Chapter 5
Information Encoding

If we want to use a quantum computer to learn from classical data—which was in the introduction referred to as the *CQ* case—we have to think about how to represent features by a quantum system. We furthermore have to design a recipe for "loading" data from a classical memory into the quantum computer. In quantum computing, this process is called *state preparation* um machine learning algorithm.

Classical machine learning textbooks rarely discuss matters of data representation and data transfer to the processing hardware (although considerations of memory access become important in big data applications). For quantum algorithms, these questions cannot be ignored (see Fig. 5.1). The strategy of how to represent information as a quantum state provides the context of how to design the quantum algorithm and what speedups one can hope to harvest. The actual procedure of encoding data into the quantum system is part of the algorithm and may account for a crucial part of the complexity.[1] Theoretical frameworks, software and hardware that address the interface between the classical memory and the quantum device are therefore central for technological implementations of quantum machine learning. Issues of efficiency, precision and noise play an important role in performance evaluation. This is even more true since most quantum machine learning algorithms deliver probabilistic results and the entire routine—including state preparation—may have to be repeated many times. These arguments call for a thorough discussion of "data encoding" approaches and routines, which is why we dedicate this entire chapter to questions of data representation with quantum states. We will systematically go through the four encoding methods distinguished in Sect. 3.4 and discuss state preparation routines and the consequences for algorithm design.

[1] This is not only true for quantum machine learning algorithms. For example, the classically hard *graph isomorphism* problem is efficiently solvable on a quantum computer if a superposition of isomorph graphs can be created efficiently [1].

M. Schuld and F. Petruccione, *Supervised Learning with Quantum Computers*,
Quantum Science and Technology, https://doi.org/10.1007/978-3-319-96424-9_5

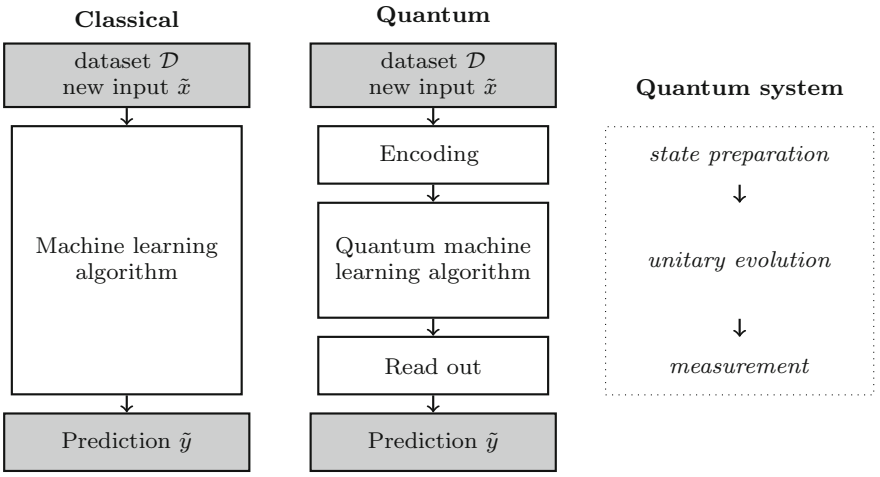


Fig. 5.1 In order to solve supervised machine learning tasks based on classical datasets, the quantum algorithm requires an information encoding and read out step that are in general highly non-trivial procedures, and it is important to consider them in the runtime. Adapted from [2]

Table 5.1 Comparison of the four encoding strategies for a dataset of M inputs with N features each. While basis, amplitude and Hamiltonian encoding aim at representing a full data set by the quantum system, qsample encoding works a little different in that it represents a probability distribution over random variables. It therefore does not have a dependency on the number of inputs M. **Only certain datasets or models can be encoded in this time. See text for details.*

Encoding	Number of qubits	Runtime of state prep	Input features
Basis	N	$\mathcal{O}(MN)$	Binary
Amplitude	$\log N$	$\mathcal{O}(MN)/\mathcal{O}(\log(MN))$*	Continuous
Qsample	N	$\mathcal{O}(2^N)/\mathcal{O}(N)$*	Binary
Hamiltonian	$\log N$	$\mathcal{O}(MN)/\mathcal{O}(\log(MN))$*	Continuous

A central figure of merit for state preparation is the asymptotic runtime, and an overview of runtimes for the four encoding methods is provided in Table 5.1. In machine learning, the input of the algorithm is the data, and an efficient algorithm is efficient in the dimension of the data inputs N and the number of data points M. In quantum computing an efficient algorithm has a polynomial runtime with respect to the number of qubits. Since data can be encoded into qubits or amplitudes, the expression "efficient" can have different meanings in quantum machine learning, and this easily gets confusing. To facilitate the discussion, we will use the terms introduced in Sect. 4.1 and call an algorithm either *amplitude-efficient* or *qubit-efficient*, depending on what we consider as an input. It is obvious that if the data is encoded

into the amplitudes or operators of a quantum system (as in amplitude and Hamiltonian encoding), amplitude-efficient state preparation routines are also efficient in terms of the data set. If we encode data into qubits, qubit-efficient state preparation is efficient in the data set size. We will see that there are some very interesting cases in which we encode data into amplitudes or Hamiltonians, but can guarantee qubit-efficient state preparation routines. In these cases, we prepare data in time which is logarithmic in the data size itself. Of course, this requires either a very specific access to or a very special structure of the data.

Before we start, a safe assumption when nothing more about the hardware is known is that we have a n-qubit system in the ground state $|0...0\rangle$ and that the data is accessible from a classical memory. In some cases we will also require some specific classical preprocessing. We consider data sets $\mathcal{D} = \{x^1, ..., x^M\}$ of N-dimensional real feature vectors. Note that many algorithms require the labels to be encoded in qubits entangled with the inputs, but for the sake of simplicity we will focus on unlabelled data in this chapter.

5.1 Basis Encoding

Assume we are given a binary dataset $\mathcal{D}$ where each pattern $x^m \in \mathcal{D}$ is a binary string of the form $x^m = (b_1^m, ..., b_N^m)$ with $b_i^m \in \{0, 1\}$ for $i = 1, ..., N$. We can prepare a superposition of basis states $|x^m\rangle$ that qubit-wise correspond to the binary input patterns,

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} |x^m\rangle. \tag{5.1}$$

For example, given two binary inputs $x^1 = (01,\ 01)^T, x^2 = (11,\ 10)^T$, where features are encoded with a binary precision of $\tau = 2$, we can write them as binary patterns $x^1 = (0110), x^2 = (1110)$. These patterns can be associated with basis states $|0110\rangle, |1110\rangle$, and the full data superposition reads

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{2}}|0101\rangle + \frac{1}{\sqrt{2}}|1110\rangle. \tag{5.2}$$

The amplitude vector corresponding to State (5.1) has entries $\frac{1}{\sqrt{M}}$ for basis states that are associated with a binary pattern from the dataset, and zero entries otherwise. For Eq. (5.2), the amplitude vector is given by

$$\alpha = (0, 0, 0, 0, 0, \frac{1}{\sqrt{2}}, 0, 0, 0, 0, 0, 0, 0, 0, \frac{1}{\sqrt{2}}, 0)^T$$

Since—except in very low dimensions—the total number of amplitudes $2^{N\tau}$ is much larger than the number of nonzero amplitudes M, basis encoded datasets generally give rise to sparse amplitude vectors.

5.1.1 Preparing Superpositions of Inputs

An elegant way to construct such 'data superpositions' in time linear in M and N has been introduced by Ventura, Martinez and others [3, 4], and will be summarised here as an example of basis encoded state preparation. The circuit for one step in the routine is shown in Fig. 5.2. We will simplify things by considering binary inputs in which every bit represents one feature, or $\tau = 1$.

We require a quantum system

$$|l_1, ..., l_N;\ a_1, a_2;\ s_1, ..., s_N\rangle$$

with three registers: a *loading register* of N qubits $|l_1, ..., l_N\rangle$, the *ancilla register* $|a_1, a_2\rangle$ with two qubits and the N-qubit *storage register* $|s_1, ..., s_N\rangle$. We start in the ground state and apply a Hadamard to the second ancilla to get

$$\frac{1}{\sqrt{2}}|0, ..., 0;\ 0, 0;\ 0, ..., 0\rangle + \frac{1}{\sqrt{2}}|0, ..., 0;\ 0, 1;\ 0, ..., 0\rangle.$$

The left term, flagged with $a_2 = 0$, is called the *memory branch*, while the right term, flagged with $a_2 = 1$, is the *processing branch*. The algorithm iteratively loads patterns into the loading register and 'breaks away' the right size of terms from the processing

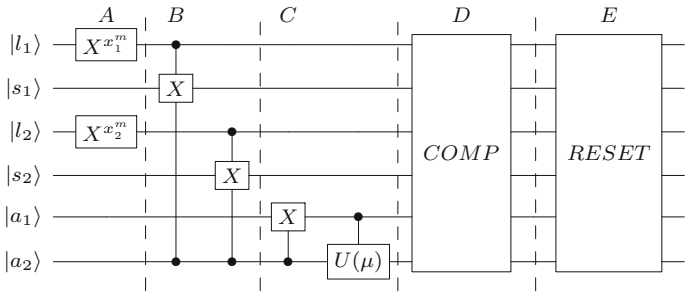


Fig. 5.2 Circuit for one step of Ventura and Martinez state preparation routine as described in the text for an example of 2-bit input patterns. The storage and loading qubits are in a slightly different order to facilitate the display. (A) The pattern is written into the loading register by NOT gates. Taking the gate to the power of the bit value x_i^m is a convenient way to apply the flip only when $x_i^m = 1$. (B) Transfer the pattern to the storage register of the processing branch. (C) Split the processing branch. (D) Flip the first ancilla back conditioned on a successful comparison of loading and storage branch. (E) Reset the registers to prepare for the next patterns

branch to add it to the memory branch (Fig. 5.3). This way the superposition of patterns is 'grown' step by step.

To explain how one iteration works, assume that the first m training vectors have already been encoded after iterations $1, ..., m$ of the algorithm. This leads to the state

$$|\psi^{(m)}\rangle = \frac{1}{\sqrt{M}} \sum_{k=1}^{m} |0, ..., 0; 00; x_1^k, ..., x_N^k\rangle + \sqrt{\frac{M-m}{M}} |0, ..., 0; 01; 0, ..., 0\rangle. \tag{5.3}$$

The memory branch stores the first m inputs in its storage register, while the storage register of the processing branch is in the ground state. In both branches the loading register is also in the ground state.

To execute the $(m+1)$th step of the algorithm, write the $(m+1)$th pattern $x^{m+1} = (x_1^{m+1}, ..., x_N^{m+1})$ into the qubits of the loading register (which will write it into both branches). This can be done by applying an X gate to all qubits that correspond to nonzero bits in the input pattern. Next, in the processing branch the pattern gets copied into the storage register using a CNOT gate on each of the N qubits. To limit the execution to the processing branch only we have to control the CNOTs with the second ancilla being in $a_2 = 1$. This leads to

$$\frac{1}{\sqrt{M}} \sum_{k=1}^{m} |x_1^{m+1}, ..., x_N^{m+1}; 00; x_1^k, ..., x_N^k\rangle + \sqrt{\frac{M-m}{M}} |x_1^{m+1}, ..., x_N^{m+1}; 01; x_1^{m+1}, ..., x_N^{m+1}\rangle.$$

Using a CNOT gate, we flip $a_1 = 1$ if $a_2 = 1$, which is only true for the processing branch. Afterwards apply the single qubit unitary

$$U_{a_2}(\mu) = \begin{pmatrix} \sqrt{\frac{\mu-1}{\mu}} & \frac{1}{\sqrt{\mu}} \\ \frac{-1}{\sqrt{\mu}} & \sqrt{\frac{\mu-1}{\mu}} \end{pmatrix}$$

with $\mu = M + 1 - (m+1)$ to qubit a_2 but controlled by a_1. On the full quantum state, this operation amounts to

$$\mathbb{1}_{\text{loading}} \otimes c_{a_1} U_{a_2}(\mu) \otimes \mathbb{1}_{\text{storage}}.$$

This splits the processing branch into two subbranches, one that can be "added" to the memory branch and one that will remain the processing branch for the next step,

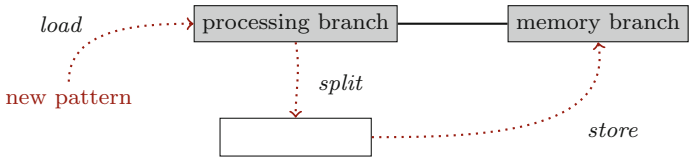


Fig. 5.3 Illustration of Ventura and Martinez state preparation routine [3]. The superposition is divided into a processing and memory term, flagged by an ancilla. New training input patterns get successively loaded into the processing branch, which gets 'split' by a Hadamard on an ancilla, and the pattern gets 'merged' into the memory term

$$\frac{1}{\sqrt{M}}\sum_{k=1}^{m}|x_1^{m+1},...,x_N^{m+1};00;x_1^k,...,x_N^k\rangle + \frac{1}{\sqrt{M}}|x_1^{m+1},...,x_N^{m+1};10;x_1^{m+1},...,x_N^{m+1}\rangle + \frac{\sqrt{M-(m+1)}}{\sqrt{M}}|x_1^{m+1},...,x_N^{m+1};11;x_1^{m+1},...,x_N^{m+1}\rangle$$

To add the subbranch marked by $|a_1a_2\rangle = |10\rangle$ to the memory branch, we have to flip a_1 back to 1. To confine this operation to the desired subbranch we can condition it on an operation that compares if the loading and storage register are in the same state (which is only true for the two processing subbranches), and that $a_2 = 1$ (which is only true for the desired subbranch). Also, the storage register of the processing branch as well as the loading register of both branches has to be reset to the ground state by reversing the previous operations, before the next iteration begins. After the $(m+1)$th iteration we start with a state similar to Eq. (5.3) but with $m \to m+1$. The routine requires $\mathcal{O}(MN)$ steps, and succeeds with certainty.

There are interesting alternative proposals for architectures of *quantum Random Access Memories*, devices that load patterns 'in parallel' into a quantum register. These devices are designed to query an index register $|m\rangle$ and load the mth binary pattern into a second register in basis encoding, $|m\rangle|0...0\rangle \to |m\rangle|x^m\rangle$. Most importantly, this operation can be executed in parallel. Given a superposition of the index register, the quantum random access memory is supposed to implement the operation

$$\frac{1}{\sqrt{M}}\sum_{m=0}^{M-1}|m\rangle|0...0\rangle \to \frac{1}{\sqrt{M}}\sum_{m=0}^{M-1}|m\rangle|x^m\rangle. \tag{5.4}$$

We will get back to this in the next section, where another step allows us to prepare amplitude encoded quantum states. Ideas for architectures which realise this kind of

'query access' in logarithmic time regarding the number of items to be loaded have been brought forward [5], but a hardware realising such an operation is still an open challenge [6, 7].

5.1.2 Computing in Basis Encoding

Acting on binary features encoded as qubits gives us the most computational freedom to design quantum algorithms. In principle, each operation on bits that we can execute on a classical computer can be done on a quantum computer as well. The argument is roughly the following: A Toffoli gate implements the logic operation

Input	Output
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 **0**	1 1 **1**
1 1 **1**	1 1 **0**

and is a universal gate for Boolean logic [8]. Universality implies that *any* binary function $f : \{0, 1\}^{\otimes N} \rightarrow \{0, 1\}^{\otimes D}$ can be implemented by a succession of Toffoli gates and possibly some 'garbage' bits. The special role of the Toffoli gate stems from the fact that it is reversible. If one only sees the state after the operation, one can deduce the exact state before the operation (i.e., if the first two bits are 11, flip the third one). No information is lost in the operation, which is in physical terms a non-dissipative operation. In mathematical terms the matrix representing the operation has an inverse. Reversible gates, and hence also the Toffoli gate, can be implemented on a quantum computer. In conclusion, if any classical algorithm can efficiently be formulated in terms of Toffoli gates, and these can always be implemented on a quantum computer, this means that any classical algorithm can efficiently be translated to a quantum algorithm. The reformulation of a classical algorithm with Toffoli gates may however have a large polynomial overhead and slow down the routines significantly.

Note that once encoded into a superposition, the data inputs can be processed in quantum parallel (see Sect. 3.2.4). For example, if a routine

$$\mathcal{A}(|x\rangle \otimes |0\rangle) \rightarrow |x\rangle \otimes |f(x)\rangle$$

is known to implement a machine learning model f and write the binary prediction $f(x)$ into the state of a qubit, we can perform inference in parallel on the data superposition,

$$\mathcal{A}\left(\frac{1}{\sqrt{M}}\sum_{m=1}^{M}|x^m\rangle\otimes|0\rangle\right)\rightarrow\frac{1}{\sqrt{M}}\sum_{m=1}^{M}|x^m\rangle\otimes|f(x^m)\rangle.$$

From this superposition one can extract statistical information, for example by measuring the last qubit. Such a measurement reveals the expectation value of the prediction of the entire dataset.

5.1.3 *Sampling from a Qubit*

As the previous example shows, the result of a quantum machine learning model can be basis encoded as well. For binary classification this only requires one qubit. If the qubit is in state $|f(x)\rangle = |1\rangle$ the prediction is 1 and if the qubit is in state $|f(x)\rangle = |0\rangle$ the prediction is 0. A superposition can be interpreted as a probabilistic output that provides information on the uncertainty of the result.

In order to read out the state of the qubit we have to measure it, and we want to briefly address how to obtain the prediction estimate from measurements, as well as what number of measurements are needed for a reliable prediction. The field of reconstructing a quantum state from measurements is called *quantum tomography*, and there are very sophisticated ways in which samples from these measurements can be used to estimate the density matrix that describe the state. Here we consider a much simpler problem, namely to estimate the probability of measuring basis state $|0\rangle$ or $|1\rangle$. In other words, we are just interested in the diagonal elements of the density matrix, not in the entire state. Estimating the ouptut of a quantum model in basis encoding is therefore a 'classical' rather than a 'quantum task'.

Let the final state of the output qubit be given by the density matrix

$$\rho = \begin{pmatrix}\rho_{00} & \rho_{01}\\ \rho_{10} & \rho_{11}\end{pmatrix}.$$

We need the density matrix formalism from Chap. 3 here because the quantum computer may be in a state where other qubits are entangled with the output qubit, and the single-qubit-state is therefore a mixed state. We assume that the quantum algorithm is error-free, so that repeating it always leads to precisely the same density matrix ρ to take measurements from. The diagonal elements ρ_{00} and ρ_{11} fulfil $\rho_{00} + \rho_{11} = 1$ and give us the probability of measuring the qubit in state $|0\rangle$ or $|1\rangle$ respectively. We associate ρ_{11} with the probabilistic output $f(x) = p$ which gives us the probability that model f predicts $y = 1$ for the input x.

To get an estimate $\hat{p}$ of p we have to repeat the entire algorithm S times and perform a computational basis measurement on the output qubit in each run. This produces a set of samples $\Omega = \{y_1, ..., y_S\}$ of outcomes, and we assume the samples stem from a distribution that returns 0 with probability $1 - p$ and 1 with probability p. Measuring a single qubit is therefore equivalent to sampling from a Bernoulli distribution, a problem widely investigated in statistics.[2] There are various strategies of how to get an estimate $\hat{p}$ from samples Ω. Most prominent, maximum likelihood estimation leads to the rather intuitive 'frequentist estimator' $\hat{p} = \bar{p} = \frac{1}{S}\sum_{i=1}^{S} y_i$ which is nothing else than the average over all outcomes.

An important question is how many samples from the single qubit measurement we need to estimate p with error ϵ. In physics language, we want an 'error bar' ϵ of our estimation $\hat{p} \pm \epsilon$, or the *confidence interval* $[\hat{p} - \epsilon, \hat{p} + \epsilon]$. A confidence interval is valid for a pre-defined confidence level, for example of 99%. The confidence level has the following meaning: If we have different sets of samples $\mathcal{S}_1, ..., \mathcal{S}_S$ and compute estimators and confidence intervals for each of them, $\hat{p}_{\mathcal{S}_1} \pm \epsilon_{\mathcal{S}_1}, ..., \hat{p}_{\mathcal{S}_S} \pm \epsilon_{\mathcal{S}_S}$, the confidence level is the proportion of sample sets for which the true value p lies within the confidence interval around $\hat{p}$. The confidence level is usually expressed by a so called z-value, for example, a z-value of 2.58 corresponds to a confidence of 99%. This correspondence can be looked up in tables.

Frequently used is the Wald interval which is suited for cases of large S and $p \approx 0.5$. The error ϵ can be calculated as

$$\epsilon = z\sqrt{\frac{\hat{p}(1-\hat{p})}{S}}.$$

This is maximised for $p = 0.5$, so that we can assume the overall error of our estimation ϵ to be at most

$$\epsilon \leq \frac{z}{2\sqrt{S}}$$

with a confidence level of z. In other words, for a given ϵ and z we need $\mathcal{O}(\epsilon^{-2})$ samples of the qubit measurement. If we want to have an error bar of at most $\epsilon = 0.1$ and a confidence level of 99% we need about 167 samples, and an error of $\epsilon = 0.01$ with confidence 99% requires at most 17, 000 samples. This is a vast number, but only needed if the estimator is equal to 0.5, which is the worst case scenario of an undecided classifier (and we may not want to rely on the decision very much in any case). One can also see that the bound fails for $p \to 0, 1$ [9].

There are other estimates that also work when p is close to either zero or one. A more refined alternative is the Wilson score interval [10] with the following estimator for p,

$$\hat{p} = \frac{1}{1 + \frac{z^2}{S}}\left(\bar{p} + \frac{z^2}{2S}\right),$$

[2] Bernoulli sampling is equivalent to a (biased) coin toss experiment: We flip a coin S times and want to estimate the bias p, i.e. with what probability the coin produces 'heads'.

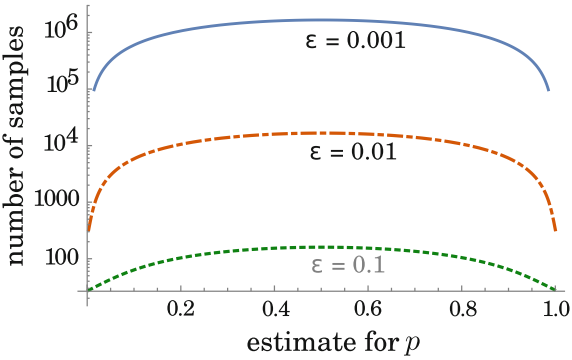


Fig. 5.4 Relationship between the sample size S and the mean value $\bar{p} = \frac{1}{S}\sum_{i=1}^{S} y_i$ for different errors ϵ for the Wilson score interval of a Bernoulli parameter estimation problem as described in the text

and the error

$$\epsilon = \frac{z}{1+\frac{z^2}{S}}\left(\frac{\bar{p}(1-\bar{p})}{S} + \frac{z^2}{4S^2}\right)^{\frac{1}{2}},$$

with $\bar{p}$ being the average of all samples as defined above. Again this is maximised for $\bar{p} = 0.5$ and with a confidence level z we can state that the overall error of our estimation is bounded by

$$\epsilon \leq \sqrt{z^2\frac{S+z^2}{4S^2}}.$$

This can be solved for S as

$$S \leq \frac{\epsilon^2\sqrt{\frac{z^4(16\epsilon^2+1)}{\epsilon^4}} + z^2}{8\epsilon^2}.$$

With the Wilson score, a confidence level of 99% suggests that we need 173 single qubit measurements to guarantee an error of less than 0.1. However, now we can test the cases $\bar{p} = 0, 1$ for which $S = z^2(\frac{1}{2\epsilon} - 1)$. For $\epsilon = 0.1$ we only need about 27 measurements for the same confidence level (see Fig. 5.4).

5.2 Amplitude Encoding

An entire branch of quantum machine learning algorithms encode the dataset into the amplitudes of a quantum state

$$|\psi_{\mathcal{D}}\rangle = \sum_{m=1}^{M}\sum_{i=1}^{N} x_i^m |i\rangle|m\rangle$$
$$= \sum_{m=1}^{M} |\psi_{x^m}\rangle|m\rangle. \tag{5.5}$$

This quantum state has an amplitude vector of dimension NM that is constructed by concatenating all training inputs, $\alpha = (x_1^1, ..., x_N^1, ..., x_1^M, ..., x_N^M)^T$. As discussed in Sect. 3.4.2, the dataset has to be normalised so that the absolute square of the amplitude vector is one, $|\alpha|^2 = 1$. The training outputs can either be basis encoded in an extra qubit $|y^m\rangle$ entangled with the $|m\rangle$ register, or encoded in the amplitudes of a separate quantum register,

$$|\psi_y\rangle = \sum_{m=1}^{M} y^m |m\rangle.$$

Amplitude encoding of datasets therefore requires the ability to prepare an arbitrary state

$$|\psi\rangle = \sum_i \alpha_i |i\rangle, \tag{5.6}$$

both efficiently and robustly.

The main advantage of amplitude encoding is that we only need $n = \log NM$ qubits to encode a dataset of M inputs with N features each. If the quantum machine learning algorithm is polynomial in n (or qubit-efficient), it has a logarithmic runtime dependency on the data set size. Promises of exponential speedups from qubit-efficient quantum machine learning algorithms sound strange to machine learning practitioners, because simply loading the NM features from the memory hardware takes time that is of course linear in NM. And indeed, the promises only hold if state preparation can also be done qubit-efficiently [11]. This is in fact possible in some cases that exhibit a lot of structure. As a trivial example, if the goal is to produce a vector of uniform entries we simply need to apply n Hadamard gates - a qubit-efficient state preparation time. Similarly, s-sparse vectors can be prepared with the routines from the previous section in time sn. On the other hand, there are subspaces in a Hilbert space that cannot be reached from a given initial state with qubit-efficient algorithms [12] (see Fig. 5.5). It is therefore an important and nontrivial open question which classes of relevant states for machine learning with amplitude encoding can be prepared qubit-efficiently.

Similar caution is necessary for the readout of all amplitudes. If the result of the computation is encoded in one specific amplitude α_i only, the number of measurements needed to retrieve it scale with the number of amplitudes (and to measure the

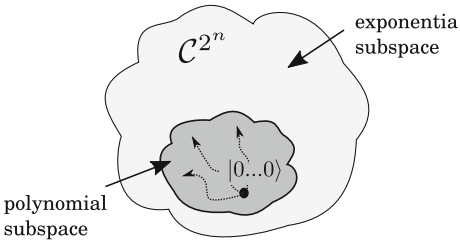


Fig. 5.5 Starting from the n-qubit ground state $|0...0\rangle$ some quantum states in $\mathbb{C}^{2^n}$ can be reached by a quantum algorithm that grows polynomially with n, while others require algorithms that grow exponentially with n. In amplitude encoding the number of features is 2^n, and state preparation routines that are polynomial in the number of qubits are therefore logarithmic in the number of features

state corresponding to that amplitude once requires on average $\mathcal{O}(1/|\alpha_i|^2)$ repetitions of the entire routine). This is why the result of quantum machine learning algorithms based on amplitude encoding is often designed in a different manner, for example through a single-qubit measurement as explained in the previous section,.

How can we prepare an arbitrary vector like in Eq. (5.6)? We will first discuss a scheme that is amplitude-efficient (and thereby efficient in N,M) and then look at qubit-efficient schemes and their limitations.

5.2.1 State Preparation in Linear Time

Given an n qubit quantum computer, the theoretical lower bound of the depth of an arbitrary state preparation circuit is known to be $\frac{1}{n}2^n$ [13–17]. Current algorithms perform slightly worse with slighly less than 2^n parallel operations, most of which are expensive 2-qubit gates. To illustrate one way of doing state preparation in linear time, let us have a look at the routine presented by Möttönen et al. [18]. They consider the reverse problem, namely to map an arbitrary state $|\psi\rangle$ to the ground state $|0...0\rangle$. In order to use this algorithm for our purpose we simply need to invert each and every operation and apply them in reverse order.

The basic idea is to control a rotation on qubit q_s by all possible states of the previous qubits q_1, ..., q_{s-1}, using sequences of so called *multi-controlled rotations*. In other words, we explicitly do a different rotation for each possible branch of the superposition, the one in which the previous qubits were in state $|0..0\rangle$ to the branch in which they are in $|1...1\rangle$. This is comparable to tossing $s-1$ coins and manipulating the sth coin differently depending on the measurement outcome via a look-up table.

A full sequence of multi-controlled rotations around vectors v^i with angles β_i consists of the successive application of the 2^{s-1} gates

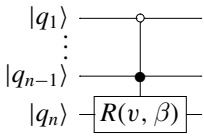
$$
\begin{aligned}
c_{q_1=0}\cdots c_{q_{s-1}=0}\; R_{q_s}(v^1,\beta_1) \quad & |q_1 \ldots q_{s-1}\rangle|q_s\rangle \\
c_{q_1=0}\cdots c_{q_{s-1}=1}\; R_{q_s}(v^2,\beta_2) \quad & |q_1 \ldots q_{s-1}\rangle|q_s\rangle \\
& \vdots \\
c_{q_1=1}\cdots c_{q_{s-1}=1}\; R_{q_s}(v^{2^{s-1}},\beta_{2^{s-1}}) \quad & |q_1 \ldots q_{s-1}\rangle|q_s\rangle .
\end{aligned}
$$

For example, for $s = 3$ a full sequence consists of the gates

$$
\begin{aligned}
c_{q_1=0}c_{q_2=0}\; R_{q_3}(v^1,\beta_1) \quad & |q_1 q_2\rangle|q_3\rangle \\
c_{q_1=0}c_{q_2=1}\; R_{q_3}(v^2,\beta_2) \quad & |q_1 q_2\rangle|q_3\rangle \\
c_{q_1=1}c_{q_2=0}\; R_{q_3}(v^3,\beta_3) \quad & |q_1 q_2\rangle|q_3\rangle \\
c_{q_1=1}c_{q_2=1}\; R_{q_3}(v^4,\beta_4) \quad & |q_1 q_2\rangle|q_3\rangle ,
\end{aligned}
$$

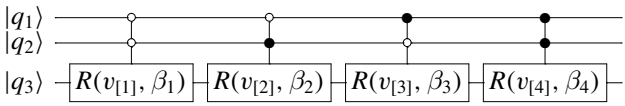
which rotate q_3 in a different way for all four branches of the superposition of q_1, q_2.

The circuit symbol of a single multi-controlled rotation is



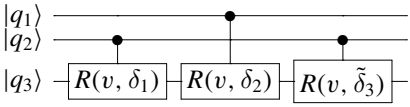
(5.7)

where white circles indicate a control on qubit q being in state 1, or $c_{q=1}$, and black circles a control on qubit q being in state 0, $c_{q=0}$. The circuit diagram of a full sequence of multi-controlled rotations on the third of three qubits is



(5.8)

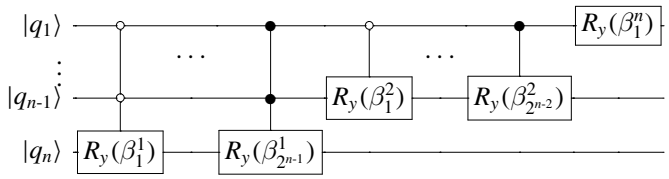
Of course, we need a prescription to decompose sequences of multi-controlled rotations into elementary gates. If there are $s-1$ control qubits, this is possible with 2^s CNOTs and 2^s single qubit gates [14, 18]. For example, a multi-controlled rotation applied to the third of three qubits would have the decomposition



into three single-controlled rotations.

For a general quantum state one has to apply two *cascades* of such operations, where each cascade is a sequences of multi-controlled rotations that run trough all

qubits q_1 to q_n. The first cascade uses R_z-rotations (which fixes the rotation axis v) and has the effect of 'equalising' the phases until the state only has one global phase which we can ignore (remember, we are doing reverse state preparation). The second cascade applies R_y rotations and has the effect of 'equalising' all amplitudes to result in the ground state. Since we are usually interested in real amplitude vectors, we can neglect the first cascade and end up with the circuit



The choice of the rotation angles β is related to the amplitudes of the original state as follows:

$$\beta_j^s = 2 \arcsin \left(\frac{\sqrt{\sum_{l=1}^{2^{s-1}} |\alpha_{(2j-1)2^{s-1}+l}|^2}}{\sqrt{\sum_{l=1}^{2^s} |\alpha_{(j-1)2^s+l}|^2}} \right) \tag{5.9}$$

Example 5.1 We want to prepare the state

$$|\psi\rangle = \sqrt{0.2}|000\rangle + \sqrt{0.5}|010\rangle + \sqrt{0.2}|110\rangle + \sqrt{0.1}|111\rangle,$$

with the amplitudes $a_0 = \sqrt{0.2}, a_2 = \sqrt{0.5}, a_6 = \sqrt{0.2}, a_7 = \sqrt{0.1}$. The circuit requires seven multi-controlled y-rotations

$$\begin{aligned}
c_{q_1=0}c_{q_2=0}\, R_{y,q_3}(\beta_1^1), &\qquad \beta_1^1 = 0 \\
c_{q_1=0}c_{q_2=1}\, R_{y,q_3}(\beta_2^1), &\qquad \beta_2^1 = 0 \\
c_{q_1=1}c_{q_2=0}\, R_{y,q_3}(\beta_3^1), &\qquad \beta_3^1 = 0 \\
c_{q_1=1}c_{q_2=1}\, R_{y,q_3}(\beta_4^1), &\qquad \beta_4^1 = 1.231.. \\
c_{q_1=0}\, R_{y,q_2}(\beta_1^2), &\qquad \beta_1^2 = 2.014.. \\
c_{q_1=1}\, R_{y,q_2}(\beta_2^2), &\qquad \beta_2^2 = 3.142.. \\
R_{y,q_1}(\beta_1^3), &\qquad \beta_1^3 = 1.159..
\end{aligned}$$

We are left with only four gates to apply, and with the approximate values

$$\begin{aligned}
\sin\left(\frac{1.231}{2}\right) &= 0.577, \quad \cos\left(\frac{1.231}{2}\right) = 0.816 \\
\sin\left(\frac{2.014}{2}\right) &= 0.845, \quad \cos\left(\frac{2.014}{2}\right) = 0.534 \\
\sin\left(\frac{3.142}{2}\right) &= 1.000, \quad \cos\left(\frac{3.142}{2}\right) = 0 \\
\sin\left(\frac{1.159}{2}\right) &= 0.548, \quad \cos\left(\frac{1.159}{2}\right) = 0.837,
\end{aligned}$$

these gates can be written as

$$c_{q_1=1}c_{q_2=1}\ R_{y,q_3}(\beta_4^1) = \begin{pmatrix}
1&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&0\\
0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0.816&0.577\\
0&0&0&0&0&0&-0.577&0.816
\end{pmatrix},$$

$$c_{q_1=0}\ R_{y,q_2}(\beta_1^2) = \begin{pmatrix}
0.534&0&0.845&0&0&0&0&0\\
0&0.534&0&0.845&0&0&0&0\\
-0.845&0&0.534&0&0&0&0&0\\
0&-0.845&0&0.534&0&0&0&0\\
0&0&0&0&1&0&0&0\\
0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&1
\end{pmatrix},$$

$$c_{q_1=1}\ R_{y,q_2}(\beta_2^2) = \begin{pmatrix}
1&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&1\\
0&0&0&0&-1&0&0&0\\
0&0&0&0&0&-1&0&0
\end{pmatrix},$$

and

$$R_{y,q_1}(\beta_1^3) = \begin{pmatrix} 0.837 & 0 & 0 & 0 & 0.548 & 0 & 0 & 0 \\ 0 & 0.837 & 0 & 0 & 0 & 0.548 & 0 & 0 \\ 0 & 0 & 0.837 & 0 & 0 & 0 & 0.548 & 0 \\ 0 & 0 & 0 & 0.837 & 0 & 0 & 0 & 0.548 \\ -0.548 & 0 & 0 & 0 & 0.837 & 0 & 0 & 0 \\ 0 & -0.548 & 0 & 0 & 0 & 0.837 & 0 & 0 \\ 0 & 0 & -0.548 & 0 & 0 & 0 & 0.837 & 0 \\ 0 & 0 & 0 & -0.548 & 0 & 0 & 0 & 0.837 \end{pmatrix}.$$

Applying these gates to the amplitude vector $(\sqrt{0.2}, 0, \sqrt{0.5}, 0, 0, 0, \sqrt{0.2}, \sqrt{0.1})^T$, the effect of the first operation is to set the last entry to zero, the second operation sets the third entry to zero and the third operation swaps the 5th element with the 7th. The last operation finally cancels the 5th element and results in $(1, 0, 0, 0, 0, 0, 0, 0)^T$ as desired. State preparation means to apply the inverse gates in the inverse order.

5.2.2 Qubit-Efficient State Preparation

The above routine is always possible to use for amplitude-efficient state preparation, but requires an exponential number of operations regarding the number of qubits. In the gate model of quantum computing, a number of proposals have been brought forward to prepare specific classes of states qubit-efficiently, or in $\log(MN)$. We want to present some of those ideas and the conditions to which they apply.

5.2.2.1 Parallelism-Based Approach

Grover and Rudolph suggest a scheme that is linear in the number of qubits for the case that we know an efficiently integrable one-dimensional probability distribution $p(a)$ of which the state is a discrete representation [19]. More precisely, the desired state has to be of the form

$$|\psi\rangle = \sum_{i=1}^{2^n} \sqrt{p(i\Delta a)}|i\rangle = \sum_{i=1}^{2^n} \sqrt{p_i}|i\rangle,$$

with $\Delta a = \frac{1}{2^n}$. The desired quantum state is a coarse grained qsample (see Sect. 3.4.3) of a continuous distribution $p(a)$ (see Fig. 5.6). The "efficiently integrable" condition means that we need an algorithm on a classical computer that calculates definite integrals of the probability distribution efficiently, like for the Gaussian distribution.

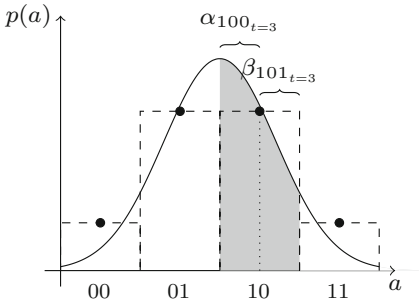


Fig. 5.6 Grover and Rudolph's state preparation algorithm for efficiently integrable probability distributions [19]. After step $t = 2$ the domain is divided into four regions $i = 00, 01, 10, 11$ of size $\Delta x = 1/2^2$, which defines a 2-qubit quantum state with amplitudes $p_{00}, p_{01}, p_{10}, p_{11}$ (indicated by the black dots). In the third step, each of the four regions (here demonstrated for $i = 10$) gets split into two. The parameters α and β are the probability to find a random variable in the left or right part of the region. This procedure successively prepares a finer and finer discretisation of the probability distribution

The idea of the scheme is to successively rotate each of the n qubits and get a finer and finer coarse graining of the distribution. Assume Step $(t-1)$ prepared a qsample

$$|\psi^{(t-1)}\rangle = \sum_{i=1}^{2^{t-1}} \sqrt{p_i^{(t-1)}}|i\rangle|00...0\rangle.$$

The index register $|i\rangle$ contains the first $t-1$ qubits. The next step t rotates the t'th qubit according to

$$|\psi^{(t)}\rangle = \sum_{i=1}^{2^t} \sqrt{p_i^{(t-1)}}|i\rangle(\sqrt{\alpha_i}|0\rangle + \sqrt{\beta_i}|1\rangle)|0...0\rangle,$$

such that α_i, $[\beta_i]$ are the probabilities for a random variable to lie in the left [right] half of the i'th interval of the probability distribution. In the tth step, the input domain is discretised into 2^t equidistant intervals which is visualised in Fig. 5.6. This defines a new qsample

$$|\psi^{(t)}\rangle = \sum_{i=1}^{2^t} \sqrt{p_i^{(t)}}|i\rangle|0...0\rangle,$$

where the index register now has t qubits and the remaining qubits in the ground state are reduced by one. With each step, this process prepares an increasingly fine discretisation of the probability distribution $p(a)$.

The Grover and Rudolph suggestion is in fact rather similar to the state preparation routine in the previous section, but this time we do not have to hardcode the 'lookup-table' for each state of the first $t-1$ qubits, but we can use quantum parallelism to compute the values α_i, β_i for all possible combinations. This more general version of the idea was proposed by Kaye and Mosca [20], who do not refer to probability distributions but demand that the conditional probability $p(q_k = 1|q_1...q_{k-1})$, which is the chance that given the state $q_1...q_{k-1}$ of the previous qubits, the k'th qubit is in state 1, is easy to compute. It is an interesting task to find classical datasets which allow for this trick.

5.2.2.2 Oracle-Based Approach

Soklakov and Schack [21] propose an alternative qubit-efficient scheme to approximately prepare quantum states whose amplitudes $\alpha_i = \sqrt{p_i}$ represent a discrete probability distribution p_i, $i = 1, ..., 2^n = N$, and all probabilities p_i are of the order of $1/\eta N$ for $0 < \eta < 1$. With this condition, they can use a Grover-type algorithm which with probability greater than $1-\nu$ has an error smaller than ϵ in the result, and which is polynomial in η^{-1}, ϵ^{-1}, ν^{-1}. To sketch the basic idea (Fig. 5.7), a series of oracles is defined which successively marks basis states whose amplitudes are increased by amplitude amplification. The first oracle only marks basis states $|i\rangle$ for which the desired probability p_i is larger than a rather high threshold. With each step (defining a new oracle) the threshold is lowered by a constant amount. In the end, large amplitudes have been grown in more steps than small ones, and the final distribution is as fine as the number of steps allows. One pitfall is that we need to know the optimal number of Grover iterations for a given oracle. For this quantum counting can be applied to estimate the number of marked states in the current step. As with all oracle-based algorithms, the algorithm requires an implementation of the oracle to be used in practice.

5.2.2.3 Quantum Random Access Memory

Under the condition that the states to prepare are sufficiently uniform, a more generic approach is to refer to the quantum random access memory introduced in Eq. (5.4). The quantum random access memory allows access of classically stored information in superposition by querying an index register and is referred to by many authors in quantum machine learning [22–25]. In Sect. 5.1.1 this was used to prepare a dataset in basis encoding, but with one additional step we can extend its application to amplitude encoding as well. This step involves a conditional rotation and *branch selection* procedure that we already encountered in Step 3 of the quantum matrix inversion routine of Sect. 3.5.3. Let us assume the quantum random access memory prepared a state

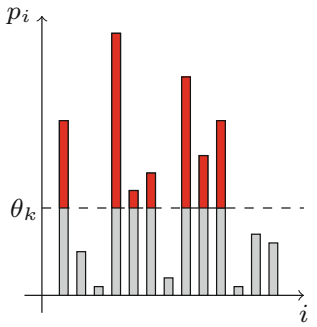


Fig. 5.7 In the k'th step of the routine of Soklakov and Schack [21], an oracle is applied to mark the states whose probabilities p_i are larger than a certain threshold θ_k (here in red). These states are amplified with the Grover iterator, resulting in a state that looks qualitatively like the red bars only. In the next step, the threshold is lowered to include the states with a slightly smaller probability in amplitude amplification, until the desired distribution of the p_i is prepared

$$\frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle |x_i\rangle |0\rangle,$$

where $x_i \leq 1$ are the entries of a classical real vector we want to amplitude encode. Now rotate the ancilla qubit conditional on the $|x_i\rangle$ register to get

$$\frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle |x_i\rangle (\sqrt{1 - |x_i|^2}|0\rangle + x_i|1\rangle). \tag{5.10}$$

The details of this step depend on how x_i is encoded into the register. If we assume binary fraction encoding, such a conditional rotation could be implemented by τ single-qubit conditional rotations on the ancilla that are controlled by the $q_1 ... q_\tau$ qubits of the second register. For qubit q_k the rotation would turn the amplitude closer to $|1\rangle$ by a value of $\frac{1}{2^k}$.

Now the ancilla has to be measured to 'select the desired branch' of the superposition (see Fig. 5.8). If the measurement results in $|1\rangle$ we know that the resulting state is in

$$\frac{1}{\sqrt{N p_{\text{acc}}}} \sum_{i=1}^{N} x_i |i\rangle |x_i\rangle |1\rangle,$$

where p_{acc} is the success or acceptance probability that renormalises the state. Discarding the last register and uncomputing the last but one (which is possible because the superposition is 'kept up' by the index register and no unwanted interference happens), we get a state that amplitude encodes a vector $x = (x_1, ..., x_N)^T$. Of course, one could also start with a non-uniform superposition and would get a product of the

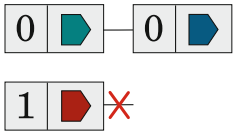


Fig. 5.8 Schematic illustration of the postselective measurement of the branch selection procedure. The two states 0 and 1 of the ancilla qubit are entangled with two different states marked in red and blue. A post-selective measurement selects only one branch of the superposition (here the 0-branch). The state entangled with this branch is formally renormalised

initial and new amplitude. If the measurement results in $|0\rangle$, we have to repeat the entire routine.

The routine for arbitrary state preparation with a quantum random access memory succeeds only with a probability of $p_{\text{acc}} = \sum_i |x_i|^2/N$ which obviously depends on the state to encode. A uniform distribution with $x_i = 1$ ensures that $p_{\text{acc}} = \sum_i \frac{1}{N} = 1$, and in the extreme case of only one non-zero amplitude of 1 we get $p_{\text{acc}} = \frac{1}{N}$, which is exponentially small in the number of qubits n. In other words, we would have to repeat the measurement $\mathcal{O}(N = 2^n)$ times on average to prepare the correct state. Luckily, very sparse states can be prepared by other methods. For example, for a one-sparse vector one can simply flip the qubit register from the ground state into a basis state representing the index i. Zhao et al. [26] therefore propose that in case of s sparse vectors, one does not apply the quantum random access memory to a uniform superposition, but a superposition

$$\frac{1}{\sqrt{s}} \sum_{i|x_i \neq 0} |i\rangle$$

of the basis states representing indices of non-zero amplitudes (Fig. 5.7).

There are many other possible ways to prepare quantum states beyond quantum circuits. An interesting perspective is offered by Aharanov et al. [1]. They present the framework of "adiabatic state generation" as a natural (and polynomially equivalent) alternative to state preparation in the gate model. The idea is to perform an adiabatic evolution of a quantum system that is initially in the ground state of a generic Hamiltonian, to the ground state of a final Hamiltonian. If the evolution is performed slow enough, we end up with the system in the ground state of the final Hamiltonian, which is the state we wish to prepare. This translates the question of which initial states can be easily prepared to the question of which ground states of Hamiltonians are in reach of adiabatic schemes, i.e. have small spectral gaps between the ground and the first excited state. A somewhat related idea is to use the unique stationary states of a dissipative process in an open quantum system.

In summary, quantum state preparation for amplitude encoded states relevant to machine learning algorithms is a topic that still requires a lot of attention, and claims of exponential speedups from qubit-efficient state preparation algorithms should therefore be viewed with a pinch of skepticism.

5.2.3 Computing with Amplitudes

In contrast to basis encoding, amplitude encoding radically limits the type of computations we can perform. As we saw in Chap. 3, there are two general operations for the manipulation of amplitudes in the formalism of quantum theory: unitary transformations and measurements.

As stated in detail in Chap. 3, unitary transformations are linear transformations of the amplitude vector that preserve its length. Even if we consider subsystems (where the evolution is not unitary any more), we still get linear dynamics, for example expressed by the so called Kraus formalism (see [27]). In fact, it has been shown that introducing a 'coherent' nonlinearity to quantum evolution implies the ability to solve NP-hard problems [28]. Essentially this is because we could increase exponentially small signals in a superposition to efficiently measurable information. With this, Grover search could be done exponentially faster. It has also been claimed that any nonlinear map would allow for super-luminal communication [29] and negate the laws of thermodynamics [30]. In short, a nonlinear version of quantum theory can be considered highly unlikely.[3]

A projective measurement is clearly a nonlinear operation, but its outcome is stochastic and if averaging over several runs it will reflect the distribution of amplitudes. But not all is lost: An important way to introduce nonlinearities on the level of amplitudes are post-selective measurements that we used in the branch selection scheme of the previous section. In a post-selective measurement of a single qubit, the state of the qubit is measured in the computational basis and the algorithm is only continued if this qubit is found in a particular state. This condition works like an "if" statement in programming. Post-selection has the effect of setting the amplitudes in a branch of superpositions to zero, namely the branch that is entangled with the qubit in the state of an unsuccessful measurement. Note that in most cases we can push the post-selection to the end of the algorithm and reject final measurement outcomes if the result of said qubit is in the desired state. A second, closely related idea is to use so called *repeat-until-success circuits* [32]. Here the unsuccessful measurement does not result in repeating the entire algorithm, but prescribes a small correction, which restores the state before the postselective measurement. Postselection can therefore be repeated until success is observed. As mentioned before, these procedures are non-unitary and for runtime considerations the likeliness of success has to be taken into account.

5.3 Qsample Encoding

Qsample encoding has been introduced to associate a real amplitude vector $(\sqrt{p_1}, ..., \sqrt{p_N})^T$ with a classical discrete probability distribution $p_1, ..., p_N$. This is in a sense a hybrid case of basis and amplitude encoding, since the information

[3] Note that there are *effective* nonlinear dynamics, see for example [31].

we are interested in is represented by amplitudes, but the N features are encoded in the qubits. An amplitude-efficient quantum algorithm is therefore exponential in the input dimension N, while a qubit-efficient algorithm is polynomial in the input.

For a given probability distribution, state preparation works the same way as in amplitude encoding. In some cases the distribution may be a discretisation of a parametrised continuous and efficiently integrable distribution $p(x)$, in which case the implicit Grover-Rudolph scheme can be applied.

The probabilistic nature of quantum systems lets us manipulate the classical distribution with the quantum system and implement some standard statistic computations generically. We can prepare joint distributions easily, marginalise over qubits 'for free', and we can perform a rejection sampling step via branch selection. These methods are nothing other than an application of basic quantum theory from Sects. 3.1.2.3 and 3.1.3.5, but can be useful when regarded from the perspective of manipulating distributions represented by a qsample.

5.3.1 Joining Distributions

When joining two quantum systems representing a qsample,

$$|\psi_1\rangle = \sum_{i=1}^{2^n} \sqrt{u_i}\ |i\rangle,$$

and

$$|\psi_2\rangle = \sum_{j=1}^{2^n} \sqrt{s_j}\ |j\rangle,$$

the joint state of the total system is described as a tensor product of the two states (see Sect. 3.1.2.3),

$$|\psi_1\rangle \otimes |\psi_2\rangle = \sum_{i,j}^{2^n} \sqrt{u_i s_j}\ |i\rangle|j\rangle.$$

Sampling the binary string ij from this state is observed with probability $u_i s_j$, which is a product of the two original probabilities. In other words, the qsample of two joint qsamples is a product distribution.

5.3.2 Marginalisation

Given a qsample,

$$|\psi\rangle = \sum_{i=1}^{2^n} \sqrt{p_i}\ |i\rangle.$$

The mathematical operation of tracing over a qubit q_k corresponds to omitting the kth qubit from the description, leading to a statistical ensemble over all possible states of that qubit Sect. 3.1.2.3. The resulting state is in general (that is, unless the k'th qubit was unentangled) a mixed state.

To write down the effect of a trace operation is awkward and hides the simplicity of the concept, but we will attempt to do it anyways in order to show the equivalence to marginalisation. Remember that the ith computational basis state $|i\rangle$ represents a binary sequence, more precisely the binary representation of $i-1$ (for example, $|1\rangle$ represents $|0...0\rangle$). Let us define the shorthand $i_{\neg k}$ for the binary sequence indicated by i, but without the kth bit, and let $i_{k=0,1}$ be the same sequence, but this time the kth bit is set to either 0 or 1. Also, let $\langle i|0_k\rangle$ and $\langle i|1_k\rangle$ be the inner product between $|i\rangle$ and a state $\mathbb{1}\otimes|0\rangle\otimes\mathbb{1}$ and $\mathbb{1}\otimes|1\rangle\otimes\mathbb{1}$, respectively, where $|0\rangle, |1\rangle$ sits at the kth position. With this notation, tracing out the kth qubit has the following effect,

$$\begin{aligned}\mathrm{tr}_k\{|\psi\rangle\langle\psi|\} &= \mathrm{tr}_k\left\{\sum_i\sum_{i'}\sqrt{p(i)p(i')}\;|i\rangle\langle i'|\right\}\\ &= \sum_i\sum_{i'}\sqrt{p(i)p(i')}\;\left(\langle 0_k|i\rangle\langle i'|0_k\rangle + \langle 1_k|i\rangle\langle i'|1_k\rangle\right)\\ &= \sum_{i_{\neg k}}\sum_{i'_{\neg k}}\left(\sqrt{p(i_{k=0})p(i'_{k=0})}+\sqrt{p(i_{k=1})p(i'_{k=1})}\right)\;|i_{\neg k}\rangle\langle i'_{\neg k}|.\end{aligned}$$

The probability to measure the computational basis state $|i_{\neg k}\rangle$ is given by the corresponding diagonal element of the resulting density matrix, which is given by

$$\left(\sqrt{p(i_{k=0})p(i_{k=0})}+\sqrt{p(i_{k=1})p(i_{k=1})}\right) = \left(p(i_{k=0})+p(i_{k=1})\right).$$

This is the sum of the probability of finding k in 0 and the probability of finding k in 1, just as we would expect from common sense.

In the classical case, given a probability distribution $p(b_1...b_n)$ over n-bit strings $b_1...b_N$, marginalising over the state of the k'th bit will yield the marginal distribution

$$\begin{aligned}p(i_{\neg k}) &= p(b_1..b_{k-1}b_{k+1}..b_n) &(5.11)\\ &= p(b_1..b_k=0..b_n)+p(b_1..b_k=1..b_n). &(5.12)\end{aligned}$$

This demonstrates that the tracing in the quantum formalism is a 'classical' statistical marginalisation. In other words, dropping the k'th qubit from our algorithm effectively implements a qsample representing a marginal distribution.

5.3.3 Rejection Sampling

There is a close relationship between branch selection or postselection and classical rejection sampling, which can help to understand, design and compare quantum algorithms. Rejection sampling is a method to draw samples from a distribution $p(x)$ that is difficult to sample from, but for which the probabilities at each point x are easy to evaluate. The idea is to use a distribution $q(x)$ from which it is easy to draw samples (i.e., a Gaussian or uniform distribution), and scale it such that $p(x) \leq Kq(x)$ for an integer $K < \infty$ (see Fig. 5.9). One then draws a sample x' from $q(x)$ as well as a random number u' from the interval [0, 1]. One accepts the sample if $u'Kq(x') < p(x')$ and rejects it otherwise. In other words, the chance to accept the sample from $q(x)$ depends on the ratio of the original and the alternative distribution.

Obviously, if $Kq(x')$ and $p(x')$ are almost equal, the acceptance probability is very high, but in regions where $p(x') << Kq(x')$ we will almost never accept. When repeating this procedure, the accepted samples x' are *effectively* drawn from $p(x)$. The general probability of acceptance can be calculated as

$$\begin{aligned} p_{\text{acc}} &= \left\langle p\left(u < \frac{p(x)}{Kq(x)}\right)\right\rangle \\ &= \left\langle \frac{p(x)}{Kq(x)}\right\rangle \\ &= \int \frac{p(x)}{Kq(x)} q(x)dx \\ &= \frac{1}{K}\int p(x)dx = \frac{1}{K}, \end{aligned}$$

which shows that K is an important figure of merit for the runtime [33]. The larger it is, or the bigger the difference between the two distributions, the more samples are rejected.

In branch selection, we have a quantum state of the form

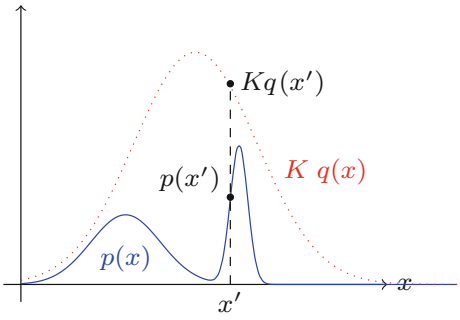


Fig. 5.9 Illustration of rejection sampling. In order to sample from the desired distribution $p(x)$ (smooth line) one samples x' instead from a distribution $K\ q(x)$ (dotted line) and accepts the sample with a probability depending on the proportion of the two distributions at that point

$$\sum_{i=1}^{N} \sqrt{a_i}|i\rangle(\sqrt{1-b_i}|0\rangle + \sqrt{b_i}|1\rangle),$$

where from Eq. (5.10) the uniform coefficients $\frac{1}{\sqrt{N}}$ were replaced by more general amplitudes $\sqrt{a_1}$, ..., $\sqrt{a_N}$, and the value x_i was replaced by $\sqrt{b_i}$ to avoid confusion with the classical notation. For ease of comparison with classical probability theory, let us assume that the a_i and b_i are all real numbers in [0, 1]. By measuring the $|i\rangle$ register we sample from the distribution $\mathcal{A} = \{a_1, ..., a_N\}$, but our goal is to sample from the 'unknown' distribution $\mathcal{B} = \{a_1 b_1, ..., a_N b_N\}$. The conditional measurement on the ancilla therefore plays the role of the rejection step if we associate

$$q(x) \leftrightarrow \mathcal{A} \tag{5.13}$$
$$p(x) \leftrightarrow \mathcal{B}. \tag{5.14}$$

To see this, consider the probability b_i of measuring the ancilla in $|1\rangle$ for state $|i\rangle$ and compare it to the 'rejection sampling' probability of success for sample i. Using the replacement of Eqs. (5.14) we get $p_{\text{acc}}(i) = p(ua_i < a_i b_i) = p(u < b_i) = b_i$. The general probability of acceptance is

$$\begin{aligned} p_{\text{acc}} &= \langle p(u < b_i) \rangle \\ &= \langle b_i \rangle \\ &= \sum_i b_i a_i, \end{aligned}$$

where the expectation value is taken with respect to the distribution $\mathcal{A}$. This is equivalent to the probability of a successful branch selection.

5.4 Hamiltonian Encoding

While the approaches discussed so far encode features explicitly into quantum states, we can choose an implicit encoding which uses the features to define the evolution of the quantum system. This approach differs a bit from the previous ones. Instead of preparing a quantum state which contains the features or a distribution in its mathematical description, they now define an evolution applied to a state.

We mentioned in Sect. 3.4.4 that there are different ways to encode data into the dynamics of a quantum system, and focused on Hamiltonian encoding. Hamiltonian encoding associates the Hamiltonian of a system with a matrix that represents the data, such as the design matrix X containing the feature vectors as rows, or the Gram matrix $X^T X$. As discussed in Sect. 3.4.2, we may have to use preprocessing tricks that make this matrix Hermitian. Section 3.5.3 presented ways in which Hamiltonian encoding allows us to extract eigenvalues of matrices, or to multiply them to an

amplitude vector. We therefore want to summarise some results on the resources needed to evolve a system by a given Hamiltonian. Similar to amplitude encoding, the general case will yield amplitude-efficient state preparation algorithms, while for some limited datasets we can get qubit-efficient schemes. Possibly not surprisingly, this is most prominently the case for sparse data.

To use Hamiltonian encoding we need to be able to implement an evolution

$$|\psi'\rangle = \mathrm{e}^{-iH_A t}|\psi\rangle \tag{5.15}$$

on a quantum computer. The Hamiltonian H_A "encodes" a Hermitian matrix A of the same dimensions, which means that the matrix representation of H_A is entry-wise equivalent to A. The initial quantum state $|\psi\rangle$ describes a system of n qubits, and one can think of the final state $|\psi'\rangle$ as a quantum state that now"contains" the information encoded into the Hamiltonian—for example H's eigenvalues in the phase of the amplitudes.

The process of implementing a given Hamiltonian evolution on a quantum computer is called *Hamiltonian simulation* [34]. Since we only consider time-independent Hamiltonians, any unitary transformation can be written as $\mathrm{e}^{-iH_A t}$. Hamiltonian simulation on a gate-based quantum computer[4] is therefore very closely related to the question of how to decompose a unitary into quantum gates. However, an operator-valued exponential function is not trivial to evaluate, and given H there are more practical ways than computing the unitary matrix $U = \mathrm{e}^{-iH_A t}$ and decomposing it into gates.

The problem of (digital) Hamiltonian simulation can be formulated as follows: Given a Hamiltonian H, a quantum state $|\psi\rangle$, an error $\epsilon > 0$, the evolution time t (which can be imagined as a scaling factor to H), and an appropriate norm $||\cdot||$ that measures the distance between quantum states, find an algorithm which implements the evolution of Eq. (5.15) so that the final state of the algorithm, $|\tilde{\psi}\rangle$, is ϵ-close to the desired final state $|\psi'\rangle$,

$$|| \, |\psi'\rangle - |\tilde{\psi}\rangle|| \leq \epsilon.$$

We want to summarise the basic ideas of Hamiltonian simulation and then state the results for qubit-efficient simulations.

5.4.1 Polynomial Time Hamiltonian Simulation

Consider a Hamiltonian H that can be decomposed into a sum of several "elementary" Hamiltonians $H = \sum_{k=1}^{K} H_k$ so that each H_k is easy to simulate. For non-commuting

[4] Hamiltonian simulation research can be distinguished into *analog* and *digital* approaches to simulation. Roughly speaking, analog simulation finds quantum systems that "naturally" simulate Hamiltonians, while digital simulation decomposes the time evolution into quantum gates, which is more relevant in the context of this book.

H_k, we cannot apply the factorisation rule for scalar exponentials, or

$$\mathrm{e}^{-i\sum_k H_k t} \neq \prod_k \mathrm{e}^{-iH_k t}.$$

An important idea introduced by Seth Lloyd in his seminal paper in 1996 [35] was to use the first-order Suzuki-Trotter formula instead,

$$\mathrm{e}^{-i\sum_k H_k t} = \prod_k \mathrm{e}^{-iH_k t} + \mathcal{O}(t^2). \tag{5.16}$$

The Trotter formula states that for small t the factorisation rule is approximately valid. This can be leveraged when we write the evolution of H for time t as a sequence of small time-steps of length Δt,

$$\mathrm{e}^{-iHt} = (\mathrm{e}^{-iH\Delta t})^{\frac{t}{\Delta t}}.$$

While for the left side, the Trotter formula has a large error, for each small time step Δt, the error in Eq. (5.16) becomes negligible. Of course, there is a trade-off: The smaller Δt, the more often the sequence has to be repeated. But overall, we have a way to simulate H by simulating the terms H_k.

This approach shows that if we know a decomposition of Hamiltonians into sums of elementary Hamiltonians that we know how to simulate, we can approximately evolve the system in the desired way. The case-specific decomposition may still be non-trivial, and some examples are summarised in [34]. One example for such a decomposition is a sum of Pauli operators. Every Hamiltonian can be written as

$$H = \sum_{k_1,\cdots,k_n=1,x,y,z} a_{k_1,\ldots,k_n}(\sigma^1_{k_1} \otimes \cdots \otimes \sigma^n_{k_n}). \tag{5.17}$$

The sum runs over all possible tensor products of Pauli operators applied to the n qubits, and the coefficients

$$a_{k_1,\cdots,k_n} = \frac{1}{2^n}\mathrm{tr}\{(\sigma^1_{k_1} \otimes \cdots \otimes \sigma^n_{k_n})H\},$$

define the entries of the Hamiltonian. For example, for two qubits we can write

$$H_2 = a_{1,1}(\sigma^1_1 \otimes \sigma^2_1) + a_{1,x}(\sigma^1_1 \otimes \sigma^2_x) + \cdots + a_{z,z}(\sigma^1_z \otimes \sigma^2_z).$$

This decomposition has $4^n = 2^n \times 2^n$ terms in general. If the evolution describes a physical problem we can hope that only local interactions—terms in which $\sigma^i = \mathbb{1}$ for all but a few neighbouring qubits—are involved. For machine learning this could also be interesting, when the features are generated by a "local" process and we can hope that correlations in the data reduce the number of terms in the sum of Eq. (5.17).

5.4.2 Qubit-Efficient Simulation of Hamiltonians

There are special classes of Hamiltonians that can be simulated in time that is logarithmic in their dimension. If the Hamiltonian encodes a data matrix, this means that the runtime is also logarithmic in the dimension of the dataset. The most prominent example are Hamiltonians that only act on a constant number of qubits, so called *strictly local Hamiltonians* [35]. This is not surprising when we remember that a local Hamiltonian can be expressed by a constant number of terms constructed from Pauli operators.

More generally, it has been shown that *sparse* Hamiltonians can be simulated qubit-efficiently [1, 36, 37]. An s-sparse Hamiltonian has at most s non-zero elements in each row and column. One usually assumes that the non-zero elements of are given by "oracular" access, which means that for integers $i, l \in 2^n \times [1...s]$, we have access to the lth non-zero element from the ith row. For example, the non-zero elements could be stored in a sparse array representation in a classical memory, which we can use to load them into a quantum register via the oracle call $|i, j, 0\rangle \rightarrow |i, j, H_{ij}\rangle$.

Recent proposals [37–39] manage to reduce the asymptotic runtime of former work considerably. They employ a combination of demanding techniques which go beyond the scope of this book, which is why we only state the result. To recapture, we want to simulate an s-sparse Hamiltonian H for time t and to error ϵ, and H's non-zero elements are accessible by an efficient oracle. Writing $\tau = s||H||_{\max}t$, the number of times we have to query the classical memory for elements of the Hamiltonian grows as

$$\mathcal{O}\left(\tau \frac{\log(\frac{\tau}{\epsilon})}{\log(\log(\frac{\tau}{\epsilon}))}\right)$$

and the number of 2-qubit gates we need grows with

$$\mathcal{O}\left(\tau \left(n + \log^{\frac{5}{2}}(\frac{\tau}{\epsilon})\right) \frac{\log(\frac{\tau}{\epsilon})}{\log(\log(\frac{\tau}{\epsilon}))}\right).$$

The runtime of the algorithm that simulates H is hence roughly linear in s and n, which is the number of qubits. This ensures a poly-logarithmic and therefore logarithmic dependency on the dimension of the matrix A we wish to encode into the Hamiltonian. It was also shown that this runtime is nearly optimal for Hamiltonian simulation of sparse Hamiltonians.

Note that there are other classes of qubit-efficiently simulable Hamiltonians. One class are Hamiltonians where the positions of the nonzero entries define a tree-like graph. For example, if a Hamiltonian has the non-zero elements at positions $(i, j) = \{(1, 2), (1, 3), (3, 6), (3, 7)\}$, we can read each index pair as an edge between nodes, and the graph structure is shown in Fig. 5.10. The proof draws on techniques of *quantum walks* [40]. How applicable such structures are to machine learning applications is an open question.

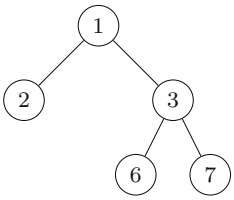


Fig. 5.10 Tree structure for the example of sparse Hamiltonian simulation in the text

5.4.3 Density Matrix Exponentiation

There are special conditions under in which we can guarantee qubit-efficiency for densely populated Hamiltonians. The most important in the context of quantum machine learning is the technique of *density matrix exponentiation*. This technique can be used for more general amplitude-efficient simulations, but becomes qubit-efficient for low-rank Hamiltonians. Instead of querying an oracle, the data is initially encoded in a density matrix, for which we can apply techniques from previous sections. Although density matrix exponentiation relies on many technical details and its understanding requires a high level of quantum computing expertise, we want to try to briefly sketch the outline of the idea. This section is consequently suited for more advanced readers.

The goal of density matrix exponentiation is to simulate a Hamiltonian $\mathrm{e}^{iH_\rho t}$ that encodes a non-sparse density matrix ρ and apply it to a quantum state σ. This is only possible because both operators are Hermitian, and every density matrix has an entry-wise equivalent Hamiltonian. It turns out that simulating H_ρ is approximately equivalent to simulating a swap operator S, applying it to the state $\rho \otimes \sigma$ and taking a trace operation. A swap operator is sparse and its simulation therefore efficient. It also means that whatever data we can encode in the entries of a density matrix, we can approximately encode it into a Hamiltonian.

The formal relation reads

$$\mathrm{tr}_2\left\{e^{-iS\Delta t}(\sigma \otimes \rho)e^{iS\Delta t}\right\} = \sigma - i\Delta t[\rho, \sigma] + \mathcal{O}(\Delta t^2) \tag{5.18}$$

$$\approx e^{-i\rho\Delta t}\sigma e^{i\rho\Delta t}. \tag{5.19}$$

Note that $\sigma - i\Delta t[\rho, \sigma]$ are the first terms of the exponential series $e^{-i\rho\Delta t}\sigma e^{i\rho\Delta t}$. In words, simulating the swap operator that 'exchanges' the state of the qubits of state ρ and σ for a short time Δt, and taking the trace over the second quantum system results in an effective dynamic as if exponentiating the second density matrix.

To prove the relation, consider two general mixed states $\rho = \sum_{i,i'=1}^{N} a_{ii'}|i\rangle\langle i'|$ and $\sigma = \sum_{jj'=1}^{N} b_{j,j'}|j\rangle\langle j'|$, where $\{|i\rangle\}, \{|j\rangle\}$ are the computational bases in the Hilbert spaces of the respective states, which have the same dimension. Now write the operator-valued exponential functions as a series

$$\mathrm{tr}_2\{e^{-iS\Delta t}(\sigma\otimes\rho)e^{iS\Delta t}\} = \mathrm{tr}_2\left\{\sum_{k,k'=0}^{\infty}\frac{(-i)^k\Delta t^k i^{k'}\Delta t^{k'}}{k!k'!}\ S^k(\sigma\otimes\rho)S^{k'}\right\}$$

and apply the swap operators. Since $S^2 = 1$, higher order terms in the series only differ in the prefactor, and for $\Delta t << 1$ the higher orders quickly vanish. We therefore only write the first few terms explicitly and summarise the vanishing tail of the series in a $\mathcal{O}(\Delta t^2)$ term,

$$\mathrm{tr}_2\Big\{\Big(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|j\rangle\langle j'|\otimes|i\rangle\langle i'|\Big) + i\Delta t\Big(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|j\rangle\langle j'|\otimes|i\rangle\langle i'|\Big)S$$
$$-i\Delta tS\Big(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|j\rangle\langle j'|\otimes|i\rangle\langle i'|\Big) + \mathcal{O}(\Delta t^2)\Big\}.$$

Next, apply the swap operator and the trace operation. Tracing out the second system means to 'sandwich' the entire expression by $\sum_k\langle k|\cdot|k\rangle$, where $\{|k\rangle\}$ is a full set of computational basis vectors in the Hilbert space of the second system. We get

$$\sum_{jj'}b_{jj'}|j\rangle\langle j'| + i\Delta t\sum_{ij}\sum_k a_{ki}b_{jk}|j\rangle\langle i| - i\Delta t\sum_{ij}\sum_k a_{ik}b_{kj}|i\rangle\langle j| + \mathcal{O}(\Delta t^2).$$

This is in fact the same as $\sigma - i\Delta t[\rho,\sigma] + \mathcal{O}(\Delta t^2)$, which can be shown by executing the commutator $[a,b] = ab - ba$,

$$\sigma + i\Delta t(\sigma\rho - \rho\sigma) + \mathcal{O}(\Delta t^2),$$

and inserting the expressions for ρ and σ,

$$\sigma + i\Delta t\Big(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|j\rangle\langle j'||i\rangle\langle i'| - \sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|i\rangle\langle i'||j\rangle\langle j'|) + \mathcal{O}(\Delta t^2\Big).$$

Note that the expressions $\rho\sigma$ and $\sigma\rho$ from the commutator are not abbreviated tensor products, but common matrix products, and we can use the relations $\langle j'|i\rangle = \delta_{j',i}$ and $\langle i'|j\rangle = \delta_{i',j}$ for orthonormal basis states. The error of the approximation is in $\mathcal{O}(\Delta t^2)$ which is negligible for sufficiently small simulation times Δt.

Density matrix exponentiation is often used in combination with phase estimation presented in Sects. 3.5.2 and 3.5.3. Once a density matrix containing data is exponentiated, we can 'apply' it to some amplitude encoded state and extract eigenvalues of ρ through a phase estimation routine using the the inverse quantum Fourier transform. However, to do so we need to be able to prepare powers of $(e^{-iH_\rho\Delta t})^k$ (compare to U^k in Sect. 3.5.2).

Lloyd et al. [41] show that this can be done by using of the order of $\mathcal{O}(\epsilon^{-3})$ copies of ρ joined with an index register of d qubits in superposition,

$$\sum_{k=1}^{2^d} |k\rangle\langle k| \otimes \sigma \otimes \rho^{(1)} \otimes \ldots \otimes \rho^{(2^d)}.$$

Instead of simulating a single swap operator, we now have to simulate a sequence of 2-qubit swap operators, each of which swaps the first state σ with the gth copy of ρ. The swap operator sequences are entangled with an index register, so that for index $|k\rangle$ the sequence of swap operators runs up to copy $\rho^{(k)}$,

$$\frac{1}{K}\sum_{k=1}^{K} |k\Delta t\rangle\langle k\Delta t| \otimes \prod_{g=1}^{k} e^{-iS_g\Delta t}$$

After taking the trace over all copies of ρ, this effectively implements the evolution

$$\sum_{k=1}^{K} |k\rangle\langle k| \otimes e^{-ikH_\rho\Delta t}\sigma\, e^{ikH_\rho\Delta t} + \mathcal{O}(\Delta t^2),$$

which is precisely in the form required to apply the quantum Fourier transform. For a proof, simply write out the expressions as seen above.

Density matrix exponentiation is qubit-efficient for low-rank matrices, given that the state ρ can be prepared qubit-efficiently. To see this, we first have to note that the desired accuracy is not necessarily a constant, but depends on the size of H_ρ, especially when we are interested in the eigenvalues of ρ. For example, consider the eigenvalues are approximately uniform. Since $\mathrm{tr}\rho = 1$, they are of the order of $\frac{1}{N}$ if N is the number of diagonal elements or the dimension of the Hilbert space of ρ. We certainly want the error to be smaller than the eigenvalues themselves, which means that $\epsilon < \frac{1}{N}$. The number of copies needed for the density matrix exponentiation in superposition hence grows with N^3, and since we have to apply swap operators to each copies, the runtime grows accordingly.

As a consequence, density matrix exponentiation for phase estimation is only polynomial in the number of qubits or qubit-efficient if the eigenvalues of H_ρ are dominated by a few large eigenvalues that do not require a small error to be resolved. In other words, the matrix has to be well approximable by a low-rank matrix. For design matrices containing the dataset, this means that the data is highly redundant.

References

1. Aharonov, D., Ta-Shma, A.: Adiabatic quantum state generation and statistical zero knowledge. In: Proceedings of the Thirty-Fifth annual ACM Symposium on Theory of Computing, pp. 20–29. ACM (2003)
2. Schuld, M., Petruccione, F.: Quantum machine learning. In: C. Sammut, G.I. Webb (eds.) Encyclopaedia of Machine Learning and Data Mining. Springer (2016)
3. Ventura, D., Martinez, T.: Quantum associative memory. Inf. Sci. **124**(1), 273–296 (2000)

4. Trugenberger, C.A.: Probabilistic quantum memories. Phys. Rev. Lett. **87**, 067901 (2001)
5. Giovannetti, V., Lloyd, S., Maccone, L.: Quantum random access memory. Phys. Rev. Lett. **100**(16), 160501 (2008)
6. Raoux, S., Burr, G.W., Breitwisch, M.J., Rettner, C.T., Chen, Y.-C., Shelby, R.M., Salinga, M., Krebs, D., Chen, S.-H., Lung, H.-L., et al.: Phase-change random access memory: a scalable technology. IBM J. Res. Dev. **52**(4.5):465–479 (2008)
7. Kyaw, T.H., Felicetti, S., Romero, G., Solano, E., Kwek, L.-C.: Scalable quantum memory in the ultrastrong coupling regime. Sci. Rep. **5**(8621) (2015)
8. Bennett, C.H.: Logical reversibility of computation. In: M. Demon (ed.) Entropy, Information, Computing, pp. 197–204 (1973)
9. Brown, L.D., Cai, T.T., DasGupta, A.: Interval estimation for a binomial proportion. Stat. Sci. 101–117 (2001)
10. Wilson, E.B.: Probable inference, the law of succession, and statistical inference. J. Am. Stat. Assoc. **22**(158), 209–212 (1927)
11. Aaronson, S.: Read the fine print. Nat. Phys. **11**(4), 291–293 (2015)
12. Kliesch, M., Barthel, T., Gogolin, C., Kastoryano, M., Eisert, J.: Dissipative quantum Church-Turing theorem. Phys. Rev. Lett. **107**(12), 120501 (2011)
13. Knill, E.: Approximation by quantum circuits (1995). arXiv:quant-ph/9508006
14. Mikko, M., Vartiainen, J.J., Bergholm, V., Salomaa, M.M.: Quantum circuits for general multiqubit gates. Phys. Rev. Lett. **93**(13), 130502 (2004)
15. Vartiainen, J.J., Möttönen, M., Salomaa, M.M.: Efficient decomposition of quantum gates. Phys. Rev. Lett. **92**(17), 177902 (2004)
16. Plesch, M., Brukner, Č.: Quantum-state preparation with universal gate decompositions. Phys. Rev. A **83**(3), 032302 (2011)
17. Iten, R., Colbeck, R., Kukuljan, I., Home, J., Christandl, M.: Quantum circuits for isometries. Phys. Rev. A **93**(3), 032318 (2016)
18. Möttönen, M., Vartiainen, J.J., Bergholm, V., Salomaa, M.M.: Transformation of quantum states using uniformly controlled rotations. Quantum Inf. Comput. **5**(467) (2005)
19. Grover, L., Rudolph,T.: Creating superpositions that correspond to efficiently integrable probability distributions (2002). arXiv:0208112v1
20. Kaye, P., Mosca, M.: Quantum networks for generating arbitrary quantum states. In: Proceedings of the International Conference on Quantum Information, OSA Technical Digest Series, pp. PB28. ICQI (2001). arXiv:quant-ph/0407102v1
21. Soklakov, A.N., Schack, R.: Efficient state preparation for a register of quantum bits. Phys. Rev. A **73**(1):012307 (2006)
22. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. Phys. Rev. Lett. **103**(15), 150502 (2009)
23. Rebentrost, P., Mohseni, M., Lloyd, S.: Quantum support vector machine for big data classification. Phys. Rev. Lett. **113**, 130503 (2014)
24. Prakash, A.: Quantum Algorithms for Linear Algebra and Machine Learning. Ph.D thesis, EECS Department, University of California, Berkeley, Dec 2014
25. Wiebe, N., Braun, D., Lloyd, S.: Quantum algorithm for data fitting. Phys. Rev. Lett. **109**(5), 050505 (2012)
26. Zhao, Z., Fitzsimons, J.K., Fitzsimons, J.F.: Quantum assisted Gaussian process regression (2015). arXiv:1512.03929
27. Breuer, H.-P., Petruccione, F.: The Theory of Open Quantum Systems. Oxford University Press (2002)
28. Abrams, D.S., Lloyd, S.: Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors. Phys. Rev. Lett. **83**(24), 5162 (1999)
29. Gisin, N.: Weinberg's non-linear quantum mechanics and supraluminal communications. Phys. Lett. A **143**(1), 1–2 (1990)
30. Peres, A.: Nonlinear variants of Schrödinger's equation violate the second law of thermodynamics. Phys. Rev. Lett. **63**(10), 1114 (1989)

31. Meyer, D.A., Wong, T.G.: Nonlinear quantum search using the Gross-Pitaevskii equation. New J. Phys. **15**(6), 063014 (2013)
32. Paetznick, A., Svore, K.M.: Repeat-until-success: Non-deterministic decomposition of single-qubit unitaries. Quantum Inf. Comput. **14**, 1277–1301 (2013)
33. Andrieu, C., De Freitas, N., Doucet, A., Jordan, M.I.: An introduction to MCMC for machine learning. Mach. Learn. **50**(1–2):5–43 (2003)
34. Georgescu, I.M., Ashhab, S., Nori, F.: Quantum simulation. Rev. Mod. Phys. **86**, 153–185 (2014)
35. Lloyd, S.: Universal quantum simulators. Science **273**(5278), 1073 (1996)
36. Childs, A.M.: Quantum information processing in continuous time. Ph.D thesis, Massachusetts Institute of Technology (2004)
37. Berry, D.W., Ahokas, G., Cleve, R., Sanders, B.C.: Efficient quantum algorithms for simulating sparse Hamiltonians. Commun. Math. Phys. **270**(2), 359–371 (2007)
38. Berry, D.W., Childs, A.M., Cleve, R., Kothari, R., Somma, R.D.: Exponential improvement in precision for simulating sparse Hamiltonians. In: Proceedings of the 46th Annual ACM Symposium on Theory of Computing, pp. 283–292. ACM (2014)
39. Berry, D.W., Childs, A.M., Kothari, R.: Hamiltonian simulation with nearly optimal dependence on all parameters. In: IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS), pp. 792–809. IEEE (2015)
40. Childs, A.M., Kothari, R.: Limitations on the simulation of non-sparse Hamiltonians. Quantum Inf. Comput. **10**(7), 669–684 (2010)
41. Lloyd, S., Mohseni, M., Rebentrost, P.: Quantum principal component analysis. Nat. Phys. **10**, 631–633 (2014)