Hi! This is my attempt at solving the second question.

In [1]:
```
import pennylane as qml
from pennylane import numpy as np
from pennylane.qnodes import PassthruQNode
from pennylane_cirq import ops as cirq_ops
import random as rand
```

The device is set up for 10 measurements and a depolarizing channel is added to simulate noise. Although Rx and Ry gates are not required on both wires I included them regardless.

In [2]:
```
dev2 = qml.device("cirq.mixedsimulator", wires=2, shots=10, analytic=Fa
lse)
noise_param=rand.random()
@qml.qnode(dev2)
def circuit(x):
    qml.RY(x[0], wires=0)
    qml.RX(x[1], wires=0)
    qml.RX(x[2], wires=1)
    qml.RY(x[3], wires=1)
    qml.CNOT(wires=[0,1])
    cirq_ops.Depolarize(noise_param, wires=0)
    cirq_ops.Depolarize(noise_param, wires=1)
    return qml.probs(wires=[0,1])
```

probs(wires=[]) will give the probability of each of the four possible states. We need the |01> and the |10> states to be equally probable, so I set up a cost function as the negative of the product of their probabilities. Minimizing this cost function should give us -0.25.

In [3]:
```
def cost(x):
    cir = circuit(x)
    return -(cir[1])*(cir[2])
```

Giving random starting values to the parametric gates:

In [4]:
```
import random as rand
init_params = np.array([rand.random(),rand.random(),rand.random(),rand.
random()])
print(cost(init_params))
```

```
-0.0
```

Using pennylane's gradient descent optimizer to minimize the cost function:

In [5]:
```
opt = qml.GradientDescentOptimizer(stepsize=0.1)

# set the number of steps
steps = 500
# set the initial parameter values
params = init_params

for i in range(steps):
    # update the circuit parameters
    params = opt.step(cost, params)

    cs = cost(params)
    if (i + 1) % 100 == 0:
        print("Cost after step {:5d}: {: .7f}".format(i + 1, cs))

print("Optimized rotation angles: {}".format(params))
```

```
Cost after step   100: -0.1200000
Cost after step   200: -0.2500000
Cost after step   300: -0.2400000
Cost after step   400: -0.2400000
Cost after step   500: -0.1600000
Optimized rotation angles: [1.61078468 0.66908669 0.0241705  3.1021175
8]
```

We get the required probabilities:

In [6]:
```
circuit (params)
```

Out[6]:
```
array([0. , 0.5, 0.5, 0. ])
```

Doing the same for 100 shots:

In [8]:
```
dev2 = qml.device("cirq.mixedsimulator", wires=2, shots=100, analytic=F
alse)
noise_param=rand.random()
@qml.qnode(dev2)
def circuit(x):
    qml.RY(x[0], wires=0)
    qml.RX(x[1], wires=0)
    qml.RX(x[2], wires=1)
    qml.RY(x[3], wires=1)
    qml.CNOT(wires=[0,1])
    cirq_ops.Depolarize(noise_param, wires=0)
    cirq_ops.Depolarize(noise_param, wires=1)
    return qml.probs(wires=[0,1])
```

In [9]:
```
def cost(x):
    cir = circuit(x)
    return -(cir[1])*(cir[2])
```

In [10]:
```
import random as rand
init_params = np.array([rand.random(),rand.random(),rand.random(),rand.
random()])
print(cost(init_params))
```

```
-0.0006
```

In [15]:
```
opt = qml.GradientDescentOptimizer(stepsize=0.15)

# set the number of steps
steps = 500
# set the initial parameter values
```

```
params = init_params

for i in range(steps):
    # update the circuit parameters
    params = opt.step(cost, params)

    cs = cost(params)
    if (i + 1) % 100 == 0:
        print("Cost after step {:5d}: {: .7f}".format(i + 1, cs))

print("Optimized rotation angles: {}".format(params))
```

```
Cost after step   100: -0.0015000
Cost after step   200: -0.0038000
Cost after step   300: -0.0024000
Cost after step   400: -0.1683000
Cost after step   500: -0.2496000
Optimized rotation angles: [0.31514478 1.55424451 0.01641144 3.1125674
6]
```

In [21]:

```
circuit (params)
```

Out[21]:

```
array([0.  , 0.51, 0.49, 0.  ])
```

Doing the same for 1000 shots:

In [22]:

```
dev2 = qml.device("cirq.mixedsimulator", wires=2, shots=1000, analytic=
False)
noise_param=rand.random()
@qml.qnode(dev2)
def circuit(x):
    qml.RY(x[0], wires=0)
    qml.RX(x[1], wires=0)
    qml.RX(x[2], wires=1)
    qml.RY(x[3], wires=1)
    qml.CNOT(wires=[0,1])
    cirq_ops.Depolarize(noise_param, wires=0)
```

```
    cirq_ops.Depolarize(noise_param, wires=1)
    return qml.probs(wires=[0,1])
```

In [23]:
```
def cost(x):
    cir = circuit(x)
    return -(cir[1])*(cir[2])
```

In [24]:
```
import random as rand
init_params = np.array([rand.random(),rand.random(),rand.random(),rand.
random()])
print(cost(init_params))
```

```
-0.00406
```

In [25]:
```
opt = qml.GradientDescentOptimizer(stepsize=0.15)

# set the number of steps
steps = 500
# set the initial parameter values
params = init_params

for i in range(steps):
    # update the circuit parameters
    params = opt.step(cost, params)

    cs = cost(params)
    if (i + 1) % 100 == 0:
        print("Cost after step {:5d}: {: .7f}".format(i + 1, cs))

print("Optimized rotation angles: {}".format(params))
```

```
Cost after step   100: -0.0525000
Cost after step   200: -0.2477050
Cost after step   300: -0.2498560
Cost after step   400: -0.2500000
Cost after step   500: -0.2499990
Optimized rotation angles: [ 1.13013116e+00  1.57646876e+00  3.13824009
e+00 -2.19498580e-03]
```

In [26]:
```
circuit (params)
```

Out[26]:
```
array([0.   , 0.486, 0.514, 0.   ])
```

This method gives us |10> and |01> with equal probablities. But we don't know if it is the |01> + |10> or |01> - |10> state. Finding a way to get the |01> + |10> state everytime proved easier in theory than in practice. If I use amplitudes instead if probabilites in my code then I could make a cost function with real parts of the amplitude of the |01> and |10> states. Minimizing the product of the real parts of their amplitudes will give me the |10> + |10> state. (by basically eliminating complex parts of the amplitudes)

I could not however, manage to get this to work. I encountered quite a few problems while trying to get the statevector, making a cost function from it and then applying gradient descent (mainly while trying to use autograd with complex parameters), so I'm submitting my solution without the bonus part.

This was an extremely fun exercise and I hope to work on similarly exciting problems if selected. Thanks!