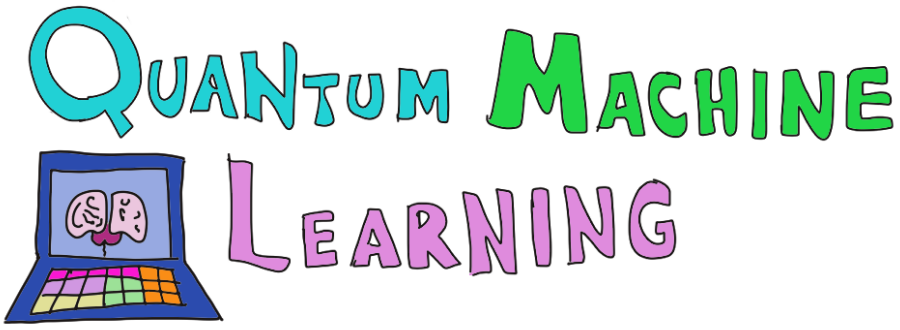

QML: Who likes the Beatles? [200 points]

Version: 1

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 [200 points]

A classification algorithm seeks to determine the class in which an element belongs, given a set of features. One of the simplest classification algorithms is the k-nearest neighbours (k-NN) algorithm, which we will use in this challenge. Our goal is to determine whether a person likes the Beatles from two features:

- their age
- the number of minutes a day that they watch television

The idea behind the k-NN algorithm is the following:

- We initially have a set of objects whose class we already know.
- Given a new point to classify, we search for the k points closest to it (the axes represent the different features).

- The class of the new object is deemed equal to the most repeated class among the k selected points.

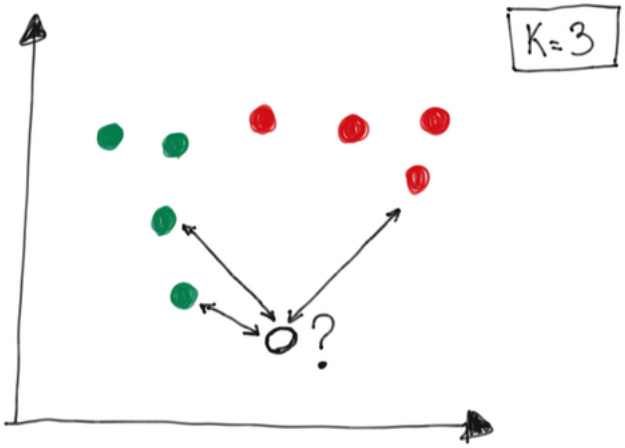

Figure 1: k-NN

On the X and Y axes in Figure 1, we represent two characteristics of the objects. The colour of the data point corresponds to the class in which they belong. In Figure 1, the new point will belong in the class of green points since two of the nearest points are green.

The most repeated operation throughout the algorithm is calculating the distance between points, since we need to determine which are the closest points. The traditional way to calculate the distance is through the Euclidean norm

$$d(\vec{A}, \vec{B}) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2},$$

where $\vec{A} = (x_A, y_A)$ and $\vec{B} = (x_B, y_B)$ are feature vectors.

However, this algorithm can be improved by calculating the distance more efficiently. When $\vec{A}$ and $\vec{B}$ are unit vectors, we can write

$$d(\vec{A}, \vec{B}) = \sqrt{2(1 - |\cos\theta_{AB}|)},$$

where θ_{AB} represents the angle between $\vec{A}$ and $\vec{B}$.

We can translate this formalism to the quantum world by switching to Dirac notation using $|A\rangle$ and $|B\rangle$:

$$d(A, B) = \sqrt{2(1 - |\langle A|B\rangle|)}.$$

Therefore, if we calculate $|\langle A|B\rangle|$, we obtain the distance between objects A and B. As a possible hint, it is possible to calculate it through the SWAP test, although there are other methods. However, most of these give you the value of $|\langle A|B\rangle|^2$, so don't forget to apply the square root.

The advantage of this quantum-based calculation of the distance given N different features is that classically we have to go through a vector of length N, whereas with this new method it will be sufficient to work with $\log_2 N$ qubits!

In the provided template called `who_likes_the_beatles_template.py`, there is a function called `distance` that you need to complete. As input, this function takes the age and television-watching time of two different people A and B (see Input section below for more information). You must construct a circuit within this function that embeds these features and calculates the square overlap $|\langle A|B\rangle|^2$ between them. Using this, you must then calculate the distance $d(A, B)$.

Input

- `list`: A list containing the new person's age and television watch time, the integer k of nearby neighbors that defines the k-NN procedure, and the information required to define a dataset with labels such as this one:

  ```
  dataset = [
    [[20,100], "YES"],
    [[13, 33], "NO"],
    [[24,40], "YES"],
    [[26, 20], "NO"],
    [[60, 300], "YES"],
    [[45, 200], "YES"],
    [[33, 10], "NO"]
  ]
  ```

  Each list element represents one person whose age, television-watching time, and like ("YES") / dislike ("NO") for the Beatles is given.

Output

- `list`: A list containing a 0 or 1 indicating if the new person likes or dislikes the Beatles, respectively, and the distance between the new person and the first element of the input dataset. We included the second number to ensure that you are doing it right!

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.