Desperate Measures

400 points

Backstory

With the resources available to them, Zenda and Reece decide that one single method is not enough to interfere with the correct functioning of Sqynet, since it can repair itself too quickly. It's time to resort to brute force methods. By firing missiles at the outer shell, they will introduce a considerable amount of depolarizing noise into Sqynet's hardware.

Trotterization of the Heisenberg model

An approximate way to model Sqynet is by considering it as a closed spin chain of length N. A spin chain contains particles of spin $1/2$ in each of its N sites. We make this model more realistic by assuming that the spins may be pointing in any direction, and we consider that there may be an external magnetic field acting on the system.

When we model a closed spin chain of length N in which spins can point in any direction, we need to use the Heisenberg Hamiltonian. In the presence of an external magnetic field of intensity h, the Hamiltonian is given by

$$H = -\sum_{i=1}^{N}(J_x X_i \otimes X_{i+1} + J_y Y_i \otimes Y_{i+1} + J_z Z_i \otimes Z_{i+1}) - h\sum_{i=1}^{N} X_i.$$

The subindices i indicate the spin site where the operators act. In a closed spin chain, we identify site $N+1$ with the first site. The coefficients J_x, J_y and J_z are known as *coupling constants* and they measure the strength of the interaction between neighbouring spins.

Sqynet's correct functioning relies on it being completely isolated from the environment, to avoid decoherence. Zenda and Reece think that, to tamper with Sqynet's correct functioning, the old way is the best way, so they'll shoot missiles at the tail of the spaceship, where the quantum device is. This will introduce noise into the gates that Sqynet executes.

Zenda and Reece need to estimate how the noise affects Hamiltonian evolution. Your task is to build a Trotterization circuit that simulates $U = \exp\left(-iHt\right)$. This circuit must only contain RX, RY, RZ, and $CNOT$ gates. The missiles will introduce noise on the target qubit of every execution of a CNOT gate. We model this via a **Depolarizing Channel** with parameter p . To quantify the effects of noise, you are asked to find the fidelity between this noisy Trotterization and the noiseless one.

Challenge code

You must complete the `heisenberg_trotter` that implements the Trotterization of the Heisenberg Hamiltonian for $N = 4$ using only the following PennyLane gates: `qml.RX` `qml.RY`, `qml.RZ`, `qml.CNOT`, and `qml.DepolarizingChannel`. This function will return a quantum state. You should also minimize the number of CNOT gates as much as you can, in order to avoid noise. To verify that the that the Trotterization that you proposed is not excessively noisy, we will calculate for you the fidelity of your output state with respect to the noiseless case using the `calculate_fidelity` function.

Input

As input to this problem, you are given:

- `couplings` (`list(float)`): An array of length 4 that contains the coupling constants and the magnetic field strength, in the order $[J_x, J_y, J_z, h]$.
- `p` (`float`): The depolarization probability on the target qubit after each CNOT gate.
- `depth` (`int`): The Trotterization depth.
- `time` (`float`): Time during which the state evolves.

Output

This code will output a `float` corresponding to the fidelity between the output states of the noisy and noiseless trotterizations, calculated from the output of `heisenberg_trotter`. The outputs in the test cases correspond to the minimal fidelity that you should achieve if you used a small enough amount of CNOT gates.

If your fidelity is larger, up to a tolerance of 0.005, of that specified in the output cases, your solution will be judged as `"Correct!"` Otherwise, you will receive a `"Wrong answer"` prompt.

Good luck!

Code

Help

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

```python
num_wires = 4
dev = qml.device("default.mixed", wires=num_wires)

@qml.qnode(dev)
def heisenberg_trotter(couplings, p, time, depth):
    """This QNode returns the final state of the spin chain afte
    under the Trotter approximation of the exponential of the He

    Args:
        couplings (list(float)):
            An array of length 4 that contains the coupling cons
            strength, in the order [J_x, J_y, J_z, h].
        p (float): The depolarization probability after each CNC
        depth (int): The Trotterization depth.
        time (float): Time during which the state evolves

    Returns:
        (numpy.tensor): The evolved quantum state.
    """

```

```python
    # Put your code here #
    return qml.state()

```

```python
def calculate_fidelity(couplings, p, time, depth):
    """This function returns the fidelity between the final stat
    noiseless Trotterizations of the Heisenberg models, using on

    Args:
        couplings (list(float)):
            A list with the J_x, J_y, J_z and h parameters in th
            defined in the problem statement.
        p (float): The depolarization probability of the depolar
                   target qubit of each CNOT gate.
        time (float): The period of time evolution simulated by
        depth (int): The Trotterization depth.

    Returns:
        (float): Fidelity between final states of the noisy and
    """
    return qml.math.fidelity(heisenberg_trotter(couplings,0,time

```

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

    ins = json.loads(test_case_input)
    output =calculate_fidelity(*ins)

    return str(output)

def check(solution_output: str, expected_output: str) -> None:
    """
    Compare solution with expected.

    Args:
            solution_output: The output from an evaluated solu
            the same type as returned.
            expected_output: The correct result for the test c

    Raises:
            ``AssertionError`` if the solution output is incor

    """
    def create_hamiltonian(params):

        couplings = [-params[-1]]
        ops = [qml.PauliX(3)]

        for i in range(3):

            couplings = [-params[-1]] + couplings
            ops = [qml.PauliX(i)] + ops

        for i in range(4):

            couplings = [-params[-2]] + couplings
            ops = [qml.PauliZ(i)@qml.PauliZ((i+1)%4)] + ops

        for i in range(4):

            couplings = [-params[-3]] + couplings
            ops = [qml.PauliY(i)@qml.PauliY((i+1)%4)] + ops

        for i in range(4):

            couplings = [-params[0]] + couplings
            ops = [qml.PauliX(i)@qml.PauliX((i+1)%4)] + ops

        return qml.Hamiltonian(couplings,ops)

    @qml.qnode(dev)
    def evolve(params, time, depth):

        qml.ApproxTimeEvolution(create_hamiltonian(params), ti

```

```python
            return qml.state()

        solution_output = json.loads(solution_output)
        expected_output = json.loads(expected_output)

        tape = heisenberg_trotter.qtape
        names = [op.name for op in tape.operations]

        random_params = np.random.uniform(low = 0.8, high = 3.0, s

        assert qml.math.fidelity(heisenberg_trotter(random_params,

        assert names.count('ApproxTimeEvolution') == 0, "Your circ

        assert set(names) == {'DepolarizingChannel', 'RX', 'RY', '

        assert solution_output >= expected_output-0.005, "Your fid
```

```python
test_cases = [['[[1,2,1,0.3],0.05,2.5,1]', '0.337239811233695'
```

```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