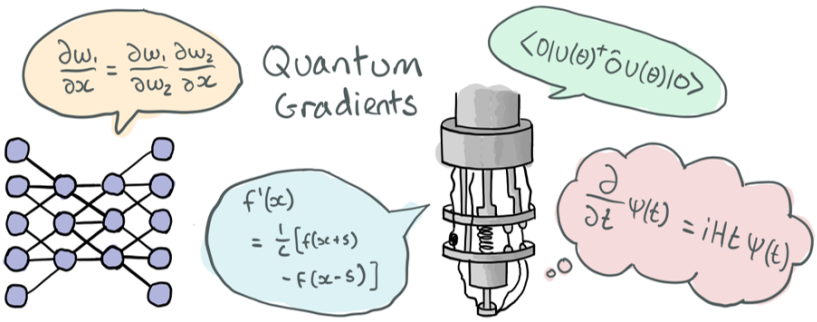

Quantum Gradients: Finding the Natural Gradient [500 points]

Version: 1

NOTE: Coding templates are provided for all challenge problems at this link. You are strongly encouraged to base your submission off the provided templates.

Overview: Quantum Gradients challenges

In this set of challenges, you'll explore various methods for computing gradients of variational quantum circuits.

Variational quantum circuits are quantum circuits that apply a parametrized unitary $U(\theta)$ to an initial state $|\psi\rangle$, and output a measurement statistic, such as the expectation value (average value) of a measurement operator $\hat{O}$. Crucially, the parameters (θ) of the circuit and the output measurement statistics are both real classical data, and the output is deterministic—given a specific parameter input θ, the expectation value will always be the same. Note that when running on quantum hardware, we estimate expectation values using a finite set of samples, so we might see some small fluctuations, but in the limit of infinite samples the expectation value is fixed. Quantum simulators allow us to estimate expectation values exactly (up to machine precision).

Variational quantum circuits are therefore differentiable functions, that map gate parameters to an expectation value. There are various ways we can differentiate a variational quantum circuit:

1. **Numerical differentiation:** We can treat the circuit as a mathematical black box, and vary the parameters an infinitesimal amount using the finite-difference method. Numerical differentiation is only an approximation and can be unstable, but it is one available technique for near-term quantum

hardware.

2. **The parameter-shift rule:** The parameter-shift rule provides a method of computing **exact** gradients of variational quantum circuits on near-term hardware, by shifting each parameter θ by a **constant**, **large** amount. Unlike the finite-difference method, it is not an approximation, but it only works for a subset of quantum gates.

3. **Backpropagation:** Backpropagation, or "reverse-mode" differentiation, powers machine-learning frameworks such as TensorFlow and PyTorch. During function execution, intermediate computations are stored, and later accumulated to produce the gradient. This introduces only a constant overhead to compute the gradient, making it more efficient than the other methods. This can easily be applied to quantum computing simulations, and there are even versions of backpropagation that are more efficient if the computation is unitary, like it is with quantum computing. However, backpropagation is not compatible with hardware, since we would need to 'peek' at the quantum state during execution.

Problem statement [500 points]

In this challenge, you will be tasked with calculating the Fubini-Study metric and using it to find the quantum natural gradient (QNG). This quantity is used within PennyLane's QNG optimizer.

Suppose we have a quantum state $|\psi(\boldsymbol{\theta})\rangle$ that depends on a collection of N parameters $\boldsymbol{\theta}$. The Fubini-Study metric allows us to quantify how $|\psi(\boldsymbol{\theta})\rangle$ behaves as we change its parameters. It is an $N \times N$ matrix F and its elements can be calculated using:

$$\begin{aligned} F_{ij}(\boldsymbol{\theta}) = \frac{1}{8}\Big[&- |\langle\psi(\boldsymbol{\theta})|\psi(\boldsymbol{\theta} + (\hat{\mathbf{e}}_i + \hat{\mathbf{e}}_j)\pi/2)\rangle|^2 \\ &+ |\langle\psi(\boldsymbol{\theta})|\psi(\boldsymbol{\theta} + (\hat{\mathbf{e}}_i - \hat{\mathbf{e}}_j)\pi/2)\rangle|^2 \\ &+ |\langle\psi(\boldsymbol{\theta})|\psi(\boldsymbol{\theta} + (-\hat{\mathbf{e}}_i + \hat{\mathbf{e}}_j)\pi/2)\rangle|^2 \\ &- |\langle\psi(\boldsymbol{\theta})|\psi(\boldsymbol{\theta} - (\hat{\mathbf{e}}_i + \hat{\mathbf{e}}_j)\pi/2)\rangle|^2 \Big], \end{aligned}$$

where $\hat{\mathbf{e}}_i$ is a unit vector along axis i, i.e., a vector of zeros with a 1 in the i-th slot.

The QNG is an N-dimensional vector that is defined as

$$F^{-1}(\boldsymbol{\theta})\nabla f(\boldsymbol{\theta}),$$

where F^{-1} is the matrix inverse of F and $\nabla f(\boldsymbol{\theta})$ is the gradient of the output of the variational circuit $f(\boldsymbol{\theta})$. Note that sometimes F is non-invertible, so

more generally one can take the matrix pseudoinverse, although this will not be required for this challenge. Check out the QNG tutorial for an understanding of why the natural gradient is useful.

PennyLane already has some functionality for calculating the natural gradient: see `qml.metric_tensor` which is used by the `qml.QNGOptimizer`. However, only the *block-diagonal approximation* of the metric tensor is presently calculated! See the original QNG paper for more context.

Your task is to use the equation above for $F_{ij}(\boldsymbol{\theta})$ to calculate the *exact* metric tensor, then use it to find the quantum natural gradient for an input value of $\boldsymbol{\theta}$.

You will be provided with a variational quantum circuit composed of two trainable layers with three parameters each. This circuit (called `variational_circuit`) is further combined with an expectation value measurement, resulting in the `qnode` function which represents $f(\boldsymbol{\theta})$. Both `variational_circuit` and `qnode` have a single `params` argument used to specify the six trainable parameters. You should edit the `natural_gradient` function to calculate the Fubini-Study metric and the gradient of the QNode, and then combine these together to find the natural gradient evaluated at the `params`.

You should calculate the Fubini-Study metric using the equation above. This may require defining another QNode which returns the quantum state before the expectation value measurement. Using the `qml.metric_tensor` function will result in a *block-diagonal approximation* of the metric tensor. However, the non-zero elements of the evaluated `qml.metric_tensor` are correct and can be used for comparison against the full Fubini-Study metric that you calculate.

Input

Your solution should be provided in the `natural_gradient` function. The inputs to this function are `params`, specifying the 6 parameters that the natural gradient should be evaluated with respect to.

Output

The `natural_gradient` function should return the natural gradient as a flat NumPy array of length 6.

Acceptance Criteria

In order for your submission to be judged as "correct":

- The outputs generated by your solution when run with a given `.in` file must match those in the corresponding `.ans` file to within the `Tolerance` specified below.
- Your solution must take no longer than the `Time limit` specified below to produce its outputs.

You can test your solution by passing the `#.in` input data to your program as stdin and comparing the output to the corresponding `#.ans` file:

```
python3 quantum_gradients_500_template.py < 1.in
```

WARNING: Don't modify any of the code in the template outside of the `# QHACK #` markers, as this code is needed to test your solution. Do not add any print statements to your solution, as this will cause your submission to fail.

Specs
Tolerance: **0.01 (1%)**
Timelimit: **20 s**

Version History

Version 1: Initial document.