2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Learning Quantum Operators From Quantum State Pairs

Neil Toronto and Dan Ventura

***Abstract*— Developing quantum algorithms has proven to be very difficult. In this paper, the concept of using classical machine learning techniques to derive quantum operators from examples is presented. A gradient descent algorithm for learning unitary operators from quantum state pairs is developed as a starting point to aid in developing quantum algorithms. The algorithm is used to learn the quantum Fourier transform, an underconstrained two-bit function, and Grover's iterate.**

I. Introduction

By making use of quantum superposition and entanglement as new computational resources, quantum computing holds the promise of superlinear speedup over classical computation. However, because of its inherent limitations—such as the unitary transformation requirement and the non-clonability of generic qubit states—developing quantum algorithms is difficult, which is evidenced by the paucity of quantum algorithms in general. One possible remedy is to enlist the help of classical learning algorithms.

This research area is relatively unexplored. The most notable investigations are successful attempts at using genetic algorithms to invent small decomposed quantum operators [1] [2] [3]. In [4], what may be characterized as a thought experiment is given, demonstrating a possible iterative method for training whole quantum operators. In [5], limitations of the algorithm as given in [4] are pointed out (i.e. no gaurantees of convergence nor a unitary result) and a genetic algorithm is developed to address them. However, the general approach of iteratively refining an estimate using a simple training rule is valid. This paper adopts that approach via gradient descent, addressing the same limitations as [5], to provide a solid basis for future iterative refinement techniques.

It is fairly easy to derive a quantum operator from a full set of basis states and the operator's outputs for those basis states. What is difficult, and currently not well understood, is deriving a quantum operator from a function that is *not* fully specified, or which implies a non-unitary operator. An algorithm that performed such a task would be quite useful for inventing new operators, for checking or approximating functions, and in general analysis and exploration. This paper develops just that: a classical gradient descent algorithm for learning quantum operators.

Neil Toronto and Dan Ventura are with the Department of Computer Science, Brigham Young University, Provo, UT 84602, USA (email ntoronto@cs.byu.edu and ventura@cs.byu.edu).

II. Learning Quantum Operators

This section starts with a formal description of the problem, presents a relatively well-known analytic solution, and explains why that solution is not satisfactory. An iterative solution is presented as a starting point for future, more robust algorithms.

A. Formal Description

Given a training set of quantum states $S = \{(|x_1\rangle, |y_1\rangle), ..., (|x_m\rangle, |y_m\rangle)\}$ in $\mathbb{C}^n$, we wish to train a quantum operator U, such that $U|x_k\rangle = |y_k\rangle$ for all $k \in [1..m]$. In other words, the goal is to find some unitary matrix U which is *consistent* with the training set.

Of course, this is not possible for all training sets (indeed, the set of all training sets is dense with members that imply non-unitary operators), so an approximation will have to suffice. First, as a notational convenience, let X and Y be $n \times m$ matrices $[x_1|x_2|...|x_m]$ and $[y_1|y_2|...|y_m]$. Then the goal becomes to find some unitary matrix U such that

$$UX \approx Y \tag{1}$$

Notice that $U = YX^{-1}$ if $m = n$ and YX^{-1} is unitary.

$UX \approx Y$ can be satisfied by defining and minimizing an error function, ϵ:

$$\epsilon(U) = \frac{1}{2}\|UX - Y\|_F^2 \tag{2}$$

$\|A\|_F^2$ denotes the squared *Frobenius* norm, which is defined as:

$$\|A\|_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{m}|a_{ij}|^2 = \mathrm{Tr}(AA^\dagger)$$

For real matrices, the squared Frobenius norm is equivalent to sum-squared error. For complex matrices, it is a sum-squared error analogue. Use of this norm is largely motivated by the fact that derivatives of traces of matrices have simple definitions (see the Appendix for details). Also, because there is no natural ordering over $\mathbb{C}$, $\epsilon(U)$ must be real-valued, as is the Frobenius norm.

B. The Orthogonal Procrustes Solution

Those familiar with factor analysis may immediately recognize the above formulation as an instance of the *orthogonal Procrustes problem*, the solution of which is proven in [6]. In short, if the singular value decomposition of $YX^\dagger = W\Sigma Z^\dagger$, the unitary matrix that minimizes $\|UX - Y\|_F^2$ is $U = WZ^\dagger$.

Though the solution is guaranteed to produce a correct answer given any set of training data, it cannot be extended

0-7803-9487-9/06/$20.00/©2006 IEEE

to problems with tighter constraints (factorizability, for example), nor can it be extended to problems involving "don't-care" outputs and ancillary bits. Error functions with such extended and complex constraints are better minimized using more general optimization techniques. One such algorithm is presented here.

C. Gradient Descent Training Rule

A gradient-descent algorithm is developed for the following reasons:

1) It is easy to extend criteria and add constraints.
2) It produces an answer with minimum perturbation from the first estimate. In the algorithm presented here, it tends to preserve ones along the main diagonal, which may be helpful in factorization and analysis.
3) Like all other searches that are informed by first-order derivatives, it requires the calculation of a gradient. How to derive the gradient of a real-valued function of complex matrices is not immediately obvious, and should be addressed.

The training rule takes this form:

$$U_{t+1} = U_t - \eta \nabla \epsilon(U_t)$$

where $\nabla\epsilon(U)$ is a *gradient matrix*, η is the *learning rate* and $0 < \eta \ll 1$. $\nabla\epsilon(U)$, the derivation of which is reproduced in the appendix, is simply

$$\nabla \epsilon(U) = (UX - Y)X^\dagger \tag{3}$$

where $X^\dagger$ denotes the conjugate transpose (Hermitian transpose) of X. The training rule becomes

$$U_{t+1} = U_t - \eta(U_t X - Y)X^\dagger \tag{4}$$

For small enough values of η, a process using this training rule will converge to a local least-error solution. When the input matrix, X, is invertible, it will converge to a *global* minimum with $\epsilon(U_t) = 0$. This is because, at the minimum, $(U_t X - Y)X^\dagger = \mathbf{0}$, or $U_t X = Y$. This is demonstrated in the following example.

D. Testing the Training Rule

We wish to train a 4×4 unitary operator using four linearly independent examples in order to verify that the training rule works at least as well as the closed-form solution $U = YX^{-1}$. The examples were produced by applying a quantum Fourier transform operator to instances of the well-known discrete $\mathrm{comb}_k(t)$ function for various integer values of k. Figure 1(a) shows the training inputs, and Figure 1(b) shows the target states. (Keep in mind that the quantum states are the matrix columns.) In this example, $U_0 = \mathbf{I}_4$ and $\eta = 0.1$. Training until $\epsilon(U_t) < 10^{-16}$ took 2018 epochs. Figure 1(c) is the result of training, which is indeed the quantum Fourier transform.

$$\frac{1}{2}\begin{bmatrix} 1 & \sqrt{2} & \sqrt{2} & 2 \\ 1 & 0 & 0 & 0 \\ 1 & \sqrt{2} & 0 & 0 \\ 1 & 0 & \sqrt{2} & 0 \end{bmatrix}$$

(a) Input state matrix

$$\frac{1}{2}\begin{bmatrix} 2 & \sqrt{2} & \sqrt{2} & 1 \\ 0 & 0 & e^{i7\pi/4} & 1 \\ 0 & \sqrt{2} & 0 & 1 \\ 0 & 0 & e^{i\pi/4} & 1 \end{bmatrix}$$

(b) Target state matrix

$$\frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

(c) Learned quantum operator

Fig. 1. Learning the quantum Fourier transform with $U_{t+1} = U_t - \eta(U_t X - Y)X^\dagger$.

E. Obtaining Unitary Results

For the learning algorithm to be useful in developing quantum algorithms, it must produce unitary operators. Conventional approaches to unitarization include the Graham-Schmidt procedure, and randomly rebuilding single basis vectors to be normal and orthogonal to the remaining $n-1$ basis vectors. However, it is not obvious how to use these types of procedures with a *penalty* method of constraint satisfaction, which this paper adopts.

The definition of unitary, $UU^\dagger = \mathbf{I}$, suggests the error measure

$$\upsilon(U) = \frac{1}{4}\|UU^\dagger - \mathbf{I}\|_F^2 \tag{5}$$

As before, the derivation of $\nabla\upsilon(U)$ is reproduced in the appendix. It is simply

$$\nabla \upsilon(U) = (UU^\dagger - \mathbf{I})U \tag{6}$$

The gradient descent rule, then, is

$$U_{t+1} = U_t - \nu(U_t U_t^\dagger - \mathbf{I})U_t \tag{7}$$

where ν is the learning rate and $0 < \nu \ll 1$.

Assuming ν is small enough, this rule can transform any *invertible* matrix into a unitary matrix. Indeed, when any U_t is singular, all successors are singular. For convenience, let $E_t = U_t U_t^\dagger - \mathbf{I}$. Substituting E_t and solving for $\det(U_{t+1})$,

$$\begin{aligned} U_{t+1} &= U_t - \nu E_t U_t \\ &= (\mathbf{I} - \nu E_t)U_t \\ \det(U_{t+1}) &= \det(\mathbf{I} - \nu E_t)\det(U_t) \end{aligned}$$

Thus, if $\det(U_t) = 0$, $\det(U_{t+1}) = 0$. It is therefore imperative that this process never produce a singular matrix. Fortunately, this can be guaranteed by setting ν to a value within the correct range.

At $\nu = 0$, $\det(\mathbf{I} - \nu E_t) = 1$. For values of ν between zero and the smallest positive root of the polynomial function

$\det(\mathbf{I} - \nu E_t) = 0$, $\det(\mathbf{I} - \nu E_t)$ is positive. If ν is in this range, the determinant cannot cross zero, so $\det(U_{t+1})$ cannot be zero. The roots of $\det(\mathbf{I} - \nu E_t) = 0$ can be found by using the identity $\det(kA) = k^n \det(A)$ to transform this problem into the eigenvalue problem:

$$(-1)^n \left(\frac{1}{\nu}\right)^n \det(\mathbf{I} - \nu E_t) = 0 \quad \Rightarrow \quad \det(E_t - \frac{1}{\nu}\mathbf{I}) = 0$$

Therefore, the roots of $\det(\mathbf{I} - \nu E_t) = 0$ are the reciprocals of the eigenvalues of E_t. A safe range for ν, then, is $0 < \nu < \max(\lambda_{E_t})^{-1}$, where λ_{E_t} denotes the set of real eigenvalues of E_t. Setting $\nu = \frac{1}{2}\max(\lambda_{E_0})^{-1}$ works well in practice, and if an iteration results in $\det(U_t) = 0$, ν can be recalculated. This happens infrequently—if at all—because as E_t approaches $\mathbf{0}$, $\max(\lambda_{E_0})^{-1}$ approaches infinity.

If U_t is at a minimum and is invertible, the minimum is global and equal to zero—or, U_t is unitary. This is because, at the minimum, $(U_t U_t^\dagger - \mathbf{I})U_t = \mathbf{0}$, or $U_t U_t^\dagger = \mathbf{I}$.

F. The Learning Algorithm

In the *penalty* method[1] of constraint satisfaction, the level of adherence to the constraint is assigned a real value. In this case, U_t's "unitariness" is determined from the error measure given in Equation 5. The penalty method has one strong advantage in this case: having derived the error measure's gradient, it is easy to add it to the existing training algorithm. The new iteration step is

$$\begin{aligned} U_{t+1} &= U_t - \eta\nabla\epsilon(U_t) - \nu\nabla\upsilon(U_t) \\ &= U_t - \eta(U_t X - Y)X^\dagger - \nu(U_t U_{t\dagger} - \mathbf{I})U_t \end{aligned}$$

Figure 2 shows a sketch of the algorithm. Notice that it performs gradient descent *twice*: once to train on the training set, and once to ensure that the result is unitary.

For simplicity, two aspects of the algorithm have been omitted. First, though rare, it is possible for the training loop to yield a singular matrix. The full algorithm would unitarize the last *invertible* matrix. Second, there is no code to recalculate ν_2 and roll back if unitarization produces a singular matrix. In practice, it is easy to select an initial value that is small enough to always stay within safe bounds. Using $\nu_2 = 0.01$ tends to work for any size matrix.

The ratio ν_1/η affects the behavior of the algorithm. If $\nu_1/\eta = 0$, there is no unitarization penalty, and all unitarization is performed after training. With greater values, satisfying the unitary constraint is given more importance. A ratio of 1 has worked well for many training sets, with (usually) $\nu_1 = \eta = 0.01$.

[1]A *preservation* method of constraint satisfaction would guarantee that every U_t is unitary, either by unitarizing U_t after each step or operating on U_t in such a way that successors are always unitary—for example, by reformulating the training step as a matrix multiplication and ensuring that U_t is always multiplied by another unitary matrix. However, such methods have not worked well in practice. They tend to converge very slowly and yield less fit operators than the penalty method presented here. Preservation methods work in a severely restricted subspace of $\mathbb{C}^{n\times n}$, which may cause many or most legal movements on the error surface to be opposite or orthogonal to the path of steepest descent. Determining this behavior precisely is outside the scope of this paper and is a subject of future work.

$U_0 \leftarrow \mathbf{I},\ t \leftarrow 0$

Training

`do`

$U_{t+1} = U_t - \eta(U_t X - Y)X^\dagger - \nu_1(U_t U_{t\dagger} - \mathbf{I})U_t$
$t \leftarrow t + 1$

`while` $|\epsilon(U_t) - \epsilon(U_{t-1})| > \varepsilon_1$

Post-Unitarization

`do`

$U_{t+1} = U_t - \nu_2(U_t U_{t\dagger} - \mathbf{I})U_t$
$t \leftarrow t + 1$

`while` $\upsilon(U_t) > \varepsilon_2$

Fig. 2. The gradient descent algorithm for learning quantum operators.

Because the error functions are sums of errors of matrix elements, both ε_1 and ε_2 should be directly proportional to the number of matrix elements. In practice, $\varepsilon_1 = 10^{-16}nm$ and $\varepsilon_2 = 10^{-8}n^2$ have worked well.

III. Results

This section presents the results of running the learning algorithm on two quantum state training sets.

A. An Underconstrained Training Set

Suppose a quantum algorithm requires an operator that tests a single qubit q and outputs $2q$ and $2q + 1$ with equal probability. In other words, the function is

$$\begin{aligned} |00\rangle &\rightarrow \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle) \\ |01\rangle &\rightarrow \frac{1}{\sqrt{2}}(|10\rangle + |11\rangle) \end{aligned}$$

The first question is whether a unitary operator exists that performs such a function. Second, if it exists, what is it? Figure 3 shows this problem expressed as quantum state pairs. With $\eta = \nu_1 = 0.01$, the algorithm arrived at the operator in Figure 3(c) in 4005 epochs. Post unitarization was not required because the result of training was at a global minimum of error (zero).

This example demonstrates a consistent feature of the learning algorithm's bias. First, of the infinite number possibilities for values on the right side of the matrix, it chose values that differ only by a phase shift. Second, of the possible phase shifts, it chose the set that preserved the phase of the right half of the main diagonal. This and other tests show that the learning algorithm tends to preserve features of U_0—in this case, the identity matrix. More complete characterization of this behavior may be a subject of future work.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

(a) Input state matrix

$$\frac{1}{2}\begin{bmatrix} \sqrt{2} & 0 \\ \sqrt{2} & 0 \\ 0 & \sqrt{2} \\ 0 & \sqrt{2} \end{bmatrix}$$

(b) Target state matrix

$$\frac{1}{2}\begin{bmatrix} \sqrt{2} & 0 & 1 & 1 \\ \sqrt{2} & 0 & -1 & -1 \\ 0 & \sqrt{2} & 1 & -1 \\ 0 & \sqrt{2} & -1 & 1 \end{bmatrix}$$

(c) Learned quantum operator

Fig. 3. Learning a unitary operator from an underconstrained training set.

B. Learning Grover's Iterate

The two examples so far have produced unitary matrices with zero error. What if the training set implies a non-unitary operator? One of the few known superclassical quantum algorithms, Grover's quantum search [7], suggests such a training set. An operator integral to the algorithm, called *Grover's iterate*, transforms a π phase shift in an otherwise uniform quantum state into amplitude. Figure 4 shows a training set that expresses this goal. Note that the closed-form solution $U = YX^{-1}$ is not unitary.

With $\eta = \nu_1 = \nu_2 = 0.01$, the learning algorithm, after 1212 training epochs and 395 unitarization epochs, produced the operator shown in Figure 4(c), which is indeed Grover's iterate. This and subsequent experiments indicate that the learning algorithm is able to discover Grover's iterate for any size quantum state.

IV. Conclusions and Future Work

Though inventing quantum algorithms is difficult, classical learning algorithms may be a partial solution to the problem. This learning algorithm uses gradient descent to find an operator that minimizes sum-squared error over the training set and uses a penalty method of constraint satisfaction to ensure it is unitary. It performs well on the problems presented here and on many others not presented.

It does not express the operator as a quantum circuit, but it may do well as a front end to a quantum compiler, such as that developed in [8] or [9]. This suggests an extension of the algorithm: biasing its results toward operators that are easy to factorize. Because criteria such as these are likely to be expressed in penalty constraints that are not differentiable, optimization methods that do not need derivative information may be required.

Future work may also include exploring this algorithm's usefulness as a general machine learning algorithm. How accurately can it generalize from limited training data? What kind of constraints does the unitary requirement put on hypothesis space, and how can they be characterized?

$$\frac{\sqrt{2}}{4}\begin{bmatrix} -1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) Input state matrix (b) Target state matrix

$$\frac{1}{4}\begin{bmatrix} -3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -3 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -3 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -3 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -3 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & -3 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & -3 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & -3 \end{bmatrix}$$

(c) Learned quantum operator

Fig. 4. Learning Grover's iterate.

When measurement is expected, phase differences are not error. Is there a *simple* training rule that disregards phase differences? Extending the algorithm to use ancillary bits would increase its usefulness. This requires that "don't care" outputs be expressed in the training set and an error measure that takes these into account. Solving these problems is also a subject of future work.

Appendix

This section contains the derivations of the matrix gradients $\nabla\epsilon(U)$ and $\nabla\upsilon(U)$. It is important to note that non-analytic functions may have gradients. The gradient of $f(U)$ is $\nabla f(U) = \frac{\partial}{\partial U_\Re} f(U) + i\frac{\partial}{\partial U_\Im} f(U)$, where $U_\Re$ and $U_\Im$ denote the real and imaginary parts of U, respectively [10].

Partial derivatives of functions of matrices are denoted $\frac{\partial}{\partial X} f(X)$, which is defined by $[\frac{\partial}{\partial X} f(X)]_{ij} = \frac{\partial}{\partial X_{ij}} f(X)$. In other words, $\frac{\partial}{\partial X} f(X)$ is a matrix that contains every partial derivative of $f(X)$. (For real-valued functions of real-valued matrices, this is the gradient matrix.) We will use many of the first- and second-order derivatives given in [10], relying heavily on the cyclic property of traces ($\mathrm{Tr}(ABC) = \mathrm{Tr}(CAB) = \mathrm{Tr}(BCA)$) and their invariance under transposition ($\mathrm{Tr}(A^T) = \mathrm{Tr}(A)$).

Each derivation of a gradient of the form $\nabla\|f(U)\|_F^2$ follows these steps:

1) Rewrite $\nabla\|f(U)\|_F^2$ as $\mathrm{Tr}(f(U)f(U)^\dagger)$,

2) Distribute the conjugate transpose, multiply to obtain $\mathrm{Tr}(g(U))$ where $g(U)$ is an nth-order polynomial function of matrices,
3) Distribute the trace through $g(U)$, apply the gradient operator to each term,
4) Derive the gradient for each term and reassemble.

To derive the gradients of these smaller terms,

1) Write each U as $(U_\Re + iU_\Im)$ and each $U^\dagger$ as $(U_\Re^T - iU_\Im^T)$,
2) Multiply and reduce terms,
3) Derive $\frac{\partial}{\partial U_\Re} f(U)$ and $\frac{\partial}{\partial U_\Im} f(U)$ by applying well-known identities,
4) Combine these partials into the gradient.

A. Derivation of $\nabla\epsilon(U)$

From equation 2,

$$\begin{aligned}\epsilon(U) &= \frac{1}{2}\|UX - Y\|_F^2 \\ 2\epsilon(U) &= \mathrm{Tr}((UX-Y)(UX-Y)^\dagger) \\ &= \mathrm{Tr}(UXX^\dagger U^\dagger - UXY^\dagger - YX^\dagger U^\dagger + YY^\dagger) \\ &= \mathrm{Tr}(U^\dagger UXX^\dagger) - \mathrm{Tr}(UXY^\dagger) \\ &\quad - \mathrm{Tr}(U^\dagger YX^\dagger) + \mathrm{Tr}(YY^\dagger)\end{aligned}$$

Because the gradient operator distributes over addition:

$$\begin{aligned}2\nabla\epsilon(U) = \nabla\,&\mathrm{Tr}(U^\dagger UXX^\dagger) - \nabla\,\mathrm{Tr}(UXY^\dagger) \\ &- \nabla\,\mathrm{Tr}(U^\dagger YX^\dagger)\end{aligned}$$

$\mathrm{Tr}(YY^\dagger)$ is ignored because it is constant. This leaves three gradients of traces, which will be considered in turn.

The simplest is $\nabla\,\mathrm{Tr}(UXY^\dagger)$. Because $\nabla f(U) = \frac{\partial}{\partial U_\Re} f(U) + i\frac{\partial}{\partial U_\Im} f(U)$, U must be expressed in its real and imaginary parts explicitly:

$$\begin{aligned}\mathrm{Tr}(UXY^\dagger) &= \mathrm{Tr}((U_\Re + iU_\Im)(XY^\dagger)) \\ &= \mathrm{Tr}(U_\Re XY^\dagger) + i\,\mathrm{Tr}(U_\Im XY^\dagger)\end{aligned}$$

Using $\frac{\partial}{\partial X}\mathrm{Tr}(XB) = B^T$ and $\frac{\partial}{\partial X}\mathrm{Tr}(B) = \mathbf{0}$,

$$\begin{aligned}\frac{\partial}{\partial U_\Re}\mathrm{Tr}(UXY^\dagger) &= (Y^\dagger)^T X^T \\ \frac{\partial}{\partial U_\Im}\mathrm{Tr}(UXY^\dagger) &= i(Y^\dagger)^T X^T \\ \nabla\,\mathrm{Tr}(UXY^\dagger) &= \frac{\partial}{\partial U_\Re}\mathrm{Tr}(UXY^\dagger) + i\frac{\partial}{\partial U_\Im}\mathrm{Tr}(UXY^\dagger) \\ &= (Y^\dagger)^T X^T - (Y^\dagger)^T X^T \\ &= \mathbf{0}\end{aligned}$$

$\nabla\,\mathrm{Tr}(U^\dagger YX^\dagger)$ proceeds similarly:

$$\begin{aligned}\mathrm{Tr}(U^\dagger YX^\dagger) &= \mathrm{Tr}((U_\Re^T - iU_\Im^T)(YX^\dagger)) \\ &= \mathrm{Tr}(U_\Re^T YX^\dagger) - i\,\mathrm{Tr}(U_\Im^T YX^\dagger)\end{aligned}$$

Using the additional identity $\frac{\partial}{\partial X}\mathrm{Tr}(X^T B) = B$,

$$\begin{aligned}\frac{\partial}{\partial U_\Re}\mathrm{Tr}(U^\dagger YX^\dagger) &= YX^\dagger \\ \frac{\partial}{\partial U_\Im}\mathrm{Tr}(U^\dagger YX^\dagger) &= -iYX^\dagger \\ \nabla\,\mathrm{Tr}(U^\dagger YX^\dagger) &= YX^\dagger + YX^\dagger \\ &= 2YX^\dagger\end{aligned}$$

Finally, $\nabla\,\mathrm{Tr}(U^\dagger UXX^\dagger)$:

$$\begin{aligned}\mathrm{Tr}(U^\dagger UXX^\dagger) = \mathrm{Tr}(U_\Re^T U_\Re XX^\dagger) &+ i\,\mathrm{Tr}(U_\Re^T U_\Im XX^\dagger) \\ - i\,\mathrm{Tr}(U_\Im^T U_\Re XX^\dagger) &+ \mathrm{Tr}(U_\Im^T U_\Im XX^\dagger)\end{aligned}$$

Using previous first-order identities and $\frac{\partial}{\partial X}\mathrm{Tr}(X^T XB) = X(B + B^T)$,

$$\begin{aligned}\frac{\partial}{\partial U_\Re}\mathrm{Tr}(U^\dagger UXX^\dagger) &= U_\Re(XX^\dagger) + U_\Re(XX^\dagger)^T \\ &\quad + iU_\Im(XX^\dagger) - iU_\Im(XX^\dagger)^T \\ \frac{\partial}{\partial U_\Im}\mathrm{Tr}(U^\dagger UXX^\dagger) &= iU_\Re(XX^\dagger)^T - iU_\Re(XX^\dagger) \\ &\quad + U_\Im(XX^\dagger) + U_\Im(XX^\dagger)^T\end{aligned}$$

$$\begin{aligned}\nabla\,\mathrm{Tr}(U^\dagger UXX^\dagger) &= U_\Re(XX^\dagger) + U_\Re(XX^\dagger)^T \\ &\quad + iU_\Im(XX^\dagger) - iU_\Im(XX^\dagger)^T \\ &\quad - U_\Re(XX^\dagger)^T + U_\Re(XX^\dagger) \\ &\quad + iU_\Im(XX^\dagger) + iU_\Im(XX^\dagger)^T \\ &= 2U_\Re XX^\dagger + 2iU_\Im XX^\dagger \\ &= 2(U_\Re + iU_\Im)XX^\dagger \\ &= 2UXX^\dagger\end{aligned}$$

The gradient of $\epsilon(U)$, then, is

$$\begin{aligned}2\nabla\epsilon(U) &= \nabla\,\mathrm{Tr}(U^\dagger UXX^\dagger) - \nabla\,\mathrm{Tr}(UXY^\dagger) \\ &\quad - \nabla\,\mathrm{Tr}(U^\dagger YX^\dagger) \\ &= 2UXX^\dagger - \mathbf{0} - 2YX^\dagger \\ \nabla\epsilon(U) &= UXX^\dagger - YX^\dagger \\ &= (UX - Y)X^\dagger\end{aligned}$$

B. Derivation of $\nabla\upsilon(U)$

From equation 5,

$$\begin{aligned}\upsilon(U) &= \frac{1}{4}\|UU^\dagger - \mathbf{I}\|_F^2 \\ 4\upsilon(U) &= \mathrm{Tr}((UU^\dagger - \mathbf{I})(UU^\dagger - \mathbf{I})) \\ &= \mathrm{Tr}(UU^\dagger UU^\dagger) - 2\,\mathrm{Tr}(UU^\dagger) + \mathrm{Tr}(\mathbf{I}) \\ 4\nabla\upsilon(U) &= \nabla\,\mathrm{Tr}(UU^\dagger UU^\dagger) - 2\nabla\,\mathrm{Tr}(UU^\dagger)\end{aligned}$$

First, to derive $\nabla \operatorname{Tr}(UU^\dagger)$,

$$\begin{aligned}\operatorname{Tr}(UU^\dagger) &= \operatorname{Tr}(U_\Re U_\Re^T - iU_\Re U_\Im^T + iU_\Im U_\Re^T + U_\Im U_\Im^T) \\ &= \operatorname{Tr}(U_\Re U_\Re^T) + \operatorname{Tr}(U_\Im U_\Im^T)\end{aligned}$$

The middle terms cancel because $\operatorname{Tr}(U_\Re U_\Im^T) = \operatorname{Tr}(U_\Im U_\Re^T)$. Using the identity $\frac{\partial}{\partial X}\operatorname{Tr}(X^T X) = \frac{\partial}{\partial X}\operatorname{Tr}(XX^T) = 2X$,

$$\begin{aligned}\frac{\partial}{\partial U_\Re}\operatorname{Tr}(UU^\dagger) &= 2U_\Re \\ \frac{\partial}{\partial U_\Im}\operatorname{Tr}(UU^\dagger) &= 2U_\Im \\ \nabla \operatorname{Tr}(UU^\dagger) &= 2U\end{aligned}$$

For the fourth-order term $\nabla \operatorname{Tr}(UU^\dagger UU^\dagger)$:

$$\operatorname{Tr}(UU^\dagger UU^\dagger) = \operatorname{Tr}((U_\Re U_\Re^T - iU_\Re U_\Im^T + iU_\Im U_\Re^T + U_\Im U_\Im^T)^2)$$

Multiplying this out gives a 16-term polynomial; however, reductions based on trace cycles and transposes yield:

$$\begin{aligned}\operatorname{Tr}(UU^\dagger UU^\dagger) = {} & \operatorname{Tr}(U_\Re U_\Re^T U_\Re U_\Re^T) + 2\operatorname{Tr}(U_\Re U_\Re^T U_\Im U_\Im^T) \\ & - 2\operatorname{Tr}(U_\Re U_\Im^T U_\Re U_\Im^T) + 2\operatorname{Tr}(U_\Re U_\Im^T U_\Im U_\Re^T) \\ & + \operatorname{Tr}(U_\Im U_\Im^T U_\Im U_\Im^T)\end{aligned}$$

Using the additional identities $\frac{\partial}{\partial X}\operatorname{Tr}(XX^T XX^T) = 4XX^T X$, $\frac{\partial}{\partial X}\operatorname{Tr}(XX^T B) = BX + B^T X$ and $\frac{\partial}{\partial X}\operatorname{Tr}(XBXC) = C^T X^T B^T + B^T X^T C^T$,

$$\begin{aligned}\frac{\partial}{\partial U_\Re}\operatorname{Tr}(UU^\dagger UU^\dagger) = {} & 4U_\Re U_\Re^T U_\Re + 4U_\Im U_\Im^T U_\Re \\ & - 4U_\Im U_\Re^T U_\Im + 4U_\Re U_\Im^T U_\Im \\ \frac{\partial}{\partial U_\Im}\operatorname{Tr}(UU^\dagger UU^\dagger) = {} & 4U_\Re U_\Re^T U_\Im - 4U_\Re U_\Im^T U_\Re \\ & + 4U_\Im U_\Re^T U_\Re + 4U_\Im U_\Im^T U_\Im \\ \nabla \operatorname{Tr}(UU^\dagger UU^\dagger) = {} & 4(U_\Re + iU_\Im)(U_\Re^T - iU_\Im^T)(U_\Re + iU_\Im) \\ = {} & 4UU^\dagger U\end{aligned}$$

Substituting back to solve $\nabla \upsilon(U)$,

$$\begin{aligned}4\nabla\upsilon(U) &= 4UU^\dagger U - 2(2U) \\ \nabla\upsilon(U) &= UU^\dagger U - U \\ &= (UU^\dagger - \mathbf{I})U\end{aligned}$$

ACKNOWLEDGMENTS

Thanks to Dr. Wayne W. Barrett for his suggestion for the unitarization component of the learning algorithm, and Adam Peterson for his help in establishing bounds on ν.

REFERENCES

[1] L. Spector, H. J. Bernstein, H. Barnum, and N. Swamy, "Finding a better-than-classical quantum AND/OR algorithm using genetic programming," in *IEEE Proceedings of 1999 Congress on Evolutionary Computation*, 1999, pp. 2239–2246.

[2] L. Spector, *Automatic Quantum Computer Programming: A Genetic Programming Approach*, ser. Genetic Programming. Kluwer Academic Publishers, 2004, vol. 7.

[3] P. Massey, J. A. Clark, and S. Stepney, "Evolving quantum circuits and programs through genetic programming," in *Genetic and Evolutionary Computation Conference: GECCO 2004, Seattle, USA, June 2004*, ser. LNCS, vol. 3103. Springer, 2004, pp. 569–580.

[4] D. Ventura, "Learning quantum operators," in *Proceedings of the Joint Conference on Information Sciences*, March 2000, pp. 750–752.

[5] G. Giraldi, R. Thess, and J. Faber, "Learning linear operators by genetic algorithms." LNCC—National Laboratory for Scientific Computing, Tech. Rep., May 2003. [Online]. Available: ftp://ftp.lncc.br/pub/report/rep03/rep0503.ps.Z

[6] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge University Press, 1985, p. 431.

[7] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the 28th Annual ACM Symposium on Theory of Computation*, 1996, pp. 212–219.

[8] C. P. Williams and A. G. Gray, "Automated design of quantum circuits," in *Quantum Computing and Quantum Communications: First NASA International Conference, QCQC'98*, ser. Lecture Notes in Computer Science, C. P. Williams, Ed., vol. 1509. Palm Springs, California, USA: Springer-Verlag GmbH, Feb. 1998, pp. 113–125.

[9] R. R. Tucci, "A rudimentary quantum compiler(2cnd ed.)," 1999. [Online]. Available: http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:quant-ph/9902062

[10] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," 2005, version 20050105. [Online]. Available: http://www2.imm.dtu.dk/pubdb/p.php?3274