Honours Project: Update

Alapan Chaudhuri

December 9, 2021

Contents

- Update on work on Quantum Computation
- OpenQASM3: IR for Quantum Programs
- Creating Arbitrary Superpositions
- Quantum Machine Learning: Training a Variational Circuit
- Motivation for QPIR
- Private Information Retrieval
- Coding Theory
- Quantum Private Information Retrieval
- Quantum Query Complexity

Update on work in Quantum Computation

Worked on Cirq:

- Currently working on implementing OpenQASM 3
- Added $r(\theta, \phi)$ gate (with Zeeshan Ahmed)

OpenQASM3

Quantum assembly languages are machine-independent languages that traditionally describe quantum computation in the circuit model. Open quantum assembly language (OpenQASM 2) was proposed as an imperative programming language for quantum circuits based on earlier QASM dialects.

OpenQASM3

- OpenQASM is a low-level quantum programming language. It is a general purpose, imperative language for describing quantum operations. It is designed for both quantum circuits and quantum state machines.
- It is designed to be extensible. It can be extended to support new quantum devices and quantum algorithms.
- OpenQASM3 is the current latest specification available.

OpenQASM3 vs OpenQASM2

- In principle, any quantum computation could be described using OpenQASM 2, but there is a need to describe a broader set of quantum circuits beyond the language of qubits and gates.
- One problem with respect to OpenQASM2 is that sometimes a gate-level description of a circuit is too high-level. We may need to specify the signals on the control wires attached to the quantum processor.
- OpenQASM2 is also timing unaware. It contemplates an abstract machine model of unitary transformations and measurements on qubits.
- It is with these use cases in mind that OpenQASM3 was proposed.

OpenQASM3 vs OpenQASM2

- Using OpenQASM3, we can now work on both circuit level as well as have full access to sample level in the control hardware.

- It interfaces the classical and quantum regimes and proposes a multi-level programming model that will allow the dynamic exploration of quantum systems.

Creating Arbitrary Superpositions

Creating any arbitrary superposition is an important problem to solve and implement in every quantum programming language.

The problem statement is to design a general circuit that accepts vectors with random positive integral values of size n with m bits in length for each element and finds the superposition of indices such that the elements of the vector at those indices have different values for every two adjacent bits in their binary representation.

Creating Arbitrary Superpositions

- Given the array *arr*, we now have to make a list of indices $i, j, ...$ such that all elements of the array at those indices have a binary representation where any of its two adjacent bits are different.

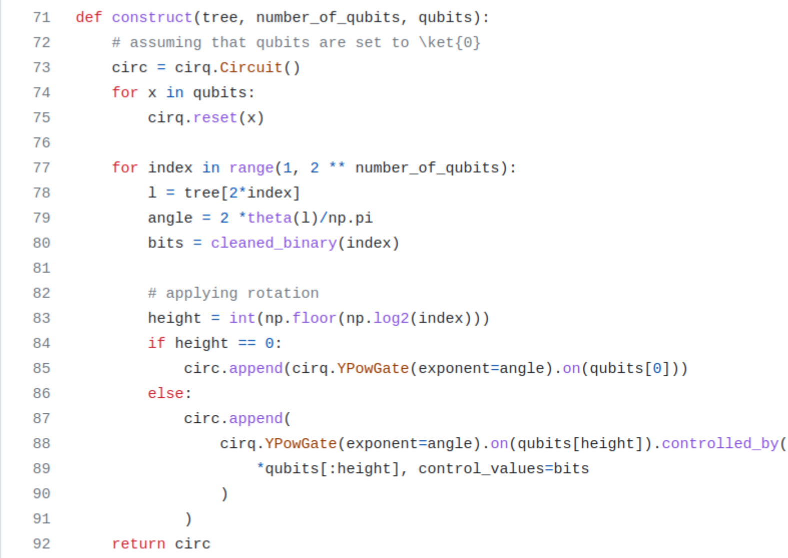
- Now, given that we have the required indices, we need to create a array (*params*) first which represents the required superposed state we want.

- Given that we have the array representation of the required state, all that we need is an efficient algorithm/oracle to convert a quantum state —000...0¿ to one that resembles our array.

Creating Arbitrary Superpositions

- In my case, I try to model the state as a probability distribution across a binary tree (similar to the bifurcation graph for QRAM) and apply controlled rotations based on the probabilities at any node.
- Final circuit constructed based on the algorithm mentioned yields the quantum state (the required superposition) we wanted.

Creating Arbitrary Superpositions



```python
def construct(tree, number_of_qubits, qubits):
    # assuming that qubits are set to \ket{0}
    circ = cirq.Circuit()
    for x in qubits:
        cirq.reset(x)

    for index in range(1, 2 ** number_of_qubits):
        l = tree[2*index]
        angle = 2 *theta(l)/np.pi
        bits = cleaned_binary(index)

        # applying rotation
        height = int(np.floor(np.log2(index)))
        if height == 0:
            circ.append(cirq.YPowGate(exponent=angle).on(qubits[0]))
        else:
            circ.append(
                cirq.YPowGate(exponent=angle).on(qubits[height]).controlled_by(
                    *qubits[:height], control_values=bits
                )
            )
    return circ
```

Figure: Code for tree construction

QML: Training a Variational Circuit

Problem: To train a quantum variational circuit that will transform according to the following map:

$$f(|0000\rangle) = |0011\rangle$$

$$f(|0001\rangle) = |0101\rangle$$

$$f(|0010\rangle) = |1010\rangle$$

$$f(|0011\rangle) = |1100\rangle$$

The input states were chosen arbitrarily, and the output states were fixed. The transformation for other states is irrelevant.

Framework used: Pennylane.

Training a Variational Circuit: QVCs

QVCs are the prime example of the intersection between quantum computing and classical machine learning.

A quantum circuit can be taken as a model with parameters of different kinds that can be trained for a transformation.

There are two kinds of parameters:

- Adaptable $\hat{\theta} \rightarrow (\theta_1, \cdots \theta_n)$
- Non-adaptable $\hat{x} \rightarrow (x_1, \cdots x_n)$

QVCs

The whole circuit can be modeled as one single unitary transformation that is parameterized by these parameters, and the transformation can be written as: $U(\hat{\theta}, \hat{x})$.

To use machine learning, we need to convert the qubit states into classical information. For this, we shall use observables. To maximize the distance between the possible outputs, we use 'PauliZ', which has $\{-1, 1\}$ as the observable values.

Note: We also need to convert the target values into the above format to draw reasonable accuracy and cost.

Training a Variational Circuit

The circuits' gates can be thought of as hyperparameters that do not change while they are being trained. To capture the function's entire essence, we need to have both single qubit and multi-qubit gates.

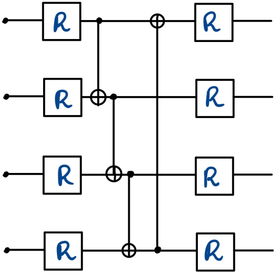


Figure: Circuit

Training a Variational Circuit

- The *CNOT* gates were required to count for the relation between different qubits.
- A saturation at 1.4 cost was observed with one layer of R gates and *CNOT* gates.
- Adding another layer of ‘R‘ gates dropped the cost to 0.6.
- Each ‘R‘ gate takes in 3 parameters:

$$R(\theta, \phi, \omega)$$

Training a Variational Circuit

- **Embedding**: Before the input can be fed into the circuit, it must be converted into a state. We will use the basis embedding:

  $$0000 \rightarrow |0000\rangle$$

  The kind of embedding determines the non-adaptable parameters of the circuit.

- **Cost Function**: Since the distance between the predicted state and the required state directly can't be made, we instead apply our loss function over the expected value of the observable `PauliZ`.

Training a Variational Circuit

Training: The QVC algorithm is now used to train the circuit, i.e. find the correct parameters for each of the gates. The parameters can be modeled into a weight matrix W.

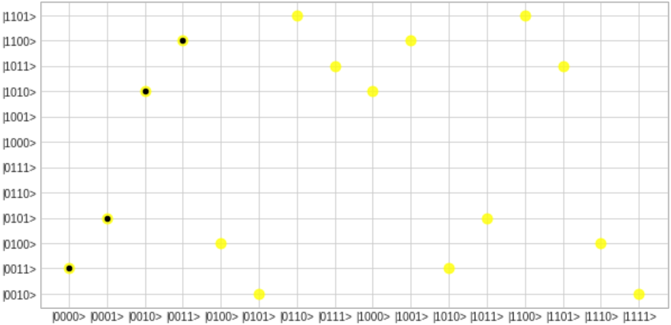
Initialization: The initial values of the matrix are set to small random values.

Optimization: The weight matrix will now have to be optimized to reduce the cost, for which we will use gradient descent paired with Nesterov Momentum optimization.

The learning rate was initially set to 0.5 which did not give low cost due to a great amount of fluctuation in the parameters. The rate was then gradually reduced and decent results were obtained at 0.01 learning rate.

Training a Variational Circuit

Final Cost: After 200 iterations the final cost was 0.6856744 with full accuracy.

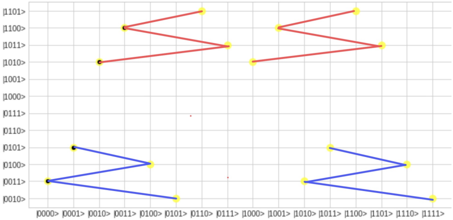


Training a Variational Circuit

- After the network was trained, it was tested for all the 16 possible classical inputs to the circuit. For visualization, it is easier to scatter plot both the prediction and the expected value. The plot will also help us draw insights about the states' transformations that aren't considered on the map.

- The diagram (next slide) shows a recurring pattern since the training samples had only the first 4 points. The rest of the values are adjusted accordingly, and the pattern is repeated.

Training a Variational Circuit



The graph doesn't change when run multiple times, that means the function generated is not random for the other points but is deterministic based on the training samples.

Motivations for QPIR

- Protecting our online privacy is a big issue nowadays.
- PIR from coded storage has gained a lot of interest recently.
- Several capacity results derived, e.g., for replicated and MDS coded storage, colluding servers, and symmetric PIR.
- Promise of Quantum: We believe that better rates of transmission are possible with quantum communication.

Private Information Retrieval

- **What is PIR?** PIR is a retrieval protocol to obtain or retrieve messages without revealing which message was requested.
- There are two fields of study within PIR (ITPIR), namely, information theoretic PIR and computational PIR (CPIR).
- ITPIR is faster (uses cheap cryptographic operations) and is information-theoretically secure but assumes non-collusion.
- **ITPIR**: The identity of the querier and the query have zero mutual information, i.e., obtaining identity given the query is impossible.

Private Information Retrieval

Theorem

If the data is stored on only one server (or equivalently, if all servers are colluding), then the only thing we can do to achieve perfect privacy is to download the entire database.

- On the other hand, if the database is replicated or coded on multiple servers that are not all communicating, then we can do better.
- Upload cost ignored: the number of bits in the files ($\sim$ download cost) is assumed to be much bigger than the number of files on the servers ($\sim$ upload cost).
- Symmetric PIR (SPIR): the user is only able to decode the file that he has requested, and learns nothing about the other files, i.e., privacy is guaranteed also for the server.

Definitions within PIR

- Correctness: there exists a functional D such that $D(A^K, Q^K, K) = x^K, \forall K$.
- t-collusion: any t servers may collude, i.e., exchange their received queries.
- t-PIR scheme: a scheme that protects against t-collusion, i.e., any set of at most t colluding nodes learns no information about the index K of the desired file.

Definitions within PIR

- PIR rate: $R = \frac{\text{number of bits in a file}}{\text{number of received bits}}$ and in case of QPIR we have number of received qubits as the denominator.
- Capacity: supremum of PIR rates of all possible PIR schemes, for a fixed parameter setting.

Coding Theory: a recap

- A linear code C is a subspace of $\mathbb{F}_q^n$.
- We call n the length and $k = dim(C)$ the dimension of C.
- The minimum distance of C is $d = min\{wt_H(c) : c \in C\{0\}\}$, where $wt_H(c)$ is the Hamming weight of c, i.e., the number of non zero coordinates in c.

Coding Theory: a recap

- When $d = n - k + 1$, C is called a maximum distance separable (MDS) code.

- The dual code C' is the dual vector space of C.

- The dual codes of MDS codes are also MDS.

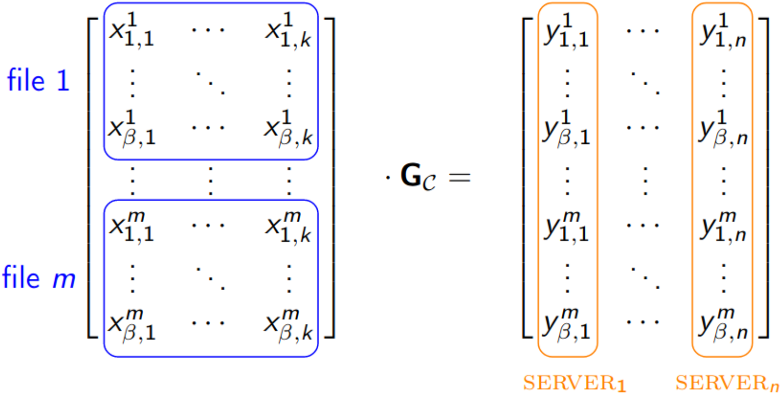
Coding Theory: a recap

- We call $G_C \in \mathbb{F}_q^{k \times n}$ is a generator matrix for C if rank $G_C = k$ and its k rows span C.

- A message $m \in \mathbb{F}_q^n$ is encoded into a codeword c by $c = mG_C$.

- m files $x_1, \cdots, x_m \in F_q^{\beta \times k}$ are encoded and stored on n servers by a storage code C.

Coding Theory: a recap



$$
\begin{array}{r|ccc|}
 & x^1_{1,1} & \cdots & x^1_{1,k} \\
\text{file } 1 & \vdots & \ddots & \vdots \\
 & x^1_{\beta,1} & \cdots & x^1_{\beta,k} \\
 & \vdots & \vdots & \vdots \\
 & x^m_{1,1} & \cdots & x^m_{1,k} \\
\text{file } m & \vdots & \ddots & \vdots \\
 & x^m_{\beta,1} & \cdots & x^m_{\beta,k}
\end{array}
\cdot \mathbf{G}_{\mathcal{C}} =
\begin{array}{|ccc|}
y^1_{1,1} & \cdots & y^1_{1,n} \\
\vdots & \ddots & \vdots \\
y^1_{\beta,1} & \cdots & y^1_{\beta,n} \\
\vdots & \vdots & \vdots \\
y^m_{1,1} & \cdots & y^m_{1,n} \\
\vdots & \ddots & \vdots \\
y^m_{\beta,1} & \cdots & y^m_{\beta,n} \\
\hline
\text{SERVER}_1 & & \text{SERVER}_n
\end{array}
$$

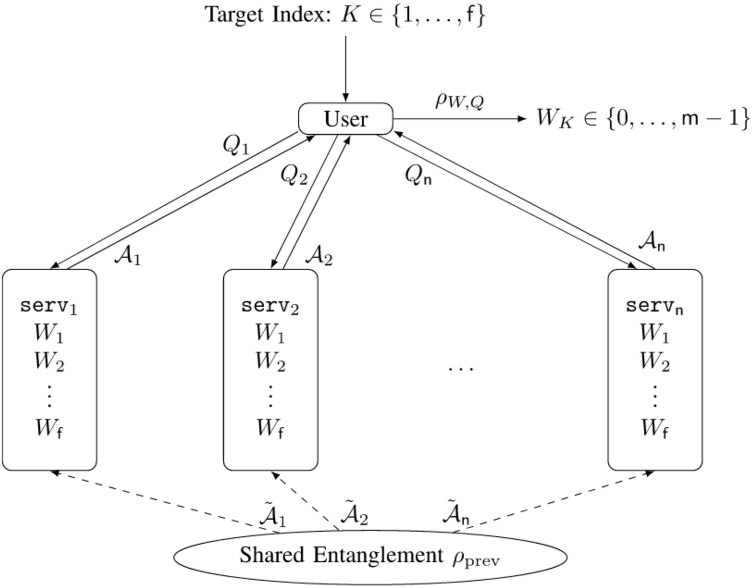
Figure: Generator

Quantum Private Information Retrieval

In the QPIR problem with multiple servers, the objective is for a user to retrieve a classical file by downloading (entangled) quantum systems from multiple replicated servers, while maintaining the privacy constraint that identity of the downloaded file remains unknown to each server.

In the quantum counterpart for the classical PIR problem, we shall be dealing with non-colluded replicated servers.

Quantum Private Information Retrieval


Target Index: $K \in \{1, \ldots, \mathsf{f}\}$
User
$\rho_{W,Q}$
$W_K \in \{0, \ldots, \mathsf{m}-1\}$
Q_1
Q_2
Q_n
$\mathcal{A}_1$
$\mathcal{A}_2$
$\mathcal{A}_\mathsf{n}$
$\mathtt{serv}_1$ W_1 W_2 $\vdots$ W_f
$\mathtt{serv}_2$ W_1 W_2 $\vdots$ W_f
$\ldots$
$\mathtt{serv}_\mathsf{n}$ W_1 W_2 $\vdots$ W_f
$\tilde{\mathcal{A}}_1$
$\tilde{\mathcal{A}}_2$
$\tilde{\mathcal{A}}_\mathsf{n}$
Shared Entanglement ρ_{prev}

Quantum Private Information Retrieval

QPIR protocol, for n-servers and f-files with m-bits each, is given by the 4-tuple $\Phi^(m)_{QPIR} = (\rho_{prev}, Enc_{user}, Enc_{serv}, Dec)$ where Enc_{user} denotes the user encoder, $Enc_{serv} := (Enc_{serv_1}, ..., Enc_{serv_n})$ denotes the collection of the server encoders, and Dec denotes the decoder.

Quantum Query Complexity

- After each query, we would like to allow the quantum algorithm to select the next query in an arbitrary way.
- To do so, we give the algorithm a work space of arbitrary size.
- That is, let W be a set representing the possible basis states of the work space. Then the quantum algorithm will act on the Hilbert space with basis states corresponding to $W \times [n] \times 0, 1$.
- The algorithm will alternate between applying an arbitrary transformation on this state which is independent of x (that is, an arbitrary unitary matrix which does not depend on the input), and applying the unitary U^x implementing the query specified by the query register.
- The unitary U^x will be extended to act as identity on the work space register.

Quantum Query Complexity

Theorem

The action of the quantum algorithm on x will be $U_T U^x U_{T-1} U^x \dots U_1 U^x U_0 |\psi\rangle$ where $|\psi\rangle$ is some fixed initial state.

- T is the number of queries the quantum algorithm makes, and that the quantum algorithm itself is specified by the matrices $U_0, U_1, \dots, U_T$.
- The unitary U_0 can transform the initial state to an arbitary state that does not depend on x.
- Thus, we can assume $|\psi\rangle = |0\rangle_W |1\rangle_I |0\rangle_B$ where we use the subscripts W , I, and B to denote the different registers, namely, work register, index register, and query-answer register.