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].

© Springer Nature Switzerland AG 2018

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

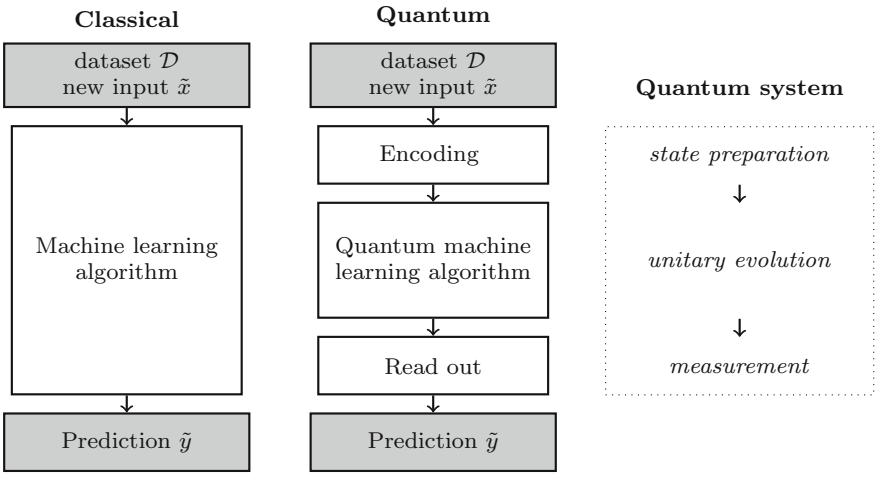

Fig. 5.1 In order to solve supervised machine learning tasks based on classical datasets, the quantum algorithm requires an information encoding and read out step that are in general highly non-trivial procedures, and it is important to consider them in the runtime. Adapted from [2]

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

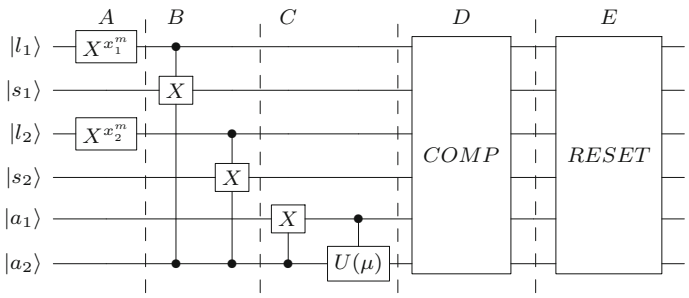

Fig. 5.2 Circuit for one step of Ventura and Martinez state preparation routine as described in the text for an example of 2-bit input patterns. The storage and loading qubits are in a slightly different order to facilitate the display. (A) The pattern is written into the loading register by NOT gates. 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,

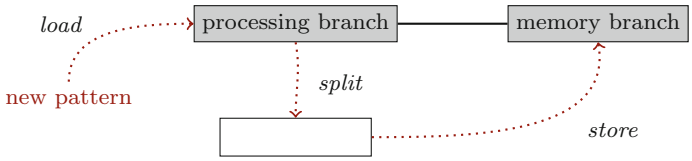

Fig. 5.3 Illustration of Ventura and Martinez state preparation routine [3]. The superposition is divided into a processing and memory term, flagged by an ancilla. New training input patterns get successively loaded into the processing branch, which gets 'split' by a Hadamard on an ancilla, and the pattern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

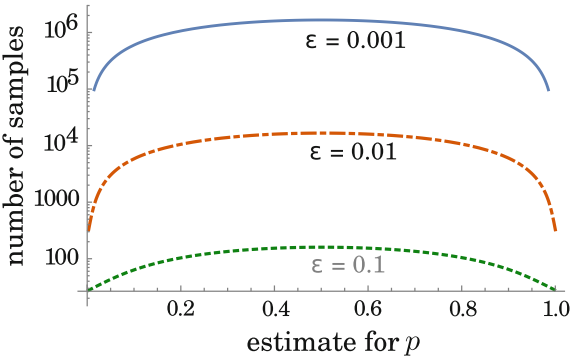

Fig. 5.4 Relationship between the sample size S and the mean value $\bar{p} = \frac{1}{S}\sum_{i=1}^{S} y_i$ for different errors ϵ for the Wilson score interval of a Bernoulli parameter estimation problem as described in the text

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