QHack

Quantum Coding Challenges

🚀 CHALLENGE COMPLETED

View successful submissions

Jump to code — Collapse text

The Super Parameter

200 points

Backstory

At Trine's Designs, the coffee machine is a programmable quantum device. It has three dials that tell the machine the type of drink it will prepare. However, two of the dials are broken. Trine, the CEO, is in despair: *"Coffee is essential for employees to function optimally."* So, as a provisional solution while they contact the manufacturer, Trine calls Zenda and Reece to quickly reprogram the device so that it works with only one dial.

Expressivity in Quantum Machine Learning

Within QML it is common to find the term expressivity, which refers to the size of all possible models that we can generate by varying our parameters. One way to increase the expressivity of our model family is usually by adding more parameters. However, this is not always a good thing, since increasing the number of parameters, and therefore the number of possible models, means that we have to perform our training on a very large set, making it more difficult to find the model that best suits our needs. Therefore, the real challenge of a good QML researcher is to find the smallest possible family of models that still contains the optimal solution. There is much more to the notion of expressivity, but in this challenge we are going to push the concept to its limits.

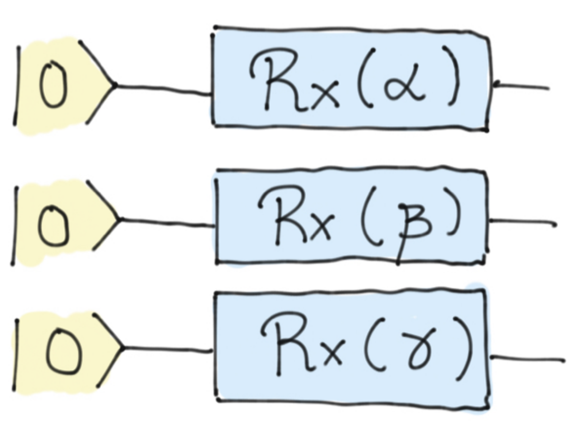
Suppose that we are in the situation where we have 3 qubits and we know that the solution to our problem is a computational basis state, i.e. an element of the set

$$\mathcal{B} = \{|000\rangle, |001\rangle, |010\rangle, \dots, |111\rangle\}.$$

We don't know exactly what the basis state is, so we would like to generate an ansatz expressive enough so that:

$$\begin{aligned} U(\vec{\theta}_0)|000\rangle &= |000\rangle \\ U(\vec{\theta}_1)|000\rangle &= |001\rangle \\ U(\vec{\theta}_2)|000\rangle &= |010\rangle \\ &\vdots \\ U(\vec{\theta}_7)|000\rangle &= |111\rangle \end{aligned}$$

for certain values of $\vec{\theta}_i$. An example of ansatz that accomplishes this would be the following circuit:



This is the fundamental concept in Basis embedding, where you can see that by taking α, β and γ properly, we can generate any basis state. However, this challenge is not going to be this easy. You are asked to build an ansatz that, with **only one parameter,** is able to generate all the basis states. To judge your solution, we will ask you to provide us with a list of the 8 values of the parameter that generate each of them. Good luck!

Challenge code

You must complete the qnode `model` that will be in charge of obtaining different outputs. This model depends on a single parameter and you must ensure that it generates all the basis states. You must also define the function `generate_coefficients`, which will return a list with the 8 values of the parameter to generate these basis states.

Output

To judge this challenge, the `generate_coefficients` function will be called first. With the output of this function (the eight coefficients), we will call the model to ensure that the generated states are the desired ones. In addition, we will check that:

- The model is continuous (small modifications of the parameter imply small modifications of the generated state). By putting the parameter inside rotation gates you will have no problems with this.
- The generated coefficients are in the interval [0,10]. Solutions that do not fit this interval will be considered incorrect.

In this challenge, we will not work with public and private tests. We will simply check that all of the above is fulfilled. Good luck!

Code

Help

```python
import json
import pennylane as qml
import pennylane.numpy as np
```

```python
dev = qml.device("default.qubit", wires=3)
```

```python
@qml.qnode(dev)
def model(alpha):
    """In this qnode you will define your model in such a way that there is a single
    parameter alpha which returns each of the basic states.

    Args:
        alpha (float): The only parameter of the model.

    Returns:
        (numpy.tensor): The probability vector of the resulting quantum state.
    """
    # Put your code here #
    return qml.probs(wires=range(3))

def generate_coefficients():
    """This function must return a list of 8 different values of the parameter that
    generate the states 000, 001, 010, ..., 111, respectively, with your ansatz.

    Returns:
        (list(int)): A list of eight real numbers.
    """
    # Put your code here #
```

```python
# These functions are responsible for testing the solution.
def run(test_case_input: str) -> str:
    return None

def check(solution_output, expected_output: str) -> None:
    coefs = generate_coefficients()
    output = np.array([model(c) for c in coefs])
    epsilon = 0.001

    for i in range(len(coefs)):
        assert np.isclose(output[i][i], 1)

    def is_continuous(function, point):
        limit = calculate_limit(function, point)

        if limit is not None and sum(abs(limit - function(point))) < epsilon:
            return True
        else:
            return False

    def is_continuous_in_interval(function, interval):
        for point in interval:
            if not is_continuous(function, point):
                return False
        return True

    def calculate_limit(function, point):
        x_values = [point - epsilon, point, point + epsilon]
        y_values = [function(x) for x in x_values]
        average = sum(y_values) / len(y_values)

        return average

    assert is_continuous_in_interval(model, np.arange(0,10,0.001))

    for coef in coefs:
```

```python
test_cases = [['No input', 'No output']]
```

```python
for i, (input_, expected_output) in enumerate(test_cases):
    print(f"Running test case {i} with input '{input_}'...")

    try:
        output = run(input_)

    except Exception as exc:
        print(f"Runtime Error. {exc}")

    else:
        if message := check(output, expected_output):
            print(f"Wrong Answer. Have: '{output}'. Want: '{expected_output}'.")

        else:
            print("Correct!")
```

Copy all

Reset Open Notebook Submit