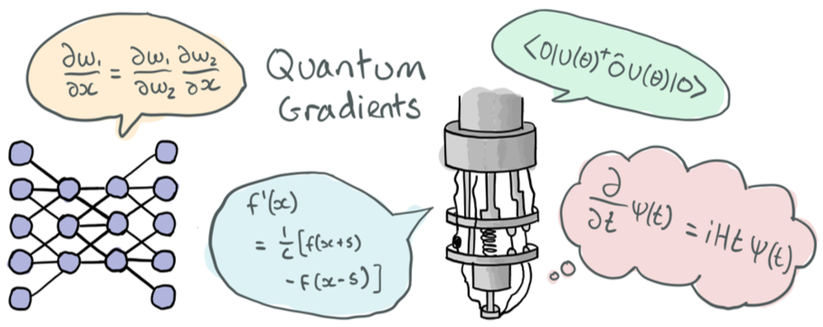

$\frac{\partial w_1}{\partial x} = \frac{\partial w_1}{\partial w_2}\frac{\partial w_2}{\partial x}$
Quantum Gradients
$\langle 0|U(\theta)^{+}\hat{O}U(\theta)|0\rangle$
$f'(x) = \frac{1}{2}[f(x+s) - f(x-s)]$
$\frac{\partial}{\partial t}\psi(t) = iHt\,\psi(t)$

Quantum Gradients: Exploring Quantum Gradients [100 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 [100 points]

Compute the gradient of the provided QNode using the parameter-shift rule:

$$\frac{\partial f}{\partial \theta_i} = \frac{f(\theta + s\hat{\mathbf{e}}_\mathbf{i}) - \mathbf{f}(\theta - \mathbf{s}\hat{\mathbf{e}}_\mathbf{i})}{2\sin(s)},$$

where $f(\theta) = \langle\psi|\mathbf{U}^\dagger(\theta)\hat{\mathbf{O}}\mathbf{U}^\dagger(\theta)|\psi\rangle$ is the output of the variational circuit (the measured expectation value), and $\hat{\mathbf{e}}_\mathbf{i}$ the ith unit vector.

In other words, to compute the gradient of the ith parameter, the expectation value is computed with the ith parameter shifted **forward** and **backward**, the difference taken, and finally divided by a constant factor $2\sin(s)$.

If you're stuck, you might find the following hints helpful:

- PennyLane documentation
- Quantum Gradients demo

Input

You will have access to an `ansatz` function that takes in trainable parameters `weights` with size `(2, 3)`.

You must then fill in the `parameter_shift` function so that it computes the gradient of `ansatz` given some values for `weights`. You may choose any shift value s you like.

Output

The output of your program should be a `np.float64` NumPy array of size `(2, 3)`, matching the input shape of the weights. Each element in the returned array is the gradient of the corresponding weight.

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_100_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: **0.001 (0.1%)**
Timelimit: **30 s**

Version History

Version 1: Initial document.