CSCI 4140 Laboratory Seven

Error Mitigation

Introduction

In the previous laboratory we examined errors in gates and suggested a way that we could reduce the impact of gate errors. In this laboratory we will investigate measurement errors and how we can mitigate these errors. The basic idea is to measure the errors for different qubit configurations and then develop a simple model that attempts to reverse those errors.

Measurement Error

Start by constructing an error model that only has measurement error. With this model all of our gates will be perfect. The import statements that we will need are:

```
from qiskit import QuantumCircuit, QuantumRegister, Aer, execute
from qiskit.providers.aer.noise import NoiseModel
from qiskit.providers.aer.noise.errors import pauli_error, depolarizing_error
from qiskit.ignis.mitigation.measurement import (complete_meas_cal,CompleteMeasFitter)
import numpy as np
import scipy.linalg as la
from qiskit.visualization import plot_histogram
```

The noise model that we are going to use is:

```
def get_noise(p):

    error_meas = pauli_error([('X',p), ('I', 1 - p)])

    noise_model = NoiseModel()
    noise_model.add_all_qubit_quantum_error(error_meas, "measure") # measurement error is applied to measurements

    return noise_model
```

We will start our exploration with 2 qubits. For each possible combination of qubits we will perform a large number of measurements, in this case 10,000, using the noise model. There are 4 possible combinations of qubit values, so this is relatively easy. The code for doing this is:

```
noise_model = get_noise(0.01)

for state in ['00','01','10','11']:
    qc = QuantumCircuit(2,2)
    if state[0]=='1':
        qc.x(1)
    if state[1]=='1':
        qc.x(0)
    qc.measure(qc.qregs[0],qc.cregs[0])
    print(state+' becomes',
          execute(qc, Aer.get_backend('qasm_simulator'),noise_model=noise_model,shots=10000).result().get_counts())
```

When we run this code, we get the following results:

```
00 becomes {'00': 9777, '10': 111, '11': 3, '01': 109}
01 becomes {'00': 112, '10': 1, '11': 109, '01': 9778}
10 becomes {'00': 100, '11': 112, '10': 9788}
11 becomes {'00': 2, '10': 113, '11': 9801, '01': 84}
```

Note that we get the correct result over 97% of the time. This is what we would expect with the fairly low error probability. We also notice that 2 qubit errors occur very rarely, if not at all. The vast majority of the errors are 1 qubit errors.

Try one more example before we start building our model. If we have something like the Bell state, we have two correct answers. Let's give that a try:

```
qc = QuantumCircuit(2,2)
qc.h(0)
qc.cx(0,1)
qc.measure(qc.qregs[0],qc.cregs[0])
print(execute(qc, Aer.get_backend('qasm_simulator'),noise_model=noise_model,shots=10000).result().get_counts())
```

```
{'00': 4900, '10': 98, '11': 4898, '01': 104}
```

Again, the two correct answers are the ones with the highest probability, but there is still a chance that we could get an incorrect answer.

All term we've taken the result with the highest number of counts as the correct one. For simple programs that has worked well, but we would like something more formal for more complicated programs.

Mitigation Algorithm

Our first experiment essentially took ideal results and produced a vector of measured results. That is, it mapped from the correct result to the measured result. For each of the 4 possible qubit combinations we got a vector of 4 measurements. We have 16 values, which suggests that we could put our results into a 4x4 matrix.

Rearranging our first vector of results to be in |00>, |01>, |10> and |11> order we have the following after we normalize the counts:

$$\begin{pmatrix} 0.9777 \\ 0.0109 \\ 0.0111 \\ 0.0003 \end{pmatrix}$$

We can do the same thing with the other three sets of results to form a complete matrix. This gives us the following Python code:

```
M = [[0.9777, 0.0112, 0.0100, 0.0002],
    [0.0112, 0.9778, 0.0001, 0.0109],
    [0.0100, 0.0, 0.9788, 0.0112],
    [0.0002, 0.0084, 0.0113, 0.9801]]

Cideal = [[5000],
          [0],
          [0],
          [5000]]

Cnoisy = np.dot(M, Cideal)
print('C_noisy =\n',Cnoisy)
```

If we take the correct result and multiply it by this matrix, we will get something close to the last experiment in the previous section. The result that we get from this code is:

```
C_noisy =
 [[4889.5]
 [ 110.5]
 [ 106. ]
 [4901.5]]
```

This is pretty close to the result that we got before. But what we really want is the opposite, we want to go from measured results to the correct results. Now that we have the matrix M, we only need to invert this matrix to get what we want. The Python code for doing this is:

```
Minv = la.inv(M)

print(Minv)
```

```
[[ 1.02304976e+00 -1.17186559e-02 -1.04513578e-02  4.09943919e-05]
 [-1.17173704e-02  1.02293597e+00  1.46531715e-04 -1.13756757e-02]
 [-1.04522210e-02  2.20044977e-04  1.02190076e+00 -1.16779885e-02]
 [ 1.21682031e-05 -8.76727366e-03 -1.17810623e-02  1.02053618e+00]]
```

When we apply this matrix, we get the following result:

```
Cmeasured = [4900, 104, 98, 4898]

Cmitigated = np.dot(Minv, Cmeasured)
print('C_mitigated =\n',Cmitigated)
```

```
C_mitigated =
 [5010.90166117   -6.73347357   -8.24551171 4996.57948692]
```

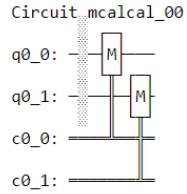
As you can see this is much closer than the measured result. There are two negative entries, which shouldn't be possible, but they are so small they can be ignored.

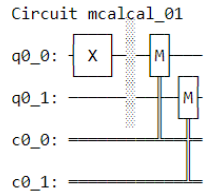
Error Mitigation in Qiskit

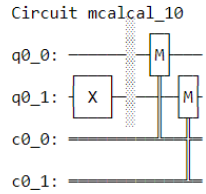
This approach is so useful that Qiskit has a set of procedures that automates it. This way we don't need to construct the matrices ourselves and Qiskit provides a better way of inverting the matrices. To start with we use the complete_meas_cal procedure to construct the circuits that we need to collect the data required for constructing the matrix. This is done in the following way:

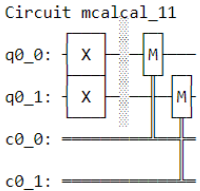
```
qr = QuantumRegister(2)
meas_calibs, state_labels = complete_meas_cal(qr=qr, circlabel='mcal')

for circuit in meas_calibs:
    print('Circuit',circuit.name)
    print(circuit)
    print()
```

In this case we are doing the calibration for two qubit circuits. The output from this code is:









This is basically what we would expect to see. Now we need to run these circuits and fit the model. The CompleteMeasFitter procedure does the fitting for us.

```
noise_model = get_noise(0.1)

backend = Aer.get_backend('qasm_simulator')
job = execute(meas_calibs, backend=backend, shots=1000, noise_model=noise_model)
cal_results = job.result()

meas_fitter = CompleteMeasFitter(cal_results, state_labels, circlabel='mcal')
print(meas_fitter.cal_matrix)
```

```
[[0.828 0.097 0.082 0.012]
 [0.09  0.808 0.014 0.089]
 [0.073 0.021 0.817 0.088]
 [0.009 0.074 0.087 0.811]]
```

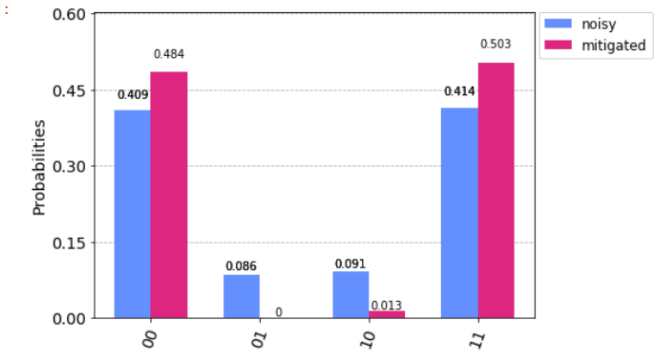
Now let's go back to the circuit that we had at the end of the first section and run it again to get the following:

```
qc = QuantumCircuit(2,2)
qc.h(0)
qc.cx(0,1)
qc.measure(qc.qregs[0],qc.cregs[0])

results = execute(qc, backend=backend, shots=10000, noise_model=noise_model).result()

noisy_counts = results.get_counts()
print(noisy_counts)
```

```
{'00': 4086, '10': 910, '11': 4143, '01': 861}
```

Now that we have our noisy results, we can use the error mitigation model to produce better results:

```
# Get the filter object
meas_filter = meas_fitter.filter

# Results with mitigation
mitigated_results = meas_filter.apply(results)
mitigated_counts = mitigated_results.get_counts(0)

plot_histogram([noisy_counts, mitigated_counts], legend=['noisy', 'mitigated'])
```



The mitigated results are much better. The incorrect results have been virtually eliminated.

Laboratory Activity

For your laboratory activity repeat the model fitting with 3 qubits. This is a small change to the code. Your test circuit will now need to have three qubits. Start with the previous circuit that produced a Bell state and just add a X gate to the third qubit. Cut and paste the resulting histogram and submit it as your laboratory report. The report must be a PDF or PNG file.