Quantum C2C

Deep learning for compression of classical data in quantum computing

TEAM PRESENTATION

Chih-Han (Robin) Huang
Data Scientist
Deep Learning, Quantum Computing

Sumitra Pundlik
Professor
Quantum Computing, Data Science

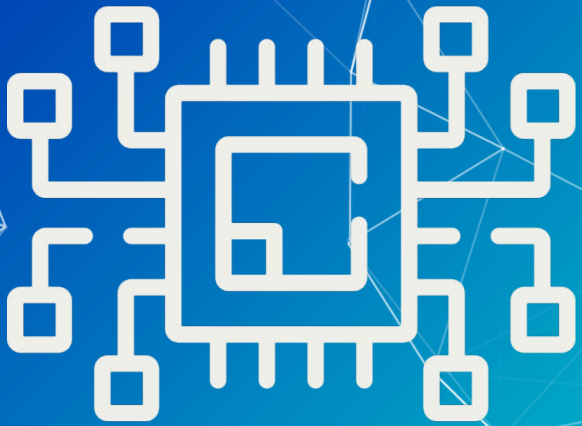
Khushwant Kumar
Engineer
Deep learning, Quantum Computing

Vardaan Sahgal
Physics Master Student
Physics,, Quantum Computing

Tathagata Majumdar
Computer Science Bachelor Student
Computer Science, Quantum Computing

Data compression of classical data is important

In a noisy intermediate-scale quantum era



Classical data compression

In previous literatures about quantum computing (QC) or quantum machine learning (QML)

- Simple or non-parametric
  - Downsampling image resolutions
  - Principal components analysis

Classical data

↓ Dimension reduction

Classical data with low dimension

↓ Quantum encoding

Quantum data

↓ Quantum computing/Quantum machine learning

Result

Suitable for any quantum circuit?

Information loss?

Quantum C2C

Designing a data compression model for your own quantum circuit!

Main idea

- We hypothesized that deep learning (DL) would learn the optimized parameters to compress classical data for QC/QML.
- Information loss would be minimized during the data compression with deep learning.
- We argued that each quantum circuit would be suitable for different data compression models (both hyperparemeters and parameters).
- One could train and design different DL data compression model structures for several quantum circuits.

Quantum C2C methodology

We first proposed to use Multitask Learning on QC/QML to simultaneously minimize the loss of autoencoder and loss of performance of QC/QML.

Steps

1

Autoencoder pre-training

2

Main training

Autoencoder pre-training

Classical training data

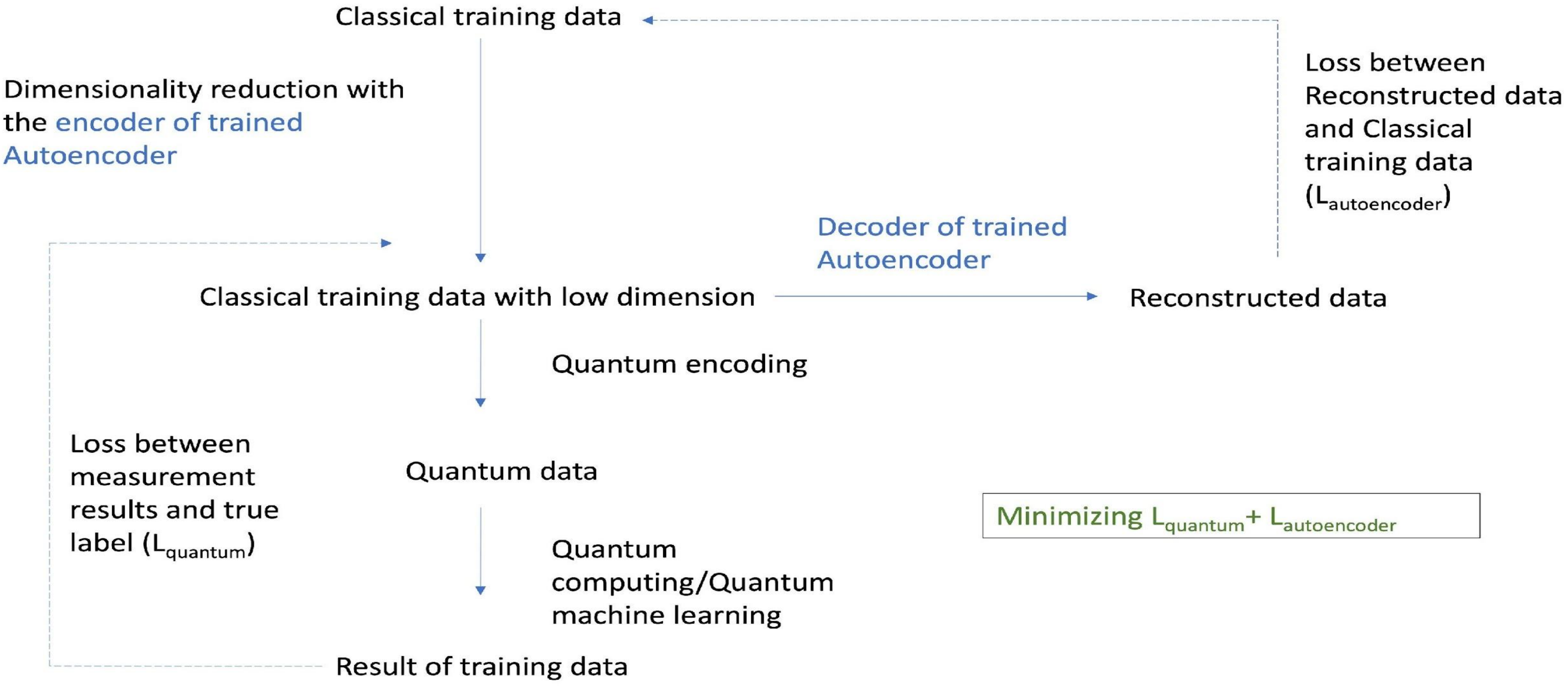
$\downarrow$ Encoder

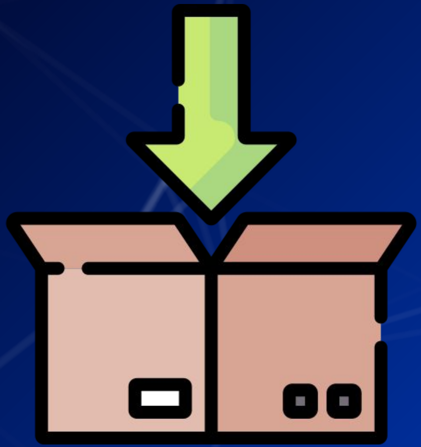
Classical training data with low dimension

$\downarrow$ Decoder

Reconstructed data

Main training





A tool of Quantum C2C is also developed

Install

```
cd lib
python setup.py install
```

Usage

```
from quantum_c2c.quantum_c2c import quantum_c2c,AutoEncoder,Hybrid
```

```
ae=AutoEncoder(input_shape=X_train.data.shape,encoded_len=encoded_len)
qc=Hybrid(qiskit.Aer.get_backend('aer_simulator'),
            100, np.pi / 2,encoded_len=encoded_len)
```

```
trained_encoder_model,y_predicted=quantum_c2c(X_train_autoencoder,
                                              X_train,
                                              X_test,
                                              autoencoder_model=ae,
                                              quantum_curcuit=qc,
                                              saving_folder='model',
                                              epochs=epochs,
                                              encoded_len=encoded_len)
```

Training process

```
Pre-training autoencoder
[1/10] Loss: 0.03536197170615196
[2/10] Loss: 0.03202573955059051 5
[3/10] Loss: 0.031863734126091
[4/10] Loss: 0.0250651277601718 9
[5/10] Loss: 0.024263398721814156
[6/10] Loss: 0.021736128255724907
[7/10] Loss: 0.025709688663482666
[8/10] Loss: 0.024877650663256645
[9/10] Loss: 0.023503243923187256
[10/10] Loss: 0.024549739435315132
Training main model
Training [10%]  Loss: 0.7039
Training [20%]  Loss: 0.6432
Training [30%]  Loss: 0.6276
Training [40%]  Loss: 0.4430
Training [50%]  Loss: 0.2569
Training [60%]  Loss: 0.1707
Training [70%]  Loss: 0.1223
Training [80%]  Loss: 0.1035
Training [90%]  Loss: 0.0857
Training [100%] Loss: 0.0741
Performance on test data:
        Loss: 0.0718
        Accuracy: 100.0%
```

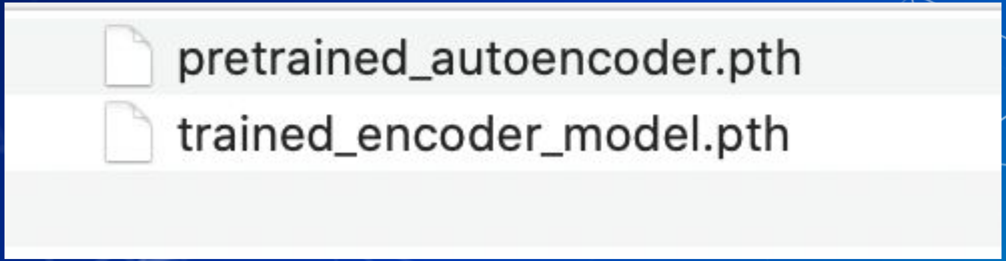
Input

- **a. X_train_autoencoder:**
  - Classical Data for the autoencoder pre-training.
- **b. X_train:**
  - Classical Data for main model training.
- **c. X_test:**
  - Classical Data for main model testing.
- **d. Autoencoder_model:**
  - PyTorch model of autoencoder.
- **e. Quantum_curcuit:**
  - Qiskit's "Quantum-Classical Class" with PyTorch
- **f. Saving_folder:**
  - Folder name for saving the models "pretrained_autoencoder.pth" and "trained_encoder_model.pth".
- **g. Epochs:**
  - Epochs for the autoencoder pre-training and main model training.
- **h. Encoded_len:**
  - length of encoded data

Output

- **a.model.encoder:**
  - Return of the function. The last trained encoder model
- **b. y_predicted:**
  - Return of the function. Predicted target values
- **c. pretrained_autoencoder.pth:**
  - The file saved in saving_folder. Pretrained autoencoder.
- **d. trained_encoder_model.pth:**
  - The file saved in saving_folder. Trained encoder model.

The files saved in saving_folder automatically



Conclusion

- We proposed a unique methodology, Quantum C2C, to compress classical data with deep learning for QC
- Quantum C2C is Multitask Learning to simultaneously minimize the loss of autoencoder and loss of performance of QC/QML
- We also developed a tool that one could simply perform end-to-end training with inputting classical data.

Future works

- Performing a systematic comparison study to testing more autoencoder frameworks (e.g. variational autoencoder and masked autoencoder) for different quantum circuits
    - Providing a reference to design a model for future researchers

Special thanks

Dr. Alberto Maldonado Romo

- Mentor

Thank you for listening