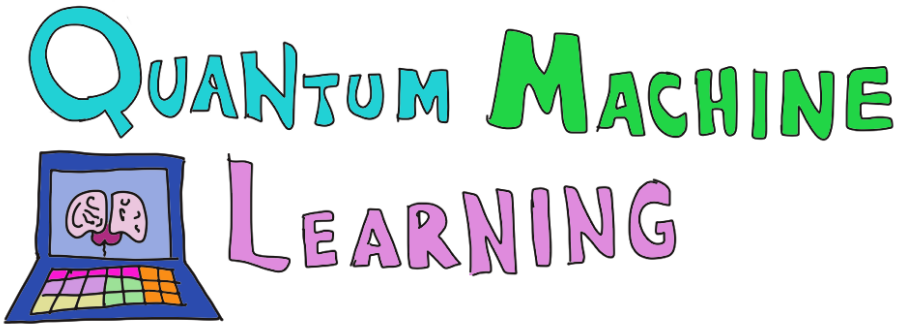

QML: Generating Fourier Basis [100 points]

Version: 2

Quantum Machine Learning

In this set of challenges, you'll explore methods and applications for training variational quantum circuits and quantum neural networks. Both play critical roles in quantum machine learning. They typically have a layered structure, and a set of tunable parameters that are learned through training with a classical optimization algorithm.

The circuit structure, optimization method, and how data is embedded in the circuit varies heavily across algorithms, and there are often problem-specific considerations that affect how they are trained. In the **Quantum Machine Learning** challenges, you'll explore how to construct and optimize a diverse set of circuits from the ground up, and become acquainted with some of today's popular quantum machine learning and optimization algorithms.

Problem statement [100 points]

Nearly all of the time, the basis that we work in is the eigenbasis of the $\hat{\sigma}^z$ operator. We will henceforth refer to this basis as "the computational basis." However, there are many other bases to work in, such as the Fourier basis, that have their individual use cases. To go from the computational basis to the Fourier basis, we use the QFT (quantum Fourier Transform) operator whose inverse QFT^{-1} returns us to the computational basis.

In this challenge, you will create the QFT operation from scratch by training a variational circuit. Your circuit ansatz (you have to code this) is given in Figure 1. You must optimize the angles θ_i in each of the z-rotation gates such that the resulting quantum state is $\text{QFT}|m\rangle$ for a certain integer m. For example, for `m = 3` and `n_qubits = 4` we would generate the state $\text{QFT}|0011\rangle$ (3 in binary is 11).

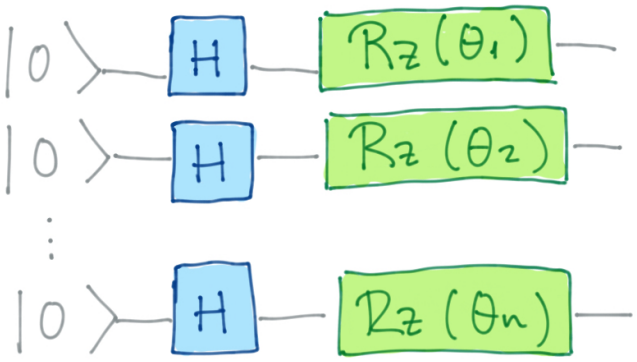

Figure 1: The circuit you must create and optimize to represent a Quantum Fourier Transform.

Specifically, your code will:

- Construct the aforementioned circuit.
- Optimize the circuit's parameters with respect to some sensible loss function.
- Output the "learned" angles.

The provided template `generating_fourier_state_template.py` contains three functions, two of which you need to complete. The `generating_fourier_state` function is where your circuit is defined and trained. The `circuit` function is where you need to construct the circuit ansatz. The `error` function acts like your loss function (the function to be minimized). Think of something sensible to return!

Input

- `list(int)`: A list containing the number of qubits / wires the circuit will have and the integer m indicating which state we want to generate a Fourier Transform of (i.e., QFT$|m\rangle$).

Output

- `list(float)`: The `generating_fourier_state` function will return the used circuit and the angles that generate the desired state. This output

will be tested with a random initial state to verify that the solution is correct. The test returns `qml.state`, which is what is in the `#.ans` files.

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 `0.001` tolerance specified below. To clarify, your answer must satisfy

$$\text{tolerance} \geq \left| \frac{\text{your solution} - \text{correct answer}}{\text{correct answer}} \right|.$$

- Your solution must take no longer than the `60s` 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 {name_of_file}.py < 1.in
```

WARNING: Don't modify the code outside of the `# QHACK #` markers in the template file, 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**
Time limit: **60 s**

Version History

Version 1: Initial document. Version 2: Tolerance typo (changed from 0.01 to 0.001). This update does not affect submissions that were evaluated as correct before this change.