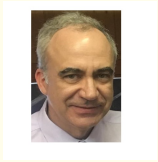
Telling quantum DoQs and quantum Qats apart

Team Quant'ronauts


Alain Chancé


Michaël Rollin

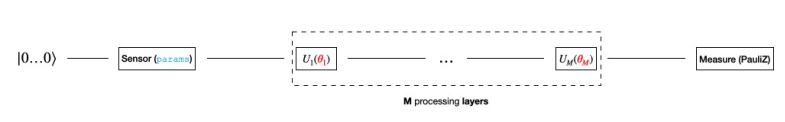

Sahar Ben Rached


Tamás Varga

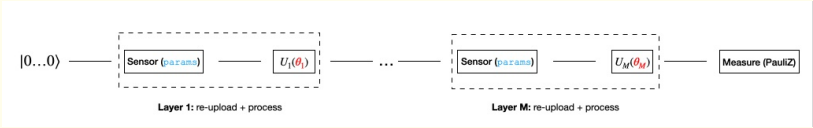

Laurent Querella

Idea: we classify regions of the Hilbert space, of quantum states of n qubits. There are 2 categories, "Qat" and "DoQ". As an example, for n=1, one hemisphere of the Bloch sphere could be labelled "Qat", the other hemisphere "DoQ". The state vectors to classify are generated as the output of a sensor, which is then fed into a classifier circuit of M layers. Note that we are NOT classifying the classical params vector of the sensor, as we could use any other sensor with different parameterization as long as it's capable of producing Qat and DoQ states. Also, we take the sensor as is, we don't try to "optimize" it.

Catch: during operation, the sensor can only produce its output once. Thus, when we calculate the accuracy on the test set, we are not allowed to make use of expected values resulting from many shots. There is only 1 shot (in the training phase, we can optimize using expected values, as training is done in our laboratory where we can recreate the sensor outputs of the training set at will). We'd like to experiment how much the accuracy drops due to this 1-shot limitation, whether it's different using simulator vs real quantum hardware, and what kind of cost function would reduce this impact.



Extra: if multiple shots are allowed, how much would a data re-uploading scheme improve the accuracy? E.g. imagine there are M identical sensors located very close to each other. When a certain physical event happens, it sets all the parameters of the M sensors at once, identically for each sensor. Then, the parameters don't change until the next event. Furthermore, there may be exponentially many parameters of the sensor, inaccessible to us. So again, we are classifying quantum states.

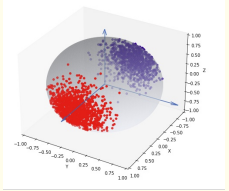


Step 1 - Set up the sensor:
We create a 1-qubit sensor that produces Qat and DoQ states. The eastern hemisphere of the Bloch sphere corresponds to the “Qat” region, and the western hemisphere to “DoQ”.

0: ──RX(sensor_theta_x)──RY(sensor_theta_y)──...

Step 2 - Classifier:
We feed the sensor's output into a 1-qubit classifier circuit that consists of 3 parameterized gates, RX, RY, and RZ, and then apply a PauliZ measurement

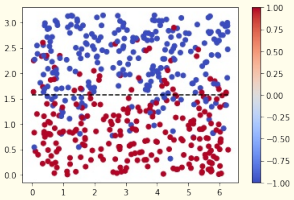
...──RX(classifier_theta_x)──RY(classifier_theta_y)──RZ(classifier_theta_z)──┤ Z



Telling quantum DoQs and quantum Qats apart

Team Quant'ronauts

Step 3 - Training:
The θ parameters of the classifier are initialized randomly, then we use the Gradient Descent optimizer to optimize the sensor's output over 50 steps. During training, the aim is for the sensor outputs labelled "Qat", the measurement result (expected value of 30 shots) should be close to 1, while for sensor outputs labelled "DoQ", the measurement would be close to -1.



For the training, we used a random training set of 500 labelled sensor outputs.

4- Results -

For testing, we used a random test set of unseen 1000 labelled sensor outputs.

We experiment with the data output after one or more shots. If we run our sensor once, meaning with one shot, the output is fed into a classifier, and the PauliZ measurement result is either 1 or -1. In this extreme case, it is not possible to reproduce the sensor output and feed it to the classifier multiple times - we cannot clone the output sensor, which is a quantum state.

Increasing the number of shots, we clearly witness an improvement in the accuracy of the classification task: 1-shot accuracy: 80.4%, 30-shot accuracy: 94.8%, 1000-shot accuracy: 98.1%

However, following this process, with a 1-qubit classifier that applies a unitary operation with parameters independent of the sensor's output, the accuracy improvement hits a limit. We assume this limitation is due to applying a single rotation gate, which doesn't actually distribute the state vectors closer to the East and West poles of the Bloch sphere.

5- Improving accuracy:

5.1 - Data re-uploading:

We tried two schemes to improve the accuracy.

The first one, is applying data re-uploading on the same 1-qubit classifier, and here is an example circuit with 2 layers:
0: ──RX(0.307)──RY(1.35)──RX(1.04)──RY(-2.64)──RZ(0.442)──RX(0.307)──RY(1.35)──RX(0.429)──RY(-2.66)──RZ(2.14)──┤ Z
Notice the ──RX(0.307)──RY(1.35) is repeated twice, followed by the RX, RY, RZ gates of the classifier.

2 layers	3 layers	4 layers	5 layers	10 layers
49.5%	69.3%	70.1%	72.4%	76.3%

Telling quantum DoQs and quantum Qats apart

Team Quant'ronauts

5.2- Multi-qubit classifier results:

The second idea we investigated is feeding the sensor's output into a classifier with more than 1 qubit. Here is an example circuit:

```
0: ──RX(0.307)──RY(1.35)────RX(0.584)──RY(-3.08)──RZ(-0.152)──  C──  ┤  Z  Z
1: ──RX(1.31)────RY(-2.87)──RZ(2.38)──────────────────────────────  X──  ┤  Z  Z
```

The sensor consists of the RX and RY gates on the top wire. Notice the CNOT gate applied in the "ring" pattern to harness the entanglement.

We apply a PauliZ x PauliZ measurement, which returns either 1 and -1. In this case, as we have more than 1 qubit, we can have multiple classifier layers. RX, RY, RZ, and the CNOT ring make up one layer. (In the picture above, we show only 1 classifier layer).

Adding multi-qubit classifiers resulted also in improving the accuracy, yet with a slower rate of improvement compared to the Data re-uploading method.

In this table, we represent the improvement of accuracy with the increased number of circuit layers using the “multi-qubit classifier” method:

	1 layer	2 layers	3 layers	4 layers	5 layers	10 layers
n=2	47.9%	82.1%	81.1%	81.9%	82.1%	82.4%
n=3	51.1%	79.1%	80.9%	80.3%	80.7%	80.8%

Perspective

To advance further on this project and seek pertinent methods to improve the testing accuracy, we mainly consider adding more layers as well as qubits, trying different measurement mechanisms, parameterized data re-uploading and different cost functions.

All the results mentioned in our project presentation is derived from Pennylane simulator “default.qubit”. In the future, we will run our trained circuits on the different simulators and quantum hardware available on AWS. We will investigate how the accuracy varies from 1 to 30 shots, and compare the fidelity of the available backends regarding our project.

Source code: https://github.com/mickahell/qhack21