

PAPER • OPEN ACCESS

## A strategy for quantum algorithm design assisted by machine learning

To cite this article: Jeongho Bang *et al* 2014 *New J. Phys.* **16** 073017

View the [article online](#) for updates and enhancements.

You may also like

- [On the robustness of bucket brigade quantum RAM](#)  
Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor et al.
- [Fault-ignorant quantum search](#)  
Péter Vrana, David Reeb, Daniel Reitzner et al.
- [Quantum gradient descent and Newton's method for constrained polynomial optimization](#)  
Patrick Rebentrost, Maria Schuld, Leonard Wossnig et al.

## A strategy for quantum algorithm design assisted by machine learning

Jeongho Bang<sup>1,2</sup>, Junghee Ryu<sup>2,3</sup>, Seokwon Yoo<sup>2</sup>, Marcin Pawłowski<sup>3,4</sup> and Jinhyoung Lee<sup>1,2</sup>

<sup>1</sup> Center for Macroscopic Quantum Control & Department of Physics and Astronomy, Seoul National University, Seoul, 151-747, Korea

<sup>2</sup> Department of Physics, Hanyang University, Seoul, 133-791, Korea

<sup>3</sup> Institute of Theoretical Physics and Astrophysics, University of Gdańsk, 80-952 Gdańsk, Poland

<sup>4</sup> Department of Mathematics, University of Bristol, Bristol BS8 1TW, UK

E-mail: [jbang@snu.ac.kr](mailto:jbang@snu.ac.kr) and [hyoung@hanyang.ac.kr](mailto:hyoung@hanyang.ac.kr)

Received 27 February 2014, revised 12 May 2014

Accepted for publication 2 June 2014

Published 14 July 2014

*New Journal of Physics* **16** (2014) 073017

doi:[10.1088/1367-2630/16/7/073017](https://doi.org/10.1088/1367-2630/16/7/073017)

### Abstract

We propose a method for quantum algorithm design assisted by machine learning. The method uses a *quantum–classical hybrid* simulator, where a ‘quantum student’ is being taught by a ‘classical teacher’. In other words, in our method, the learning system is supposed to evolve into a quantum algorithm for a given problem, assisted by a classical main-feedback system. Our method is applicable for designing quantum oracle-based algorithms. We chose, as a case study, an oracle decision problem, called a Deutsch–Jozsa problem. We showed by using Monte Carlo simulations that our simulator *can* faithfully learn a quantum algorithm for solving the problem for a given oracle. Remarkably, the learning time is proportional to the *square root* of the total number of parameters, rather than showing the exponential dependence found in the classical machine learning-based method.

Keywords: quantum learning, quantum automatic control, quantum algorithm



Content from this work may be used under the terms of the [Creative Commons Attribution 3.0 licence](https://creativecommons.org/licenses/by/3.0/). Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

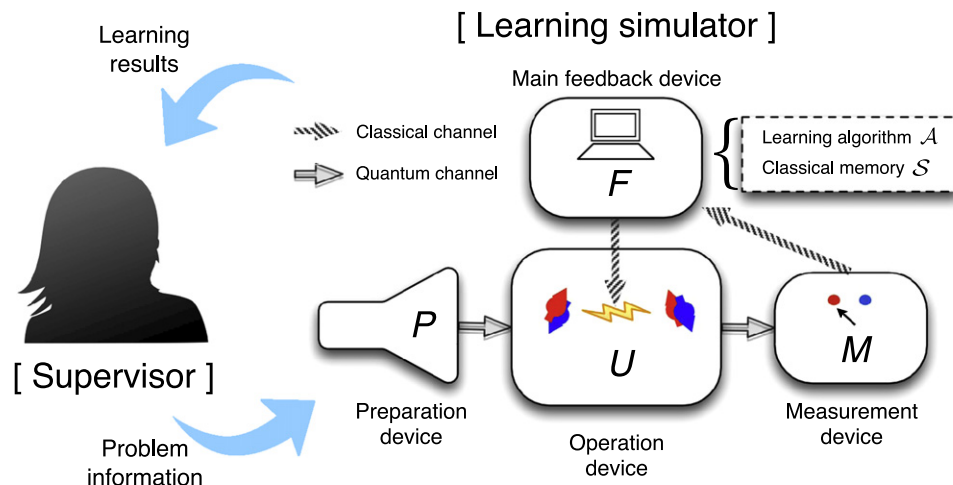
## 1. Introduction

Quantum information science has seen explosive growth in recent years, as a more powerful generalization of classical information theory [1]. In particular, quantum computation has received momentum from quantum algorithms that outperform their classical counterparts [2–5]. Thus, the development of quantum algorithms is one of the most important areas of computer science. However, unfortunately, recent research on quantum algorithm design is rather stagnant, compared to other areas in quantum information, as scarcely any new quantum algorithms have been discovered in the past few years [6]. We believe that this is due to the fact that we—the designers—are used to *classical* logic. Thus we think that quantum algorithm design should turn towards new methodology, different from that of the current approach.

Machine learning is a well-developed branch of artificial intelligence and automatic control. Although ‘learning’ is often thought of as a uniquely human trait, a machine being given feedback (taught) can improve its performance (learn) in a given task [7, 8]. In the past few decades, there has been a growing interest not only in theoretical studies of but also in a variety of applications of machine learning. Recently, many quantum implementations of machine learning have been introduced to achieve better performance for quantum information processing [9–13]. These works motivate us to look at machine learning as an alternative approach for quantum algorithm design.

Keeping our primary goal in mind, we ask whether a quantum algorithm can be found by the machine that also implements it. On the basis of this idea, we consider a machine which is able to learn quantum algorithms in a real experiment. Such a machine may discover solutions which are difficult for humans to find because of our classical way of thinking. Since we can always simulate a quantum machine on a classical computer (though not always efficiently), we can use such simulations to design quantum algorithms without the need for a programmable quantum computer. This classical machine can thus be regarded as a simulator that learns a quantum algorithm—a so-called *learning simulator*. The novelty of such a learning simulator lies in its capabilities of ‘learning’ and ‘teaching’. With regard to these abilities, we consider two internal systems: one is a learning system (the ‘student’ say), and the other is a main-feedback system (the ‘teacher’ say). While the standard approach is to assume that both student and teacher are quantum machines, here we use a *quantum–classical hybrid* simulator such that the student is a quantum machine and the teacher a classical machine. Such a hybridization is easier and more economical to realize if any algorithms can be learned.

In this paper, we employ a learning simulator for quantum algorithm design. The main question of this work is: ‘Can our learning simulator help in designing a quantum algorithm?’ The answer to this question is affirmative, as it is shown, in Monte Carlo simulations, that our learning simulator *can* faithfully *learn* appropriate elements of a quantum algorithm for solving an oracle decision problem, called the Deutsch–Jozsa problem. The algorithms found are equivalent, but not exactly equal, to the original Deutsch–Jozsa algorithm. We also investigate the learning time, as it becomes important in application not only due to the large-scale problems often arising in machine learning but also because, in its learning, our simulator will exhibit the quantum speedup (if any) of an algorithm to be found, as described later. We observe that the learning time is proportional to the *square root* of the total number of parameters, in contrast to showing the exponential tendency found in classical machine learning. We expect our learning simulator to reflect the quantum speedup of the algorithm found in its learning, possibly in synergy with the finding that the size of the parameter space



**Figure 1.** Schematic picture of our method. A supervisor defines the problem to be solved and arranges the necessary prerequisites for learning the quantum algorithm. All of this information is communicated to the learning simulator at once. The simulator encodes the information on its own elements. The simulator consists of quantum elements, i.e., preparation  $P$ , operation  $U$ , and measurement  $M$ , assisted by the classical main feedback  $F$ . The classical channels  $C_{MF}$  and  $C_{FU}$  enable one-way communication from  $M$  to  $F$  and from  $F$  to  $U$ .

can be significantly smaller for quantum algorithms than for their classical counterparts [14]. We note that the method presented is aimed at a real experiment, in contrast to the techniques of [15, 16].

## 2. The basic architecture of the learning simulator

Before discussing the details of the learning simulator, it is important to have an understanding of what machine learning is. A typical task of machine learning is to find a function  $f(x) = t_x$  for the input  $x$  and the target  $t_x$  based on the observations in supervised learning, or to find some hidden structure in unsupervised learning [7, 8]. The main difference between supervised and unsupervised learning is that in the latter case the target  $t_x$  is unknown. Throughout this paper, we consider supervised learning where the target  $t_x$  is known.

We now briefly sketch our method (see also figure 1). To begin, a supervisor defines the problem to be solved, and arranges the necessary prerequisites (e.g., the input–target pairs  $(x, t_x)$ , and a function  $Q$  produced by a non-trivial device, the so-called oracle) for learning. The preliminary information is passed to the learning simulator *at once*. The simulator encodes the communicated information on its own elements. We note here that one could consider two main tasks in designing a quantum algorithm. The first is to construct a useful form of quantum oracle, and the second is to find another incorporating quantum operation(s) to maximize the quantum advantages, such as superposition engaging parallelism [17] or entanglement [18]. We

focus here on the latter<sup>5</sup>. Note, however, that it is necessary to define a specific oracle operation (see appendix A). This task is also performed, by the supervisor, at this preliminary stage.

We then describe the basic elements of the learning simulator in figure 1. The simulator consists of two internal parts. One is the learning system which is supposed to eventually perform a quantum algorithm, and the other is the feedback system responsible for teaching the former. The learning system consists of the standard *quantum* information-processing devices: preparation  $P$  for preparing a pure quantum state, operation  $U$  for performing a unitary operation, and measurement  $M$ . Here, the chosen quantum oracle is involved in  $U$ . On the other hand, the feedback system is *classical*, as this is easier and less expensive to realize in practice. Furthermore, by employing classical feedback, we can use a well-known (classical) learning algorithm whose performance has already been proved to be reliable. Recently, a scheme for machine learning involving quantum feedback has been reported [21], but the usefulness of the quantumness has not been clearly elucidated, even though the reported results are meaningful in some applications. Moreover, it is unclear at present whether any classical feedback is applicable to quantum algorithm design. Consequently, it is preferred to use classical feedback in this work. In this sense, our simulator is a *quantum–classical hybrid*. The feedback system is equipped with a main-feedback device  $F$  which involves the classical memory  $\mathcal{S}$  and the learning algorithm  $\mathcal{A}$ .  $\mathcal{S}$  records the control parameters of  $U$  and the measurement results of  $M$ .  $\mathcal{A}$  corresponds to a series of rules for updating  $U$ .

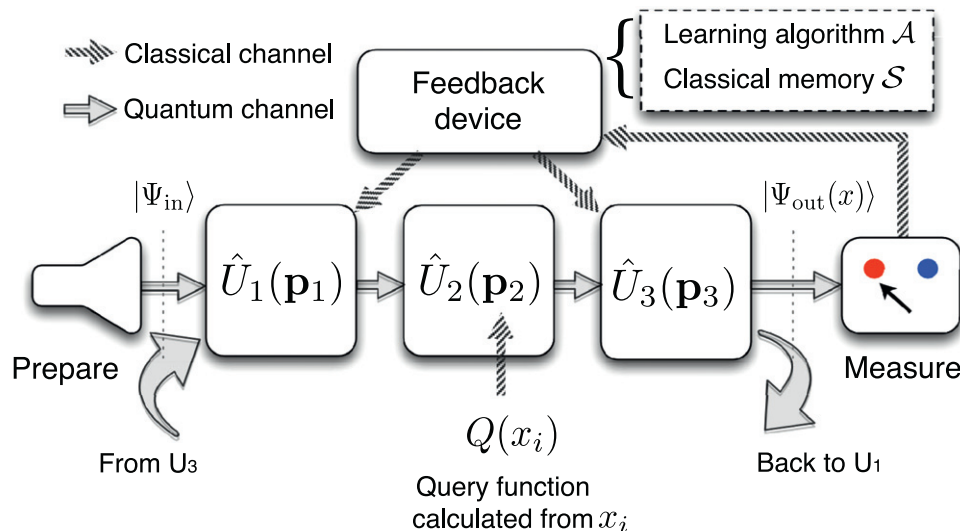
We illustrate how our simulator performs the learning. Let us start with the set of  $K$  input–target pairs communicated from the supervisor:

$$T = \left\{ (x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_K, f(x_K)) \right\}, \quad (1)$$

where  $f$  is a function that transforms the inputs  $x_i$  into their targets<sup>6</sup>. The main task of the simulator is to find  $f$ . Firstly, an initial state  $|\Psi_{\text{in}}\rangle$  is prepared in  $P$  and transformed to  $|\Psi_{\text{out}}\rangle$  by  $U$ . Then  $M$  performs measurement on  $|\Psi_{\text{out}}\rangle$  with a chosen measurement basis. The measurement result is delivered to  $F$  through  $C_{MF}$ . Note here that the information about the initial state  $|\Psi_{\text{in}}\rangle$  and the measurement basis encoded in  $P$  and  $M$  is also determined by the supervisor before the learning. Finally,  $F$  updates  $U$  on the basis of  $\mathcal{A}$ . Basically, the learning is just the repetition of these three steps. When the learning is completed, we obtain a  $P$ – $U$ – $M$  device to implement  $f$  by simply removing  $F$ . The supervisor, then, investigates whether the  $P$ – $U$ – $M$  device found provides any speedup reducing the overall oracle references, or saves any computational resources for implementing the algorithm [22]. In particular, the supervisor would standardize the identified operations  $U$  as an algorithm. Here, we clarify that the input information in  $T$  and the measurement results are classical. Nevertheless, the simulator is supposed to exploit quantum effects in learning, because the operations before measurement are all quantum. This assumption is supported by recent theoretical studies that show the improvement of learning efficiency achieved by using quantum superposition [14, 23].

<sup>5</sup> Actually, in algorithm design [19] or logic-mechanism programming [20], the important point is usually how we utilize a given oracle (or a corresponding operation for judging the positive or negative state) with other incorporated logics in order to achieve a speedup of the designed algorithm, rather than how we construct or optimize the oracle itself.

<sup>6</sup> Here,  $x_i$  ( $i = 1, 2, \dots, K$ ) can be encoded either on the state  $|\Psi_{\text{in}}\rangle$  in  $P$  or on the control parameters of  $U$ . In most cases, encoding on  $U$  is appropriate and this is the choice made for our work, as shown later.



**Figure 2.** Architecture of our simulator for learning a quantum algorithm, where the unitary operation  $U$  consists of three sub-operations (see the text).

### 3. Construction of the learning simulator

The general design of the learning simulator depicted in figure 1 works fine for problems such as number factorization. However, in problems requiring a large number of oracle references, the input is the oracle itself and, by definition, it is a (unitary) transformation rather than a string of bits. To allow for the input in the form of a unitary matrix we need to refine our simulator a little (but let us stress that this does not mean that our method is not general). The refined version depicted in figure 2 allows the simulator to learn an algorithm of iterative type. The difference in learning simulator stems directly from the formulation of the problems.

The most important aspect of the refined learning simulator is the decomposition of  $U$ . In order to deal with both classical and quantum information, we divide  $U$  into three sub-devices, such that

$$\hat{U}_{\text{tot}} = \hat{U}_3(\mathbf{p}_3) \hat{U}_2(\mathbf{p}_2) \hat{U}_1(\mathbf{p}_1), \quad (2)$$

where  $\hat{U}_{\text{tot}}$  is the total unitary operator, and  $\hat{U}_j$  ( $j = 1, 2, 3$ ) denotes the unitary operator of the  $j$ th sub-device. Here,  $\hat{U}_1$  and  $\hat{U}_3$  are  $n$ -qubit controllable unitary operators, whereas  $\hat{U}_2$  is the oracle for encoding the input  $x_i$ . By ‘controllable’ we mean here, and throughout the paper, that they can be changed by the feedback.

The unitary operators are *generally* parameterized as

$$\hat{U}(\mathbf{p}) = \exp(-i\mathbf{p}^T \mathbf{G}), \quad (3)$$

where  $\mathbf{p} = (p_1, p_2, \dots, p_{d^2-1})^T$  is a real vector in  $(d^2 - 1)$ -dimensional Bloch space for  $d = 2^n$ , and  $\mathbf{G} = (\hat{g}_1, \hat{g}_2, \dots, \hat{g}_{d^2-1})^T$  is a vector whose components are  $SU(d)$  group generators [24, 25]. The components  $p_j \in [-\pi, \pi]$  of  $\mathbf{p}$  can be directly matched to control parameters in some experimental schemes, e.g., beam-splitter and phase-shifter alignments in linear optical systems

[26] and radio-frequency (rf) pulse sequences in nuclear magnetic resonance (NMR) systems [27]. In that sense, we call  $\mathbf{p}$  a control-parameter vector. Here,  $\mathbf{p}_2$  is determined by  $Q(x_i) \mapsto \mathbf{p}_2(x_i)$ , as described above. In such a setting, we expect our simulator to learn an optimal set of  $\{\mathbf{p}_1, \mathbf{p}_3\}$ , such that  $\hat{U}_1$  and  $\hat{U}_3$  will solve a given problem.

Our simulator is actually well-suited to learning even iterative algorithms, such as Grover's [5]. We envisage using our simulator as follows: in the first stage, apply  $\hat{U}_1$  to an input state, then  $\hat{U}_2$  which is a non-trivial operation, say the oracle, and finally  $\hat{U}_3$ , to generate an output state. The feedback system updates  $\hat{U}_1$  and  $\hat{U}_3$ . Then, after a certain number of iterations which do not lead to any improvements, our simulator goes to the second stage, where the output state is fed back to be the input state to apply  $\hat{U}_1$ - $\hat{U}_2$ - $\hat{U}_3$  again. Therefore, in the second stage, the oracle is referenced twice. If it fails again, it will try to loop three times at the third stage. By some number of stages, there will be enough oracle references to solve the problem. In such a way, our simulator can learn even a quantum algorithm of iterative type<sup>7</sup>, without adopting any additional sub-devices and altering the structure in a real experiment. Thus, the scalability for the size of the search space is only concerned with the number of control parameters in  $\hat{U}_1$  and  $\hat{U}_3$ , given by  $D = 2(d^2 - 1)$ , where  $d = 2^n$ .

Here, we highlight another subsidiary question: how long does it take for our simulator to learn an (almost) deterministic quantum algorithm? Investigating this issue will become increasingly important, especially in the application of our simulator to very large-scale (i.e.,  $D \gg 1$ ) problems. One may raise the objection that our simulator runs extremely slowly for large size problems. On the other hand, however, it is also likely that, in its learning, our simulator enjoys the quantum speedup, if any, of the algorithm to be found. To see this, consider two cases: a classical algorithm and a quantum algorithm which our simulator tries to find, assuming that they are of different complexities in terms of the number of oracle queries. For instance, the quantum one may query a polynomial number of oracles, whereas the number of queries for the classical one increases exponentially with respect to the problem size. Regardless of the methods of realization, a learning simulator cannot reduce the number of stages to less than the number of oracle queries in a given algorithm to be found. This is reflected in the learning time. In other words, our simulator may show the learning speedup, exploring far fewer stages in the learning of the quantum algorithm, as long as the algorithm to be found exhibits quantum speedup. These controversial arguments require us to investigate the learning time as well as the effectiveness of our simulator.

#### 4. Application to the Deutsch–Jozsa problem

As a case study, consider an  $n$ -bit oracle decision problem, called the Deutsch–Jozsa (DJ) problem. The problem is deciding whether some binary function  $x_i: \{0, 1\}^n \rightarrow \{0, 1\}$  is constant ( $x_i$  generates the same value 0 or 1 for every input) or balanced ( $x_i$  generates 0 for exactly half of the inputs, and 1 for the rest of the inputs) [2, 3]. On a classical Turing machine,

<sup>7</sup> The procedure is not the most general one. For full generality one would also need to add some quantum memory but, to our knowledge, no existing quantum algorithm actually uses this yet.

$2^{n-1} + 1$  queries are required to solve this problem. If we use a probabilistic random classical algorithm, we can determine the function  $x_i$  with a small error, less than  $2^{-q}$ , by making  $q$  queries [28, 29].

On the other hand, the DJ quantum algorithm solves the problem with only a single query [29, 30]. The DJ quantum algorithm runs as follows: first, apply  $\hat{H}^{\otimes n}$  on the input state  $|\Psi_{\text{in}}\rangle = |00\dots 0\rangle$ , then  $\hat{U}_x$  to evaluate the input function, and finally  $\hat{H}^{\otimes n}$  again to produce an output state  $|\Psi_{\text{out}}\rangle$ . Here,  $\hat{H}$  is the Hadamard gate which transforms the qubit states  $|0\rangle$  and  $|1\rangle$  into equal superposition states  $\hat{H}|0\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$  and  $\hat{H}|1\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$  respectively.  $\hat{U}_x$  is the function-evaluation gate that calculates a given function  $x_i$ . It is defined by its action,

$$\hat{U}_x |k_1 k_2 \dots k_n\rangle = e^{i\pi x_i(k_1 k_2 \dots k_n)} |k_1 k_2 \dots k_n\rangle, \quad (4)$$

where  $k_1 k_2 \dots k_n \in \{0, 1\}^n$  is the binary sequence of the computational basis. Then, the output state is given as

$$|\Psi_{\text{out}}(x_i)\rangle = \begin{cases} \pm |00\dots 0\rangle, & \text{if } x_i \in C \\ \pm |z_1 z_2 \dots z_n\rangle, & \text{if } x_i \in B \end{cases} \quad (5)$$

where  $C$  and  $B$  are the sets of constant and balanced functions, respectively, and the binary components  $z_j \in \{0, 1\}$  ( $j = 1, 2, \dots, n$ ) depend on the  $\binom{d}{d/2}$  balanced functions (excepting that  $z_j = 0$  for all  $j$ ). In the last step, von Neumann measurement is performed on the output state. The corresponding measurement operator is given by  $\hat{M} = |00\dots 0\rangle\langle 00\dots 0|$ . The other projectors constituting the observable are irrelevant because we are interested only in the probabilities associated with the first case,

$$P_C = \langle \Psi_{\text{out}}(x_i) | \hat{M} | \Psi_{\text{out}}(x_i) \rangle = 1, \quad \text{if } x_i \in C, \quad (6)$$

and the second case,

$$P_B = \langle \Psi_{\text{out}}(x_i) | \hat{M} | \Psi_{\text{out}}(x_i) \rangle = 0, \quad \text{if } x_i \in B. \quad (7)$$

Therefore it is promised that the function  $x_i$  is either constant or balanced by only a single oracle query.

We are now ready to apply our method to the DJ problem. To begin, the supervisor prepares the set of input–target pairs,

$$T = \left\{ (x_i, f(x_i)) \mid f(x_i) = \text{'c'} \text{ if } x_i \in C \text{ and } f(x_i) = \text{'b'} \text{ if } x_i \in B \right\} \quad (8)$$

The learning simulator is to find the ‘functional’  $f$  now, by adjusting  $\hat{U}_1$  and  $\hat{U}_3$ . The input functions  $x_i$  are encoded in  $\mathbf{p}_2(x_i)$  of  $\hat{U}_2$ . Here, we chose the same form of the oracle as in equation (4), i.e., type (ii). Then  $P$  prepares an *arbitrary* initial state  $|\Psi_{\text{in}}\rangle$  and  $M$  performs the measurement on each qubit. Here we introduce a function to apply a measurement result to one of the targets (in our case, ‘c’ or ‘b’). We call this the interpretation function. Note that the interpretation function is also to be learned, because, in general, any *a priori* knowledge of the quantum algorithm to be found is completely unknown. For the sake of convenience, we



consider a Boolean function that transforms the measurement result  $z_1 z_2 \cdots z_n$  to 0 (equivalently, ‘c’) only if  $z_j = 0$  for all  $j = 1, 2, \dots, n$ , and otherwise 1 (equivalently, ‘b’). One may generalize the interpretation function to a function  $\{0, 1\}^n \rightarrow \{0, 1\}^m$  if one is interested in any other problems that contain many targets less than  $2^m$  [31].

## 5. The learning algorithm for differential evolution

One of the most important parts of our method is the choice of a learning algorithm  $\mathcal{A}$ . The efficiency and accuracy of machine learning are heavily influenced in general by the algorithm chosen. We employ so-called ‘differential evolution’, as it is known as one of the most efficient optimization methods [32]. We implement the differential evolution as follows. To begin, we prepare  $N_{\text{pop}}$  sets of the control-parameter vectors:  $\{\mathbf{p}_{1,i}, \mathbf{p}_{3,i}\}$  ( $i = 1, 2, \dots, N_{\text{pop}}$ ). Thus we have  $2N_{\text{pop}}$  parameter vectors in total. They are chosen initially at random and recorded on  $\mathcal{S}$  in  $F$ .

[L.1] Then,  $2N_{\text{pop}}$  mutant vectors  $\nu_{k,i}$  are generated for  $\hat{U}_k$  ( $k = 1, 3$ ), according to

$$\nu_{k,i} = \mathbf{p}_{k,a} + W(\mathbf{p}_{k,b} - \mathbf{p}_{k,c}),$$

where  $\mathbf{p}_{k,a}$ ,  $\mathbf{p}_{k,b}$ , and  $\mathbf{p}_{k,c}$  are randomly chosen for  $a, b, c \in \{1, 2, \dots, N_{\text{pop}}\}$ . These three vectors are chosen to be different from each other; for that,  $N_{\text{pop}} \geq 3$  is necessary. The free parameter  $W$ , called a differential weight, is a real and constant number.

[L.2] After that, all  $2N_{\text{pop}}$  parameter vectors

$$\mathbf{p}_{k,i} = (p_{k,1}, p_{k,2}, \dots, p_{k,d^2-1})_i^T$$

are reformulated as trial vectors

$$\boldsymbol{\tau}_{k,i} = (\tau_{k,1}, \tau_{k,2}, \dots, \tau_{k,d^2-1})_i^T$$

by means of the following rule: for each  $j$ ,

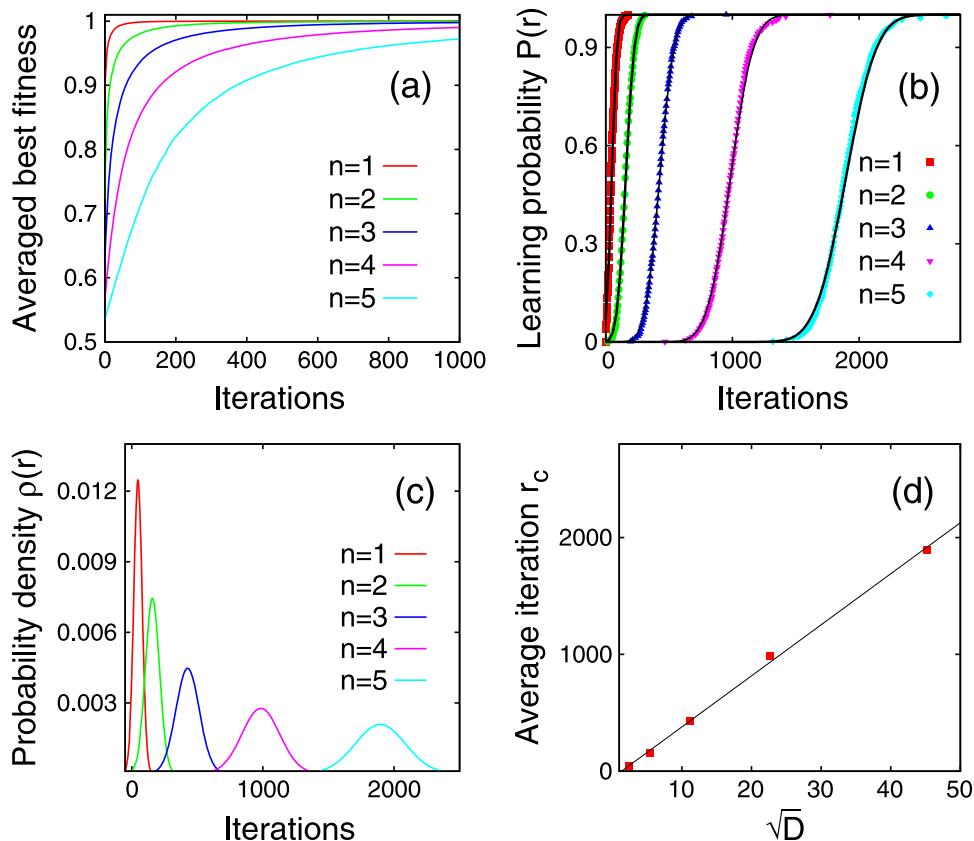
$$\begin{cases} \tau_{k,j} \leftarrow p_{k,j} & \text{if } R_j > C_r, \\ \tau_{k,j} \leftarrow \nu_{k,j} & \text{otherwise,} \end{cases} \quad (9)$$

where  $R_j \in [0, 1]$  is a randomly generated number and the crossover rate  $C_r$  is another free parameter between 0 and 1.

[L.3] Finally,  $\{\boldsymbol{\tau}_{1,i}, \boldsymbol{\tau}_{3,i}\}$  are taken for the next iteration if  $\hat{U}_1(\boldsymbol{\tau}_{1,i})$  and  $\hat{U}_3(\boldsymbol{\tau}_{3,i})$  yield a fitness value larger than that from  $\hat{U}_1(\mathbf{p}_{1,i})$  and  $\hat{U}_3(\mathbf{p}_{3,i})$ ; if not,  $\{\mathbf{p}_{1,i}, \mathbf{p}_{3,i}\}$  are retained. Here the fitness  $\xi_i$  is defined by

$$\xi_i = \frac{P_{C,i} + (1 - P_{B,i})}{2}, \quad (10)$$

where  $P_{C,i}$  and  $P_{B,i}$  are measurement probabilities for the  $i$ th set, given by equations (6) and (7). While evaluating the  $N_{\text{pop}}$  fitness values,  $F$  records on  $\mathcal{S}$  the best  $\xi_{\text{best}}$  and its corresponding parameter vector set  $\{\mathbf{p}_{1,\text{best}}, \mathbf{p}_{3,\text{best}}\}$ .



**Figure 3.** (a) Averaged best fitness  $\bar{\xi}_{\text{best}}$  versus iteration  $r$ . Each data value is averaged over 1000 simulations. It is observed that  $\bar{\xi}_{\text{best}}$  approaches unity upon iterating. (b) Learning probability  $P(r)$  for the halting condition  $\bar{\xi}_{\text{best}} \geq 0.99$ , when sampling 1000 trials.  $P(r)$  is well-fitted by an integrated Gaussian (black solid line),  $G(r) = \int_{-\infty}^r dr' \rho(r')$ . (c) Probability density  $\rho(r)$  resulting from  $P(r)$  for each  $n$ . (d) Graph of  $r_c$  versus  $\sqrt{D}$ , where  $D$  is the total number of the control parameters, and  $r_c$  is the average number of iterations needed to complete the learning. The data are well-fitted linearly by  $r_c = A\sqrt{D} + B$  with  $A \simeq 43$  and  $B \simeq -57$ .

The above steps [L.1]–[L.3] are repeated until  $\bar{\xi}_{\text{best}}$  reaches close to 1. In an ideal case, the simulator finds  $\{\mathbf{p}_{1,\text{best}}, \mathbf{p}_{3,\text{best}}\}$  yielding  $\bar{\xi}_{\text{best}} = 1$  with  $P_C = 1$  and  $P_B = 0$ . The parameters found lead to an algorithm equivalent to the original DJ one.

## 6. Numerical analysis

The simulations are done for the  $n$ -bit DJ problem, with  $n$  increasing from 1 to 5. In the simulations, we take  $N_{\text{pop}} = 10$  for all  $n$ <sup>8</sup>. The results are given in figure 3(a), where we present

<sup>8</sup> For a large size classical learning system, a huge number  $N_{\text{pop}}$  of candidate solutions are usually needed. For example, it is appropriate to choose  $N_{\text{pop}} \simeq 5D-10D$  (see [32]).

the averaged best fitness  $\bar{\xi}_{\text{best}}$ , sampling 1000 trials. It is clearly observed that  $\bar{\xi}_{\text{best}}$  approaches 1 as the iteration proceeds. There is just one stage required for all  $n$ . This implies that our simulator *can* faithfully *learn* a single-query quantum algorithm for the DJ problem, showing  $\xi \simeq 1$ . It is also notable that the algorithms found are equivalent to, but not exactly equal to, the original DJ algorithm: the  $\hat{U}_1$  and  $\hat{U}_3$  found are always different, but constitute an algorithm solving the DJ problem (see appendix B).

Then we present a learning probability  $P(r)$ , defined by the probability that the learning is completed before or at the  $r$ th iteration [33]. Here we assume a halting condition  $\bar{\xi}_{\text{best}} \geq 0.99$  for finding a *nearly deterministic* algorithm. In figure 3(b), we present  $P(r)$  for all  $n$ , each of which is averaged over 1000 simulations. We find that  $P(r)$  is well-fitted with an integrated Gaussian

$$G(r) = \int_{-\infty}^r dr' \rho(r'), \quad (11)$$

where the probability density  $\rho(r)$  is a Gaussian function  $\frac{1}{\sqrt{2\pi}\Delta r} e^{-\frac{(r-r_c)^2}{2\Delta r^2}}$ . Here,  $r_c$  is the average iteration number and  $\Delta r$  is the standard deviation over the simulations, which characterize how many iterations are sufficient for a statistical accuracy of  $\bar{\xi}_{\text{best}} \geq 0.99$ . Note that we have *finite* values of  $r_c$  and  $\Delta r$  for all  $n$ . The probability density  $\rho(r)$  is drawn in figure 3(c), resulting from  $P(r)$ .

We also investigate the learning time. As we already pointed out, the learning time becomes an intriguing issue which may be related not only to the applicability of our algorithm to large-scale problems but also to the learning speedup. Regarding  $r_c$  as a learning time, we present a graph of  $r_c$  versus  $\sqrt{D}$  in figure 3(d). Remarkably, the data are well-fitted *linearly* with  $r_c = A\sqrt{D} + B$  with  $A \simeq 43$  and  $B \simeq -57$ . This means that the learning time is proportional to the *square root* of the size of the parameter space<sup>9</sup>. This contrasts with the typical exponential tendency for classical machine learning (see, for example, [34, 35] and the references therein).

## 7. Summary and remarks

We have presented a method for quantum algorithm design based on machine learning. The simulator that we have used is a *quantum-classical hybrid*, where the quantum student is being taught by a classical teacher. We discussed such hybridization being beneficial in terms of the usefulness and the implementation cost. Our simulator was applicable in designing an oracle-based quantum algorithm. We demonstrated, as a case study, that our simulator can faithfully learn a single-query quantum algorithm that solves the DJ problem, even though it does not have to. The algorithms found are equivalent, but not exactly equal, to the original DJ algorithm with fitness  $\simeq 1$ .

We also investigated the learning time, as this would become increasingly important in application, not only due to the large-scale problems often arising in machine learning but also because, in its learning, our simulator potentially exhibits the quantum speedup, if any, of an

<sup>9</sup> It is worth noting that there is an alternative method, called semidefinite programming, which may be used for the purpose of finding a quantum algorithm. In [19], the authors have considered the problem of finding optimal unitaries given a fixed number of queries. Their algorithm was able to solve the problem in polynomial time (i.e. polynomial in the dimension  $d$ ).

algorithm to be found. In the investigation, we observed that the learning time is proportional to the *square root* of the size of the parameter space, instead of showing the exponential dependence of classical machine learning. This result is very suggestive. We expect our simulator to reflect the quantum speedup of the algorithm found in its learning, possibly in synergy with the finding from [14] that for quantum algorithms, the size of the parameter space can be significantly smaller than that for their classical counterparts: not only does their learning time scale more favorably with the size of the space, but also this size is smaller to begin with.

We hope that the proposed method will help in designing quantum algorithms, and provide an insight into learning speedup, establishing a link between the learning time and the quantum speedup of the algorithms found. However, it is an open question whether one would observe more improved behaviors in quantum algorithm design on employing a quantum feedback rather than the classical feedback.

## Acknowledgments

JB would like to thank M Żukowski, H J Briegel, and B C Sanders for discussions and comments. We acknowledge the financial support of National Research Foundation of Korea (NRF) grants funded by the Korea government (MEST; No.2010-0018295 and No.2010-0015059). JR and MP were supported by the Foundation for Polish Science TEAM project cofinanced by the EU European Regional Development Fund. JR was also supported by NCBiR-CHIST-ERA Project QUASAR. MP was also supported by the UK EP-SRC and ERC grant QOLAPS.

## Appendix A. Quantum oracle operation

As described in the main text, one could consider two different issues in designing a certain type of quantum algorithm. The first is determining a specific form of quantum oracle operation, and the second is finding another incorporating operations to maximize the quantum advantages. Although we focused on the latter in the current work, it is also necessary to inquire into the question of what kind of quantum oracle is best fitted for our learning simulator in figure 2, practically.

Dealing with the quantum oracle is a twofold task: defining the appropriate query function  $Q$  and encoding its output  $q$  for the oracle operation. The query function  $Q$  maps available inputs  $x_i$  of the problem to certain accessible values  $q_{x_i}$ ,  $Q: x_i \mapsto q_{x_i}$  ( $i = 1, 2, \dots, K$ ). Here we clarify that  $Q$  is evaluated *classically*, and independently of the construction of the oracle operation. The finite input set  $\{x_i\}$  ( $i = 1, 2, \dots, K$ ) and the query function  $Q$  are determined prior to learning, as mentioned in section 2.

Let us now consider a general process for oracle operation, such that

$$|j\rangle |x_i\rangle \rightarrow e^{i\pi\varphi_{x_i}} |j \oplus g_{x_i}\rangle |x_i\rangle, \quad (\text{A.1})$$

where  $|j\rangle$  is a computational basis, and  $|x_i\rangle$  is a quantum state of an input  $x_i$ . Here,  $\varphi_{x_i}$  and  $g_{x_i}$  are controllable parameters depending on  $x_i$ . We then determine a specific form of oracle operation by choosing either  $(\varphi_{x_i} = 0) \wedge (g_{x_i} = q_{x_i})$  or  $(\varphi_{x_i} = q_{x_i}) \wedge (g_{x_i} = 0)$ . These two types of oracle

are equivalent, in the sense that they are independent of the query function  $Q$ , and can be converted into each other without any alteration of the complexity of the algorithm found [36]. In this work, we considered the latter type of oracle operation, as it is more economical in the sense that the query function is encoded into the phase without any additional system.

## Appendix B. The variants of the original one-bit Deutsch–Jozsa algorithm

In this appendix, we discuss the original Deutsch–Jozsa algorithm and its variants for the simple case  $n = 1$  [37]. In such a case, the learning part of our simulator consists of two single-qubit unitary operations  $\hat{U}_k$  ( $k = 1,3$ ) and one oracle operation  $\hat{U}_x$ , as in equation (4). Here it is convenient to rewrite any single-qubit unitary operation  $\hat{U}_k$  as

$$\hat{U}_k(\mathbf{p}) = \exp(-i\mathbf{p}_k^T \boldsymbol{\sigma}) = \cos \Theta_k \hat{1} - i \sin \Theta_k (\mathbf{n}_k^T \boldsymbol{\sigma}), \quad (\text{B.1})$$

where  $\mathbf{p}_k = (p_{k,x}, p_{k,y}, p_{k,z})^T$  is a three-dimensional real vector, and  $\boldsymbol{\sigma} = (\hat{\sigma}_x, \hat{\sigma}_y, \hat{\sigma}_z)^T$  is nothing but the vector of Pauli operators. Here,  $\Theta_k$  is given as the Euclidean vector norm of  $\mathbf{p}_k$ , i.e.,  $\Theta_k = \|\mathbf{p}_k\| = (\mathbf{p}_k^T \mathbf{p}_k)^{\frac{1}{2}}$ , and  $\mathbf{n}_k = \frac{\mathbf{p}_k}{\|\mathbf{p}_k\|}$  is a normalized vector. All pure states are characterized as points on the surface of a unit sphere, called the ‘Bloch sphere’, and  $\hat{U}_k$  rotates a pure state (i.e., a point on the Bloch sphere) by the angle  $2\Theta_k$  around the axis  $\mathbf{n}_k$ . Such a geometric description is convenient for describing the unitary processes.

We now turn to the one-bit DJ algorithm  $\hat{U}_1\text{--}\hat{U}_x\text{--}\hat{U}_3$  which consists of three operation steps. Firstly, the unitary  $\hat{U}_1$  rotates the initial state  $|0\rangle$  to a state on *the equator of the Bloch sphere*, i.e.,  $\frac{1}{\sqrt{2}}(|0\rangle + e^{i\phi}|1\rangle)$ , where  $\phi$  is an arbitrary phase factor. The oracle  $\hat{U}_x$  then flips the state to the antipodal side if  $x_i$  is balanced, and leaves it unchanged if  $x_i$  is constant. The last unitary  $\hat{U}_3$  transforms the incoming state to the corresponding output,

$$|\Psi_{\text{out}}(x_i)\rangle = \begin{cases} \pm |0\rangle, & \text{if } x_i \text{ is constant,} \\ \pm |1\rangle, & \text{if } x_i \text{ is balanced.} \end{cases} \quad (\text{B.2})$$

Noting that the Hadamard operation  $\hat{H}$  is  $\pi$ -rotation about the axis  $\mathbf{n} = (1/\sqrt{2}, 0, 1/\sqrt{2})^T$ , it is easily checked that the phase  $\phi$  is given as zero in the original DJ algorithm. On the basis of such a description, we can infer that there are numerous sets  $\{(\Theta_k, \mathbf{n}_k)\}$  ( $k = 1,3$ ) leading the initial state  $|0\rangle$  to the desired output  $|\Psi_{\text{out}}(x_i)\rangle$  as in equation (B.2). Thus, many variants of the original DJ algorithm exist. As an example, we give  $\hat{U}_1$  and  $\hat{U}_3$  found in our simulator as follows:

$$\begin{aligned} \hat{U}_1 &\simeq \begin{pmatrix} 0.348 + 0.612i & 0.631 - 0.325i \\ -0.631 - 0.325i & 0.348 - 0.612i \end{pmatrix}, \\ \hat{U}_3 &\simeq \begin{pmatrix} -0.360 - 0.609i & -0.031 + 0.706i \\ 0.031 + 0.706i & -0.360 + 0.609i \end{pmatrix}, \end{aligned} \quad (\text{B.3})$$

with

$$\begin{cases} \theta_1 \simeq 0.552\pi, & \mathbf{n}_1 \simeq (-0.243, 0.847, -0.472)^T, \\ \theta_3 \simeq 0.476\pi, & \mathbf{n}_3 \simeq (0.043, -0.531, 0.846)^T. \end{cases} \quad (\text{B.4})$$

The algorithm constructed with the above  $\hat{U}_1$  and  $\hat{U}_2$  runs as

$$\begin{aligned} |0\rangle &\xrightarrow{\hat{U}_1} \begin{pmatrix} 0.704 \\ -0.710e^{0.18\pi} \end{pmatrix} \xrightarrow{\hat{U}_x} \begin{pmatrix} 0.704 \\ -0.710e^{0.18\pi} \end{pmatrix} \xrightarrow{\hat{U}_3} |\psi_{\text{out}}\rangle \simeq |0\rangle, \quad \text{if } x_i \in C, \\ |0\rangle &\xrightarrow{\hat{U}_1} \begin{pmatrix} 0.704 \\ -0.710e^{0.18\pi} \end{pmatrix} \xrightarrow{\hat{U}_x} \begin{pmatrix} 0.704 \\ 0.710e^{0.18\pi} \end{pmatrix} \xrightarrow{\hat{U}_3} |\psi_{\text{out}}\rangle \simeq |1\rangle, \quad \text{if } x_i \in B. \end{aligned} \quad (\text{B.5})$$

This algorithm is not exactly equal to, but equivalent to, the original one-bit DJ algorithm.

## References

- [1] Nielsen M A and Chuang I L 2000 *Quantum Computation and Quantum Information* (Cambridge: Cambridge University Press)
- [2] Deutsch D 1985 *Proc. R. Soc. London A* **400** 97
- [3] Deutsch D and Jozsa R 1992 *Proc. R. Soc. London A* **439** 553
- [4] Shor P W 1997 *SIAM J. Comput.* **26** 1484
- [5] Grover L K 1997 *Phys. Rev. Lett.* **79** 325
- [6] Shor P 2003 *J. ACM* **50** 87
- [7] Uchiyama M 1978 *Trans. Soc. Instrum. and Contr. Eng.* **14** 706
- [8] Langley P 1996 *Elements of Machine Learning* (San Francisco, CA: Morgan Kaufmann Publishers)
- [9] Assion A, Baumert T, Bergt M, Brixner T, Kiefer B, Seyfried V, Strehle M and Gerber G 1998 *Science* **282** 919
- [10] Sasaki M and Carlini A 2002 *Phys. Rev. A* **66** 022303
- [11] Bisio A, Chiribella G, D'Ariano G M, Facchini S and Perinotti P 2010 *Phys. Rev. A* **81** 032324
- [12] Hentschel A and Sanders B C 2010 *Phys. Rev. Lett.* **104** 063603
- [13] Bang J, Lee S W, Jeong H and Lee J 2012 *Phys. Rev. A* **86** 062317
- [14] Manzano D, Pawłowski M and Brukner Č 2009 *New J. Phys.* **11** 113018
- [15] Spector L, Langdon W B, O'Reilly U-M and Angeline P 1999 *Advances in Genetic Programming III* (Cambridge, MA: MIT Press)
- [16] Behrman E C, Steck J E, Kumar P and Walsh K A 2008 *Quantum Inf. Comput.* **8** 12
- [17] Cleve R, Ekert A, Macchiavello C and Mosca M 1998 *Proc. R. Soc. London A* **454** 339
- [18] Ekert A and Jozsa R 1998 *Proc. R. Soc. London A* **356** 1769
- [19] Barnum H, Saks M and Szegedy M 2003 *Proc. XVIII Annu. IEEE Conf. on Computational Complexity*
- [20] Muggleton S and Raedt L D 1994 *J. Logic Programming* **19** 629
- [21] Gammelmark S and Mølmer K 2009 *New J. Phys.* **11** 033017
- [22] Cleve R 2000 *Collected Papers on Quantum Computation and Quantum Information Theory* ed C Macchiavello, G M Palma and A Zeilinger (Singapore: World Scientific) pp 103–27
- [23] Yoo S, Bang J, Lee C and Lee J 2013 Quantum learning speedup in binary classification arXiv:1303.6055
- [24] Hioe F T and Eberly J H 1981 *Phys. Rev. Lett.* **47** 838
- [25] Son W, Lee J and Kim M S 2004 *J. Phys. A: Math. Gen.* **37** 11897
- [26] Reck M, Zeilinger A, Bernstein H J and Bertani P 1994 *Phys. Rev. Lett.* **73** 58
- [27] Kim J, Lee J and Lee S 2000 *Phys. Rev. A* **61** 032312
- [28] Arvind and Collins D 2003 *Phys. Rev. A* **68** 052301
- [29] Adcock M R A, Høyer P and Sanders B C 2009 *New J. Phys.* **11** 103035

- [30] Collins D, Kim K W and Holton W C 1998 *Phys. Rev. A* **58** 1633
- [31] Toffoli T 1980 *Automata, Languages and Programming* ed J W de Bakker and J van Leeuwen (New York: Springer) p 632
- [32] Storn R and Price K 1997 *J. Global Optim.* **11** 341
- [33] Bang J and Lee J 2008 Quantum learning machine arXiv:[0803.2976](#)
- [34] van den Bergh F and Engelbrecht A P 2004 *IEEE Trans. Evol. Comput.* **8** 225
- [35] Chu W, Gao X and Sorooshian S 2011 *Inf. Sci.* **181** 4909
- [36] Kashefi E, Kent A, Vedral V and Banaszek K 2002 *Phys. Rev. A* **65** 050304
- [37] Bang J and Yoo S 2014 Genetic algorithm based method to construct any desired unitary transformation: basic formulation and application to one-bit oracle decision problem arXiv:[1403.2827](#)