This enables machine learning researchers to leverage quantum technologies without the need of having an extensive background in quantum technologies. Just as GPUs and efficient frameworks like TensorFlow (Abadi et al. 2016) and PyTorch (Paszke et al. 2017) created an enormous community researching and deploying deep learning, we expect the same phenomenon will happen in the future with quantum processing units and collections of quantum algorithms.

Acknowledgements We would like to thank Piotr Gawron (Polish Academy of Sciences), Will Zeng and Ryan Karle (Rigetti Computing), and Joseph Fitzsimons (SUTD and CQT) for discussions.

Funding information Z. Z. received support from Singapore's Ministry of Education and National Research Foundation under NRF Award NRF-NRFF2013-01. The work of A. P.-K. is supported by Fundación Obra Social "la Caixa" (LCF/BQ/ES15/10360001), the Spanish MINECO (QIBEQI FIS2016-80773-P and Severo Ochoa SEV-2015-0522), Fundació Privada Cellex, and the Generalitat de Catalunya (SGR1381 and CERCA Program). This research was supported by Perimeter Institute for Theoretical Physics. Research at Perimeter Institute is supported by the Government of Canada through Industry Canada and by the Province of Ontario through the Ministry of Economic Development and Innovation.

[2]Information about performance measures of Rigetti's QPUs can be found in http://docs.rigetti.com/en/1.9/qpu.html.

[3]Information about performance measures of IBM's QPUs can be found in http://www.research.ibm.com/ibm-q/technology/devices/.

References

Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray DG, Steiner B, Tucker P, Vasudevan V, Warden P, Wicke M, Yu Y, Zheng X (2016) Proceedings of the 12th USENIX conference on operating systems design and implementation

Arjovsky M, Shah A, Bengio Y (2015) arXiv:1511.06464

Berry DW, Childs AM (2012) Quantum Inf Comput 12(1–2):29. https://doi.org/10.26421/QIC12.1-2

Berry DW, Childs AM, Kothari R (2015) In: Proceedings of FOCS-15, 56th annual symposium on foundations of computer science, pp 792–809. https://doi.org/10.1109/FOCS.2015.54

Biamonte J, Wittek P, Pancotti N, Rebentrost P, Wiebe N, Lloyd S (2017) Nature 549(7671):195–202. https://doi.org/10.1038/nature23474

Blundell C, Cornebise J, Kavukcuoglu K, Wierstra D (2015) In: Proceedings of the 32nd international conference on international conference on machine learning - volume 37 (JMLR.org), ICML'15, pp 1613–1622

Bradshaw J, Matthews AGDG, Ghahramani Z (2017) arXiv:1707.02476

Cao Y, Daskin A, Frankel S, Kais S (2012) Mol Phys 110(15-16):1675–1680. https://doi.org/10.1080/00268976.2012.668289

Cao Y, Papageorgiou A, Petras I, Traub J, Kais S (2013) New J Phys 15(1):013021. https://doi.org/10.1088/1367-2630/15/1/013021

Childs AM (2010) Commun Math Phys 294(2):581. https://doi.org/10.1007/s00220-009-0930-1

Childs AM, Cleve R, Deotto E, Farhi E, Gutmann S, Spielman DA (2003) In: Proceedings of STOC-03, 35th annual ACM Symposium on Theory of computing, pp 59–68. https://doi.org/10.1145/780542.780552

Cho Y, Saul LK (2009) In: Advances in neural information processing systems, pp 342–350

Coles PJ, Eidenbenz S, Pakin S, Adedoyin A, Ambrosiano J, Anisimov P, Casper W, Chennupati G, Coffrin C, Djidjev H, Gunter D, Karra S, Lemons N, Lin S, Lokhov A, Malyzhenkov A, Mascarenas D, Mniszewski S, Nadiga B, O'Malley D, Oyen D, Prasad L, Roberts R, Romero P, Santhi N, Sinitsyn N, Swart P, Vuffray M, Wendelberger J, Yoon B, Zamora R, Zhu W (2018) arXiv:1804.03719

Cross AW, Bishop LS, Smolin JA, Gambetta JM (2017) arXiv:1707.03429

Daniely A, Frostig R, Singer Y (2016) arXiv:1602.05897

Farhi E, Neven H (2018) arXiv:1802.06002

Furrer R, Genton MG, Nychka D (2006) J Comput Graph Stat 15(3):502. https://doi.org/10.1198/106186006x132178

Gal Y, Ghahramani Z (2016) In: Balcan M. F., Weinberger K. Q. (eds) Proceedings of the 33rd international conference on machine learning, proceedings of machine learning research, vol 48. (PMLR, New York, New York, USA), Proceedings of Machine Learning Research, vol 48, pp 1050–1059

Ghahramani Z (2015) Nature 521(7553):452–459. https://doi.org/10.1038/nature14541

Glorot X, Bordes A, Bengio Y (2011) In: Gordon G., Dunson D., Dudík M (eds) Proceedings of the 14th international conference on artificial intelligence and statistics, proceedings of machine learning research, vol 15, pp 315–323. (PMLR, Fort Lauderdale, FL, USA), Proceedings of Machine Learning Research

Gottesman D, Chuang I (2001) arXiv:quant-ph/0105032

Grosse K, Pfaff D, Smith MT, Backes M (2017) arXiv:1711.06598

Harrow AW, Hassidim A, Lloyd S (2009) Phys Rev Lett 103:150502. https://doi.org/10.1103/PhysRevLett.103.150502

Hyland S, Rätsch G (2017) In: AAAI conference on artificial intelligence

Khoshaman A, Vinci W, Denis B, Andriyash E, Amin MH (2018) arXiv:1802.05779

Kimmel S, Lin CYY, Low GH, Ozols M, Yoder TJ (2017) npj Quantum Inf 3(1):13. https://doi.org/10.1038/s41534-017-0013-7

Kitaev AY (1995) arXiv:quant-ph/9511026

Lee J, Sohl-Dickstein J, Pennington J, Novak R, Schoenholz S, Bahri Y (2018) In: International conference on learning representations

Liu D, Ran SJ, Wittek P, Peng C, García RB, Su G, Lewenstein M (2017) arXiv:1710.04833

Lloyd S (1996) Science 273(5278):1073–1078. https://doi.org/10.1126/science.273.5278.1073

Matthews AGdeG, Rowland M, Hron J, Turner RE, Ghahramani Z (2018) Gaussian process behaviour in wide deep neural networks. In: Proceedings of the 6th international conference on learning representations. arXiv:1804.11271

Neal RM (1994) Priors for infinite networks. Tech. Rep. crg-tr-94-1 University of Toronto

Nielsen MA, Chuang IL (2000) Quantum computation and quantum information. Cambridge University Press, Cambridge

Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in Pytorch. In: Workshop Proceedings of the 31st conference on neural information processing systems

Rasmussen CE, Williams CKI (2006) Gaussian processes for machine learning. MIT press, Cambridge

Rebentrost P, Mohseni M, Lloyd S (2014) Phys Rev Lett 113:130503. https://doi.org/10.1103/PhysRevLett.113.130503

Rebentrost P, Schuld M, Wossnig L, Petruccione F, Lloyd S (2016) arXiv:1612.01789

Rebentrost P, Steffens A, Marvian I, Lloyd S (2018) Phys Rev A 97(1):012327. https://doi.org/10.1103/PhysRevA.97.012327

Schuld M, Killoran N (2018) Phys Rev Lett 101103:122. https://doi.org/10.1103/PhysRevLett.122.040504

Schuld M, Sinayskiy I, Petruccione F (2014) Quantum Inf Process 13(11):2567. https://doi.org/10.1007/s11128-014-0809-8

Smith RS, Curtis MJ, Zeng WJ (2016) arXiv:1608.03355

Stoudenmire EM (2018) Quantum Sci Technol 3(3):034003. https://doi.org/10.1088/2058-9565/aaba1a

Suzuki M (1992) Phys Lett A 165(5-6):387. https://doi.org/10.1016/0375-9601(92)90335-J

Tacchino F, Macchiavello C, Gerace D, Bajoni D (2018) arXiv:1811.02266v1

Torrontegui E, Garcia-Ripoll JJ (2018) arXiv:1801.00934

Trabelsi C, Bilaniuk O, Zhang Y, Serdyuk D, Subramanian S, Santos JF, Mehri S, Rostamzadeh N, Bengio Y, Pal CJ (2017) arXiv:1705.09792

Verdon G, Broughton M, Biamonte J (2017) arXiv:1712.05304

Verdon G, Pye J, Broughton M (2018) arXiv:1806.09729

Wang Y, Li Y, Yin ZQ, Zeng B (2018) npj Quantum Inf 4(1):46. https://doi.org/10.1038/s41534-018-0095-x

Wiebe N, Berry D, Høyer P, Sanders BC (2010) J Phys A Math Theor 43(6):065203. https://doi.org/10.1088/1751-8113/43/6/065203

Wittek P, Tan CL (2011) Trans Pattern Anal Mach Intell 33(10):2039–2050. https://doi.org/10.1109/TPAMI.2011.28

Wossnig L, Zhao Z, Prakash A (2018) Phys Rev Lett 120:050502. https://doi.org/10.1103/PhysRevLett.120.050502

Zhao Z, Fitzsimons JK, Osborne MA, Roberts SJ, Fitzsimons JF (2018) arXiv:1803.10520

Zhao Z, Fitzsimons JK, Fitzsimons JF (2015) arXiv:1512.03929