QHack

Quantum Coding Challenges

CHALLENGE COMPLETED View successful submissions

Jump to code — Collapse text

A Pauli-Worded Problem

300 points

Backstory

Now Zenda and Reece know where Trine is in hyperjail, and how to evade the quantum guard who patrols the hypercube. The only question is how to get there! Doc Trine's journal explains that the portal to hyperjail is held open by exotic matter, and the quantum sensor not only helps avoid the guard, but can be used to detect this matter! But the galaxy is a big place. How do Zenda and Reece find the entrance to hyperjail?

Thankfully, they stumble onto a section of Trine's journal entitled 'How to build a robot swarm'. This not only directs them to an old storage cupboard with hundreds of jetpack-equipped robots, but instructions for coordinating them using a special entangled state. Zenda and Reece need to search the office and see if this state can be found! There are several mysterious boxes which, at the push of a button, output a quantum state ρ. Zenda and Reece would like to figure out if any of these states will do. Unfortunately, noise makes it harder to tell what the states are!

Blurry shadows

Whenever Zenda and Reece push the button on a box and output a state in order to test it, it goes into a noisy circuit, where each qubit is subject to depolarizing noise, Δ_λ. If ρ is a single-qubit density matrix, Δ_λ is defined by

$$\Delta_\lambda[\rho] = (1-\lambda)\rho + \frac{\lambda}{2}I,$$

and with probability λ, the state is deleted and replaced with something random. Zenda and Reece suspect that noisy is making the states coming out of the box very hard to distinguish from random, and would like some way to test just how badly blurred they are.

To explore this, we first note that any density matrix on n qubits can be written as a linear combination of a special set of "Pauli" density matrices. These have the form

$$\rho_P = \frac{1}{2^n}(I+P),$$

where $P \in \{I, X, Y, Z\}^{\otimes n}$ is a tensor product of n single-qubit Pauli operators, called a Pauli word. We'll let $\rho_P(\lambda) = \Delta_\lambda^{\otimes n}[\rho_P]$ label the result of applying a layer of depolarizing noise to the Pauli density ρ_P.

If adding noise makes a Pauli density matrix look random, a combination of Pauli densities — in other words, any matrix! — will look random. Here, "looks random" means "the expectation of any measurement is similar to the maximally mixed density matrix $\rho_0 = I/2^n$". Remarkably, we can capture all expectations at once using something called *trace distance* between density matrices. This is defined as

$$T(\rho, \sigma) = \frac{1}{2}\mathrm{Tr}|\rho - \sigma|,$$

where $|A| = \sqrt{A^\dagger A}$ for a generic matrix A (to calculate $|\rho - \sigma|$ you will be provided with the function `abs_dist`). For any (projective) measurement M, the trace distance between two density matrices ρ and σ bounds the difference in expectations:

$$\langle M\rangle_\rho - \langle M\rangle_\sigma = \mathrm{Tr}[M(\rho-\sigma)] \le T(\rho,\sigma).$$

If the trace distance is small, the two states are hard to tell apart with *any* measurement.

Zenda and Reece suspect that the noise in their circuitry is blurring the states and making them hard to distinguish. Your goal is to write a function which verifies the bound

$$T(\rho_P(\lambda), \rho_0) \le (1-\lambda)^{|P|},$$

by computing the difference between the right-hand side and left-hand side, where $|P|$ is the number of **non-identity** operators in the Pauli word P. You should find this is always positive! Since a Pauli density matrix gets *exponentially* blurry, and all states can be built from these Pauli densities, most states will be exponentially hard to distinguish.

Challenge code

In the code below, you are given various functions:

- `word_dist`: which counts the number of non-identity operators in a Pauli word.
- `abs_dist`: which computes the distance $|\rho - \sigma|$ between density matrices (`rho` and `sigma`).
- `noisy_Pauli_density`: a helper subcircuit which produces the density matrix ρ_P associated with a Pauli word P (`word`) and applies depolarizing noise to each qubit with parameter `lmbda`. It is merely a collection of gates, and should not return anything. **You must complete this function.**
- `maxmix_trace_dist`: a helper function which calculates the trace distance $T(\rho_P(\lambda), \rho_0)$, from the noisy ρ_Q (specified by `word` and `lmbda`) to the maximally mixed density ρ_0. **You must complete this function.**
- `bound_verifier`: a function which computes the difference $(1-\lambda)^{|P|} - T(\rho_P(\lambda), \rho_0)$, with both terms specified by `lmbda` and `P`. **You must complete this function.**

Inputs

The functions `noisy_Pauli_density`, `maxmix_trace_dist` and `bound_verifier` take as input a Pauli word (`word (str)`) represented as a string of characters `I`, `X`, `Y` and `Z`, and a noise parameter `lmbda (float)` giving probability of erasing the state of a qubit.

Note that, for `noisy_Pauli_density`, you are working with the `default.mixed` device and can create a density matrix using `qml.QubitDensityMatrix`.

Output

Your function `bound_verifier` must correctly compute the difference between the upper bound $(1-\lambda)^{|P|}$ and the trace distance $T(\rho_P(\lambda), \rho_0)$ on test cases.

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

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

```python
def abs_dist(rho, sigma):
    """A function to compute the absolute value |rho - sigma|."""
    polar = scipy.linalg.polar(rho - sigma)
    return polar[1]

def word_dist(word):
    """A function which counts the non-identity operators in a Pauli word"""
    return sum(word[i] != "I" for i in range(len(word)))


# Produce the Pauli density for a given Pauli word and apply noise

def noisy_Pauli_density(word, lmbda):
    """
       A subcircuit which prepares a density matrix (I + P)/2**n for a given Pauli
       word P, and applies depolarizing noise to each qubit. Nothing is returned.

    Args:
            word (str): A Pauli word represented as a string with characters I,  X, Y and Z.
            lmbda (float): The probability of replacing a qubit with something random.
    """

```

```python
    # Put your code here #
    n = len(word)
    #print(n)
    P = qml.pauli.string_to_pauli_word(word)
    #print(P)
    I = qml.Identity([0])
    for i in range(1,n):
        I = I @ qml.Identity([i])
    rho_p = (P + I)*(0.5)**n
    rho_p_matrix = qml.matrix(rho_p, wire_order=range(n))

    qml.QubitDensityMatrix(rho_p_matrix, wires=range(n))
    for i in range(n):
        qml.DepolarizingChannel(lmbda, i)
```

```python
# Compute the trace distance from a noisy Pauli density to the maximally mixed density

def maxmix_trace_dist(word, lmbda):
    """
       A function compute the trace distance between a noisy density matrix, specified
       by a Pauli word, and the maximally mixed matrix.

    Args:
            word (str): A Pauli word represented as a string with characters I, X, Y and Z.
            lmbda (float): The probability of replacing a qubit with something random.

    Returns:
            float: The trace distance between two matrices encoding Pauli words.
    """

```

```python
    # Put your code here  n = len(word)
    n = len(word)
    dev = qml.device('default.mixed', wires=n)
    @qml.qnode(dev)
    def circuit():
    # Put your code here #
        noisy_Pauli_density(word, lmbda)
        return qml.density_matrix(wires=range(n))

    rho_p_matrix = circuit()

    I = qml.Identity([0])
    for i in range(1,n):
        I = I @ qml.Identity([i])

    rho_0 = (I)*(0.5)**n
    rho_0_matrix = qml.matrix(rho_0, wire_order=range(n))
    distance = abs_dist(rho_p_matrix, rho_0_matrix)

    return 0.5*np.trace(distance)
```

```python
def bound_verifier(word, lmbda):
    """
       A simple check function which verifies the trace distance from a noisy Pauli density
       to the maximally mixed matrix is bounded by (1 - lambda)^|P|.

    Args:
            word (str): A Pauli word represented as a string with characters I, X, Y and Z.
            lmbda (float): The probability of replacing a qubit with something random.

    Returns:
            float: The difference between (1 - lambda)^|P| and T(rho_P(lambda), rho_0).
    """

```

```python
    # Put your code here #
    return (1-lmbda)**(word_dist(word)) - maxmix_trace_dist(word, lmbda)
```

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

    word, lmbda = json.loads(test_case_input)
    output = np.real(bound_verifier(word, 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 trace distance isn't quite right!"
```

```python
test_cases = [['["XXI", 0.7]', '0.0877777777777777'], ['["XXIZ", 0.1]', '0.4035185185185055'], ['["YI
```

```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