Chapter 18

Quantum Machine Learning

18.1 Introduction

Machine learning tries to extract patterns and regularities from given data for the purposes of prediction and understanding. In a slogan, one could say: ML = data + optimization. The data is what you learn from; the optimization finds a good model or hypothesis for the given data, that hopefully has some generalization power. ML has gone through several ups and downs over the years, but currently is booming thanks to the success of so-called “deep learning,” based on neural networks.\(^1\) ML is often subdivided into three subareas, depending on the data one has:

1. In *supervised learning* we are given labelled data, for instance pictures of animals annotated with the kind of animal that’s on the picture, and we want to learn how to predict the label.

2. In *unsupervised learning* we are just given unlabeled data, and need to find patterns in it. The canonical example is the *clustering* problem, where we are given unlabelled data items that we want to group into “similar” subsets. For example, it could be that our data consists of pictures of different kinds of animals (not labeled with the type of animal), and we somehow want to cluster the cat-pictures together, the wolf-pictures together, etc. We may or may not know in advance what the number of clusters should be.

3. In *reinforcement learning* the learner actually interacts with the environment, receiving rewards or penalties for desirable or undesirable behavior, and tries to learn from this interactive data to behave more successfully in the environment. This is roughly how a child learns.\(^2\)

It is a very interesting question to see how quantum computing changes and helps machine learning. Here the learner would be a quantum computer, and the data may be classical or quantum. Quantum ML is by now a rather large area, and in this chapter we will go over a few representative results and methods for supervised and unsupervised learning, mostly with classical output. See [90] for quantum applications to reinforcement learning, and [47, 23, 200] for much more.

\(^1\)Machine learning based on neural networks has been studied for decades but quite suddenly became much more successful starting around 2012, due to the availability of more data, stronger computing hardware (incl. special-purpose GPUs for fast parallel matrix-vector calculations), and better software to do the training.

\(^2\)It’s also how a computer can learn to play games. One of the big breakthroughs of machine learning was in 2016 when the AlphaGo program learned to play the game of Go so well that it beat one of the world’s best human Go players. Computers have been better than humans at chess already since the late 1990s (the IBM program “Deep Blue” beat Kasparov in 1997), but Go was viewed as a much more complicated game than chess.
18.2 Supervised learning from quantum data

18.2.1 The PAC model of learning

Let us first describe a mathematical model of what it means to learn from labeled data. This is Valiant’s PAC model \[215\], for “probably approximately correct” learning (see \[201, 171\] for more).

Assume for simplicity that the labels are just binary: 0 or 1. Our goal is to learn a Boolean function \(f : \mathcal{X} \to \{0, 1\}\) from examples of the form \((x, f(x))\), where \(x \in \mathcal{X}\). A typical case would be \(\mathcal{X} = \{0, 1\}^n\). The last bit \(f(x)\) of the example is called the label. Think for instance about the case where we are given 1000 \(\times\) 1000-pixel black-and-white pictures \((n = 1000, 000)\) whose labels \(f(x)\) indicate whether \(x\) is the picture of a wolf or not. We would like to learn \(f\), or some good approximation of it, to be able to recognize pictures of wolves in the future. Some \(x\)’s are more important and more likely to appear as examples than others: many 1000 \(\times\) 1000-grids don’t depict anything. The assumption in PAC learning is that the examples are generated (independent and identically distributed) according to some distribution \(D\) on \(\mathcal{X}\). The idea is that this \(D\) represents “the world” or “Nature,” which provides us with examples. We assume \(f\) cannot be completely arbitrary (in that case there would be an \(f\) consistent with every possible sequence of labeled examples) but comes from some known “concept class” \(\mathcal{C}\) of Boolean functions. For instance, \(\mathcal{C}\) could be a set of small logical formulas \(f\) on \(n\) Boolean variables, or a set of small-depth or small-size decision trees on \(n\) input bits, or neural networks with a restricted number of nodes or depth.

A learning algorithm should generate a “hypothesis” \(h : \mathcal{X} \to \{0, 1\}\) that has small error compared to the unknown \(f\) that we’re trying to learn, measured under the same distribution \(D\) that generated the data.\(^3\) The generalization error of \(h\) w.r.t. the target function \(f\) is defined as

\[
\text{err}_D(f, h) = \Pr_{x \sim D}[f(x) \neq h(x)].
\]

This error measures how well we’ve generalized the examples, and how well we can predict the labels of future examples. We say that \(h\) is “approximately correct” if this error is small, at most some specified \(\varepsilon\). The goal in PAC learning is to output an \(h\) that is probably approximately correct:

**Definition 4** An \((\varepsilon, \delta)\)-PAC learner for a concept class \(\mathcal{C}\) w.r.t. distribution \(D\) on \(\mathcal{X}\), is an algorithm that receives \(m\) labeled examples \((x_1, f(x_1)), \ldots, (x_m, f(x_m))\) for a target function \(f \in \mathcal{C}\), where each \(x_i \sim D\), and that outputs a hypothesis \(h\) such that

\[
\Pr[\text{err}_D(f, h) \leq \varepsilon] \geq 1 - \delta.
\]

The learning algorithm has to satisfy the above for every possible target function \(f \in \mathcal{C}\), and the probability is over both the choice of the examples and over the internal randomness of the algorithm.

An \((\varepsilon, \delta)\)-PAC learner for a concept class \(\mathcal{C}\) is an algorithm that is an \((\varepsilon, \delta)\)-PAC learner for \(\mathcal{C}\) w.r.t. every possible distribution \(D\).

Note that the first part of the definition is about learners that are only required to work correctly for one specific distribution \(D\) (for instance, the uniform distribution over \(\mathcal{X}\)), while the second part is “distribution-independent”: here we want a learner that works well irrespective of what

\(^3\)It is important to be taught and tested according to the same distribution \(D\). Imagine a quantum-computing course whose lectures focused on the mathematics of quantum algorithms, but with an exam that focuses on physics questions about how to implement qubits and gates—that would clearly be very unreasonable.
(unknown) distribution $D$ generates the data. This is in keeping with the usual attitude towards algorithms in computer science: these should work well even for a worst-case input. We don’t require the class $\mathcal{H}$ of possible hypotheses $h$ to equal the class $\mathcal{C}$ of possible target functions $f$ (if we add this requirement, then it’s called *proper* PAC learning). This allows us for instance to use neural networks to learn target functions that come from some other class $\mathcal{C}$, say logical formulas.

The number of examples $m$ that a particular learning algorithm uses is called its “sample complexity,” and the overall time or number of elementary operations it takes to output $h$ is its “time complexity.” Clearly the latter upper bounds the former, since we need at least one operation to process one example. The sample complexity of a concept class $\mathcal{C}$ (as a function of $\varepsilon, \delta$) is the minimal sample complexity among all PAC learners for $\mathcal{C}$. Ideally, a good learner for $\mathcal{C}$ has both small sample complexity and small time complexity (say, polynomial in $n$). For some concept classes $\mathcal{C}$ efficient distribution-independent PAC learners exist, for example the class of logical formulas in $k$-Conjunctive Normal Form (i.e., each $f$ would be the AND of several ORs, each of at most $k$ variables or negated variables) or the class of regular languages (with the added help of so-called “membership queries”), but there are also many $\mathcal{C}$ that are not efficiently learnable.

### 18.2.2 Learning from quantum examples under the uniform distribution

There are different ways to define learning from quantum data. One natural way, due to Bshouty and Jackson [60], is to replace each classical random example $(x, f(x))$, with $x \sim D$, by a superposition. Focusing on the typical case $X = \{0, 1\}^n$, a quantum example would be the $(n+1)$-qubit state $\sum_{x\in\{0,1\}^n} \sqrt{D(x)}|x, f(x)\rangle$.

Of course, the world doesn’t usually present us with quantum examples, in contrast with the abundance of classical data for machine learning. So this model is only relevant in special cases, for example if have a physical experiment producing such states.

One thing we could do with a quantum example is measure it in the computational basis, but that would just give us back a classical example $(x, f(x))$ with $x \sim D$. A more clever thing we can do is *Fourier sampling*. Suppose $D$ is the uniform distribution. Exercise 1 shows how to convert a quantum example (with probability 1/2) into an $n$-qubit state where the labels are $\pm 1$-phases: $\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle$.

If we apply $n$ Hadamard gates to this state, then we get $\sum_{s\in\{0,1\}^n} \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{x \cdot s} (-1)^{f(x)}|s\rangle = \sum_{s\in\{0,1\}^n} \alpha_s |s\rangle$.

If we measure this state, then we’ll see outcome $s \in \{0, 1\}^n$ with probability $\alpha_s^2$. The amplitudes $\alpha_s$ are called the Fourier coefficients of the function $(-1)^{f(x)}$, whence the name “Fourier sampling.” In some cases Fourier sampling gives a lot of information about the $f$ we’re trying to learn.

**Learning linear functions.** A perfect illustration of Fourier sampling is for the following class: $\mathcal{C} = \{f_s \mid s \in \{0, 1\}^n, \forall x : f_s(x) = x \cdot s \mod 2\}$,
these are the linear functions modulo 2. It is easy to calculate that if we do Fourier sampling on a quantum example for function $f_s$, then $\alpha_s = 1$ and $\alpha_{s'} = 0$ for all $s' \neq s$. So one Fourier sample already tells us what $s$ is! Hence we can learn $f_s$ exactly (i.e., for $\varepsilon = 0$), with high probability, using $O(1)$ examples and $O(n)$ elementary gates. In contrast, learning linear functions from classical examples under the uniform distribution requires $\Theta(n)$ examples (see Exercise 2).

**Learning DNF.** A richer concept class that can be learned efficiently from uniform quantum examples is the class of $s$-term Disjunctive Normal Form (DNF) formulas on $n$ Boolean variables. These are formulas of the form $f(x) = (x_1 \land \neg x_3) \lor (x_2 \land x_3 \land x_5)$, i.e., an OR of up to $s$ different ANDs of variables or negations of variables. The concept class $C$ of $s$-term DNF is not known to be efficiently PAC learnable w.r.t. the uniform distribution $D$ classically. However, Bshouty and Jackson [60] showed that $s$-term DNF can be learned in polynomial time (in $s$ and $n$) from uniform quantum examples. Roughly speaking, they use Fourier sampling to produce a linear function that is weakly correlated with the target DNF function $f$, and then use a classical “boosting” algorithm to combine multiple such weak hypotheses into one good hypothesis $h$. We’ll skip the details here.

**18.2.3 Learning from quantum examples under all distributions**

We saw a few cases where quantum examples reduce the sample and/or time complexity of learning algorithms w.r.t. a fixed data-generating distribution $D$, namely uniform $D$. But in the PAC model we ideally want a *distribution-independent* learner that works well for every possible distribution $D$. Can allowing quantum instead of classical examples significantly reduce the sample complexity of learning a class $C$ in the distribution-independent setting? It turns out the answer is ‘no’.

Classically, the number of examples that is necessary and sufficient for $(\varepsilon, \delta)$-PAC learning a concept class $C$ is known to be

$$m = \Theta \left( \frac{\text{VC dim}(C)}{\varepsilon} + \frac{\log(1/\delta)}{\varepsilon} \right), \quad (18.1)$$

where VC dim$(C)$ is the so-called VC-*dimension* of $C$, defined as follows. We say that a set $S \subseteq \{0, 1\}^n$ is shattered by $C$ if for each of the $2^{|S|}$ possible labelings $\ell : S \rightarrow \{0, 1\}$, there is a function $f \in C$ that has the same labeling of $S$ (i.e., $f|_S = \ell$). VC dim$(C)$ is the size of a largest $S$ shattered by $C$. Intuitively, larger VC-dimension corresponds to a more complex or “richer” (and hence harder to learn) concept class. We won’t prove the characterization of Eq. (18.1) here, but Exercises 4 and 5 go most of the way towards the claimed upper and lower bounds on $m$, respectively.

It was proven in [25] that in fact the same formula Eq. (18.1) determines the number of quantum examples that are necessary and sufficient for learning $C$. So, up to constant factors, quantum examples are not more useful than classical examples for distribution-independent PAC learning.

**18.2.4 Learning quantum states from classical data**

One can generalize PAC learning from Boolean-valued to real-valued target functions $f : \mathcal{X} \rightarrow [0, 1]$, and then consider a hypothesis $h : \mathcal{X} \rightarrow [0, 1]$ to be approximately correct (for some small $\gamma$) if

$$\text{err}_{D, \gamma}(f, h) = \Pr_{x \sim D}[|f(x) - h(x)| > \gamma] \leq \varepsilon.$$ 

So now a good hypothesis $h$ is supposed to be close to $f$ (rather than equal) for most $x$. 

152
An interesting example is the problem of learning an unknown $n$-qubit quantum state $\rho$ from measurement data. Let $\mathcal{X}$ be the set of measurement elements, i.e., psd matrices $M$ with $\|M\| \leq 1$. If we measure $\rho$ with some POVM of which $M$ is one element, then the probability to get the outcome corresponding to $M$, is $\text{Tr}(M\rho)$. Accordingly, we can define $f : \mathcal{X} \rightarrow [0,1]$ as $f(M) = \text{Tr}(M\rho)$ and consider the class $\mathcal{C}$ of all such functions (one $f$ for each possible $\rho$, so this class is uncountable). Aaronson [1] showed that this $\mathcal{C}$ is classically PAC learnable from $O(n)$ examples of the form $(x, f(x))$ (with some polynomial dependence of the sample complexity on $\gamma, \varepsilon$, and exponential time complexity). Note that we are not really learning $\rho$ itself, but rather learn to predict the measurement probabilities. In contrast, learning a good approximation of $\rho$ itself (with small error in trace distance) requires a number of copies of $\rho$ that is exponential in $n$ [114, 179]. Some positive results for learning specific classes of quantum states can be found in [163, 16, 147].

18.3 Unsupervised learning from quantum data

In this section we will look at an example of unsupervised learning from quantum data, namely for a clustering problem. Suppose we are given $m$ vectors $v_1, \ldots, v_m \in \mathbb{R}^d$, say unit vectors for simplicity, that we want to cluster into $k$ sets. Assume $k$ is much smaller than $d$, say at most $\text{polylog}(d)$. One way to do the clustering is to find $k$ suitable unit vectors $c_1, \ldots, c_k \in \mathbb{R}^d$ (which may or may not be in $\{v_i\}$ themselves), view each as the “center” of its own cluster, and assign the data-vector $v_i$ to the center that is closest or that has most overlap with $v_i$. How to find those $k$ centers? One method that often (though not always) works well is to find the $k$ eigenvectors corresponding to the $k$ largest eigenvalues of the following $d \times d$ “correlation matrix”:

$$A = \sum_{i=1}^{m} v_i v_i^T.$$  

Those $k$ eigenvectors are called the $k$ “principal components” of $A$. They intuitively correspond to the $k$ most important directions in the data, and we can choose them as the centers of the clusters.

Classically, we can find those $k$ eigenvectors by diagonalizing $A$, which takes times polynomial in $d$. In the quantum case we can do something very different, under the (very strong) assumption that we can efficiently, say in time $\text{polylog}(d)$, prepare the $[\log(d)]$-qubit quantum states $|v_i\rangle$ corresponding to the vectors $v_i$. By choosing $i \in [m]$ uniformly at random and preparing $|v_i\rangle$, we prepare the following $[\log(d)]$-qubit mixed state, which is proportional to the correlation matrix:

$$\rho = \frac{1}{m} \sum_{i=1}^{m} |v_i\rangle \langle v_i| = \frac{1}{m} A.$$  

Let’s say this has (unknown) spectral decomposition $\rho = \sum_{j=1}^{d} \lambda_j |c_j\rangle \langle c_j|$ with $\lambda_1 \geq \cdots \geq \lambda_d \geq 0$, where the first $k$ eigenvalues sum to something close to 1, and are not too close together, at least $1/\text{poly}(k)$ apart.\footnote{All these assumptions make this principal-component analysis a heuristic method for clustering, not something that provably always works. It is anyway not clear in this case what a correct or optimal output would be: because this is unlabeled data, we do not have a clear standard for correctness.} We would now like to find the top-$k$ eigenstates $|c_1\rangle, \ldots, |c_k\rangle$ of this $\rho$.

Note that the unitary $U = e^{i \rho}$ has the same eigenstates as $\rho$ itself, with every eigenvalue $\lambda_j$ of $\rho$ translating into eigenvalue $e^{i \lambda_j}$ of $U$. Lloyd et al. [157] (with more precise analysis and matching lower bound in [139]) showed that we can actually implement $U^t$ up to error $\varepsilon$ using $O(t^2/\varepsilon)$ copies
of the state $\rho$ (see Exercise 6). We now use phase estimation with the unitary $U$ on a copy of $\rho$ itself, with additive error $\delta = 1/\text{poly}(k)$. By Section 4.6, phase estimation with additive error $\delta$ corresponds to running controlled versions of $U^t$ for $t$ up to $O(1/\delta)$. Under our earlier assumptions, this only takes $\text{poly}(k) = \text{polylog}(d)$ time. Ignoring for simplicity the small errors ($\leq \delta$) that phase estimation makes in estimating the values $\lambda_j$, phase estimation transforms the copy of $\rho$ and a few auxiliary $|0\rangle$-qubits into the state

$$\sum_{j=1}^{m} \lambda_j |c_j\rangle \langle c_j| \otimes |\lambda_j\rangle \langle \lambda_j|.$$ 

If we measure the second register, then we obtain state $|c_j\rangle \otimes |\lambda_j\rangle$ with probability $\lambda_j$.\footnote{Actually, in the second register we will see (with high probability) a $\delta$-close approximation of $\lambda_j$ rather than $\lambda_j$ itself, but if we assume $\delta$ is smaller than the spacing between the eigenvalues $\lambda_1, \ldots, \lambda_k$, then such a $\delta$-approximation is good enough to assign the state in the first register to the correct cluster.} Doing this $\text{poly}(k)$ many times, we learn the $k$ largest values $\lambda_1, \ldots, \lambda_k$, and for each of those $\lambda_j$’s we’ll have a number of copies of the eigenstate $|c_j\rangle$. This is a quantum form of Principal Component Analysis.

This collection of eigenstates determines a clustering in $k$ subsets, but it is not very explicit. Suppose we receive a new unit vector $v$ (which again we assume we can prepare efficiently as a state $|v\rangle$). How do we decide to which of the $k$ subsets this $v$ belongs? One thing we can do is use a few copies of each $|c_i\rangle$ to approximate $|\langle c_i|v\rangle|^2$ for each $i$ using the SWAP-test, and then assign $v$ to cluster $i$ where this squared overlap is maximized. This “quantum PCA” has a lot of drawbacks, but at least it shows some genuinely quantum tricks that we can use for unsupervised learning under the assumption that our input vectors can be efficiently prepared as quantum states.

### 18.4 Optimization

In the previous two sections we assumed quantum data: either the data is already given as a superposition, or we can efficiently put given classical data in superposition. However, in most real-world applications of machine learning we have classical data without the means to efficiently make this quantum. Remembering the slogan ML = data + optimization, if there’s any room left for quantum improvements when data is classical, it would be in the optimization to find a well-fitting model for the data. We’ll look at some examples where quantum computing might help.

#### 18.4.1 Variational quantum algorithms

One approach that has received a lot of attention is to optimize over parametrized circuits. Suppose we have a quantum circuit $U(\theta)$ with a vector $\theta$ of parameters. This could for instance be a circuit where CNOTs and single-qubit rotations are already in place, but the angles of the single-qubit gates are parameters that we can tweak. This $U(\theta)$ is then applied to a fixed starting state, say $|0\rangle$, yielding a final state $|\psi(\theta)\rangle = U(\theta)|0\rangle$. The goal is now to minimize the expected value of some observable $M$, i.e., to find a $\theta$ to minimize the function $f(\theta) = \langle \psi(\theta)|M|\psi(\theta)\rangle$. In the case of supervised learning applications, $U(\theta)$ could for instance represent some hypothesis (i.e., a way to predict labels of $x$’s), $M$ could incorporate the given labeled examples $(x, f(x))$, and $f(\theta)$ could be the “empirical error”: the fraction of mis-predicted labels among the given examples.

Note that $f(\theta)$ can be computed approximately (for classically given $\theta$) on a quantum computer by repeatedly preparing $|\psi(\theta)\rangle$ and measuring the observable $M$. If the circuits $U(\theta)$ are relatively
simple (say, few qubits, few gates, low depth) and $M$ is relatively easy to measure (say, a sum of a few $n$-qubit Pauli matrices with few non-identity terms) then this could already be done on a relatively small and simple quantum computer. Variational quantum algorithms (VQAs) are typically hybrid classical-quantum algorithms: the minimization over $\theta$ is usually done by a classical outer loop that iteratively improves $\theta$. Using the ability to approximately compute $f$ we can for instance try to do approximate gradient descent (move $\theta$ by some step-size in the direction of steepest descent of $f$) or some other method. This is analogous to the iterative way the weights in neural networks are optimized, and these variational quantum approaches are sometimes (with a keen sense for marketing) called “quantum neural networks” or “quantum deep learning.” For combinatorial optimization, a very structured version of the variational approach is the Quantum Approximate Optimization Algorithm (QAOA) \cite{97}. See \cite{67} for a general overview of VQAs.

One interesting application of this variational idea is in trying to find the smallest eigenvalue of a given Hamiltonian $H$. For example, $H$ could describe the energy of a chemical system as a function of the locations of the particles (nuclei and electrons) of the system; the smallest eigenvalue of $H$ would be the “ground-state energy” of the system, which is an important quantity in chemistry. We know from Chapter 13 that in general this problem of determining or even well-approximating this ground state energy is QMA-hard, even in the special case where $H$ is a sum of 2-local terms, so in general this shouldn’t be efficiently solvable on a quantum computer. However, suppose that from some general physics or chemistry intuition we have a rough idea of what the ground state of our particular Hamiltonian $H$ should look like, something we can prepare using a simple parametrized circuit $U(\theta)$. The set of states $|\psi(\theta)\rangle = U(\theta)|0\rangle$ that we are limiting ourselves to, is called an “Ansatz” (German for “approach” or “attempt”). We can now try to optimize the parameters $\theta$ in order to minimize the expected value $f(\theta) = \langle \psi(\theta) | H | \psi(\theta) \rangle$, i.e., the energy of the state $|\psi(\theta)\rangle$. This approach is called the “variational quantum eigensolver” (VQE) \cite{181}, and is one of the best hopes for applying smallish, near-term quantum computers to problems in chemistry.

\subsection{Some provable quantum speedups for optimization}

The variational approach is rather heuristic: it very much depends on how good the “Ansatz” (the choice of the class of parametrized circuits $U(\theta)$) happens to be for the particular problem at hand. Here we mention some other approaches, which yield provable (albeit usually only polynomial) quantum speedups under some assumptions on how the input is given.

- There are many quantum speedups for optimization problems on graphs, typically using Grover search (Section 7.2), Grover-based minimum-finding (Exercise 7.9), amplitude amplification (Section 7.3), or amplitude estimation (Exercise 7.7) as a subroutine. Examples are finding shortest paths \cite{91} and approximating minimum cuts or graph sparsification \cite{20}.

- Solving linear systems and other basic linear algebra is ubiquitous in classical optimization algorithms. Since quantum states are vectors and quantum operations are matrices, one can try to improve such classical algorithms using quantum algorithms. Examples are HHL (Chapter 10) and the block-encoding approach (Section 9.4 and \cite{108}, Section 3.2.4). The trouble with this approach is that it often assumes the input is a quantum state (which is not always practical) and/or that it produces the output as a quantum state (which is not always useful). For example, HHL has both of these features. See \cite{2} for more discussion.

One interesting application of “quantum linear algebra” (with classical inputs and outputs!) is the quantum recommendation system of Kerenidis and Prakash \cite{137}, which can generate
recommendations of type “you might also like” to a user of systems like Amazon or Netflix, based on the user’s and other users’ earlier behavior. Initially [137] was believed to give an exponential speedup over classical recommendation systems, until Tang showed how to “dequantize” their quantum algorithm under similar classical access assumptions [211, 70].

• In convex optimization we minimize a convex function \( f : \mathbb{R}^n \to \mathbb{R} \), either over all \( x \in \mathbb{R}^n \) or over all \( x \) that are constrained to lie in some convex domain \( X \subset \mathbb{R}^n \). This covers a big part of continuous optimization. Convexity ensures that the only local minima are also global minima, but such methods often still work to find good local minima for non-convex problems (such as training neural networks). Iterative first-order methods like gradient descent use the gradient of \( f \) at a given point, which in some cases can be computed more efficiently by quantum algorithms [133, 109, 84]. Second-order methods often solve a linear system involving the Hessian (the \( n \times n \) matrix of partial second derivatives at a given point), and we can try to use quantum linear algebra. If the matrix is symmetric and diagonally dominant and the output needs to be classical, then we could use the linear solver of [20].

Quantum algorithms are known for the specific cases of linear programming (LPs) and semidefinite programming (SDPs) [53, 18, 52, 19, 17], for learning support vector machines (SVMs) [185, 199, 192, 8, 193], and for least-squares linear regression with an \( \ell_1 \)-regularizer [69].

Exercises

1. Suppose that for some unknown Boolean function \( f : \{0,1\}^n \to \{0,1\} \) and amplitude-vector \( (\alpha_x)_{x \in \{0,1\}^n} \), you are given one copy of the \((n + 1)\)-qubit state

\[
\sum_{x \in \{0,1\}^n} \alpha_x |x| f(x).
\]

Show how you can convert this into state

\[
\sum_{x \in \{0,1\}^n} \alpha_x (-1)^f(x) |x| 1
\]

with success probability 1/2, in such a way that you know when you succeeded.

2. Consider again the concept class \( C \) of linear functions mod 2.

(a) Give a classical learning algorithm to learn a linear function exactly with high success probability (\( \varepsilon = 0, \delta = 1/3 \)) using \( O(n) \) uniform random examples and \( O(n^3) \) time.

(b) Argue that every classical PAC learner for \( C \) under uniform \( D \), with \( \varepsilon < 1/4 \), needs \( \Omega(n) \) examples.

3. In the model of exact learning with membership queries, the goal is to exactly learn a target function \( f \in C \) from queries to \( f \) (so there are no examples in this setting, or rather the learner can choose their own examples).

Show that if \( C \) is the concept class of linear functions, then a target function \( f \in C \) can be learned with 1 quantum membership query, but requires \( \Omega(n) \) classical membership queries.
4. Consider a concept class $\mathcal{C}$ of functions $f : \mathcal{X} \rightarrow \{0,1\}$, with $\mathcal{X} = \mathbb{N}$ and $VCdim(\mathcal{C}) = d$.

(a) Consider the following simple (and probably not very time-efficient) learning algorithm:

Draw $m$ examples for target function $f$; output a $h \in \mathcal{C}$ consistent with these examples.

Let $h \in \mathcal{C}$ be a function with $err_D(f,h) > \varepsilon$. Show that at the end of this algorithm, the probability that $h$ is still consistent with the $m$ examples is $< (1 - \varepsilon)^m$.

(b) Set $m = \lceil \log(3|\mathcal{C}|)/\log(1/(1 - \varepsilon)) \rceil$. Show that with probability $\geq 2/3$, the only $h$ that are consistent with the $m$ examples have $err_D(f,h) \leq \varepsilon$.

(c) Derive an upper bound $m = O(d \log(N)/\varepsilon)$ on the classical sample complexity of $(\varepsilon,1/3)$-PAC learning the class $\mathcal{C}$ using Sauer’s lemma, which says that $|\mathcal{C}| \leq \sum_{i=0}^{d} \binom{N}{i}^3$.

5. Suppose the set $S = \{x_1, \ldots, x_d\} \subseteq \mathcal{X}$ is shattered by concept class $\mathcal{C}$. Consider a distribution $D$ that puts $1 - 4\varepsilon$ probability on $x_1$ and $4\varepsilon/(d - 1)$ probability on each of $x_2, \ldots, x_d$.

(a) Let $f \in \mathcal{C}$ be the target function. Show that you need $\Omega((d - 1)/\varepsilon)$ examples ~ $D$ to see (with probability $\geq 2/3$) $(x_i, f(x_i))$ for at least 50% of the $i \in \{2, \ldots, d\}$.

(b) Show that the sample complexity of every $(\varepsilon, 1/3)$-PAC learner for the class $\mathcal{C}$ is at least $\Omega((d - 1)/\varepsilon)$.

6. (H) Let $\sigma$ and $\rho$ be $k$-qubit mixed states, $\varepsilon > 0$ small, $t \geq 0$, and $U = e^{i\theta}$ be a unitary. Our goal in this exercise is to apply $U^t$ to $\sigma$ (with error $\leq \varepsilon$ in trace norm) at the expense of using some copies of the state $\rho$.

(a) Let $V$ be the 2-qubit SWAP-gate (which maps $|a\rangle|b\rangle \rightarrow |b\rangle|a\rangle$ for all $a,b \in \{0,1\}$). Show that $V^{-2\eta/\pi} = e^{-i\eta}e^{iV\eta}$ for all $\eta \geq 0$.

(b) Let $W$ be the $2k$-qubit unitary that swaps the first $k$ qubits with the last $k$ qubits, i.e., it maps $|a\rangle|b\rangle \rightarrow |b\rangle|a\rangle$ for all $a,b \in \{0,1\}^k$. Show that for all $\eta \geq 0$, $e^{iW\eta}$ can be implemented with $k$ 2-qubit gates.

(c) Show that for small $\eta \geq 0$, $U^n\sigma U^{-n} = \sigma + i\eta(\sigma\rho - \rho\sigma) + E$, where $\|E\|_1 = O(\eta^2)$.

(d) Let $\sigma'$ be the $k$-qubit local state of the first register after applying the unitary $e^{iW\eta}$ to the $2k$-qubit state $\sigma \otimes \rho$. Show that $\sigma' = \sigma + i\eta(\sigma\rho - \rho\sigma) + E'$, where $\|E'\|_1 = O(\eta^2)$.

(e) Show that $\|\sigma' - U^n\sigma U^{-n}\|_1 = O(\eta^2)$.

(f) Show that you can implement $U^t$ on $\sigma$ with error $\varepsilon$ in trace norm, using $O(t^2/\varepsilon)$ copies of $\rho$ and $O(kt^2/\varepsilon)$ elementary gates.
distance exactly \( n/2 \). Use part (a) to argue that Alice needs to send pairwise orthogonal states for those \( n \) inputs, and hence her message-space must have dimension at least \( n \).

3. Use the fact that 2 non-orthogonal states cannot be distinguished perfectly (Exercise 2), and that a set of \( 2^n \) vectors that are pairwise orthogonal must have dimension \( 2^n \).

5. Invoke the quantum random access lower bound, Theorem 3 of Section 14.2.

7.b. Let Alice send a random row of \( C(x) \) (with the row-index) and let Bob send a random column of \( C(y) \) (with the column-index).

8.a. Two distinct polynomials, each of degree \( \leq d \), are equal on at most \( d \) points of the domain \( \mathbb{F}_p \).

9.b. Run the protocol of part (a) on an initial state where Bob has a well-chosen superposition over many \( |y \rangle \).

10.b. You can derive this from one of the communication lower bounds mentioned in this chapter, you don’t need to prove this from scratch.

11. The matching \( M \) induces a projective measurement that Bob can do on the message he receives.

12.d. Alice could send a uniform superposition over all \( h \in H \).

Chapter 16

1.b. You could write this out, but you can also get the answer almost immediately from part (a) and the fact that \( H^T = H^{-1} \).

2.b. It’s helpful here to write the EPR-pair in the basis \( |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), |\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \).

4. For every fixed input \( x, y \), there is a classical strategy that gives a wrong output only on that input, and that gives a correct output on all other possible inputs. Use the shared randomness to randomly choose one of those deterministic strategies.

6.b. Argue that \( \frac{1}{2} \langle \psi | C | \psi \rangle = \Pr[\text{win}] - \Pr[\text{lose}] \).

6.c. Use that \( A_x^2 \) and \( B_y^2 \) are the \( k \)-qubit identity matrix.

6.d. Use Cauchy-Schwarz to show \( (\langle \psi | C | \psi \rangle)^2 \leq \langle \psi | C^2 | \psi \rangle \), and then upper bound the latter.

6.e. \( \cos(\pi/8)^2 = \frac{1}{2} + \frac{1}{\sqrt{8}} \).

Chapter 17

6. Use the encoding of Exercise 5, so that Alice and Bob need to cooperate to learn the key used to change \( \rho \).

7. Show that a unitary on Alice’s side of the state won’t change Bob’s local density matrix \( \rho_B \).

9.a. The singular value decomposition (see end of Appendix A.5) of the \( d \times d \) matrix \( M \) whose entries are \( M_{ij} = \alpha_{ij} \) can be computed in polynomial time, you may assume this without proof.

Chapter 18

6.a. You can diagonalize \( V \) by something like a Hadamard gate on the “middle two” basis states, \( |01\rangle \) and \( |10\rangle \).

6.b. \( W \) consists of \( k \) 2-qubit SWAP-gates.

6.c. It’s helpful to write \( U^{i\eta} = I + i\eta F \) for some matrix \( F \) with \( \|F\|_1 = O(\eta^2) \). This follows from Taylor series, you don’t need to prove this. Here the trace norm \( \|A\|_1 \) of a matrix \( A \) is defined as the sum of \( A \)’s singular values.

6.d. You can first prove this for the case where \( \sigma = |a\rangle\langle a| \) and \( \rho = |b\rangle\langle b| \) are pure states, and then extend to general mixed states by linearity.
6.f. Apply part (d) \( r = O(t^2/\varepsilon) \) times with \( \eta = O(\varepsilon/t) \), choosing the constants in the \( O(\cdot) \) such that \( r\eta = t \) and hence \( (U\eta)^r = U^t \). Upper bound the overall error using triangle inequality.

Chapter 19

1. Compute the trace \( \text{Tr}(E^*E) \) in two ways, and use the fact that \( \text{Tr}(AB) = 0 \) if \( A \) and \( B \) are distinct Paulis, and \( \text{Tr}(AB) = \text{Tr}(I) = 2 \) if \( A \) and \( B \) are the same Pauli.

5. Given an unknown qubit \( \alpha|0\rangle + \beta|1\rangle \) encoded using this code, you could split the \( 2k \) qubits into two sets of \( k \) qubits each, and use each to recover a copy of the unknown qubit.
Bibliography


