Lecture 10: Classical simulation
November 8, 2018

1 Classical Simulation

Today, we will study classical simulations of quantum computation. What we care about the most is the technique that allows for efficient simulation of quantum circuits. This is an important subject to study because

- first of all, it allows us to execute a quantum program/circuit when quantum devices are not readily at hand, and thus allowing for testing and verifications before running on a quantum computer.
- secondly, it helps shine more light on the incredibly puzzling and not-so-well-understood computing power boundary between classical computing and quantum computing.

We want to understand how much of quantum processes can be efficiently simulated on a classical computer. Understanding classical simulation can give you insights on what are the ingredients in quantum computers that give computing power.

1.1 What is classical simulation trying to solve?

Given a quantum circuit $U$,

1) with $n$ input qubits as one bit string (e.g. $|0|^n$).
2) with size $N$ number of gates of circuits. (usually poly($n$))

In the quantum case, if we measure the qubits, we would read out some sample bits: $\alpha \in \{0, 1\}^n$, with probability: $P(\alpha) = |\langle \alpha | U | 000...0 \rangle|^2$.

Now, instead of bring it to a quantum computer lab, which can give you the result from measuring qubits at the end, we are interested in doing it classically. Let’s define more carefully what do we mean by doing it classically.

1.2 Two possible definitions for the notion of classical simulation

- Def 1: **Strong simulation**
  - The aim is to calculate the probabilities of the output measurement outcomes efficiently with high accuracy using a classical computer.
  - Within strong simulations,
there are also two kinds, namely to calculate all amplitudes or just one amplitude. More formally,

1. All amplitudes: evaluate the probability distribution by calculating all \( P(\alpha) \), \( \forall \alpha \).
2. One amplitudes: evaluate the probability of one of the outcomes, e.g. \( P(0\ldots0) \).

- **Def 2: Weak simulation**
  - The aim is to be able to *sample once* from the output distribution efficiently using a classical computer.
  - Each time you simulate a circuit weakly, you will obtain an outcome \( \alpha \) that is in accordance with the true probability distribution \( P(\alpha) \) had you done it quantumly.

Some important remarks:

- Strong and weak simulations are fundamentally different notions.
  - i.e. we can find circuits that are trivially simulable weakly, but simulating strongly is \#p-complete! [Nes08]

- **Strong simulation \Rightarrow weak simulation.**
  - if the probabilities are calculated, then you can sample according to the probabilities. But if you can sample once in poly time and there are exponential possibilities, it is not immediately clear how to recover all amplitudes with accuracy.

- **After all, weak simulation is what we are interested in physically, because a quantum device gives you a sample after all.** Strong simulation (especially all amplitudes) may be too strong when comparing quantum v.s. classical computing power.

- **Efficient simulations for a restricted class of circuits is helpful too.** It tells us what additional features in quantum computing enables full universal power.

With that being said, we are going to introduce a number of simulation techniques. In particular, they are all strong simulation techniques. In fact, most simulations that has been developed are strong simulations, and very little is known about weak simulations still.

### 2 Simulation Techniques

The simulation techniques that we are going to cover are:

- Full state vector and matrix multiplication
- Tensor networks
- Undirected graphical model
- Stabilizer formalism
2.1 Full state vector

Naively, we can just compute the state vector at each stage of the circuit, by multiplying it with the unitary matrix of the corresponding gate. Unfortunately, the size of state vector (i.e. dim of vector) grows exponentially with the number of qubits \(2^n\), and the matrices are \((2^n \times 2^n)\) in size. For a depth \(d\) circuit, need to apply matrix to the vector \(d\) times, suffering exponential slowdown in simulation.

2.1.1 Improvement

This technique can be significantly improved if all states are in product states:

\[
|\psi\rangle = |q_1\rangle \otimes |q_2\rangle \otimes ... \otimes |q_n\rangle
\]

where each qubit carries 2 amplitudes, e.g. \(|q_1\rangle = \alpha_1 |0\rangle + \beta_1 |1\rangle\). So there are only \(2n\) number of amplitudes to keep track of, rather than all \(2^n\) of them. As for the unitary matrix, note that being in product state also means that if the gate is local you only need to update the local bit of the amplitudes. (To see why you only need to update locally, note that you are now storing the amplitudes in the above way with \(2n\) complex numbers, instead of the original ket vector.)

Key take away:

\(\rightarrow\) poly time to update

\(\rightarrow\) Entanglement is key resource for quantum computing power. But it is not sufficient! As we will see later, we can construct a circuit that can entangle just fine, yet still are efficiently simulable on a classical computer. \([JL03, \text{Vid03, Joz06, VdNDVB07}]\)

2.2 Tensor Networks \([\text{MS08}]\)

2.2.1 What is a tensor?

It is a multi-dimensional generalization of a matrix. In this context, the dimension is called rank.
You can think of a rank-$k$ tensor, as some mathematical object where an entry in the object is indexed by $k$ indices. Note that in the graphical representation, a tensor is a vertex, and the rank of the tensor is represented as the number of edges connecting to the vertex. In other words, each edge represent an index. We can label the edges with the name of the index.

Examples:
Rank-0 tensor: A number
Rank-1 tensor: A vector (indexed by the position in the vector)
Rank-2 tensor: A matrix (indexed by row number and column number)
Rank-3 tensor: A movie ticket (indexed by screen number, row number, seat number)

2.2.2 From quantum circuit to tensor networks

Now, how do we transform a quantum circuit into a graph of tensors?

- Qubit state: vector $\rightarrow$ 1-d tensor.
- Single-qubit gate: $2 \times 2$ matrix (i.e. qubit input (col) and qubit output (row)) $\rightarrow$ 2-d tensor.
- Two-qubit gate: instead of a $4 \times 4$ matrix, we can index an entry by 4 indices: qubit 1 input, qubit 1 output, qubit 2 input, and qubit 2 output $\rightarrow$ 4-d tensor.
2.2.3 What does connecting two tensors mean?

By definition, it would mean that two tensors share a common edge (i.e. a common index). When two tensors share a common index, we can contract that edge away by summing over all possible values of that index. That seems like something we do in matrix multiplication! Indeed, tensor network contraction can be thought of as the multi-dimensional generalization of matrix multiplication.

In the example below, we have two matrices (i.e., rank-2 tensors) $A$ and $B$. Let’s denote $A^i_j$ as $i$-th row, $j$-th col of matrix $A$. Similarly for $B$. Now, we can contract the edge $j$, by essentially the matrix multiplication rule:

$$\sum_j A^i_j B^k_j = C^k_i$$

Note that the end result is another rank-2 tensor $C$ indexed by $i$ and $k$. 
2.2.4 Complexity of contraction

Contracting one edge take \( \exp(O(d)) \) time where \( d \) is the max rank of tensors involved. To see this, we notice that in our case, each index can take values 0 or 1, so contracting the edge corresponding to that index will yield a summation with 2 terms. Now consider the case where two tensors are connected by \( k \) edges, combining these two tensors would mean summing over \( d \) different indices each of which can take 2 values. So the summation has \( 2^d \) terms in total.

Imagine you could contract some tensors in parallel, it is therefore the maximum rank of tensors you encounter during the process of contraction that determines the complexity. So, can we avoid encountering large rank tensor by contracting the given graph cleverly? The answer is yes. To see this, consider the following two different contraction strategies.

Strategy 1:

Strategy 2:
Contraction order matters! If you want to efficiently simulate a circuit, it is therefore important to specify a clever contraction order that yields low max-rank.

There are a number of techniques that can help keeping the rank low \cite{MS08}. For instance, you can split a tensor into a number of smaller ones (e.g. by singular value decomposition) to allow for more degrees of freedom in the contraction orders. You can also make approximations by dropping some indices when you consider them as unimportant.

### 2.3 Undirected Graphical Model

One of the ways to improve the graph in tensor network representation is the undirected graphical model \cite{BISN17}. In essence, it is a model derived from tensor networks and Feynman path integral.

So what’s the difference of the graphs in tensor networks v.s. in undirected graphical model? Let’s take a look at an example first:

#### 2.3.1 Example

What is the benefit of representing a graph this way? And how do we achieve the simpler graph at the bottom right using Feynman path integral method? The key observation here is that CZ gate is diagonal, and we are doing redundant work!
Let’s expand the product in the above equation for this example circuit. Suppose we want to figure out what is the amplitude on outcome $|11\rangle$:

$$
\langle 11|U|00\rangle = \sum_{i_3,j_3,i_2,j_2} \langle 11|H \otimes H|i_3j_3\rangle \langle i_3j_3|CZ|i_2j_2\rangle \langle i_2j_2|H \otimes H|00\rangle
$$

$$
= \sum_{i_2,j_2} \langle 11|H \otimes H|i_2j_2\rangle \langle i_2j_2|CZ|i_2j_2\rangle \langle i_2j_2|H \otimes H|00\rangle
$$

*Note: $\langle i_3j_3|CZ|i_2j_2\rangle$ is non-zero iff $i_3j_3 = i_2j_2$.

Therefore, a nice simplification to the graph would be:

This technique, in many cases, can drastically reduce the number of indices we need to sum over, and thus yield a more efficient simulation of the same circuit.

### 2.4 Stabilizer formalism

Now, let’s switch gear and talk about a an entirely different framework for simulating quantum circuits. This time, we are considering efficient simulations of restricted class of quantum circuit.

When we say “efficient”, we mean $\text{poly}(n)$ where $n$ is the number of qubits, and the “restricted class” here means stabilizer circuits (clifford circuits).

The key idea here is to more compactly represent a quantum state, rather than storing all exponential number of amplitudes of the state vector.

#### 2.4.1 What is the class?

Def: Quantum circuits are called stabilizer circuits if they are made of stabilizer gates on input $|00...0\rangle$, and measurements in the computational basis.

Def: A gate is a stabilizer gate if it is generated from group $\{\text{CNOT, H, S}\}$ under multiplication.

$$
S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}.
$$
Example: Pauli gates:
\[ X = HZH \]
\[ Y = iXZ \]
\[ Z = SS \]

2.4.2 Stabilizer state

Def: A state is a stabilizer state if it can be generated from \(|00...0\rangle\) using stabilizer circuits.

Example. Single-qubit stabilizer states (6 of them) vertices on the octahedron.

2.4.3 Stabilizer

Def: \(|\psi\rangle\) is stabilized by \(U\), if \(U|\psi\rangle = |\psi\rangle\).

Stabilizer is used for error correction. It maps something that is slightly noisy back to it should be. Stabilizer code for simulation and error correction.

Examples.

- I: stabilize everything. \(I|\psi\rangle = |\psi\rangle\)
- X: \(+\). \(X|+\rangle = X\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) = \frac{|1\rangle+|0\rangle}{\sqrt{2}} = |+\rangle\)
- \(-|Z\rangle\): \(|1\rangle\). \(-Z|1\rangle = -(−|1\rangle) = |1\rangle\).

Two-qubit state:

- What dose \(X \otimes I = XI\) stabilize?
  \(|+\rangle \otimes |0\rangle, |+\rangle \otimes |+\rangle, ...\)

We want to uniquely represent a state, say \(|+\rangle \otimes |0\rangle\) with its stabilizer(s). Clearly \(XI\) is not enough, because it stabilizers multiple states and we cannot allow that ambiguity.
• Who else stabilize $|+\rangle \otimes |0\rangle$?
  \{II, XI, IZ, XZ\}

Note that $II$ stabilizers all two-qubit state, and $XZ$ is the product of $XI$ and $IZ$. So essentially, to uniquely represent our $|+\rangle \otimes |0\rangle$ state, we only need to keep track of the two stabilizers $XI$ and $IZ$. Remarkably, there is a theorem says that the number of stabilizers we need to keep track of is only $O(n)!$

### 2.4.4 Gottesman-knill theorem

**Thm.** [Got] There exists classical algorithm that simulates any stabilizer circuit in polynomial time.

In simulation, we don’t need to keep track of the amplitudes of state vector anymore. We can keep track of the stabilizer group.

### 2.4.5 How to update?

$|+\rangle \otimes |0\rangle \xrightarrow{I \otimes H} |+\rangle \otimes |+angle$

\{II, XI, IZ, XZ\} $\rightarrow$ \{II, XI, IX, XX\}

Rules:

• H gate: $X \rightarrow Z, Z \rightarrow X$
• S gate: $X \rightarrow Y, Z \rightarrow Z$
• CNOT: $XI \rightarrow XX, IX \rightarrow IX, ZI \rightarrow ZI, IZ \rightarrow ZZ$

### 2.4.6 Other ingredients enabling efficiently simulation [AG04]

• Tableau representation $(l \times 2n)$ matrix, where $l$ is the number of stabilizer = $O(n)$.
• Measurement is efficient: poly($n$) time to update tableau.
• Inner product for compute amplitude: efficient. $\langle x|U|00...0\rangle$

## References


