Quantum algorithms for training Gaussian processes

Zhikuan Zhao, Jack K. Fitzsimons, Michael A. Osborne, Stephen J. Roberts, and Joseph F. Fitzsimons

Phys. Rev. A **100**, 012304 — Published 8 July 2019

DOI: 10.1103/PhysRevA.100.012304

Quantum algorithms for training Gaussian Processes

Zhikuan Zhao,[1,2,3,*] Jack K. Fitzsimons,[4] Michael A. Osborne,[4] Stephen J. Roberts,[4] and Joseph F. Fitzsimons[2,3,5,†]

[1]*Department of Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zürich*
[2]*Singapore University of Technology and Design, 8 Somapah Road, Singapore 487372*
[3]*Centre for Quantum Technologies, National University of Singapore, 3 Science Drive 2, Singapore 117543*
[4]*Department of Engineering Science, University of Oxford, Oxford OX1 3PJ, UK*
[5]*Horizon Quantum Computing, 79 Ayer Rajah Crescent, Singapore 139955*

Gaussian processes (GPs) are important models in supervised machine learning. Training in Gaussian processes refers to selecting the covariance functions and the associated parameters in order to improve the outcome of predictions, the core of which amounts to evaluating the logarithm of the marginal likelihood (LML) of a given model. LML gives a concrete measure of the quality of prediction that a GP model is expected to achieve. The classical computation of LML typically carries a polynomial time overhead with respect to the input size. We propose a quantum algorithm that computes the logarithm of the determinant of a Hermitian matrix, which runs in logarithmic time for sparse matrices. This is applied in conjunction with a variant of the quantum linear system algorithm that allows for logarithmic time computation of the form $\mathbf{y}^T A^{-1}\mathbf{y}$, where $\mathbf{y}$ is a dense vector and A is the covariance matrix. We hence show that quantum computing can be used to estimate the LML of a GP with exponentially improved efficiency under certain conditions.

I. INTRODUCTION

The goal of supervised machine learning is to infer a function from a labelled set of input and output example points, known as the training data [1]. Gaussian processes (GPs) represent an approach to supervised learning that models the underlying functions associated with the outputs in an inference problem as an infinite-dimensional generalisation of a Gaussian distribution [2].

A limiting factor for many applications of machine learning is the computational complexity of the underlying algorithms. This can prove prohibitive when working with large datasets. Machine learning has naturally arisen as a potential application for quantum computation. In recent years, significant progress has been made on quantum algorithms for machine learning tasks from several directions [3]. Quantum algorithms have been proposed for many of the operations commonly used in machine learning, including matrix inversion [4], principal component analysis [5] and clustering [6, 7]. Quantum algorithms for support vector machines [8], accelerated deep learning [9] and pattern classification based on linear regression [10] have also emerged, and the use of quantum annealers and adiabatic quantum computation has begun to show promise for machine learning tasks [11–13]. More recent advances in the field are well summarized in [14]. Along the same line of effort in applying quantum algorithms to machine learning, a quantum algorithm for Gaussian process regression (GPR) has previously been presented in [15]. The proposed Quantum Gaussian Process Algorithm (QGPA) leverages the exponential speed-up achievable in the quantum linear system algorithm [4], and estimates the mean and variance of the predictive distribution given by the GP. Comparing to many aforementioned approaches, GPs offer a learning model with several desirable traits, including the ability to naturally quantify the uncertainty associated to a prediction, the suitability for capturing a wide range of behaviours governed only by a simple set of parameters and admitting an intuitive Bayesian interpretation. As a result, GP models have played significant roles across a broad domain of applications, ranging from climate modelling, astrophysics, geostatistics, robotic engineering to price prediction in the financial markets. Moreover, a novel connection between GPs and deep neural networks was recently revealed in [16] and later used to formulate quantum deep learning without the need of back-propagation [17]. In this work, we complete the quantum Gaussian process procedure by providing a similarly efficient quantum routine to select the covariance functions and the associated parameters necessary to train a GP model.

An important aspect of many supervised learning approaches amounts to the ability to efficiently select preferred variations of the model, in order to achieve better predictions. A significant amount of attention in machine learning research has rightfully been devoted to this process of model selection which is a somewhat under explored topic in the existing quantum machine learning literature. In the context of GPs, this amounts to choosing a covariance function, known as the GP kernel. In practice, a family of functions are usually considered. The parameters of the family of kernels are referred to as kernel hyper-parameters. For instance, if given a Gaussian kernel of the form $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$, the hyper-parameter is γ; if given a Gaussian Spectral Kernel of the form $k(\mathbf{x}, \mathbf{x}') = \cos(2\pi\mu\|\mathbf{x} - \mathbf{x}'\|)\exp(-2\pi^2\sigma^2\|\mathbf{x} - \mathbf{x}'\|^2)$, the hyper-parameters are $\{\gamma, \sigma\}$. A range of optimisers are used in order to tune these hyper-parameters based on the observed data. This process is commonly known as the training of a Gaussian process.

Since model selection typically involves repeated evaluation of certain cost functions that characterise how well a given model is performing on the problem, it generally carries a runtime overhead that scales polynomially with the input size. As QGPA provides a speed-up in computing predictions given a fixed kernel, it is desirable to also have a corre-

* zhikuan.zhao@inf.ethz.ch
† joe.fitzsimons@nus.edu.sg

spondingly efficient quantum routine for kernel and parameter selection. In particular, it would be desirable to evaluate a measure of the model's performance with a quantum routine that supplements the main learning algorithm. With the above motivation, we propose a quantum approach to improve the efficiency of GP training based on evaluating the logarithm of marginal likelihood (LML) of the Gaussian distribution of the observed data.

II. TRAINING IN GAUSSIAN PROCESSES

We begin by reviewing the basics of Gaussian processes with an emphasis on kernel selection and hyper-parameter tuning. We will then introduce the first algorithm of the paper, namely a quantum log determinant algorithm, which computes the logarithm of the determinant of a Hermitian matrix. The second algorithm builds on the well-known quantum linear system algorithm [4] and allows us to compute $\mathbf{y}^T A^{-1}\mathbf{y}$ for sparse matrix A and dense vector $\mathbf{y}$. In conjunction, the two algorithms can be applied to improve the efficiency of evaluating the LML of a GP model, potentially yielding an exponential improvement in performance. We conclude with a discussion on the potential sources of estimation errors and the practical application of our procedure.

Following the convention of [2], we begin by introducing the fundamentals of GPs and how model selection is typically implemented. Consider a supervised learning problem with a set of training data $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=0}^{n-1}$, which contains n d-dimensional inputs, $\{\mathbf{x}_i\}_{i=0}^{n-1}$, and their corresponding outputs, $\{y_i\}_{i=0}^{n-1}$. We are interested in modelling the underlying function which generated the dataset. We refer to this as the latent function $f(\mathbf{x})$. This latent function is related to the output data by

$$y = f(\mathbf{x}) + \epsilon_{\text{noise}},$$

where $\epsilon_{\text{noise}} \sim \mathcal{N}(0, \sigma_n^2)$ represents independent and identically distributed Gaussian noise, with zero mean and variance σ_n^2. When given a new input, $\mathbf{x}_*$, we aim to have a predictive distribution of $f_* = f(\mathbf{x}_*)$. A GP method models $\{f(\mathbf{x_i})\}_{i=0}^{n-1}$ as a joint multi-dimensional Gaussian distribution, which can be completely specified by a mean function,

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})],$$

and a covariance function (or kernel function),

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))],$$

with a fixed set of hyper-parameters. For the sake of simplicity, we can choose to set the GP model to have zero prior mean. An arbitrary parametric prior mean could be equally well learned by the techniques to follow. We write the predictive distribution of f_*, conditioned on $\{\mathbf{x}_i, y_i\}_{i=0}^{n-1}$ and the target input $\mathbf{x}_*$, as a multi-variable Gaussian distribution:

$$p(f_*|\mathbf{x}_*, \mathcal{T}) \sim \mathcal{N}(\bar{f}_*, \mathbb{V}[f_*]),$$

where $\bar{f}_*$ and $\mathbb{V}[f_*]$ denotes the mean and the variance of the Gaussian distribution respectively.

The aim of training is to achieve a better predictive distribution for a given problem by selecting the form of the covariance function and varying its parameters. The natural heuristic for the suitability of a supervised learning model is the marginal likelihood, $p(\mathbf{y}|K + \sigma_n^2 I)$, where $K \in \mathbb{R}^{n\times n}$ denotes the covariance matrix between the input points, which is by definition symmetric positive semi-definite. We have employed a vectorised notation for the dataset, such that $\mathbf{y} \in \mathbb{R}^n$ contains entries given by the outputs. Intuitively, we wish to tune the covariance function and respective hyper-parameters by maximising the probability of the observed data given the GP prior. Since we have chosen the model to have zero prior mean, we can write down the distribution of $\mathbf{y}$ as $\mathbf{y} \sim \mathcal{N}(0, K+\sigma_n^2 I)$. It follows that the logarithm of marginal likelihood $\text{LML} = \log[p(\mathbf{y}|K + \sigma_n^2 I)]$ is then convenient to compute using the following identity [2]:

$$\begin{aligned}\text{LML} = &-\frac{1}{2}\log\det[K + \sigma_n^2 I] - \frac{1}{2}\mathbf{y}^T(K + \sigma_n^2 I)^{-1}\mathbf{y} \\ &- \frac{n}{2}\log 2\pi. \end{aligned} \tag{1}$$

Note that $\sigma_n^2 I$ is added to account for the fact that although our GP models the latent function as a Gaussian distribution, the output signal is potentially noisy, and hence there is additional variance in the predicted distribution of the output. The first term of LML only depends on the covariance matrix with an identity noise entry, and amounts to a penalty on the complexity of the model. It will disfavour models which overfit the dataset. The second term in LML is the only one which involves the observed output data, and is therefore responsible for indicating how well the model is actually fitting the data. The final term, $-\frac{n}{2}\log 2\pi$ is an easily computable normalisation constant. Therefore only the first two terms in LML requires quantum algorithms for speed-up.

Classically, the runtime in computing LML is dominated by computing matrix multiplication and determinant, and hence scales with the data size as $O(n^3)$ in typical implementations. A possible improvement can be achieved with optimised CW-like algorithms [18], cutting the runtime down to $O(n^{2.373})$, although such scaling is difficult to achieve in practice. Due to this high computational cost, a number of approximate methods have arisen within the machine learning community. Very often GPs are constrained to a have fixed rank in order to make the computation more manageable. In these cases a range of methods can be used to reduce the complexity of training to $O(nr^2)$, where r is the rank of the covariance matrix [19]. Unfortunately, this limits the complexity of the functions which can be modelled by the GP and ultimately hinders the model's predictive performance. Other approaches, such as hierarchical matrix factorisation [20], work well in low-dimensional spaces but do not scale well to high dimensional datasets which are often of interest.

More recently, stochastic trace estimation approaches have become popular [21, 22]. These methods utilise the equality relationship between the log determinant of a matrix and the trace of the log of the matrix. Using this relationship, the logarithm of a matrix is approximated either by truncating the Taylor series of the matrix logarithm, or by approx-

imating it using a Chebyshev polynomial approximation of some user specified degree d. The advantage of a trace estimation approach is that raising A, or $(I - A)$ for that matter, still requires matrix multiplication but the speed-up arises as the product $z^\dagger \log(A) z$ can be computed in $O(n^2)$ for some $z \in \mathbb{R}^n$. As such the 'probing vectors', z, are chosen such that $\mathbb{E}[z^\dagger \log(A) z] = \mathrm{Tr}(\log(A))$ and can be done so in a number of ways [23, 24]. Note that there are two sources of error which occur in such an approach, namely due to the approximation of $\log(A)$ and due to the stochastic trace estimation. We draw particular attention to these stochastic trace estimation methods as the approach considered here may be seen as an extension of this class of algorithms. Relative to those algorithms, our approach offers both less error, due to an exact representation of $\log(K + \sigma_n^2 I)$ to machine precision, and an exponential reduction in computation time over classical algorithms.

III. QUANTUM TRAINING ALGORITHMS

A. Logarithmic determinant algorithm

We address the first term in (1) by describing a quantum procedure to efficiently sample the eigenvalues of an $n \times n$ Hermitian matrix A uniformly at random, based on phase estimation [25–28]. This can be seen as a finite dimensional analogue of the continuous variable model proposed in [29]. For simplicity, we will assume that $n = 2^N$. The algorithm then proceeds as follows:

Step 1. Prepare N qubits in maximally-mixed state, $\frac{1}{n}\sum_{i=1}^{n} |i\rangle\langle i|$, and store this in a first register. This can be achieved simply by preparing the register in a random computational basis state. Note that a maximally-mixed state is maximally-mixed in any basis, hence we can choose to represent the density matrix for the system it in the eigenbasis $\{|e_i\rangle\}$ of matrix A:

$$\frac{1}{n}\sum_{i=1}^{n} |e_i\rangle\langle e_i|.$$

Step 2. Append a second register in a superposition state given by $\frac{1}{\sqrt{T}}\sum_{\tau=1}^{T} |\tau\rangle$, so that the composite system is in the state

$$\frac{1}{nT}\sum_{\tau,\tau'=1}^{T}\sum_{i=1}^{n} |e_i\rangle\langle e_i| \otimes |\tau\rangle\langle\tau'|.$$

Here T is chosen to be some sufficiently large to ensure accurate phase estimation as described in [26].

Step 3. Treating $(-A)$ as a Hamiltonian (which is possible since A is Hermitian), evolution under $(-A)$ for time specified by the second register is simulated on the state stored in the first register. This is achieved by applying the conditional unitary evolution $\sum_{\tau=1}^{T} \mathrm{e}^{iAt_0\tau/T} \otimes |\tau\rangle\langle\tau|$, where $t_0 = O(1/\epsilon)$ is chosen with respect to the ϵ-bounded error required in the algorithm. We thus obtain the state

$$\frac{1}{nT}\sum_{\tau,\tau'=1}^{T}\sum_{i=1}^{n} \mathrm{e}^{i\lambda_i t_0(\tau-\tau')/T} |e_i\rangle\langle e_i| \otimes |\tau\rangle\langle\tau'|.$$

Step 4. Perform a quantum Fourier transform of the second register. The resulting estimated eigenvalues of A, $\{\lambda_i\}$, are then stored in the second register as a binary bit-string up to a finite precision associated with the phase estimation procedure. Thus this results in the system being in state

$$\frac{1}{n}\sum_{i=1}^{n} |e_i\rangle\langle e_i| \otimes |\lambda_i\rangle\langle\lambda_i|.$$

Step 5. Measure the second register in computational basis to obtain a random λ_i.

This sampling method can then be turned to the task of estimating the log determinant of A, by making use of the identity

$$\langle \log \lambda_i \rangle = \frac{1}{n}\sum_{i=1}^{n} \log \lambda_i = \frac{1}{n}\mathrm{Tr}[\log(A)] = \frac{1}{n}\log[\det(A)].$$

Hence the desired quantity $\log[\det(A)]$ is given by $n\langle \log \lambda_i \rangle$ which will need to be estimated by sampling eigenvalues of A on repeated runs of the above procedure. The "penalty" term of the LML can now be estimated using the eigenvalue sampling procedure described here, by setting $A = K + \sigma_n^2 I$.

The dominant time cost of this procedure is the Hamiltonian simulation step, which carries a runtime of $\tilde{O}(s \log n)$ for an s-sparse matrix. This is achieved by using the method described in [30] which is based on Szegedy quantum walks [31]. Here we have used the notational shorthand convention $\tilde{O}(x)$ to denote $O(x \log^k(x))$ for any constant k, such that slower growing contributions are omitted. The optimised phase estimation procedure [26, 27] comes with an error, ϵ_{λ_i}, which scales as $O(1/t_0)$ in estimating each λ_i. This implies the error associated with the logarithm of a single eigenvalue scales as $\epsilon = \left|\frac{d\log\lambda_i}{d\lambda_i}\epsilon_{\lambda_i}\right| = O\left(\frac{1}{\lambda_i t_0}\right)$. Note also that in the context of GPs, there generally exists a $\sigma_n^2 I$ noise contribution to the covariance matrix, which accounts for the uncertainty in the observed data. This noise entry effectively bounds the lowest eigenvalue away from zero by a constant, so that we have $\lambda_{\min} \geq \sigma_n^2$. Hence the bounded-error runtime for obtaining a sample for estimating the penalty term of LML is given by $t = \tilde{O}\left(\frac{s \log n}{\sigma_n^2 \epsilon}\right)$.

B. Modified linear system algorithm

The second term of (1), $\frac{1}{2}\mathbf{y}^T A^{-1}\mathbf{y}$, which is often referred to as the "data fit" term and relates the dense outputs $\mathbf{y}$ to the assumed sparse covariance matrix K. We will show how a modified version of the quantum linear system algorithm (QLSA) [4] can be used in order to calculate this term. The QLSA operates similarly to phase estimation discussed earlier

in this paper, with the addition of an ancilla qubit which is rotated conditioned on the values of $f(\lambda_i)$, the eigenvalues of A, in the linear equations $A|\mathbf{x}\rangle = |\mathbf{b}\rangle$, and the non-linear function f. In the case of [4], f is simply the inverse of the eigenvalues. Post-selecting this ancilla qubit followed by the reversal of the phase estimation step results in finding $A^{-1}|\mathbf{b}\rangle$ with success probability $\langle\mathbf{b}|(A^{-1})^\dagger A^{-1}|\mathbf{b}\rangle$. As the authors of [4] note, the transformation $f(A)$ may be replaced using any computable function f.

Here we apply an modified version of the QLSA by choosing $f(\lambda) = \frac{1}{\sqrt{\lambda}}$ instead of the original inversion. The procedure for estimating the data fit term is given as follows:

Step 1. Use QRAM queries to prepare $|\mathbf{y}\rangle = \frac{\mathbf{y}}{\|\mathbf{y}\|}$.

Step 2. Set $|\mathbf{b}\rangle = |\mathbf{y}\rangle$ and $A = K + \sigma_n^2 I$, and run the modified QLSA with $f(\lambda) = \frac{1}{\sqrt{\lambda}}$, to obtain

$$\sum_i \gamma_i |e_i\rangle \left(\frac{c}{\sqrt{\lambda_i}}|0\rangle + \sqrt{1 - \frac{c^2}{\sqrt{\lambda_i}}}|1\rangle \right), \tag{2}$$

where $|\mathbf{y}\rangle = \gamma_i|e_i\rangle$ is written in the eigenbasis, and the constant is chosen such that $c \leq \sqrt{\lambda_{min}}$.

Step 3. Measure the last qubit and the probability of obtaining the outcome $|0\rangle$ is given by $\langle\mathbf{y}|A^{-1}|\mathbf{y}\rangle$.

Step 4. Sample on multiple runs of Steps 1-3 to obtain a Monte Carlo estimate of the data fit term with mean $\mathbf{y}^T A^{-1}\mathbf{y}$ with a variance bounded by $\frac{1}{4}\|\mathbf{y}\|^2\sigma_n^{-2}$.

As the original QLSA, the above procedure also requires a runtime of $\tilde{O}(\log(n)s\kappa/\epsilon)$ to produce the intermediate state (2), which amounts to the dominant cost of the above procedure.

C. Overall runtime

The full quantum estimation of LML is obtained by combining the penalty and the data fit terms. Thus the overall runtime for obtaining a single estimation sample of LML is given by $\tilde{O}(\log(n)(\kappa + \sigma_n^{-2})s/\epsilon)$. However, this single-run evaluation needs to be repeated multiple times in order to achieve a bounded estimation variance. In the following, we address the expected number of repetition needed for the purpose of GP training.

In each training step, a hyper-parameter, θ of the model is varied by an amount $\delta\theta$, where the prefix δ denotes the change in a quantity between steps. Then the corresponding variation, δLML is evaluated. The aim is to tune θ toward a maximal value of LML. The presented quantum algorithms are responsible for estimating the variation δLML with respect to each step of changing θ. Thus the appropriate figure of merit for the estimation error is the relative variance, as it quantifies the amount of dispersion between the estimated and the actual variation of δLML. In order to demonstrate the quantum advantage, it therefore suffices to show that the relative variance with respect to a change in hyper-parameter, $\delta\theta$, does not scale up with n. The relative variance of δLML is given by the following,

$$\frac{\mathrm{Var}\left[\delta\mathrm{LML}\right]}{\left[\delta\mathrm{LML}\right]^2} = \frac{\mathrm{var}\left[\log[\det(A)]\right] + \mathrm{var}\left[\mathbf{y}^T A^{-1}\mathbf{y}\right]}{\left[\frac{\partial}{\partial\theta}\left(\log[\det(A)] + \mathbf{y}^T A^{-1}\mathbf{y}\right)\delta\theta\right]^2}. \tag{3}$$

Now we write the $\mathbf{y}$ as a linear combination of the eigenvectors, $\mathbf{e}_i$ of A, such that $\mathbf{y} = \sum_i \gamma_i \mathbf{e}_i$, and $\mathbf{y}^T A^{-1}\mathbf{y} = \sum_i |\gamma_i|^2\lambda_i^{-1}$, we have

$$\begin{aligned}\frac{\mathrm{Var}\left[\delta\mathrm{LML}\right]}{\left[\delta\mathrm{LML}\right]^2} &\leq \frac{n^2\left(\mathrm{var}\left[\log\lambda_i\right] + \frac{1}{4}\left\langle y_i^2\right\rangle\sigma_n^{-2}\right)}{\left[\frac{\partial}{\partial\theta}\left(\sum_i \log\lambda_i + \sum_i |\gamma_i|^2\lambda_i^{-1}\right)\delta\theta\right]^2} \\ &\leq \frac{\left\langle(\log\lambda_i)^2\right\rangle + \frac{1}{4}\left\langle y_i^2\right\rangle\sigma_n^{-2}}{\left\langle\delta\lambda_i/\lambda_i + \delta\left(|\gamma_i|^2/\lambda_i\right)\right\rangle^2},\end{aligned} \tag{4}$$

where the expectation value notation is used to denote the average over all choices of i. We observe that the factor of n^2 in (4) is canceled in the last step of the inequality. Hence the relative variance in estimating the variation of LML with respect to a training step has no explicit dependence on the dimensionality.

Note that the number of hyperparameters is dependent only on the functional form of the kernel, and the number of training steps involved has no explicit dependence on the number of data points. Provided we are working to constant precision, the number of iterations which require computing LML is independent of the dimensionality n.

IV. DISCUSSION

Due to the linear sparseness dependence from the Hamiltonian simulation step, our algorithm performs best when the covariance matrix is some constant s-sparse, in which case our algorithm provides an exponential speed-up over the classical GP training procedure. Such sparsely constructed GPs have found applications in a range of interesting problems, especially when large size datasets are involved [32]. For examples, a sparse Gaussian process is used to construct a unified framework for robotic mapping in [33]. It was also applied to realistic action recognition problems in [34]. When applied to a non-sparse covariance matrix, a singular value estimation scheme which circumvents the Hamiltonian simulation step can be applied to achieve a runtime that scales as $\tilde{O}(\sqrt{n}\log n)$, assuming the spectral norm of A is bounded by a constant [35–37]. This provides a polynomial speed-up over the best classical counterpart.

Returning to the comparison with classical stochastic trace estimation methods, it is clear that the quantum algorithm offers a precise method to compute $\log(A)$ rather than either the truncated Taylor series or Chebyshev polynomial approximations. When measurements of the second register are taken, a single $\log(\lambda_i)$ is computed and hence our proposed approach can be seen as quantum stochastic trace estimation. The main advantage, however, comes from the reduction in computation time from polynomial to sub-linear. A natural question

which arises is whether the complete GP training can scale sub-linearly in n, since if not, an exponential improvement in computing the LML in each step would yield only a polynomial improvement in precision. Note that the number of hyper-parameters is dependent only on the kernel, and thus independent of the number of data points. Provided we are working to constant precision, the number of optimisation steps which require LML computation is upper bounded by a constant.

We have shown a quantum algorithm that improves the efficiency of calculating LML from a classical $O(n^3)$ to a logarithmic scaling with respect to the size of input under certain conditions. Specifically, if the structure of the covariance matrix is constant s-sparse, our algorithm provides an exponential speed-up. Even in cases when the Hamiltonian simulation step necessarily consumes a linear time overhead, this quantum algorithm still achieves a polynomial speed-up over its best-known classical counter-part.

Acknowledgements— The authors thank Mahboobeh Houshmand Kaffashian, Nana Liu and Liming Zhao for useful comments on the manuscript. JFF acknowledges support from the Ministry of Education Singapore, and the Air Force Office of Scientific Research under AOARD grant no. FA2386-15-1-4082. This material is based on research supported by the Singapore National Research Foundation under NRF Award No. NRF-NRFF2013-01.

[1] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning* (MIT press, 2012).
[2] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning series)* (The MIT Press, 2005), ISBN 026218253X.
[3] M. Schuld, I. Sinayskiy, and F. Petruccione, Contemporary Physics **56**, 172 (2015).
[4] A. W. Harrow, A. Hassidim, and S. Lloyd, Physical Review Letters **103**, 150502 (2009).
[5] S. Lloyd, M. Mohseni, and P. Rebentrost, Nature Physics **10**, 631 (2014).
[6] E. Aïmeur, G. Brassard, and S. Gambs, Machine Learning **90**, 261 (2013).
[7] S. Lloyd, M. Mohseni, and P. Rebentrost, arXiv preprint arXiv:1307.0411 (2013).
[8] P. Rebentrost, M. Mohseni, and S. Lloyd, Physical review letters **113**, 130503 (2014).
[9] N. Wiebe, A. Kapoor, and K. M. Svore, arXiv preprint arXiv:1412.3489 (2014).
[10] M. Schuld, I. Sinayskiy, and F. Petruccione, arXiv preprint arXiv:1601.07823 (2016).
[11] H. Neven, V. S. Denchev, G. Rose, and W. G. Macready, arXiv preprint arXiv:0811.0416 (2008).
[12] K. L. Pudenz and D. A. Lidar, Quantum information processing **12**, 2027 (2013).
[13] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko, arXiv preprint arXiv:1601.02036 (2016).
[14] J. Adcock, E. Allen, M. Day, S. Frick, J. Hinchliff, M. Johnson, S. Morley-Short, S. Pallister, A. Price, and S. Stanisic, arXiv preprint arXiv:1512.02900 (2015).
[15] Z. Zhao, J. K. Fitzsimons, and J. F. Fitzsimons, arXiv preprint arXiv:1512.03929 (2015).
[16] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, arXiv preprint arXiv:1711.00165 (2017).
[17] Z. Zhao, A. Pozas-Kerstjens, P. Rebentrost, and P. Wittek, arXiv preprint arXiv:1806.11463 (2018).
[18] V. V. Williams, E-mail address: jml@ math. tamu. edu (2011).
[19] J. Quiñonero-Candela and C. E. Rasmussen, Journal of Machine Learning Research **6**, 1939 (2005).
[20] V. Minden, A. Damle, K. L. Ho, and L. Ying, arXiv preprint arXiv:1603.08057 (2016).
[21] R. K. Pace and J. P. LeSage, Computational Statistics & Data Analysis **45**, 179 (2004).
[22] C. Boutsidis, P. Drineas, P. Kambadur, and A. Zouzias, arXiv preprint arXiv:1503.00374 (2015).
[23] H. Avron and S. Toledo, Journal of the ACM (JACM) **58**, 8 (2011).
[24] J. K. Fitzsimons, M. A. Osborne, S. J. Roberts, and J. F. Fitzsimons, arXiv preprint arXiv:1608.00117 (2016).
[25] A. Y. Kitaev, arXiv preprint quant-ph/9511026 (1995).
[26] V. Bužek, R. Derka, and S. Massar, Physical review letters **82**, 2207 (1999).
[27] A. Luis and J. Peřina, Physical review A **54**, 4564 (1996).
[28] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca, in *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* (The Royal Society, 1998), vol. 454, pp. 339–354.
[29] N. Liu, J. Thompson, C. Weedbrook, S. Lloyd, V. Vedral, M. Gu, and K. Modi, arXiv preprint arXiv:1510.04758 (2015).
[30] D. W. Berry, A. M. Childs, and R. Kothari, in *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on* (IEEE, 2015), pp. 792–809.
[31] D. W. Berry and A. M. Childs, arXiv preprint arXiv:0910.4157 (2009).
[32] A. Melkumyan and F. Ramos, in *IJCAI* (2009), vol. 9, pp. 1936–1942.
[33] S. Kim and J. Kim, in *Field and Service Robotics* (Springer, 2015), pp. 319–332.
[34] L. Liu, L. Shao, F. Zheng, and X. Li, Pattern Recognition **47**, 3819 (2014).
[35] L. Wossnig, Z. Zhao, and A. Prakash, Physical review letters **120**, 050502 (2018).
[36] I. Kerenidis and A. Prakash, arXiv preprint arXiv:1603.08675 (2016).
[37] I. Kerenidis and A. Prakash, arXiv preprint arXiv:1704.04992 (2017).