QHack

Quantum Coding Challenges

CHALLENGE COMPLETED

View successful submissions

Jump to code — Collapse text

Cascadar

200 points

Backstory

Zenda and Reece have determined Doc Trine's cell number in hyperjail. Searching through Trine's notebooks, they find another note, explaining how the hypercube is patrolled by a fearsome quantum warden, which is able to place itself in a superposition and inspect multiple cells at once. To avoid detection and rescue Doc Trine, they need to build a quantum radar!

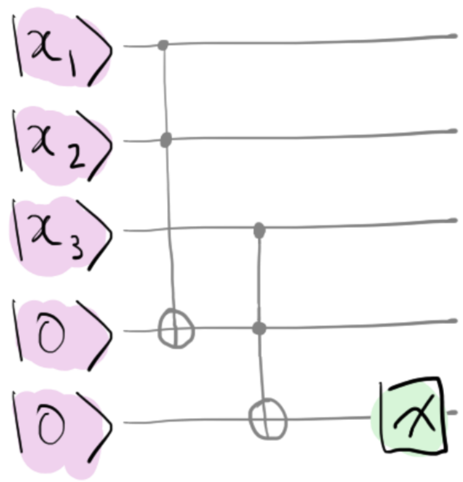
A quantum radar

The quantum guard can place itself in a superposition

$$|\text{guard}\rangle = \sum_x g_x |x\rangle,$$

where $x \in \{0,1\}^5$ ranges over all cell numbers, and g_x are complex-valued amplitudes. Seen in this way, $|g_x|^2$ is the probability that the guard is at position $|x\rangle$. They know that Doc Trine is located in a cell $c = (1, 1, 0, 0, 1)$. Ideally, they would like to wait until the guard's attention, captured by the probability $|g_c|^2$, is sufficiently low.

In this challenge, we will look for a way to be able to measure $|g_c|^2$. Unfortunately, there isn't much equipment in the office, and what is there is noisy! But Trine has left a collection of "Toffoli cascades" lying around, circuits made from a string of noisy Toffoli gates. Here is an example for three input qubits $|x_1\rangle|x_2\rangle|x_3\rangle$:



Measuring the last qubit in the computational basis gives $|(x_1 \cdot x_2 \cdot x_3)\rangle$ with probability 1, where $x_1 \cdot x_2 \cdot x_3$ indicates the *product* of classical bits x_1, x_2, and x_3. There is a Toffoli cascade acting on 5 input qubits (and with four auxiliary qubits) that Zenda and Reece can use, as well as some Pauli X gates. All are subject to *depolarizing noise*, such that after each gate, the state on each qubit is replaced with something random with probability λ.

Your task: use noisy Toffoli cascades and noisy-Pauli X gates to build a *quantum radar*, which outputs $|g_c|^2$, the guard's attention on Trine's cell. The guard state will be an input, along with four auxiliary qubits starting in the $|0\rangle$ state.

Challenge code

In the code below, you are given various functions:

- `noisy_PauliX`: which applies the Pauli-X gate and then a layer of depolarizing noise with parameter `lmbda`. (The noise is added for you.)
- `Toffoli_cascade`: a cascade of noisy Toffoli gates (noise parameter `lmbda`) which help compute a product, as in the circuit pictured above, with the input qubits on `in_wires` and auxiliary system `aux_wires`. (The noise is added for you.)
- `cascadar`: which takes a `guard_state (numpy.tensor)` and returns $|g_c|^2$, using noisy equipment with parameter `lmbda`. **You must complete this function**.

Inputs

The noisy quantum radar `cascadar` takes as input the guard state `guard_state (numpy.tensor)`, and a noise parameter `lmbda (float)` controlling the depolarizing noise.

Output

Your `cascadar` function should gives the correct probability $|g_c|^2$ for test cases, including the effects of noise.

If your solution matches the correct one within the given tolerance specified in `check` (in this case it's a `1e-4` relative error tolerance), the output will be `"Correct!"` Otherwise, you will receive a `"Wrong answer"` prompt.

Code

Help

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

```python
def noisy_PauliX(wire, lmbda):
    """A Pauli-X gate followed by depolarizing noise.

    Args:
        lmbda (float): The parameter defining the depolarizing c
        wire (int): The wire the depolarizing channel acts on.
    """
    qml.PauliX(wire)
    qml.DepolarizingChannel(lmbda, wires=wire)

def Toffoli_cascade(in_wires, aux_wires, lmbda):
    """A cascade of noisy Toffolis to help compute the product.

    Args:
        in_wires (list(int)): The input qubits.
        aux_wires (list(int)): The auxiliary qubits.
        lmbda (float): The probability of erasing the state of a
    """
    n = len(in_wires)
    qml.Toffoli(wires=[in_wires[0], in_wires[1], aux_wires[0]])
    qml.DepolarizingChannel(lmbda, wires=in_wires[0])
    qml.DepolarizingChannel(lmbda, wires=in_wires[1])
    qml.DepolarizingChannel(lmbda, wires=aux_wires[0])
    for i in range(n - 2):
        qml.Toffoli(wires=[in_wires[i + 2], aux_wires[i], aux_wi
        qml.DepolarizingChannel(lmbda, wires=in_wires[i + 2])
        qml.DepolarizingChannel(lmbda, wires=aux_wires[i])
        qml.DepolarizingChannel(lmbda, wires=aux_wires[i + 1])

# Build a quantum radar to check how much attention is on Trine'
def cascadar(guard_state, lmbda):
    """Return the squared amplitude |g_c|^2 of the guard state,

    Args:
        guard_state (numpy.tensor): A 2**5 = 32 component vector
        lmbda (float): The probability of erasing the state of a

    Returns:
        (float): The squared amplitude of the guard state on the
    """
    dev = qml.device("default.mixed", wires = 5 + 4)

    @qml.qnode(dev)
    def circuit():
        """
        Circuit that will use the Toffoli_cascade and the noisy_
        It will return a measurement on the last qubit.
        """

        qml.QubitStateVector(guard_state, range(5))

```

```python
        # Put your code here #

        return

    output = circuit()

    # if you want to post-process the output, put code here alsc

    return

```

```python
# These functions are responsible for testing the solution.
def run(test_case_input: str) -> str:

    guard_state, lmbda = json.loads(test_case_input)
    output = cascadar(guard_state, lmbda)

    return str(output)

def check(solution_output: str, expected_output: str) -> None:

    solution_output = json.loads(solution_output)
    expected_output = json.loads(expected_output)
    assert np.allclose(
        solution_output, expected_output, rtol=1e-4
    ), "Your quantum radar isn't quite working properly!"

```

```python
test_cases = [['[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

```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: '{expe

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

Copy all

Submit

Open Notebook

Reset