Calculating the Gradient of a Cost Function for a Parametric Quantum Circuit in FIVE EASY PIECES

Robert R. Tucci
P.O. Box 226
Bedford, MA 01730
tucci@ar-tiste.com
www.artiste-qb.net

March 24, 2019

Hybrid Quantum Classical (HQC) computation as being pursued by Rigetti Inc. involves minimizing the mean value of a Hermitian operator, wherein that mean value is calculated empirically from the data yielded by Rigetti's qc. One very popular method of minimizing a function is using back propagation (BP). The software PennyLane by Xanadu Inc. and the software Qubiter that I manage for Artiste-qb.net Inc. can already do BP to a limited extent on the Rigetti virtual and real qc's. The goal of this brief article is to discuss some ideas that might allow us to exploit BP to its full potential in HQC in the future.

Suppose that you want to minimize a cost function $C(x)$ where x has components x_k. BP is a way of calculating $\nabla C(x)$ which one then uses to update the value of x iteratively using what is called gradient descent:

$$x_k^{new} = x_k - \eta \frac{\partial C(x)}{\partial x_k} \ , \tag{1}$$

where $\eta > 0$. The idea behind gradient descent is that the increment in cost is $dC = (x_k^{new} - x_k)(dx_k)(-1/\eta)$, which we expect to be negative since $(x_k^{new} - x_k)(dx_k) \approx (dx_k)^2 > 0$. If $dC < 0$ each time, then we expect the cost will move towards a minimum. At this point, the question arises, what $C(x)$ should one use for HQC and how does one calculate it and its gradient?

Qubiter currently minimizes $C(x) = C_{exact}(x)$, where $C_{exact}(x)$ is the cost function calculated exactly (theoretically) from the wavefunction calculated by the Qubiter simulator. This is an interesting case because, to calculate ∇C_{exact}, one can do BP distributively, using GPU and TPU, via softwares like TensorFlow and PyTorch. It is known that the computational complexity of back propagation (BP) and forward propagation (FP) are about the same. Since doing a classical simulation of a quantum circuit (in other words, doing FP) blows up exponentially with the number of qubits, the same will be true if we attempt to do BP. So why do either? The motivations for doing BP to calculate ∇C_{exact} on a quantum simulator are the same as those for doing FP on a quantum simulator. One does it to generate new ideas and to test various things on a smallish number of qubits.

Ultimately, the HQC people will want to minimize $C(x) = C_{emp}(x)$, instead of $C(x) = C_{exact}(x)$, where $C_{emp}(x)$ is the cost function calculated empirically, from the data yielded by the qc hardware. Presumably, qc hardware can calculate $C_{emp}(x)$ and its gradient much faster that classical hardware can calculate $C_{exact}(x)$ and it gradient. Calculating the gradient of $C_{emp}(x)$ has many potential benefits, but it is not obvious what is the best way of doing this. Finding a good way will require some novel and careful thinking. If one does something like BP, this will require calculating the derivatives $\frac{\partial}{\partial x_k}$ of each gate that depends on the parameters x of the quantum circuit. C_{emp} is a statistical quantity compiled from many "shots" (samples), so it fluctuates, and a naive calculation of the derivatives of a gate by a finite difference method, with no other type of averaging, is bound to fail. It's difficult to calculate meaningfully the difference of two values that are very close and fluctuating.

Next, I will show how to calculate the derivative of a quantum gate with respect to one of its parameters, by calculating 5 separate mean values, and obtaining the derivative as a linear combination of those 5 mean values. The authors of PennyLane have come up with a similar scheme, but their method is different to mine. To tackle a general U(2) transformation, they first decompose it into an Euler product of 3 rotations along the standard X,Y or Z axes, and then they take the derivatives of each of those 3 rotations. In this paper, I give a method that can handle an arbitrary U(2) transformation, without having to do the Euler decomposition first. Nevertheless, check their method out! You might like it more than the method I propose here.

Lucky for us, the parameters of a quantum gate almost always appear inside a 2-dim unitary matrix (an element of the group U(2)). So, from here on, we will only concern ourselves with calculating the derivatives of a U(2) matrix.

Let's start easy, with a rotation about a standard axis, X, Y, Z, instead of a general U(2) matrix. If we let σ_k for $k = 1, 2, 3$ denote the Pauli matrices, then a rotation about the Z axis is

$$U(\theta_3) = e^{i\sigma_3\theta_3} = C + i\sigma_3 S \ , \tag{2}$$

where θ_3 is some real number and we abbreviate $S = \sin\theta_3, C = \cos\theta_3$. Then

$$\frac{dU}{dt} = \dot{\theta}_3(-S + i\sigma_3 C) \ . \tag{3}$$

Hence

$$\frac{dU}{d\theta_3} = -S + i\sigma_3 C = e^{i(\frac{\pi}{2}+\theta_3)\sigma_3} = U(\frac{\pi}{2} + \theta_3) \ . \tag{4}$$

Thus, for a rotation along a standard axis, one can evaluate the derivative of a gate simply by replacing that gate by that gate with its angle advanced by $\frac{\pi}{2}$. No need to take finite differences. This begs the question, can we calculate the gradient of a general U(2) matrix, in an exact, closed form that is just as convenient? Yes we can, as I will show next.

Now let us consider the most general U(2). We will parameterize it as

$$U = e^{i(\theta_0+\theta_1\sigma_X+\theta_2\sigma_Y+\theta_3\sigma_Z)} \ , \tag{5}$$

where θ_k for $k = 0, 1, 2, 3$ are real numbers. Derivatives with respect to θ_0 are trivial so we will set $\theta_0 = 0$ henceforth. Expressing things using the Einstein summation convention,

$$U = e^{i\sigma_k\theta_k} = C + i\sigma_k\frac{\theta_k}{\theta}S \ , \tag{6}$$

where we are abbreviating

$$\theta = \sqrt{\theta_k\theta_k}, S = \sin\theta, C = \cos\theta \ . \tag{7}$$

Then, it's easy to show that

$$\frac{dU}{dt} = -S\frac{\theta_k}{\theta}\dot{\theta}_k + i\sigma_k\dot{\theta}_r\left[\frac{\theta_k\theta_r}{\theta^2}C + \frac{S}{\theta}(-\frac{\theta_k\theta_r}{\theta^2} + \delta_{k,r})\right] . \tag{8}$$

Eq.(8) has been checked numerically by Qubiter's code. and is already coded into Qubiter's implementation of Autograd.

In the rest of this article, we will try to recast the right hand side of Eq.(8) into a form that is more convenient for empirical calculation from qc data. Before embarking on this task, let us introduce some notation. As physicists are fond of doing, we will represent a unit vector by a letter with a caret above it: $\hat{a} = \frac{\vec{a}}{|\vec{a}|}$. Also, for any 3-dim vector $\vec{a}$, let

$$\sigma_{\vec{a}} = \vec{a}\cdot\vec{\sigma} . \tag{9}$$

Eq.(9) is a natural generalization of the Pauli matrix notation. If $\hat{e}_A$ is the unit vector in direction A for $A = X, Y, Z$, then $\hat{e}_A \cdot \vec{\sigma} = \sigma_A$ for $A = X, Y, Z$. Expressed in the notation of Eq.(9), two familiar Pauli matrix identities are

$$\sigma_{\vec{a}}\sigma_{\vec{b}} = \vec{a}\cdot\vec{b} + i\sigma_{\vec{a}\times\vec{b}} , \tag{10}$$

where $\vec{a}$ and $\vec{b}$ are any two 3-dim vectors, and

$$e^{i\theta\sigma_{\hat{n}}} = \cos\theta + i\sigma_{\hat{n}}\sin\theta , \tag{11}$$

where θ is a real number and $\hat{n}$ is a 3-dim unit vector. Eq.(11) can be proven by Taylor expanding, and using $\sigma_{\hat{n}}^2 = 1$.

We will also use the following notation for the projectors P_0 and P_1 along the direction of $|0\rangle$ and $|1\rangle$, respectively, in a 1-qubit space.

$$\begin{array}{l} n = P_1 = |1\rangle\langle 1|, \\ \bar{n} = 1 - n = P_0 = |0\rangle\langle 0| \end{array} . \tag{12}$$

n is often called the number operator. Whenever we say $\Omega(\alpha)$ for a 1-qubit operator Ω, we mean Ω applied to qubit α.

As usual, let $U \in U(2)$. Our next goal is to express , $\dot{U}$, the derivative of U with respect to a parameter t, as a linear combination of unitary operators U_k, where the real numbers x_k sum to one:

$$\dot{U} = \sum_k x_k U_k , \tag{13a}$$

$$\sum_k x_k = 1, \;\; U_k U_k^\dagger = 1 \quad \forall k \; . \tag{13b}$$

It's important to note that we don't require $x_k > 0$ for all k, so the x_k are not probabilities.

Why do we want to express $\dot{U}$ in the form of Eqs.(13)? Because in a qc, $\dot{U}$ will often be subject to 1 or more controls. Controls are easy to deal with if $\dot{U}$ can be expressed as in Eqs.(13), because then

$$\dot{U}(0)^{n(1)n(2)} = \left[\sum_k x_k U_k\right]^{n(1)n(2)} = \sum_k x_k U_k(0)^{n(1)n(2)} \; . \tag{14}$$

The fact that Eq.(14) holds can be easily verified by considering the two cases $n(1)n(2) = 0, 1$ separately. If LHS=left hand side, RHS=right hand side, refer to the two sides of Eq.(14):

$$\begin{array}{lll} n(1)n(2) = 0: & LHS = 1, & RHS = \sum_k x_k = 1 \\ n(1)n(2) = 1: & LHS = \dot{U}(0), & RHS = \sum_k x_k U_k(0) \end{array} \; . \tag{15}$$

Eq.(8) for the general form of $\dot{U}$ when parameterized as Eq.(5) (with $\theta_0 = 0$) is fairly opaque. To clarify Eq.(8), we start by specializing it to $t = \theta_1$. The cases $t = \theta_2, \theta_3$ can be obtained from the case $t = \theta_1$ simply by replacing 1 subscripts in our final result by 2 or 3. So, setting $t = \theta_1$ in Eq.(8), we immediately get:

$$\frac{\partial U}{\partial \theta_1} = \left\{ \begin{array}{l} -\frac{\theta_1 S}{\theta^2}\left[i\sigma_{\hat{\theta}}\right] \\ +\frac{\theta_1}{\theta}\left[-S + i\sigma_{\hat{\theta}} C\right] \\ +\frac{S}{\theta}\left[i\sigma_1\right] \end{array} \right. \; . \tag{16}$$

If we define

$$p_1 = \frac{\theta_1}{\theta}, \;\; p_S = \frac{S}{\theta} \; , \tag{17}$$

then

$$\frac{\partial U}{\partial \theta_1} = \left\{ \begin{array}{l} -p_1 p_S \left[e^{i\frac{\pi}{2}\sigma_{\hat{\theta}}}\right] \\ +p_1 \left[e^{i(\frac{\pi}{2}+\theta)\sigma_{\hat{\theta}}}\right] \\ +p_S \left[e^{i\frac{\pi}{2}\sigma_1}\right] \end{array} \right. \; , \tag{18}$$

which is equivalent to

$$\boxed{\frac{\partial U}{\partial \theta_1} = \left\{ \begin{array}{l} \frac{1}{2}(1-p_1)(1-p_S)[1] \\ +\frac{1}{2}(1-p_1)(1-p_S)[-1] \\ -p_1 p_S \left[e^{i\frac{\pi}{2}\sigma_{\hat{\theta}}}\right] \\ +p_1 \left[e^{i(\frac{\pi}{2}+\theta)\sigma_{\hat{\theta}}}\right] \\ +p_S \left[e^{i\frac{\pi}{2}\sigma_1}\right] \end{array} \right. .} \tag{19}$$

But note that

$$p_1 + p_S - p_1 p_S + (1-p_1)(1-p_S) = 1 \ . \tag{20}$$

If we let U_k equal the unitary matrices inside the square brakets in the RHS of Eq.(19), then it is clear that Eq.(19) satisfies

$$\frac{\partial U}{\partial \theta_1} = \sum_{k=1}^{5} x_k U_k \ , \tag{21a}$$

and

$$\sum_{k=1}^{5} x_k = 1, \quad U_k U_k^\dagger = 1 \quad \forall k \ . \tag{21b}$$

Just what we wanted! $\frac{\partial U}{\partial \theta_1}$ in five easy pieces. And replace 1 by 2 or 3 in Eq.(19) to get partials of U with respect to θ_2 and θ_3.