CPEN 400Q: Gate-Model Quantum Computing

Nicholas Rees

27 February 2023

This is a set of notes based on the CPEN400Q class offered at UBC during the Jan-April 2023 term, taught by Olivia di Matteo. It largely follows the slides and demos that can be found here, as well as the reading materials noted at the end of the slides (with about one subsection roughly corresponding to one lecture). I have also added material I found interesting or insightful from the quizzes/assignments, however with any answers omitted. The prerequisites for understandng these notes are what I had going into the course: a decent background in basic linear algebra and python (and probably some calculus, basic computer science, and classical physics), pretty similar to this. It goes without saying that any error here are my own, and if found, please email me at imnichh@gmail.com.

Contents

§1 Single Qubit

In this section, we will introduce the mathematics/physics underlying the simplest quantum system, the single qubit, and the software development kit we will use throughout this course, PennyLane. The simultaneity of teaching these two things will help us understand the other. On one hand, an understanding of the theory behind quantum computers will help us program functions that can run on quantum circuits, and on the other, seeing actual implementation of what are normally abstract and theoretical ideas can help use understand what is normally and unintutive subject, quantum mechanics.

§1.1 Why Quantum Computers

Before we embark on our journey to learn the theory of quantum computers, lets provide some motivation for this whole quest in the first place. After all, even though I think quantum computers are an interesting to study as a subject for itself, there some good reasons why we would want an operating quantum computer too.

First, note that there is a current problem in our current classical-proccessor world. As humans, we want to do things fast on our computers. And the way we make our computers faster has historically looked like more transistors on a single chip with smaller transistors (see Moore's Law), however this has been stagnating recently, and using more processors in parallel. But this can only take us so far. There are some problems that classical computers (those only using 0s and 1s, unlike their quantum counterpart) will always just be computationally hard. This could look like searching large spaces, or trying to simulate quantum systems, such as molecules.

The thing is, it has been shown that using a model of computation that will work on a quantum computer, we can take advantage of algorithms that are *exponentially* faster. We even have current (but still debated) evidence that quantum computers are already faster at those tasks. But it should be emphasized *at those tasks*. Quantum computers won't replace classical computers at their job. This is because quantum computers, aside from being more expensive, run into a lot more difficulty that we don't have when we don't go this small. These problems are things, such as short coherence times (before the "state collapses", whatever that means), high error rates in the gates, and hard to scale hardware. These difficulties have prevented us from making full use of this "quantum advantage" today. Our current computers that suffer from these problems are called **noisy, intermediate scale quantum** (NISQ) devices. And even if we have not completely acquired a **fault-tolerant quantum computing**, we have found use cases for the devices around today (so that we can at the very least convince others the field is useful enough until we figure out how to stop our qubits from decohering). We will algorithms that are designed for both types, both assuming a feasible quantum computer, and those that can work on a NISQ device.

§1.2 What is a Qubit?

We will first look to defining what a qubit is mathematically, and see what this means for computation. The quantum bit, or the **qubit**, is simply a generalization of the classical bit used in classical information theory and current computers. A classical bit is just something that can be in a "0" state, or a "1" state. In our computers, this is usually implemented as two different voltage levels, however, the underlying mathematics do not really care how this is accomplished.

We can extend the concept of a classical bit to a probablistic bit. This is where we represent the value of a bit as a probability distribution. Since there are two possible

values, that means that a probablistic bit is determined by the probability p_0 of the "0" state (and the "1" state is just $p_1 = 1 - p_0$). Our probability distributions can evolve over time too, as we see in the example.

Example 1.1 (Evolving Probability Bits)

Say we represent the probability that it is raining / not raining as a 2-dimensional vector on $\mathbb{R}$. This would look like Today's weather, $\vec{w}$, is given by

$$\vec{w} = \begin{pmatrix} \text{Prob. rain} \\ \text{Prob. rain} \end{pmatrix} = \begin{pmatrix} p_r \\ p_r \end{pmatrix} = p_r \begin{pmatrix} 1 \\ 1 \end{pmatrix} = (1 - p_r) \begin{pmatrix} 0 \\ 0 \end{pmatrix} = p_r \vec{r} + (1 - p_r)\vec{n}$$

where $\vec{r}, \vec{n}$ are the standard unit vectors.

Given the conditions:

- Rain today implies a 70% chance it rains tomorrow
- No rain today implies a 20% chance it rains tomorrow

we wish to find the probabilty of it raining two days from now, supposing it rains today.

We can represent the time evolution of this probablistic bit with the matrix

$$P = \begin{pmatrix} 0.7 & 0.2 \\ 0.3 & 0.8 \end{pmatrix}$$

(where recall that each column is the following today for each basis). So a time step of a single day looks like

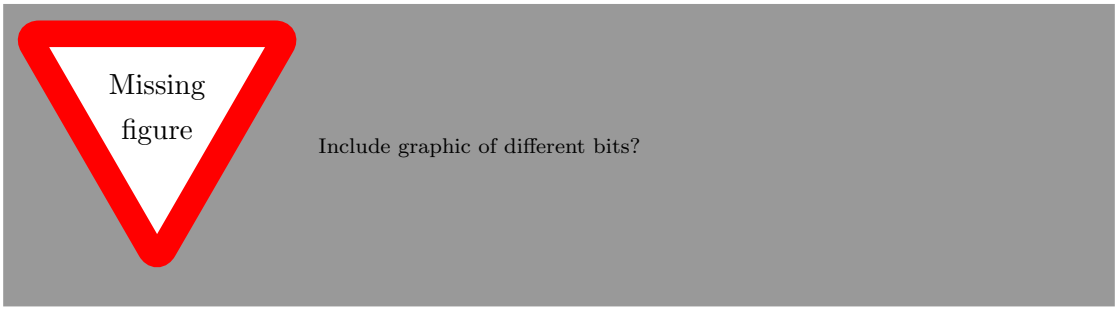
$$\vec{w} \to P\vec{w}$$

Thus, the probability is found from applying this operation twice, and since operation should be linear (since we're modelling this as a vector space), we can directly compute the new probability vector $\vec{a}$:

$$\begin{aligned} \vec{a} &= PP\vec{w} = P(0.7\vec{r} + 0.3\vec{n}) \\ &= 0.7P\vec{r} + 0.3P\vec{n} \\ &= 0.7(0.7\vec{r} + 0.3\vec{n}) + 0.3(0.2\vec{r} + 0.8\vec{n}) \\ &= 0.55\vec{r} + 0.45\vec{n} \end{aligned}$$

So we get 55%.

Now conceptually, if you did not want to learn any of the math behind them or if it helps, you could continue thinking of qubits as these probablistic bits, and our computer gates as some functions that change these probabilities, but you would lose the special features of quantum systems that have given us the so-called "quantum advantage." And besides, it is not a large step to extend probability bits to a qubit. But now, instead of vectors and operators over $\mathbb{R}$, we will be using $\mathbb{C}$.



We provide our mathematical definition of a qubit, and explain what it means after.

Definition 1.2. A **qubit** is a vector in a 2-dimensional Hilbert space.

For those who do not know the mathematical vocab, this can seem more obscuring than clarifying. But essentially (for our purposes), a Hilbert space is just a vector space over the complex field with some inner product defined on it. Since it is 2-dimensional, all of its components can just be represented as a linear combination of two basis vectors.

Similar to how we evolve probability bits with a linear operation (matrix), we can do the same with our qubits, however our matrices are now restricted to unitary matrices. We'll talk more about this after we do some work with representing different qubits.

superposition

manipulating qubits

§1.3 Quantum Circuits

Hello world, I'm inside a subsection

This section is not complete, I'm just flexing some of the features in the style package

Theorem 1.3 (Fancy Theorem)

Hello world, I'm inside a theorem. For all $n, m \in \mathbb{Z}$

$$n \leq m \Rightarrow m|n!$$

We also have code

```
import numpy as np

message1 = "Hello world, I'm inside code"

pi = np.pi

def foo(a,b):
    a = b

message2 = string(np.pi)

print(message1)
print(message2)
```