Qiskit | Global Summer School 2021

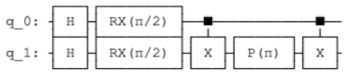
In this lab, you will see how noise affects a typical parameterized quantum circuit used in machine learning using quantum process tomography.

For grading purposes, please specify all simulator arguments (*noise_model=noise_thermal, seed_simulator=3145, seed_transpiler=3145, shots=8192*) in the ***execute*** function.

In [34]:
```
# General tools
import numpy as np
import matplotlib.pyplot as plt

# Qiskit Circuit Functions
from qiskit import  execute,QuantumCircuit, QuantumRegister, ClassicalRegister, Aer, transpile
import qiskit.quantum_info as qi

# Tomography functions
from qiskit.ignis.verification.tomography import process_tomography_circuits, ProcessTomographyFitter
from qiskit.ignis.mitigation.measurement import complete_meas_cal, CompleteMeasFitter

import warnings
warnings.filterwarnings('ignore')
```

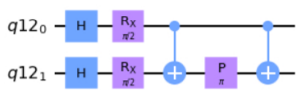
Question 1

- Make this Quantum Circuit



In [ ]:
```

```

In [35]:
```
# YOUR CODE HERE
q = QuantumRegister(2)
circ =QuantumCircuit(q)
circ.h(0)
circ.h(1)
circ.rx(np.pi/2,0)
circ.rx(np.pi/2,1)
circ.cx(0,1)
circ.p(np.pi,1)
circ.cx(0,1)
target_unitary = qi.Operator(circ)
circ.draw('mpl')
```



In [36]:
```
from qc_grader import grade_lab5_ex1

# Note that the grading function is expecting a quantum circuit with no measurements
grade_lab5_ex1(target)
```

```
Traceback (most recent call last):
```

```
  File "<ipython-input-36-b1ac46773a51>", line 4, in <module>
    grade_lab5_ex1(target)
NameError: name 'target' is not defined

Use %tb to get the full traceback.
```

Search for solution online

Quantum Process Tomography with Only Shot Noise

Here we will now use the `qasm_simulator` to simulate a Quantum Process Tomography Circuit

Question 2a

- Using the Process Tomography Circuits function built into qiskit, create the set of circuits to do quantum process tomography and simulation with a qasm simulator (with shot noise only). For this please use the execute function of the QPT Circuits with `seed_simulator=3145`, `seed_transpiler=3145` and `shots=8192`.
- *Hint: The appropriate function, process_tomography_circuits, has been imported above. When complete you should have a total of 144 circuits that are given to the `qasm_simulator` via the `execute` function. You can find out the number of circuits created using `len(qpt_circs)`.*

In [ ]:
```
#YOUR CODE HERE
qpt_circs = process_tomography_circuits(circ, q)

simulator = Aer.get_backend('qasm_simulator')
qpt_job=execute(qpt_circs,simulator,seed_simulator=3145,seed_transpiler=3145,shots=8192)
qpt_result = qpt_job.result()
```

Question 2b

- Using a least squares fitting method for the Process Tomography Fitter, determine the fidelity of your target unitary
- *Hint: First use the ProcessTomographyFitter function above to process the results from question 2a and use ProcessTomographyFitter.fit(method='...') to extract the "Choi Matrix", which effectively describes the measured unitary operation. From here you will use the average_gate_fidelity function from the quantum information module to extract the achieved fidelity of your results*

In [37]:
```
# YOUR CODE HERE
qpt_tomo = ProcessTomographyFitter(qpt_result,qpt_circs)
qpt_lstsq = qpt_tomo.fit(method="lstsq")
fidelity = qi.average_gate_fidelity(qpt_lstsq,target_unitary)
fidelity
```

```
0.9926697818280223
```

In [ ]:
```

```

In [38]:
```
# ALL CODE HERE
q = QuantumRegister(2)
circ = QuantumCircuit(q)
circ.h(0)
circ.h(1)
circ.rx(np.pi/2,0)
circ.rx(np.pi/2,1)
circ.cx(0,1)
circ.p(np.pi,1)
circ.cx(0,1)
target_unitary = qi.Operator(circ)

#circ.h(q[0])
qpt_circs = process_tomography_circuits(circ, q)

simulator = Aer.get_backend('qasm_simulator')
qpt_job=execute(qpt_circs,simulator,seed_simulator=3145,seed_transpiler=3145,shots=8192)
qpt_result = qpt_job.result()

qpt_tomo = ProcessTomographyFitter(qpt_result,qpt_circs)
qpt_lstsq = qpt_tomo.fit(method="lstsq")
fidelity = qi.average_gate_fidelity(qpt_lstsq,target_unitary)
fidelity
```

```
0.9926697818280223
```

In [39]:
```
from qc_grader import grade_lab5_ex2

# Note that the grading function is expecting a floating point number
grade_lab5_ex2(fidelity)
```

```
Submitting your answer for lab5/ex2. Please wait...
Congratulations 🎉! Your answer is correct and has been submitted.
```

In [ ]:

Quantum Process Tomography with a T1/T2 Noise Model

For the sake of consistency, let's set some values to characterize the duration of our gates and T1/T2 times:

In [40]:
```
# T1 and T2 values for qubits 0-3
T1s = [15000, 19000, 22000, 14000]
T2s = [30000, 25000, 18000, 28000]

# Instruction times (in nanoseconds)
time_u1 = 0   # virtual gate
time_u2 = 50  # (single X90 pulse)
time_u3 = 100 # (two X90 pulses)
time_cx = 300
time_reset = 1000  # 1 microsecond
time_measure = 1000 # 1 microsecond
```

In [41]:
```
from qiskit.providers.aer.noise import thermal_relaxation_error
from qiskit.providers.aer.noise import NoiseModel
```

Question 3

- Using the Thermal Relaxation Error model built into qiskit, define `u1`, `u2`, `u3`, `cx`, `measure` and `reset` errors using the values for qubits 0-3 defined above, and build a thermal noise model.
- *Hint: The Qiskit tutorial on building noise models will prove to be useful, particularly where they add quantum errors for `u1`, `u2`, `u3`, `cx`, `reset`, and `measure` errors (please include all of these).*

In [42]:
```
# QuantumError objects
errors_reset = [thermal_relaxation_error(t1, t2, time_reset)
                for t1, t2 in zip(T1s, T2s)]
errors_measure = [thermal_relaxation_error(t1, t2, time_measure)
                  for t1, t2 in zip(T1s, T2s)]
errors_u1  = [thermal_relaxation_error(t1, t2, time_u1)
              for t1, t2 in zip(T1s, T2s)]
errors_u2  = [thermal_relaxation_error(t1, t2, time_u2)
              for t1, t2 in zip(T1s, T2s)]
errors_u3  = [thermal_relaxation_error(t1, t2, time_u3)
              for t1, t2 in zip(T1s, T2s)]
errors_cx = [[thermal_relaxation_error(t1a, t2a, time_cx).expand(
             thermal_relaxation_error(t1b, t2b, time_cx))
              for t1a, t2a in zip(T1s, T2s)]
               for t1b, t2b in zip(T1s, T2s)]

# Add errors to noise model
noise_thermal = NoiseModel()
for j in range(4):
    noise_thermal.add_quantum_error(errors_reset[j], "reset", [j])
    noise_thermal.add_quantum_error(errors_measure[j], "measure", [j])
    noise_thermal.add_quantum_error(errors_u1[j], "u1", [j])
    noise_thermal.add_quantum_error(errors_u2[j], "u2", [j])
    noise_thermal.add_quantum_error(errors_u3[j], "u3", [j])
    for k in range(4):
        noise_thermal.add_quantum_error(errors_cx[j][k], "cx", [j, k])

print(noise_thermal)
```

```
NoiseModel:
  Basis gates: ['cx', 'id', 'u2', 'u3']
  Instructions with noise: ['reset', 'u2', 'measure', 'cx', 'u3']
  Qubits with noise: [0, 1, 2, 3]
  Specific qubit errors: [('reset', [0]), ('reset', [1]), ('reset', [2]), ('reset', [3]), ('measure', [0]), ('measure', [1]),
('measure', [2]), ('measure', [3]), ('u2', [0]), ('u2', [1]), ('u2', [2]), ('u2', [3]), ('u3', [0]), ('u3', [1]), ('u3',
[2]), ('u3', [3]), ('cx', [0, 0]), ('cx', [0, 1]), ('cx', [0, 2]), ('cx', [0, 3]), ('cx', [1, 0]), ('cx', [1, 1]), ('cx', [1,
2]), ('cx', [1, 3]), ('cx', [2, 0]), ('cx', [2, 1]), ('cx', [2, 2]), ('cx', [2, 3]), ('cx', [3, 0]), ('cx', [3, 1]), ('cx',
[3, 2]), ('cx', [3, 3])]
```

In [43]:
```
from qc_grader import grade_lab5_ex3

# Note that the grading function is expecting a NoiseModel
grade_lab5_ex3(noise_thermal)
```

```
Submitting your answer for lab5/ex3. Please wait...
Congratulations 🎉! Your answer is correct and has been submitted.
```

Question 4.

- Get a QPT fidelity using the noise model,but without using any error mitigation techniques. Again, use `seed_simulator=3145`, `seed_transpiler=3145` and `shots=8192` for the `execute` function
- *Hint: The process here should be very similar to that in question 2a/b, except you will need to ensure you include the noise model from question 3 in the `execute` function*

In [44]:
```
np.random.seed(0)
#target_unitary = qi.Operator(circ)

# YOUR CODE HERE
qpt_circs_4 = process_tomography_circuits(circuit=circ, measured_qubits=[0,1])

simulator_4 = Aer.get_backend('qasm_simulator')
qpt_job_4 = execute(qpt_circs_4,simulator_4,seed_simulator=3145,seed_transpiler=3145,shots=8192,noise_model=noise_thermal)
qpt_result_4 = qpt_job_4.result()

qpt_fitter_4 = ProcessTomographyFitter(qpt_result_4, qpt_circs_4)
choi_matrix_4 = qpt_fitter_4.fit(method='lstsq')

fidelity_4= qi.average_gate_fidelity(choi_matrix_4,target_unitary)
fidelity_4
```

```
0.895962996242963
```

In [45]:
```
from qc_grader import grade_lab5_ex4

# Note that the grading function is expecting a floating point number
grade_lab5_ex4(fidelity_4)
```

```
Submitting your answer for lab5/ex4. Please wait...
Congratulations 🎉! Your answer is correct and has been submitted.
```

In [ ]:
```

```

Question 5.

- Use the `complete_meas_cal` function built into qiskit and apply to the QPT results in the previous question. For both, use the `execute` function and `seed_simulator=3145`, `seed_transpiler=3145` and `shots=8192`. Also include the noise model from question 3 in the `execute` function.
- *Hint: The Qiskit textbook has a very good chapter on `readout error mitigation`. Specifically, you will want to use the `complete_meas_cal` function to generate the desired set of circuits to create the calibration matrix with `CompleteMeasFitter` function. This can then be used to generate a correction matrix `meas_filter`. Apply this function to the results from question 4.*

In [70]:
```
meas_EX5_circs, state_labels=complete_meas_cal(qubit_list = [0,1] )

meas_EX5_job = execute(meas_EX5_circs,simulator,seed_simulator=3145,seed_transpiler=3145,shots=8192,noise_model=noise_thermal)
meas_EX5_result = meas_EX5_job.result()

meas_EX5_fitter = CompleteMeasFitter(meas_EX5_result, state_labels)
meas_EX5_filter = meas_EX5_fitter.filter

mit_result = meas_EX5_filter.apply(qpt_result_4)
mit_tomo = ProcessTomographyFitter(mit_result, qpt_circs)

mit_lst = mit_tomo.fit(method='lstsq')
fidelity = qi.average_gate_fidelity(mit_lst, target_unitary)
fidelity
```

```
0.9599277489994111
```

In [71]:
```
from qc_grader import grade_lab5_ex5

# Note that the grading function is expecting a floating point number
grade_lab5_ex5(fidelity)
```

```
Submitting your answer for lab5/ex5. Please wait...
Congratulations 🎉! Your answer is correct and has been submitted.
```

Exploratory Question 6.

- Test how the gate fidelity depends on the CX duration by running noise models with varying cx durations (but leaving everything else fixed).

(Note: this would ideally be done using the scaling technique discussed in the previous lecture, but due to backend availability limitations we are instead demonstrating the effect by adjusting duration of the CX itself. This is not exactly how this is implemented on the hardware itself as the gates are not full CX gates.)

In [81]:
```
print ("4:09 صباحا | 28.7.2021 | تم بحمد لله تعالى")
```

تم بحمد الله تعالى | 28.7.2021 | صباحا 4:09

In [ ]: