QHack

Quantum Coding Challenges

CHALLENGE COMPLETED View successful submissions

Jump to code — Collapse text

Secrets in Spacetime

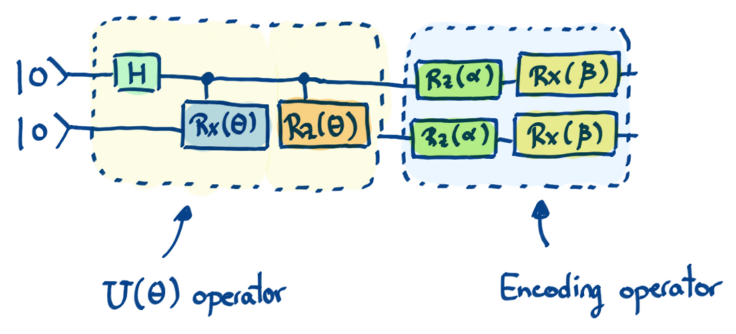
300 points

Backstory

Now Zenda and Reece have a cute way to send each other private messages using entangled qubits. Trine applauds them. "*Good work! But now that I think of it, superdense coding can be reversed, in a manner of speaking, to send quantum information using entanglement and classical bits. This will not only bring us to the last Law of Infodynamics, but teach us some basic facts about spacetime! Certain things have to be hidden from Nature itself.*" Zenda and Reece look perplexed. Trine smiles: "*Wait until I show you what timbits can do!*"

From causality to encryption

Zenda needs to send quantum states to Reece over a channel where someone could intercept the messages. They decide to encode the states they want to send with rotations on all of the qubits. To do this, they have chosen two real numbers, α and β, in advance, so that the states can be encoded as follows:



In this case, $U(\theta)$ is defined as the gate that generates the state $|\psi\rangle$ — what Zenda wants to send to Reece — that depends on a real number θ. Thus, if someone intercepts the message, instead of getting state $|\psi\rangle$ they will get state $(R_X(\beta)R_Z(\alpha))^{\otimes 2}|\psi\rangle$.

Although it seems like a super secure encoding procedure, it is not perfect! Once α and β have been chosen, there are certain values of θ that could make $(R_X(\beta)R_Z(\alpha))^{\otimes 2}|\psi\rangle = |\psi\rangle$ for certain states. This is a big problem — it means that someone is going to intercept the hidden state!

We will say that α and β are ϵ-unsafe values if there exists a θ such that

$$|\langle 0|U^{\dagger}(\theta)(R_X(\beta)R_Z(\alpha))^{\otimes 2}U(\theta)|0\rangle|^2 \geq 1-\epsilon.$$

Your goal is to determine if α and β are unsafe values given ϵ.

▾ Laws of Infodynamics Part III: The Fourth Law

This box contains information that is not essential to solving the problem. Superdense coding sends two classical bits (cbits) via a qubit and half of an entangled Bell pair (ebit). Teleportation is a converse protocol, sending a qubit with two cbits and an ebit. Suppose Zenda has a state $|\psi\rangle_{Z'}$ she wants to send to Reece, and they share a Bell state $|\beta(0,0)\rangle_{ZR}$, where

$$|\beta(j,k)\rangle_{ZR} = \frac{1}{\sqrt{2}}(|0\rangle_Z|k\rangle_R + (-1)^j|1\rangle_Z|k\oplus 1\rangle_R).$$

We use the notation $|\Phi\rangle = |\beta(0,0)\rangle$ for the "canonical" Bell pair. Here, Z' denotes Zenda's qubit where the state for teleportation is initially stored, and Z the qubit which is initially entangled with Reece. Some algebra shows that the state of the whole system is

$$|\psi\rangle_{Z'}|\Phi\rangle_{ZR} = \frac{1}{2}\sum_{jk}|\beta(j,k)\rangle_{Z'Z}X^kZ^j|\psi\rangle_R.$$

Note that in the operators X^jZ^j, k comes *before* j. If Zenda performs a Bell measurement (i.e. measure in the basis $\{|\beta(j,k)\rangle\}$) on her system, she will learn two bits j and k, and Reece will have Zenda's state in the disguised form $X^kZ^j|\psi\rangle$. For instance, if Zenda measures $j=k=0$, we apply the projector

$$P = |\beta(0,0)\rangle\langle\beta(0,0)|_{Z'Z},$$

normalize, and obtain a post-measurement state

$$|\beta(0,0)\rangle_{Z'Z}|\psi\rangle_R.$$

After she measures the cbits j and k, Zenda can send them to Reece, who takes off the disguise X^kZ^j to find $|\psi\rangle$. Since an ebit and two classical bits suffice to teleport a qubit in an arbitrary state, we have the Fourth Law of Infodynamics:

$$1\text{ ebit} + 2\text{ cbits} \geq 1\text{ qubit} \qquad (4)$$

where $x \geq y$ means having resource x also provides resource y. The disguising operators X^kZ^j seem like a nuisance, but turn out to be essential to maintaining the fabric of spacetime! Since entanglement acts in a spooky, instantaneous way at a distance, if Zenda could magically teleport $|\psi\rangle_{Z'}$ to Reece without the disguise, she could send information faster than light. If Reece knows nothing about j and k, it turns out that the state is perfectly disguised, in the sense that

$$\rho_R = \frac{1}{4}\sum_{jk}X^kZ^j|\psi\rangle\langle\psi|(X^kZ^j)^{\dagger} = \frac{1}{2}\mathbb{I}.$$

This inspired Zenda and Reece to play around with X and Z rotations as a way of concealing information.

Challenge code

In the code below, you are given a function called `is_unsafe`. **You must complete this function** by coming up with a way — you are given total freedom, from making a variational circuit to finding an analytical solution — to determine if the given values of α and β values are ϵ-unsafe.

Inputs

As input to this problem, you are given a `list(float)` containing the values of α, β, and ϵ, in that order.

Output

This code must output a boolean — `True` or `False` — corresponding to whether the values of α and β are ϵ-unsafe. For example, if you determine that the given values of α and β *aren't* ϵ-unsafe, your code must output `False`.

If your solution is correct, the output will be `"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
def U_psi(theta):
    """
    Quantum function that generates |psi>, Zenda's state wants to send to Reece.

    Args:
        theta (float): Parameter that generates the state.

    """
    qml.Hadamard(wires = 0)
    qml.CRX(theta, wires = [0,1])
    qml.CRZ(theta, wires = [0,1])

def is_unsafe(alpha, beta, epsilon):
    """
    Boolean function that we will use to know if a set of parameters is unsafe.

    Args:
        alpha (float): parameter used to encode the state.
        beta (float): parameter used to encode the state.
        epsilon (float): unsafe-tolerance.

    Returns:
        (bool): 'True' if alpha and beta are epsilon-unsafe coefficients. 'False' in the other case.

    """
```

```python
    # Put your code here #

    def f(theta, alpha, beta):
            # here is where I can put the function I have. Either keep as it is or convert using
            # cos theta/2 = x,  sin theta/2 = np.sqrt(1-x**2)
            # e^(i\theta/2) = cos(theta/2) + i sin(theta/2) =  x + 1j*np.sqrt(1-x**2)
            # e^(-i\theta/2) = cos(theta/2) - i sin(theta/2) =  x - 1j*np.sqrt(1-x**2)
            r = np.exp(-1j*alpha)*np.cos(beta/2)**2 - (1j*np.cos(theta/2)*np.exp(-1j*theta/2)*np.sin(be
                + np.cos(theta/2)*np.exp(1j*theta/2)*(np.cos(theta/2)*np.cos(beta/2)**2*np.exp(-1j*theta
                - 1j*np.sin(theta/2)*np.exp(-1j*theta/2)*np.exp(-1j*alpha)*(np.sin(beta/2)**2*np.exp(-1
            return abs(r/2)**2

    def cost(theta, alpha, beta, epsilon):
            return 1 - epsilon - f(theta, alpha, beta)

    def minimize(cost, x0, alpha, beta, epsilon, conv_tol=1e-08, step_size=0.2, max_iterations=500, prog
        opt = qml.AdamOptimizer(stepsize=step_size)
        #opt = qml.GradientDescentOptimizer(stepsize=step_size)
        #opt = qml.QNGOptimizer(stepsize=step_size, approx="block-diag")
        theta = np.array(x0, requires_grad=True)
        # store the values of the cost function
        energy = [cost(theta, alpha, beta, epsilon)]
        # store the values of the circuit parameter
        params = [theta]

        for n in range(max_iterations):
            theta, prev_energy = opt.step_and_cost(cost, theta, alpha=alpha, beta=beta, epsilon=epsilon
            energy.append(cost(theta, alpha, beta, epsilon))
            params.append(theta)

            conv = np.abs(energy[-1] - prev_energy)
            if progress:
                print(f"Step = {n},  Energy = {energy[-1]:.8f} Ha")
            if conv <= conv_tol:
                break
        status = f'finished in {n} iterations'

        return params[-1], energy[-1], status
    #from scipy.optimize import minimize

    x0 = np.pi

    #res = minimize(cost, x0, args=(alpha,beta,epsilon), method='Nelder-Mead', tol=1e-6)
    x, val, statsu = minimize(cost, x0, alpha, beta, epsilon)
    if val <= 0 and abs(val)>0.02:
        return True
    else:
        return False
```

```python
# These functions are responsible for testing the solution.
def run(test_case_input: str) -> str:
    ins = json.loads(test_case_input)
    output = is_unsafe(*ins)
    return str(output)

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

    def bool_to_int(string):
        if string == "True":
            return 1
        return 0

    solution_output = bool_to_int(solution_output)
    expected_output = bool_to_int(expected_output)
    assert solution_output == expected_output, "The solution is not correct."
```

```python
test_cases = [['[0.1, 0.2, 0.3]', 'True'], ['[1.1, 1.2, 0.3]', 'False'], ['[1.1, 1.2, 0.4]', 'True'],
```

```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