Ph 220: Quantum Learning Theory
Lecture 2: Review of Quantum Many-Body Physics

Hsin-Yuan Huang (Robert)

Caltech

1 Introduction to Quantum Many-Body Problems

In this lecture, we shift our focus from single-qubit sensing to the complex world of quantum many-body physics. A **local Hamiltonian** on n qubits is a sum of terms, where each term acts non-trivially on only a few qubits:

$$H = \sum_i h_i$$

Here, each h_i is a Hermitian operator that is the identity on all but a constant number of qubits. The ground state, $|\psi_g\rangle$, is the eigenvector corresponding to the minimum eigenvalue E_g. A fundamental problem in physics is the **Ground State Energy Problem**:

Problem: Given a local Hamiltonian $H = \sum_i h_i$ and two real numbers $a < b$ with a promise that the ground state energy E_g is either $E_g < a$ or $E_g > b$, decide which is the case.

A fundamental question is the following:

Q: How computationally difficult is this problem?

The difficulty depends on the nature of the local terms h_i:

- If h_i are diagonal in the computational basis (e.g., from the set $\{I, Z\}^{\otimes n}$), the problem is known to be **NP-complete**.
- If h_i can be arbitrary (e.g., from $\{I, X, Y, Z\}^{\otimes n}$), the problem is **QMA-complete**, the quantum analogue of NP-complete believed to be hard even for quantum computers.

Given this hardness, we focus on a specific, physically relevant class of ground states that are more tractable: **Short-Range Entangled (SRE) states**.

2 Short-Range Entangled (SRE) States

SRE states have a limited amount of entanglement, confined to local regions. From a quantum information perspective, SRE states can be defined as states that can be prepared from a simple product state, like $|0\rangle^{\otimes n}$, by a **constant-depth quantum circuit**. Imagine starting with all qubits in the $|0\rangle$ state. We then apply a sequence of layers of local quantum gates (e.g., two-qubit gates). The "depth" of the circuit is the number of such layers. For an SRE state, this depth is a constant, $O(1)$, meaning it does not grow as the number of qubits n increases. This structural property is key to their simplicity. The key question for this lecture is whether such states can be efficiently simulated on a classical computer.

Q: Is it classically easy to simulate short-range entangled states?

To answer this, we introduce the powerful language of **tensor networks**.

Tensor Network Representation

Instead of using diagrams, we can think of tensor networks algebraically. A tensor is a multi-dimensional array of numbers. A vector is a 1D tensor, and a matrix is a 2D tensor. A general tensor can have any number of indices. A tensor network represents a complex quantum state by breaking it down into a collection of smaller, interconnected tensors. The connections between tensors represent a mathematical operation called **tensor contraction**, which is a generalization of matrix multiplication. When two tensors are connected along an index, it means we are summing over all possible values of that index. This process allows us to calculate properties of the large quantum state by manipulating the smaller tensors.

For a 1D quantum system, like a chain of qubits, the constant-depth circuit structure that creates an SRE state can be translated directly into a specific type of tensor network known as a **Matrix Product State (MPS)**. In an MPS, the quantum state is described by a collection of tensors, one for each qubit. The amplitude for any computational basis state, like $|x_1 x_2 \dots x_n\rangle$, is a scalar value found by multiplying a sequence of matrices and vectors:

$$\langle x_1 \dots x_n|\psi\rangle = M_1^{x_1} M_2^{x_2} \cdots M_n^{x_n} \implies |\psi\rangle = \sum_{x_1,\dots,x_n\in\{0,1\}} (M_1^{x_1} M_2^{x_2} \cdots M_n^{x_n})\, |x_1,\dots,x_n\rangle \,.$$

Here, the tensors $M_i^{x_i}$ have specific dimensions: $M_1^{x_1}$ is a row vector of size $1\times\chi$, $M_n^{x_n}$ is a column vector of size $\chi\times 1$, and all matrices in between, $M_i^{x_i}$ for $i = 2,\dots,n-1$, are square matrices of size $\chi\times\chi$. The integer χ is the **bond dimension** of the MPS. For 1D SRE states, a fact is that they can be represented by an MPS with a small, constant bond dimension, $\chi = O(1)$.

3 Classical Simulation of 1D SRE States

Now we address the central question: Can we efficiently sample from the measurement outcomes of a 1D SRE state given its MPS representation? This means sampling from the probability distribution $p(x_1,\dots,x_n) = |\langle x_1 \dots x_n|\psi\rangle|^2$.

The answer is **yes**, and the classical algorithm can run in just $\mathcal{O}(n)$ time. The strategy is to sample the outcomes sequentially using the chain rule of probability:

$$p(x_1,\dots,x_n) = p(x_1)\cdot p(x_2|x_1)\cdot p(x_3|x_1,x_2)\cdots p(x_n|x_1,\dots,x_{n-1}).$$

Algorithm

The efficiency of the MPS sampling algorithm comes from avoiding the exponential cost of summing over all unmeasured qubits. We use a dynamic programming approach with two phases: a pre-computation phase (right-to-left) and a sampling phase (left-to-right).

1. **Pre-computation: $\mathcal{O}(n)$ time**

   First, we pre-calculate and store a series of matrices, A_k, which represent the summed probability of all possible outcomes for qubits from site k to n. We compute these matrices iteratively, starting from the right end of the chain.

   - Let $A_{n+1} = [1]$ be a 1×1 matrix (scalar). This serves as a trivial boundary condition.

- Then, for $k = n, n-1, \ldots, 2$, we compute A_k using A_{k+1}:

$$A_k = \sum_{x_k \in \{0,1\}} (M_k^{x_k})^\dagger A_{k+1} M_k^{x_k}.$$

  This is a rank-2 tensor.

Each step takes $\mathcal{O}(\chi^3) = \mathcal{O}(1)$ time. Repeating this $n-1$ times gives a total pre-computation time of $\mathcal{O}(n)$. We now have stored the list of matrices $A_2, A_3, \ldots, A_{n+1}$.

2. **Sequential Sampling: $\mathcal{O}(n)$ time**

   Now we sample outcomes from left to right. We will build up B_k, which is a rank-2 tensor incorporating the previous sampled measurement outcomes $x_1, \ldots, x_k$.

   - We begin with a trivial rank-2 tensor, the 1×1 matrix $B_0 = [1]$.
   - **Sample x_k:** Suppose we have already determined the tensor B_{k-1} from the first $k-1$ measurements. To find the probability of measuring x_k, we combine B_{k-1}, the matrix for site k: $M_k^{x_k}$, and the pre-computed rank-2 tensor A_{k+1}. The probability for sampling $x_1, \ldots, x_{k-1}, x_k$ is given by the following expression:

$$p(x_1, \ldots, x_{k-1}, x_k) = \mathrm{Tr}((M_k^{x_k})^\dagger B_{k-1} M_k^{x_k} A_{k+1}).$$

     We compute this value for both $x_k = 0$ and $x_k = 1$. We normalize these two values to get the conditional probability distribution for x_k,

$$p(x_k | x_1, \ldots, x_{k-1}) = \frac{p(x_1, \ldots, x_{k-1}, x_k)}{p(x_1, \ldots, x_{k-1}, 0) + p(x_1, \ldots, x_{k-1}, 1)},$$

     and then sample x_k from this conditional probability distribution.
   - Once x_k is sampled, we update our rank-2 tensor for the next step:

$$B_k = (M_k^{x_k})^\dagger B_{k-1} M_k^{x_k}.$$

     We then repeat the sampling step for x_{k+1}.

   We repeat this process until all n qubits have been sampled.

Each sampling step involves a few constant-time matrix-vector operations. Since there are n steps, the total time for the sampling phase is $\mathcal{O}(n)$. Combined with the pre-computation, the entire algorithm for generating one sample string $(x_1, \ldots, x_n) \in \{0,1\}^n$ runs in $\mathcal{O}(n)$ time.

4 Shallow Quantum Neural Networks (QNNs)

The structural results we have discussed have direct and important implications for the field of Quantum Machine Learning (QML). A popular model in QML is the **Quantum Neural Network (QNN)**, which is a quantum circuit with tunable parameters, analogous to a classical neural network. The goal is to "train" the circuit's parameters to perform a task, like classifying input states, generating bitstrings, simulating quantum dynamics.

A **shallow QNN** is a QNN with constant depth. From our discussion, it is clear that a 1D shallow QNN is structurally identical to the circuit that prepares a 1D SRE state. The only difference is that shallow QNNs do not have a specified input state of $|0^n\rangle$. Instead of an MPS representation, a shallow QNN admits an efficient **Matrix Product Operator (MPO)** representation.

$$U_{\text{QNN}} = \sum_{\substack{x_1,\ldots,x_n\in\{0,1\}\\ y_1,\ldots,y_n\in\{0,1\}}} (M_1^{x_1,y_1} M_2^{x_2,y_2} \cdots M_n^{x_n,y_n})\, |x_1,\ldots,x_n\rangle\langle y_1,\ldots,y_n| \tag{1}$$

Here, the tensors $M_i^{x_i,y_i}$ have specific dimensions: $M_1^{x_1,y_1}$ is a row vector of size $1\times\chi$, $M_n^{x_n,y_n}$ is a column vector of size $\chi\times 1$, and all matrices in between, $M_i^{x_i,y_i}$ for $i = 2,\ldots,n-1$, are square matrices of size $\chi\times\chi$. Almost all tasks performed by a 1D shallow QNN can be simulated efficiently on a classical computer. This includes both evaluating the QNN's output for a given product state input (inference) and calculating the gradients needed to train it. It tells us that to find a quantum advantage in machine learning, we must look beyond this simple, one-dimensional architecture.

5 What about in 2D?

Q: Are 2D SRE states and 2D shallow QNNs classically easy?

The situation changes dramatically in two dimensions. Simulating these systems is believed to be classically hard. Given that they are always quantumly easy, this provides the foundation for demonstrating quantum advantage in machine learning tasks.

A brief review of complexity classes

To understand this hardness, we need to introduce some powerful computational complexity classes.

- BPP and BQP: These are the standard complexity classes associated with the sets of decision problems that can be solved in polynomial time by randomized classical algorithms (BPP) and quantum algorithms (BQP).
- $\mathsf{PostBPP}$: The class of problems solvable by a probabilistic classical computer with the (physically unrealistic) ability of postselection: forcing a random bit to yield a specific outcome.
- $\mathsf{PostBQP}$: The class of problems solvable by a quantum computer with the (physically unrealistic) ability of postselection: forcing a measurement to yield a specific outcome. A landmark result by Scott Aaronson shows that $\mathsf{PostBQP} = \mathsf{PP}$.
- PP (Probabilistic Polynomial Time): A problem is in PP if there is a randomized algorithm that accepts with probability $> 1/2$ for "yes" instances and $< 1/2$ for "no" instances. Unlike BPP, the gap can be exponentially small, making it akin to counting solutions (counting how many possible assignments of the random bits correspond to "yes" vs "no").
- The **Polynomial Hierarchy** (PH): An infinite tower of complexity classes that generalizes NP. We define $\Sigma_0 = \mathsf{P}$, $\Sigma_1 = \mathsf{NP}$, $\Sigma_2 = \mathsf{NP}^{\mathsf{NP}}$, $\Sigma_k = \mathsf{NP}^{\Sigma_{k-1}}$, and so on. It is strongly believed that this hierarchy is infinite, that is, $\Sigma_k \neq \Sigma_{k+1}$ for any k. If PH were to "collapse" to a finite level, it would be a revolutionary result in computer science.

To gain a bit of intuition for why the Polynomial Hierarchy is believed to be an infinite tower of progressively harder problems, we can look at its lower levels. The notation $\mathsf{C}_1^{\mathsf{C}_2}$ denotes an "oracle" complexity class: the set of problems solvable by a machine that runs in the time complexity of C_1 but has access to a magical subroutine (an oracle) that can instantly solve any problem in class C_2.

Giving a machine access to an oracle can substantially increase its power. For example, giving a polynomial-time machine a polynomial-time oracle (P^{P}) doesn't help: the machine could have just solved the subroutine itself, so $\mathsf{P}^{\mathsf{P}} = \mathsf{P}$.

However, the situation changes when the oracle is more powerful. Consider the relationship between NP and its complement, coNP. A problem is in NP if "yes" instances have short, verifiable proofs. The classic example is the **Boolean Satisfiability Problem (SAT)**: given a logical formula like $(x_1 \vee \neg x_2) \wedge x_3$, is there at least one assignment of "True" or "False" to the variables that makes the whole formula true? If the answer is "yes," the proof is simple: the satisfying assignment itself. You don't have to check every possibility, just the one that works. A problem is in coNP if "no" instances have short proofs. The canonical example here is the **Tautology Problem (TAUTOLOGY)**: is a given logical formula true for *every* possible assignment of its variables? If the answer is "no," the proof is again simple: a single counterexample assignment that makes the formula false.

It is widely believed that $\mathsf{NP} \neq \mathsf{coNP}$ because there is a fundamental asymmetry between these two problems. Finding a single satisfying assignment for SAT (a proof for a "yes" answer) seems computationally much easier than proving that a formula is a tautology, which requires showing it's true for *all* assignments. The short proof for one problem doesn't seem to help you find a short proof for the other. However, they are indeed closely related. It is well known that NP and coNP are both contained within P^{NP}.

The containment of coNP is particularly instructive. Let's trace the algorithm for solving a coNP problem (like TAUTOLOGY) using a polynomial-time machine with access to an NP oracle.

1. Our machine is given a Boolean formula ϕ and must decide if it is a tautology.
2. The key logical step is this: ϕ is a tautology if and only if its negation, $\neg\phi$, is *unsatisfiable.*
3. The question "Is $\neg\phi$ unsatisfiable?" is a coNP problem. However, its complement, "Is $\neg\phi$ *satisfiable*?" is an instance of SAT, which is in NP.
4. Our machine constructs the formula $\neg\phi$ (a simple, polynomial-time operation).
5. It then asks the NP oracle: "Is this new formula, $\neg\phi$, satisfiable?" The oracle, which can solve any NP problem, answers "yes" or "no" instantly.
6. Our machine takes the oracle's one-bit answer and flips it to get the final result:
   - If the oracle says "yes" ($\neg\phi$ is satisfiable), it means there's a counterexample to ϕ being a tautology. Our machine outputs "no."
   - If the oracle says "no" ($\neg\phi$ is unsatisfiable), it means there are no counterexamples, so ϕ must be a tautology. Our machine outputs "yes."

This algorithm runs in polynomial time, proving that any problem in coNP can be solved in P^{NP}. Since P^{NP} contains both NP and coNP, it is considered strictly more powerful than either class alone (assuming they are different). The next level of the hierarchy, $\Sigma_2 = \mathsf{NP}^{\mathsf{NP}}$, allows a nondeterministic machine to access an NP oracle. This is believed to be a significant leap in computational power over plain NP, providing the intuition that each level of the hierarchy is genuinely harder than the last.

A celebrated result by Seinosuke Toda, known as **Toda's Theorem**, provides a shocking link between these classes:

$$\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$$

This means that any problem in the entire Polynomial Hierarchy can be solved efficiently by a classical computer that has access to a "PP oracle", a black box that can solve any problem in PP instantly. This shows the immense power of counting-based complexity.

Proving $\mathsf{PostBPP} \neq \mathsf{PostBQP}$ assuming PH does not collapse

Why do we need all of these complexity classes? The main usage is to derive that $\mathsf{PostBPP} \neq \mathsf{PostBQP}$, contingent on a standard conjecture in complexity theory. The argument proceeds by contradiction.

1. **Equalities and Inclusions:** From landmark results, we know:
   - $\mathsf{PostBQP} = \mathsf{PP}$ (Aaronson's Theorem).
   - $\mathsf{PostBPP} \subseteq \mathsf{BPP}^{\mathsf{NP}}$ (See *Threshold Computation and Cryptographic Security*; Intuition: an NP oracle can be used to find accepting paths to postselect upon, hence probabilistic classical machines with postselection can be simulated by a probabilistic classical machines with access to a magical machine that can solve any NP problems).
   - $\mathsf{BPP} \subseteq \mathsf{NP}^{\mathsf{NP}}$ (Sipser-Gowers-Lautemann Theorem).

2. **Placing $\mathsf{PostBPP}$ in the Polynomial Hierarchy:** We can combine the inclusions above.

   $$\mathsf{PostBPP} \subseteq \mathsf{BPP}^{\mathsf{NP}} \subseteq (\mathsf{NP}^{\mathsf{NP}})^{\mathsf{NP}} = \mathsf{NP}^{\mathsf{NP}} = \Sigma_2$$

   This places classical postselection firmly within the second level of the Polynomial Hierarchy.

3. **The Power of a $\mathsf{PostBPP}$ Oracle:** If we give a polynomial-time machine an oracle for $\mathsf{PostBPP}$, its power is contained within that of a Σ_2 oracle:

   $$\mathsf{P}^{\mathsf{PostBPP}} \subseteq \mathsf{P}^{\Sigma_2} = \mathsf{P}^{\mathsf{NP}^{\mathsf{NP}}}$$

   This class, $\mathsf{P}^{\mathsf{NP}^{\mathsf{NP}}}$, is contained in the third level, Σ_3, of the Polynomial Hierarchy.

4. **Contradiction:** Let us assume for a moment that $\mathsf{PostBPP} = \mathsf{PostBQP}$.
   - If this were true, then since $\mathsf{PostBQP} = \mathsf{PP}$, we would have $\mathsf{PostBPP} = \mathsf{PP}$.
   - This implies that a polynomial-time machine with a $\mathsf{PostBPP}$ oracle is as powerful as one with a PP oracle: $\mathsf{P}^{\mathsf{PostBPP}} = \mathsf{P}^{\mathsf{PP}}$.
   - From Toda's Theorem, we know that the Polynomial Hierarchy is contained in P^{PP}.
   - Combining these points, our assumption leads to: $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}} = \mathsf{P}^{\mathsf{PostBPP}} \subseteq \Sigma_3$.

   The result, $\mathsf{PH} \subseteq \Sigma_3$, means that the entire, supposedly infinite, Polynomial Hierarchy collapses to its third level. This is a major, widely disbelieved conjecture. Therefore, our initial assumption $\mathsf{PostBPP} = \mathsf{PostBQP}$ must be false.

This leads to the final result: assuming the Polynomial Hierarchy does not collapse, we must have

$$\mathsf{PostBPP} \neq \mathsf{PostBQP}.$$

Coming up next, we will use this relation to prove that 2D SRE states and 2D shallow QNN offer quantum advantage (classically hard but quantumly easy).