Task 2 by Mudassir Moosa

Sec 1: Introduction

Our goal in this exercise is to make a circuit that returns either $|01\rangle$ or $|10\rangle$ with equal (50%) probability. We are only allowed to use $CNOT$, R_x, and R_y quantum gates in our circuit. Moreover, to make things interesting, we are also supposed to include a random noise in our circuit.

As part of a bonus question, we wanted a circuit that prepares

$$\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$$

rather than a more general superposition

$$\frac{1}{\sqrt{2}}|01\rangle + \frac{e^{i\phi}}{\sqrt{2}}|10\rangle \, .$$

Therefore, we will, from now on, denote our **target state** as $|\psi_T\rangle$ which we define as

$$|\psi_T\rangle \equiv \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle \, .$$

To do this exercise, we will start with a variational circuit and will perform the gradient descent method to find the optimum parameters. The cost function that we will use will guide us how close we are from the target state $|\psi_T\rangle$.

We will do this excercise on Cirq (https://cirq.readthedocs.io/en/stable/#).

The rest of this document is organized as follows. We perform all our calculations and present our codes in Sec. (2). In Sec. (2.2), we discuss a simple model for noise that we consider in this exercise, we make a variational noisy circuit that we need to optimize in Sec. (2.3). We present a suitable cost function in Sec. (2.4) and discuss how to measure its gradients in Sec. (2.5). We perform the gradient descent analysis in Sec. (2.5). We combine all of these and implement the desired circuit in Sec. (2.6). In Sec. (3), we run the codes that we have written in Sec. (2) when the only possible noise available is the *phase-flip*. We verify that the indeed only returns $|01\rangle$ and $|10\rangle$ with equal probabilities. We end with some concluding remarks in Sec. (4).

Sec 2: Noisy Variational Circuit and Gradient Descent

Sec 2.1: A noise-free circuit

Before we make a variational ansatz and dicuss a suitable noise model, let us observe that if there were **no** noise in our circuit, then we already know (based on our previous knowledge of quantum gates) what circuit would prepare $|\psi_T\rangle$. This circuit takes two qubits in $|00\rangle$ state, rotates the first qubit by amount $\pi/2$ around the y-axis, rotates the second qubit by amount π around the x-axis, and then apply the CNOT gate on these qubits. This circuit can be implemented on Cirq as shown below:

In [1]:
```
import cirq
import numpy as np

# Defining two qubits and the circuit
q0 , q1 = cirq.LineQubit.range(2)
circuit = cirq.Circuit( )


#Applying rotation on qubit q0 by amount pi/2 around y_axis
circuit.append(cirq.ry(np.pi/2).on(q0))
#Applying rotation on qubit q1 by amount pi around x_axis
circuit.append(cirq.rx(np.pi).on(q1))
#Applying the CNOT gate
circuit.append(cirq.CNOT(q0,q1))

# Displaying the circuit
print(circuit)
```

```
0: ───Ry(0.5π)───@───
                 │
1: ───Rx(π)──────X───
```

To ensure that this circuit indeed prepares $|\psi_T\rangle$, we can simulate the circuit.

In [2]:
```
s = cirq.Simulator()
print(s.simulate(circuit))
```

```
measurements: (no measurements)
output vector: -0.707j|01⟩ - 0.707j|10⟩
```

This confirms that the prepared state (up to a global phase of $-i$) is the same as $|\psi_T\rangle$.

The point of discussing this **noise-free** circuit is that it provides us a hint of what our variational circuit in the presence of noise should look like. The ansatz that we make is that the variational circuit is the same as above but the angles in R_x and R_y are variational parameters. We denote these angles by θ_x and θ_y respectively.

Before we implement this variational circuit on Cirq, we discuss how we choose to include noise in our circuit.

Sec 2.2: A model for noise

We tried to make the model for the noise in our circuit as simple as possible (so it is not difficult) but also non-trivial (so it is not boring). To do this, we make the following assumptions:

- The noise is only associated with the rotation gates, R_x and R_y, but not with the $CNOT$ gate.
- The noise acts on both qubits independently.
- The only possible types of noise present are either bit-flip or phase-flip.

(However, the code that we have written (see below) can easily be modified for other types of noise like 'phase-damping' or for'non-local' noise.)

Sec 2.3: Variational Circuit

As we discussed in Sec. (2.1), the variational circuit that we make is similar to the *noise-free* circuit of Sec. (2.1) but the angles of rotation, θ_x and θ_y, are taken to be our variational parameters. Now based on our model in Sec. (2.2), both qubits will independently experience either bit_flip or phase_flip (with random but pre-determined probability) before the $CNOT$ gate is applied.

To implement this circuit, we write a function that we call `noisy_variational_circuit` . This circuit takes four inputs:

- theta*x : This is the value of the variational parameter $\theta*{x}$* and it is given as a float.
- theta*y : This is the value of the variational parameter $\theta*{y}$* and it is given as a float.
- noise_type : This is a list of two strings. The elements of this list are either 'bit_flip' or 'phase_flip'.
- noise_probability : This is a list of two floats. The elements of this list are a float number between 0 and 1.

The code for this function is presented here:

In [3]:
```
def noisy_variational_circuit(theta_x,theta_y,noise_type,noise_probability):

    circuit = cirq.Circuit()
    q0,q1 = cirq.LineQubit.range(2)

    # First moment of the circuit:
    # We apply rotations by amount theta_x and theta_y
    circuit.append(cirq.ry(theta_y).on(q0))
    circuit.append(cirq.rx(theta_x).on(q1))

    # Now we add the noise on qubit 0:
    if noise_probability[0]:
        if noise_type[0] == 'bit_flip':
            circuit.append(cirq.bit_flip(noise_probability[0]).on(q0))
        elif noise_type[0] == 'phase_flip':
            circuit.append(cirq.phase_flip(noise_probability[0]).on(q0))
        # Note: This can be easily modified to
        # include other types of noise.

    # Now we add the noise on qubit 1:
    if noise_probability[1]:
        if noise_type[1] == 'bit_flip':
            circuit.append(cirq.bit_flip(noise_probability[1]).on(q1))
        elif noise_type[1] == 'phase_flip':
            circuit.append(cirq.phase_flip(noise_probability[1]).on(q1))


    # Second moment of the circuit:
    # We apply the CNOT gate.
    circuit.append(cirq.CNOT(q0,q1))

    return circuit
```

Now given this variational circuit, our goal is to perform the gradient descent to find the optimum values of θ_x and θ_y. However, to perform gradient descent, we first need to discuss the cost function.

Sec 2.4: Cost function

Since we are dealing with a noisy circuit, it is natural for us to work with the density matrices. Let us suppose that the output of the `noisy_variational_circuit` is a density matrix $\rho(\theta_x, \theta_y)$ that depends on the variational parameters θ_x and θ_y. We need to make sure that this density matrix is close to the target state $\rho_T \equiv |\psi_T\rangle\langle\psi_T|$. We can estimate how close states $\rho(\theta_x, \theta_y)$ and ρ_T are by calculating the cost function $C(\theta_x, \theta_y)$ which we define as

$$C(\theta_x, \theta_y) \equiv 1 - \mathrm{tr}\big(\rho_T \cdot \rho(\theta_x, \theta_y)\big) \, .$$

Equivalently, we can write this as

$$C(\theta_x, \theta_y) = 1 - \, \langle\psi_T|\rho(\theta_x, \theta_y)|\psi_T\rangle \, .$$

Note that $C(\theta_x, \theta_y)$ is non-negative and it vanishes if and only if the state $\rho(\theta_x, \theta_y)$ is the same as the target state ρ_T.

Since our goal is to minimize the cost function, we can safely ignore the constant term. From now on, therefore, we will take the cost function to be given by

$$C(\theta_x, \theta_y) = - \, \langle\psi_T|\rho(\theta_x, \theta_y)|\psi_T\rangle \, .$$

Now we need to figure out how to measure this cost function? If we know the exact density matrix $\rho(\theta_x, \theta_y)$, we can calculate the expectation value and hence, the cost function. However, there is no way to find the exact density matrix $\rho(\theta_x, \theta_y)$ because of the laws of quantum mechanics. The best we can do if to measure the outcome of a circuit in a computational basis. Therefore, to measure this cost function, we first have to write it in terms of an outcome of a circuit on a computational basis.

To do this, note that we can write the target state $|\psi_T\rangle$ as

$$|\psi_T\rangle \ = \ CNOT \cdot \left(R_y(-\pi/2) \otimes I\right) |11\rangle.$$

Using this observation, we can write the cost function $C(\theta_x, \theta_y)$ as

$$C(\theta_x, \theta_y) \ = \ -\langle 11| \left(R_y(\pi/2) \otimes I\right) \cdot CNOT \cdot \rho(\theta_x, \theta_y) \cdot CNOT \cdot \left(R_y(-\pi/2) \otimes I\right) |11\rangle.$$

Equivalently, we can write this as

$$C(\theta_x, \theta_y) \ = \ -\langle 11| \, \tilde{\rho}(\theta_x, \theta_y) \, |11\rangle$$

where $\tilde{\rho}(\theta_x, \theta_y)$ is defined as

$$\tilde{\rho}(\theta_x, \theta_y) \equiv \ \left(R_y(\pi/2) \otimes I\right) \cdot CNOT \cdot \rho(\theta_x, \theta_y) \cdot CNOT \cdot \left(R_y(-\pi/2) \otimes I\right) .$$

This modified formula for the cost function is exactly what we needed. According to this formula, the cost function $C(\theta_x, \theta_y)$ can be calculated by first implementing a (noisy) circuit that prepares the state $\tilde{\rho}(\theta_x, \theta_y)$ and then calculating the probability of measuring $|11\rangle$. Therefore, now we need to implement a circuit that prepares $\tilde{\rho}(\theta_x, \theta_y)$.

From the definition of $\tilde{\rho}(\theta_x, \theta_y)$ given above, we can deduce that the state $\tilde{\rho}(\theta_x, \theta_y)$ can be constructed by first applying a $CNOT$ gate on $\rho(\theta_x, \theta_y)$ and then rotating the first qubit by amount $\pi/2$ around the y-axis. This means that we can start with a noisy variational circuit that we implemented in Sec. (2.3) and add a $CNOT$ gate followed by a $R_y(\pi/2)$ gate on the first qubit. We can implement this modified circuit on cirq using the following lines of codes:

```
#We first make the noisy variational circuit.
circuit = noisy_variational_circuit(theta_x,theta_y,noise_type,noise_probability)
#Then we add the CNOT gate:
circuit.append(cirq.CNOT(q0,q1))
#Add then the rotation gate on the first qubit
circuit.append(cirq.ry(np.pi/2).on(q0))
```

Now we write a function that we call `cost_function` which first implements the above circuit and then calculates the cost function by calculating the probability of measuring $|11\rangle$. This function takes five inputs. Four of these are the same as the inputs of the function `noisy_variational_circuit` which we defined in Sec. (2.3). The extra input is 'no_of_measurements' which is the number of measurements that we do to calculate the probability of measuring $|11\rangle$.

The code for this function is presented here:

In [4]:
```
def cost_function(theta_x,theta_y,no_of_measurements,noise_type,noise_probabil
ity):

    #We first make the noisy variational circuit.
    circuit = noisy_variational_circuit(theta_x,theta_y,noise_type,noise_proba
bility)
    #Then we add the CNOT gate:
    circuit.append(cirq.CNOT(q0,q1))
    #Add then the rotation gate on the first qubit
    circuit.append(cirq.ry(np.pi/2).on(q0))

    #We then add the measurement
    circuit.append(cirq.measure(q0, q1, key='result'))


    s = cirq.DensityMatrixSimulator()
    # We make measurements. The number of measurements is no_of_measurements
    samples = s.run(circuit,repetitions=no_of_measurements)
    probability_of_11 = (samples.histogram(key='result')[3])/no_of_measurement
s

    cost = -1.0*probability_of_11

    return cost
```

Now given the cost function (and a code that implements it), our next task is to take its gradients with respect to the variational parameters θ_x and θ_y.

Sec 2.5: Gradients of the cost function

To perform the gradient descent method, we need to take the gradients of our cost function with respect to the variational parameters. We do this by using the **parameter-shift rule** of this paper (https://arxiv.org/pdf/2008.06517.pdf) (also see this paper (https://arxiv.org/pdf/1811.04968.pdf)). Let us briefly review this method. Suppose we have a function $F(\theta)$ of parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$ which is of the form

$$F(\theta) \;=\; \langle 0|U^\dagger(\theta)\, M\, U(\theta)|0\rangle\,,$$

where M is an arbitrary Hermitian operator and $U(\theta) = V_1 R(\theta_1) \ldots V_1 R(\theta_1)$ where V_i and $R_i(\theta_i)$ is an arbitrary constant unitary circuit and an arbitrary rotation circuit respectively. Then the parameter-shift rule says that the gradient of $F(\theta)$ is given by

$$\frac{\partial F(\theta)}{\partial \theta_i} \;=\; \frac{F(\theta + t\mathbf{e}_i) - F(\theta - t\mathbf{e}_i)}{2\sin(t)} \qquad \text{for any} \quad t \neq \pi \times \mathbb{Z}\,,$$

where $\mathbf{e}_i$ is a unit vector in the direction of θ_i.

Note that the parameter-shift rule is an exact formula that can be derived using trigonometric identities and properties of the Pauli matrices. Instead of rederiving this rule here, I want to argue in this subsection that this rule is also applicable for the cost function $C(\theta_x, \theta_y)$ that we have defined in Sec. (2.4) even in the **presence** of noise. To argue this, all we have to show is that the cost function $C(\theta_x, \theta_y)$ that we have defined in Sec. (2.4) can be written as $F(\theta)$ that we have defined above.

(**Note:** After we did our calculation and showed that the parameter-shift rule is also applicable in the presence of noise, we found that this paper (https://arxiv.org/pdf/2006.06303.pdf) and this article (https://pennylane.ai/qml/demos/tutorial_noisy_circuit_optimization.html) where this has also been shown recently. Nevertheless, we still present our analysis as we will use it later in our analysis in Sec. (3.2))

Let us sweep through the noisy_variational_circuit and understand what the state looks like after each layer or moment of the circuit. After the rotation operators are applied, the state is given by

$$|\psi_0(\theta_x, \theta_y)\rangle \;=\; U(\theta_x, \theta_y)\,|00\rangle$$

where the unitary operator $U(\theta_x, \theta_y)$ is given by

$$U(\theta_x, \theta_y) \;=\; R_y(\theta_y) \,\otimes R_x(\theta_x)\,.$$

We can equivalently write this as a density matrix

$$\rho_0(\theta_x, \theta_y) \;\equiv |\psi_0(\theta_x, \theta_y)\rangle\langle\psi_0(\theta_x, \theta_y)| \;=\; U(\theta_x, \theta_y)\,|00\rangle\langle 00|\,U^\dagger(\theta_x, \theta_y)\,.$$

The noisy quantum gates act on the density matrix as a quantum channel. Let us assume that the probability that the noisy gate N will be applied is p where $0 \leq p \leq 1$. Then the action of a noisy gate N on a density matrix is described in terms of a quantum channel $\mathcal{N}$ such that

$$\mathcal{N}: \quad \rho \;\to\; \mathcal{N}[\rho] \;=\; (1-p)\,\rho \;+\; p\,N\,\cdot\,\rho\,\cdot\,N^\dagger\,.$$

Now let us assume that in the `noisy_variational_circuit`, the noise gate acting on qubit 0 is N_0 with probability p_0 whereas the noise gate acting of qubit 1 is N_1 with probability p_1. (**Note:** In this exercise, we only take N_0 and N_1 to be a bit-flip and/or a phase-flip gate. However, our analysis in this subsection is for an arbitrary type of noise.) Then the state of two-qubits after passing through the noisy gates is given by

$$\begin{aligned}
\rho_1(\theta_x, \theta_y) \;=\; & (1-p_0)(1-p_1)\,\rho_0(\theta_x, \theta_y) \\
& +\; p_0(1-p_1)\,(N_0 \otimes I)\,\rho_0(\theta_x, \theta_y)\,(N_0^\dagger \otimes I) \\
& +\; p_1(1-p_0)\,(I \otimes N_1)\,\rho_0(\theta_x, \theta_y)\,(I \otimes N_1^\dagger) \\
& +\; p_0\,p_1\,(N_0 \otimes N_1)\,\rho_0(\theta_x, \theta_y)\,(N_0^\dagger \otimes N_1^\dagger)\,.
\end{aligned}$$

After the noisy gates, the qubits in the noisy_variational_circuit pass through the $CNOT$ gate. The final state after passing through the $CNOT$ gate is then given by

$$\begin{aligned}\rho(\theta_x,\theta_y) \;=\; & (1-p_0)(1-p_1)\,CNOT\,\rho_0(\theta_x,\theta_y)\,CNOT \\ & +\; p_0(1-p_1)\,CNOT\,(N_0\otimes I)\,\rho_0(\theta_x,\theta_y)\,(N_0^\dagger\otimes I)\,CNOT \\ & +\; p_1(1-p_0)\,CNOT\,(I\otimes N_1)\,\rho_0(\theta_x,\theta_y)\,(I\otimes N_1^\dagger)\,CNOT \\ & +\; p_0\,p_1\,CNOT\,(N_0\otimes N_1)\,\rho_0(\theta_x,\theta_y)\,(N_0^\dagger\otimes N_1^\dagger)\,CNOT\,.\end{aligned}$$

Now we use this density matrix and compute the cost function $C(\theta_x,\theta_y)$ that we have defined in Sec. (2.4):

$$C(\theta_x,\theta_y) \;=\; -\,\langle\psi_T|\,\rho(\theta_x,\theta_y)\,|\psi_T\rangle \;=\; -\,\mathrm{tr}\big(\rho_T\cdot\rho(\theta_x,\theta_y)\big)\,.$$

Now using the above expression for $\rho(\theta_x,\theta_y)$ and the cyclic property of the trace, we can simplify the cost function and can write it as

$$C(\theta_x,\theta_y) \;=\; -\,\mathrm{tr}\big(M_T\cdot\rho_0(\theta_x,\theta_y)\big)\,,$$

where M_T is a Hermitian matrix given by

$$\begin{aligned}M_T \;\equiv\; & (1-p_0)(1-p_1)\,CNOT\,\rho_T\,CNOT \\ & +\; p_0(1-p_1)\,(N_0^\dagger\otimes I)\,CNOT\,\rho_T\,CNOT\,(N_0\otimes I) \\ & +\; p_1(1-p_0)\,(I\otimes N_1^\dagger)\,CNOT\,\rho_T\,CNOT\,(I\otimes N_1) \\ & +\; p_0\,p_1\,(N_0^\dagger\otimes N_1^\dagger)\,CNOT\,\rho_T\,CNOT\,(N_0\otimes N_1)\,.\end{aligned}$$

Now since $\rho_0(\theta_x,\theta_y) \;=\; U(\theta_x,\theta_y)\,|00\rangle\langle 00|\,U^\dagger(\theta_x,\theta_y)$, we can write the cost function as

$$C(\theta_x,\theta_y) \;=\; -\,\langle 00|\,U^\dagger(\theta_x,\theta_y)\,M_T\,U(\theta_x,\theta_y)\,|00\rangle\,.$$

This proves that the cost function $C(\theta_x,\theta_y)$ is of the same form as $F(\theta)$ that we have introduced at the beginning of this subsection. This implies that we can apply the parameter-shift rule to the cost function $C(\theta_x,\theta_y)$ even though there was noise in our variational circuit.

Having established that the parameter-shift rule is applicable for our purpose, we apply it on our cost function to get

$$\frac{\partial C(\theta_x,\theta_y)}{\partial\theta_x} \;=\; \frac{C(\theta_x+t,\theta_y)-C(\theta_x-t,\theta_y)}{2\sin(t)}\,,$$

and

$$\frac{\partial C(\theta_x,\theta_y)}{\partial\theta_y} \;=\; \frac{C(\theta_x,\theta_y+t)-C(\theta_x,\theta_y-t)}{2\sin(t)}\,.$$

Even though these formulas are applicable for all $t\neq\pi\mathbb{Z}$, a convenient choice is to fix $t=\pi/2$ so that $\sin(t)$ drops out from the denominator. We choose to take $t=\pi/2$ in our codes for this reason.

Now we write a function which we call `finding_gradients` which calculates the gradients of the cost function using these formulas. This function takes five inputs which are the same as the inputs for the `cost_function` that we defined in Sec. (2.4). As an output, this function returns a tuple whose elements are the gradients of the cost function: $[\partial_{\theta_x}C(\theta_x,\theta_y)\,,\;\partial_{\theta_y}C(\theta_x,\theta_y)]$.

The code for this function is:

In [5]:
```
def finding_gradients(theta_x,theta_y,no_of_measurements,noise_type,noise_prob
ability):

    t = 0.5*np.pi # A convenient choice for the constant shift parameter

    # Gradient w.r.t theta_x
    # Calculating C(theta_x + t)
    cost_plus = cost_function(theta_x+t,theta_y,no_of_measurements,noise_type,
noise_probability)
    # Calculating C(theta_x - t)
    cost_minus = cost_function(theta_x-t,theta_y,no_of_measurements,noise_type
,noise_probability)
    # Using parameter-shift rule
    x_gradient = (cost_plus-cost_minus)/(2)

    # Gradient w.r.t theta_y
    # Calculating C(theta_y + t)
    cost_plus = cost_function(theta_x,theta_y+t,no_of_measurements,noise_type,
noise_probability)
    # Calculating C(theta_y - t)
    cost_minus = cost_function(theta_x,theta_y-t,no_of_measurements,noise_type
,noise_probability)
    # Using parameter-shift rule
    y_gradient = (cost_plus-cost_minus)/(2)

    return x_gradient , y_gradient
```

We will use this function in this next subsection in which we finally perform gradient descent to optimize the cost function $C(\theta_x, \theta_y)$.

Sec 2.6: Gradient descent

In this subsection, we perform the gradient descent method to find the values of variational parameters θ_x and θ_y for which the cost function attains the minimum value. Gradient descent is an iterative algorithm in which we modify our variational parameters at each step. In particular, we change our variational parameters in the direction opposite to the gradient of the cost function as this is the direction in which the cost function decreases the most. More precisely, at each iterative step, we change our parameters like this:

$$\theta_x^{(\text{new})} = \theta_x^{(\text{old})} - \eta^{(\text{old})} \, \partial_{\theta_x} C(\theta_x, \theta_y) \, \big|_{\theta_x^{\text{old}}, \theta_y^{\text{old}}} ,$$

and

$$\theta_y^{(\text{new})} = \theta_y^{(\text{old})} - \eta^{(\text{old})} \, \partial_{\theta_y} C(\theta_x, \theta_y) \, \big|_{\theta_x^{\text{old}}, \theta_y^{\text{old}}} .$$

The parameters η is called the **learning rate**. In practice, it can be different at each step. However, we will use the same learning rate at each step for the sake of simplicity.

To calculate new parameters at each iterative step, we write a function that we call `new_parameters` . This function takes six inputs. Five of these are the same as the inputs of the `cost_function` and the `finding_gradients` functions. The extra input for this function is the learning rate. This function returns updated variational parameters as elements of a tuple. The tuple will be of the form $[\theta_x^{(\text{new})}, \theta_y^{(\text{new})}]$.

The code for this function is:

In [6]:
```
def new_parameters(theta_x,theta_y,learning_rate,no_of_measurements,noise_type
,noise_probability):

    # Find gradients of the cost functions
    gradients = finding_gradients(theta_x,theta_y,no_of_measurements,noise_typ
e,noise_probability)

    # Update variational parameters
    theta_x = theta_x - learning_rate*gradients[0]
    theta_y = theta_y - learning_rate*gradients[1]

    return theta_x, theta_y
```

Ideally, the learning rate should not be too high otherwise we may have to worry about *overshooting* the minimum point. In our analysis, we found by trial-and-error that the value of $\eta = 0.2$ yields desired results. For this reason, we will stick with this learning rate.

In the gradient descent algorithm, as discussed above, we perform the update of parameters iteratively. To do this, we write a function that we call `gradient_descent`. This function takes four inputs. Three of the inputs are *no_of_measurements*, *noise_type*, and *noise_probability* which are also inputs for the previously defined functions. The fourth input for this function is *no_of_iterations*. This is the number of times the function updates the variational parameters. This function starts the optimization procedure from random initial variational parameter θ_x and θ_y. Then it uses the function `new_parameters` to find modified parameters. As an output, this function returns a tuple of optimum values of parameters.

The code for this function is given here:

In [7]:
```
import random

def gradient_descent(no_of_iterations,no_of_measurements,noise_type,noise_prob
ability):

    # choosing random initial values
    # for the variational parameters.
    theta_x = (random.random())*np.pi
    theta_y = (random.random())*np.pi


    learning_rate = 0.2 # Just a convenient choice because it seemed to work.

    # performing the iterative modification of parameters.
    for i in range(no_of_iterations):
        new_angles = new_parameters(theta_x,theta_y,learning_rate,no_of_measur
ements,noise_type,noise_probability)
        theta_x = new_angles[0]
        theta_y = new_angles[1]

    return theta_x,theta_y
```

Before we proceed further, it may be instructive at this stage to run our codes for a test case for which we already know the correct answer and see if they yield the expected result. The simplest test case for us is the one where there is no noise in our circuit. In this case, we know from Sec. (2.1) that the optimum values of variational parameters are $\theta_x = \pi$ and $\theta_y = \pi/2$. We now run our codes for the case when there is no noise.

In [8]:
```
# Noise-free setup means that
# the probability of noise is 0.0
noise_probability = [0.0,0.0]

# The type of noise does not
# as they will never be applied.
noise_type = ['bit_flip','phase_flip']

# Find the optimum value using
# gradient descent
no_of_iterations = 100
no_of_measurements = 1000
print(gradient_descent(no_of_iterations,no_of_measurements,noise_type,noise_pr
obability))
```

```
(3.140300438102256, 1.557059481651797)
```

We can see from the output that we get the expected result. This gives us confidence that our code is working as expected.

Sec 2.7: Main circuit

We are now finally in a position to write a function that returns a circuit that we wanted. That is, it prepares a circuit that returns $|01\rangle$ and $|10\rangle$ with 50% probabilities. We call this function `main_circuit` . This function takes four inputs which are the same as the inputs for the `gradient_descent` function. The code for this function is here:

In [9]:
```
def main_circuit(no_of_iterations,no_of_measurements,noise_type,noise_probabil
ity):

    # Find optimum parameters using
    # gradient descent
    parameters = gradient_descent(no_of_iterations,no_of_measurements,noise_ty
pe,noise_probability)

    return noisy_variational_circuit(parameters[0],parameters[1],noise_type,no
ise_probability)
```

It may again be useful to run this code for the case when there is no noise. We expect that we will end up with the noise-free circuit of Sec. (2.1).

In [10]:
```
# Noise-free setup means that
# the probability of noise is 0.0
noise_probability = [0.0,0.0]
print(main_circuit(100,1000,['bit_flip','phase_flip'],noise_probability))
```

```
0: ──Ry(0.502π)──@──
                 │
1: ──Rx(π)───────X──
```

This is the same as the noise-free circuit from Sec. (2.1). This gives us some more confidence that our code is working as expected.

This finishes our task of implementing a circuit that returns $|01\rangle$ and $|10\rangle$ with 50% probabilities. In the next section, we try this circuit out in the presence of noise and observe how it performs.

Sec 3: Testing Our Circuit With Phase-Flip Noise

In this section, we run our circuit with the assumption that the only possible type of noise present is the 'phase-flip'. Moreover, we assume that the phase of each qubit is flipped about 10% of the time.

In [11]:
```
# One each qubit, there is a 0.1 probability
# of phase-flip:
noise_type = ['phase_flip','phase_flip']
noise_probability = [0.1,0.1]
```

Now we run our `main_circuit` and measure both of the qubits. We want to find out what fraction of times we get $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$. This can be done using the following lines of codes:

In [13]:
```
# Measuring the output of our main_circuit

# We make 1000 measurements
no_of_measurements = 1000

# We make 100 iterations in our gradient descent
no_of_iterations = 100

# Preparing out circuit
circuit = main_circuit(no_of_iterations,no_of_measurements,noise_type,noise_pr
obability)

# Adding a measurement gate
circuit.append(cirq.measure(q0,q1,key='result'))

# Running the circuit
noisy_sim = cirq.DensityMatrixSimulator()
fin_result = noisy_sim.run(circuit,repetitions=no_of_measurements)

# Displaying our results
print('Fraction of times we get |00> is ' + str((fin_result.histogram(key='res
ult')[0])/no_of_measurements))
print('Fraction of times we get |01> is ' + str((fin_result.histogram(key='res
ult')[1])/no_of_measurements))
print('Fraction of times we get |10> is ' + str((fin_result.histogram(key='res
ult')[2])/no_of_measurements))
print('Fraction of times we get |11> is ' + str((fin_result.histogram(key='res
ult')[3])/no_of_measurements))
```

```
Fraction of times we get |00> is 0.0
Fraction of times we get |01> is 0.508
Fraction of times we get |10> is 0.492
Fraction of times we get |11> is 0.0
```

As we wanted, our circuit only returns $|01\rangle$ and $|10\rangle$ and with equal probabilities.

Therefore, we have been able to achieve the assigned task for the case when only the phase-flip noise is present.

It is also instructive to compare our results for different numbers of measurements. We do this for $[1000, 100, 10, 1]$ number of measurements using the following code:

In [14]:
```
# Measuring the output of our main_circuit
# for various values of number of measurements:

no_of_measurements = [1000,100,10,1]

# We make 100 iterations in our gradient descent
no_of_iterations = 100

for number in no_of_measurements:
    # Preparing out circuit
    circuit = main_circuit(no_of_iterations,number,noise_type,noise_probabilit
y)

    # Adding a measurement gate
    circuit.append(cirq.measure(q0,q1,key='result'))

    # Running the circuit
    noisy_sim = cirq.DensityMatrixSimulator()
    fin_result = noisy_sim.run(circuit,repetitions=number)

    # Displaying our results
    print('For '+str(number)+' number of measurements: ')
    print('Fraction of times we get |01> is ' + str((fin_result.histogram(key=
'result')[1])/number))
    print('Fraction of times we get |10> is ' + str((fin_result.histogram(key=
'result')[2])/number))
```

```
For 1000 number of measurements:
Fraction of times we get |01> is 0.518
Fraction of times we get |10> is 0.482
For 100 number of measurements:
Fraction of times we get |01> is 0.55
Fraction of times we get |10> is 0.45
For 10 number of measurements:
Fraction of times we get |01> is 0.4
Fraction of times we get |10> is 0.6
For 1 number of measurements:
Fraction of times we get |01> is 1.0
Fraction of times we get |10> is 0.0
```

We find that the probability of measuring $|01\rangle$ and $|10\rangle$ is closer to each other when we make larger number of measurements. This is not an unexpected result.

Even though we have been successful in achieving what we wanted, that is, implementing a circuit that only returns $|01\rangle$ and $|10\rangle$ and with equal probabilities, we should also figure out if we have indeed produced the target state $|\psi_T\rangle$. This is part of the bonus question. We discuss this in the following subsection.

Sec 3.1: Does our circuit indeed produces $|\psi_T\rangle$? (BONUS QUESTION)

In this subsection, we describe a procedure which one can follow to verify if the output state of our `main_circuit` is indeed $|\psi_T\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$ or some other equal probability superposition of $|01\rangle$ and $|10\rangle$. We then follow this procedure in this subsection and present our results.

To verify the output state of our circuit, we can repeat the *trick* that we used in Sec. (2.4) in our analysis of the cost function. As we discussed in Sec. (2.4), the state $|\psi_T\rangle$ can be written as

$$|\psi_T\rangle = CNOT \cdot \left(R_y(-\pi/2) \otimes I\right)|11\rangle .$$

We can invert this to write

$$|11\rangle = \left(R_y(\pi/2) \otimes I\right) \cdot CNOT\,|\psi_T\rangle .$$

This observation suggests that we should *append* our `main_circuit` by a CNOT gate followed by the rotation of the first qubit by amount $\pi/2$ around the y-axis. (This is precisely the same as what we did in Sec. (2.4).) Then we should measure the state of the two qubits. If we get $|11\rangle$ with 100%, then this would guarantee that our circuit indeed produces $|\psi_T\rangle$.

We perform this analysis using the following code:

In [15]:
```
#We start with our circuit:
no_of_measurements = 1000
no_of_iterations = 100
circuit = main_circuit(no_of_iterations,no_of_measurements,noise_type,noise_pr
obability)

#Then we add the CNOT gate:
circuit.append(cirq.CNOT(q0,q1))
#And then the rotation gate on the first qubit
circuit.append(cirq.ry(np.pi/2).on(q0))
#And then the measurement gate
circuit.append(cirq.measure(q0, q1, key='result'))

# We now run our modified circuit
# and make measurements.
s = cirq.DensityMatrixSimulator()
samples = s.run(circuit,repetitions=no_of_measurements)

#Display our results:
print('Fraction of times we get |00> is ' + str((samples.histogram(key='resul
t')[0])/no_of_measurements))
print('Fraction of times we get |01> is ' + str((samples.histogram(key='resul
t')[1])/no_of_measurements))
print('Fraction of times we get |10> is ' + str((samples.histogram(key='resul
t')[2])/no_of_measurements))
print('Fraction of times we get |11> is ' + str((samples.histogram(key='resul
t')[3])/no_of_measurements))
```

```
Fraction of times we get |00> is 0.0
Fraction of times we get |01> is 0.099
Fraction of times we get |10> is 0.0
Fraction of times we get |11> is 0.901
```

This shows that we only measure $|11\rangle$ about 90% of the times. This implies that our circuit prepares $|\psi_T\rangle$ with a success rate of 90%.

Sec 3.2: Interpreting our results

In Sec. (2), the cost function that we chose attains it **minimum** value when the state prepared by our `noisy_variational_circuit` is the same as the target state $|\psi_T\rangle$. Then we performed the gradient descent algorithm so that we can find the values of variational parameters at which the cost function is minimum. However, we find that our `noisy_variational_circuit` does not prepare $|\psi_T\rangle$ with 100% success probability. What is the reason for this? Does this mean there is something wrong with our implementation of gradient descent? We will argue in this subsection that gradient descent has performed just as expected and it has indeed returned the **stationary point** of the cost function.

To see this, let us start by looking at what values of variational parameters our `gradient_descent` returns when there is a 10% chance of phase-flip on each qubit. This can be done with the following lines of codes:

In [18]:
```
# One each qubit, there is a 0.1 probability
# of phase-flip:
noise_type = ['phase_flip','phase_flip']
noise_probability = [0.1,0.1]

# Defining iterations parameters
no_of_iterations = 100
no_of_measurements = 1000

#Displaying optimal values of theta_x, theta_y
print(gradient_descent(no_of_iterations,no_of_measurements,noise_type,noise_pr
obability))
```

```
(3.1437447196323722, 1.5648643472527735)
```

Surprisingly, we have found that the optimal values for the variational parameters in the presence of noise are the same as the values of the variational parameters when there was no noise (see Sec. (2.1)).

Now let us find out what is the actual value of the cost function for these parameters. This can be done using the `cost_function` :

In [21]:
```
theta_x , theta_y = gradient_descent(no_of_iterations,no_of_measurements,noise
_type,noise_probability)
print(cost_function(theta_x,theta_y,no_of_measurements,noise_type,noise_probab
ility))
```

```
-0.901
```

Interestingly, the value of the cost function, in this case, is greater than the minimum possible value of -1.0. These two observations tell us that the values of the variational parameters are unchanged by the presence of the noise whereas the actual value of the cost function has changed.

In the following, we will explain why these values indeed correspond to the **stationary point** of the cost function, and hence, the gradient descent algorithm is indeed performing as it should be. To see this, recall from Sec. (2.5) that the cost function can be written as

$$C(\theta_x, \theta_y) \;=\; -\langle 00|\, U^\dagger(\theta_x, \theta_y)\, M_T\, U(\theta_x, \theta_y)\, |00\rangle\,,$$

where M_T is a Hermitian matrix given by a sum of four terms $M_T \;=\; M_T^{(1)} \;+\; M_T^{(2)} \;+\; M_T^{(3)} \;+\; M_T^{(4)}$, and (we have taken $N_0 \;=\; N_1 \;=\; Z$ since we are only considering phase flip in this section)

$$\begin{aligned}
M^{(1)})_T \;&=\; (1-p_0)(1-p_1)\, CNOT\, |\psi_T\rangle\langle\psi_T|\, CNOT \\
M_T^{(2)} \;&=\; p_0(1-p_1)\, (Z_0 \otimes I)\, CNOT\, |\psi_T\rangle\langle\psi_T|\, CNOT\, (Z \otimes I) \\
M_T^{(3)} \;&=\; p_1(1-p_0)\, (I \otimes Z)\, CNOT\, |\psi_T\rangle\langle\psi_T|\, CNOT\, (I \otimes Z) \\
M_T^{(4)} \;&=\; p_0\, p_1\, (Z \otimes Z)\, CNOT\, |\psi_T\rangle\langle\psi_T|\, CNOT\, (Z \otimes Z)
\end{aligned}$$

This means we can also write the cost function as a sum of four terms

$$C(\theta_x, \theta_y) \;=\; (1-p_0)(1-p_1)\, C^{(1)}(\theta_x, \theta_y) \;+\; p_0(1-p_1)\, C^{(2)}(\theta_x, \theta_y) \;+\; p_1(1-p_0)\, C^{(3)}(\theta_x, \theta_y) \;+\; p$$

where

$$\begin{aligned}
C^{(1)}(\theta_x, \theta_y) \;&=\; -\left|\left|\langle\psi_T|\, CNOT\, U(\theta_x, \theta_y)\, |00\rangle\right|\right|^2, \\
C^{(2)}(\theta_x, \theta_y) \;&=\; -\left|\left|\langle\psi_T|\, CNOT\, (Z \otimes I)\, U(\theta_x, \theta_y)\, |00\rangle\right|\right|^2, \\
C^{(3)}(\theta_x, \theta_y) \;&=\; -\left|\left|\langle\psi_T|\, CNOT\, (I \otimes Z)\, U(\theta_x, \theta_y)\, |00\rangle\right|\right|^2, \\
C^{(4)}(\theta_x, \theta_y) \;&=\; -\left|\left|\langle\psi_T|\, CNOT\, (Z \otimes Z)\, U(\theta_x, \theta_y)\, |00\rangle\right|\right|^2.
\end{aligned}$$

Now we evaluate these cost functions for $\theta_x = \pi$ and $\theta_y = \pi/2$. We get

$$\begin{aligned}
C^{(1)}(\theta_x = \pi, \theta_y = \pi/2) \;&=\; -1\,, \\
C^{(2)}(\theta_x = \pi, \theta_y = \pi/2) \;&=\; 0\,, \\
C^{(3)}(\theta_x = \pi, \theta_y = \pi/2) \;&=\; -1\,, \\
C^{(4)}(\theta_x = \pi, \theta_y = \pi/2) \;&=\; 0\,.
\end{aligned}$$

This means that we get the minimum possible values (-1.0) for $C^{(1)}$ and for $C^{(3)}$ and the maximum possible values (0.0) for $C^{(2)}$ and for $C^{(4)}$. Therefore, the gradients of these individual cost functions vanish at $\theta_x = \pi$ and $\theta_y = \pi/2$. Then by linearity, the gradient of the total cost function $C(\theta_x, \theta_y)$ also vanish at $\theta_x = \pi$ and $\theta_y = \pi/2$. This confirms that $\theta_x = \pi$ and $\theta_y = \pi/2$ is a stationary point of the cost function. Hence, our gradient descent yields the correct answer.

Furthermore, we find that the actual value of the cost function at $\theta_x = \pi$ and $\theta_y = \pi/2$ is

$$C(\theta_x = \pi, \theta_y = \pi/2) \;=\; -(1-p_0)(1-p_1) \;-\; p_1(1-p_0) \;=\; -(1-p_0)\,.$$

When $p_0 \;=\; 0.1$, the cost function at the stationary point becomes $C(\theta_x = \pi, \theta_y = \pi/2) \;=\; -0.9$. This is indeed the value that we found above using our `cost_function` function.

(**Note**: After we did our analysis, we found this (https://pennylane.ai/qml/demos/tutorial_noisy_circuit_optimization.html) where similar discussion is presented.)

The above analysis shows that our implementations of the gradient descent function work exactly as it should. It is unfortunate that we were not able to make our circuit to prepare $|\psi_T\rangle$ with 100% probability. We expect some other optimization problem which is more suitable for the noisy circuit will resolve this issue.

Sec 4: Conclusions

In this project, we have implemented a noisy circuit that returns $|01\rangle$ and $|10\rangle$ with equal probabilities. We started with a variational circuit and performed gradient descent to find the optimal parameters. The codes for our project are presented in Sec. (2). We ran our code in Sec. (3) for a case when there is 10% probability of phase-flip of each qubit. We found that our circuit indeed returns $|01\rangle$ and $|10\rangle$ with equal probabilities. We also checked in Sec. (3) whether our circuit returns $|\psi_T\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$ or some other equal-probability superposition. We found that our circuit prepares $|\psi_T\rangle$ most of the times but not always. We discussed why this is consistent with the approach of the gradient_descent.

References

1. Estimating the gradient and higher-order derivatives on quantum hardware, by A. Mari, T.R. Bromley, and N. Killoran: *https://arxiv.org/pdf/2008.06517.pdf (https://arxiv.org/pdf/2008.06517.pdf)*.
2. PennyLane: Automatic differentiation of hybrid quantumclassical computations: *https://arxiv.org/pdf/1811.04968.pdf (https://arxiv.org/pdf/1811.04968.pdf)*
3. A variational toolbox for quantum multi-parameter estimation, by J.J. Meyer, J. Borregaard, and J. Eisert: *https://arxiv.org/pdf/2006.06303.pdf (https://arxiv.org/pdf/2006.06303.pdf)*
4. Optimizing noisy circuits with Cirq: *https://pennylane.ai/qml/demos/tutorial_noisy_circuit_optimization.html (https://pennylane.ai/qml/demos/tutorial_noisy_circuit_optimization.html)*

In [ ]: