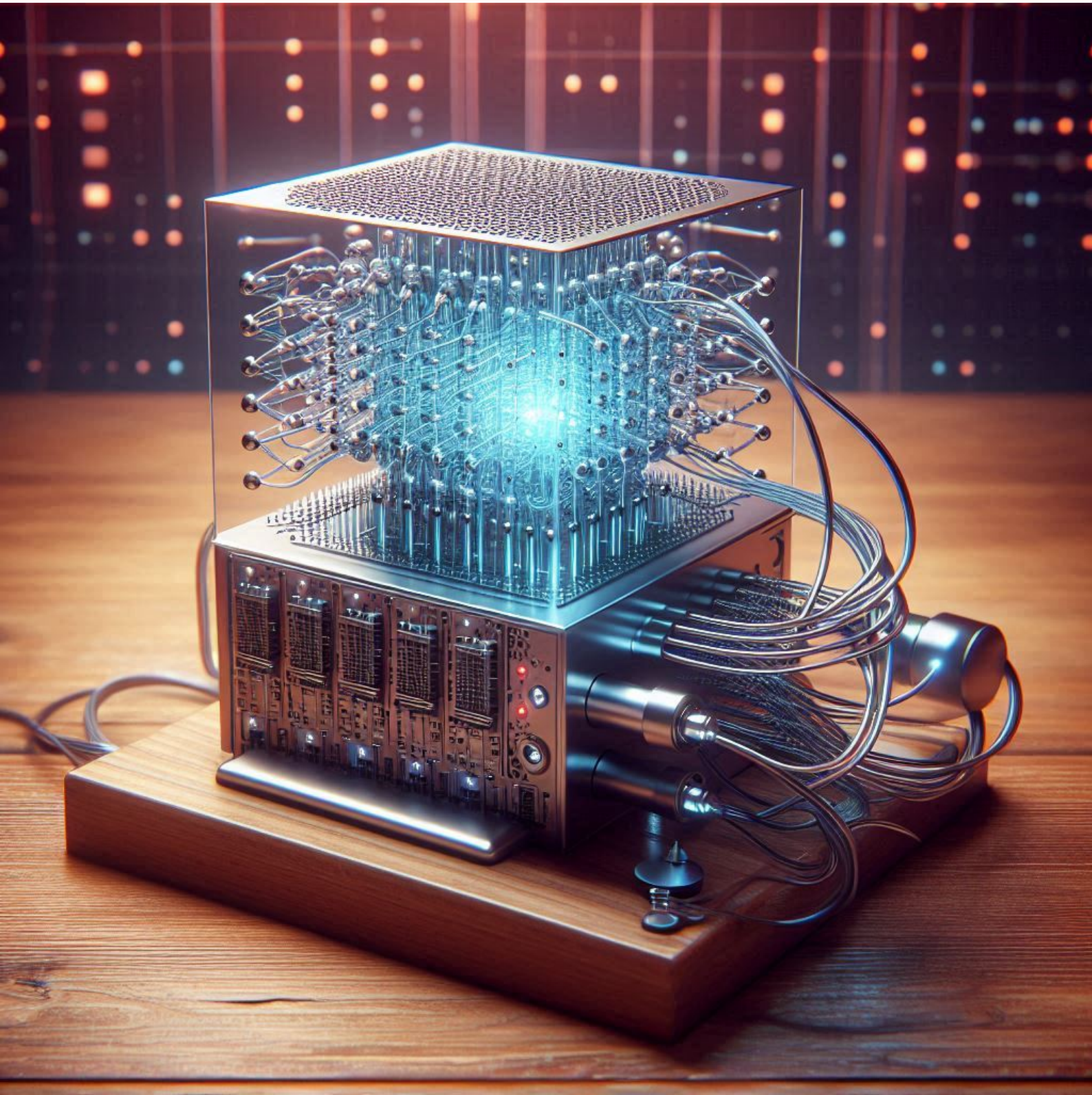
Quantum Machine Learning
Lecture Notes

Institute for Integrated Circuits and Quantum Computing
Johannes Kepler University Linz, Austria

Winter Term 2025/26

Johannes Kofler



Cite as:
Johannes Kofler, *Quantum Machine Learning Lecture Notes*, Johannes Kepler University Linz, Austria, 2025/26.

The cover page picture was generated with the Microsoft Designer Image Creator.

Contents

1. Introduction

Quantum Machine Learning (QML) is an emerging field at the intersection of quantum computing and machine learning. It aims to harness the principles of quantum mechanics – such as superposition and entanglement – to develop algorithms that can process and analyze data in ways that may surpass the capabilities of classical computers. While both quantum computing and machine learning are well-established disciplines on their own, their synthesis is comparatively new and rapidly evolving, driven by theoretical and practical advances in both fields.

Machine learning has become a cornerstone of modern data science, powering applications in natural language processing, image recognition, recommendation systems, and many more. However, classical machine learning algorithms sometimes struggle with challenges such as high-dimensional data. On the other hand, quantum computing offers a fundamentally different model of computation, which can process information in exponentially large Hilbert spaces. This raises the question: Can quantum computers provide advantages for machine learning tasks?

QML can be categorized into four main tasks, where data are either classical or quantum, and where the algorithm is either classical or quantum (see Fig. 1.1). It can enhance a plethora of applications (see Fig. 1.2). Two primary ingredients for many supervised, unsupervised, and reinforcement learning schemes are quantum neural networks (QNNs) and quantum kernels (see Fig. 1.3).

This course provides an introduction to Quantum Machine Learning. It will not attempt to give a complete overview of the field but will rather choose to focus on a few specific topics. Prior knowledge in quantum information and/or quantum computing as well as in supervised machine learning is expected. These lecture notes are being written during Winter Term 2025/26, when this course is held for the first time.

Johannes Kofler

Institute for Integrated Circuits and Quantum Computing
Johannes Kepler University Linz, Austria
Winter Term 2025/26

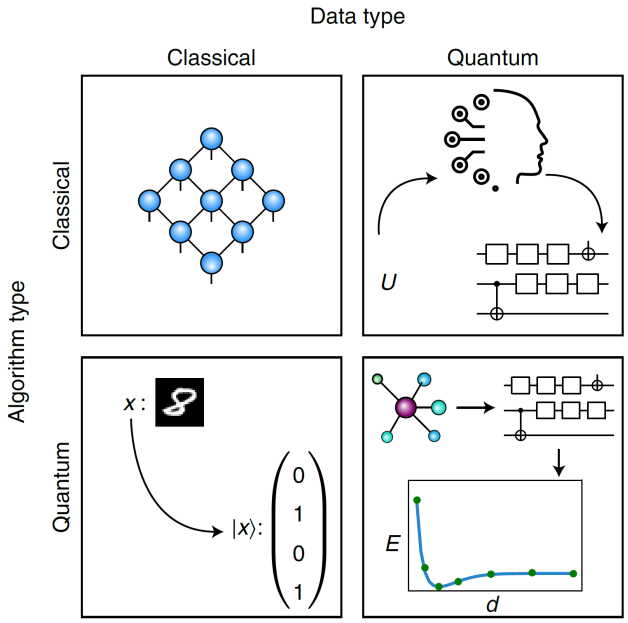


Figure 1.1: QML tasks. QML is usually considered for four main tasks. These include tasks where the data are either classical or quantum, and where the algorithm is either classical or quantum. Top left: Tensor networks are quantum-inspired classical methods that can analyze classical data. Top right: Unitary time-evolution data U from a quantum system can be classically compiled into a quantum circuit. Bottom left: Handwritten digits can be mapped to quantum states for classification on a quantum computer. Bottom right: Molecular ground-state data can be classified directly on a quantum computer. The figure shows the dependence of ground-state energy E on the distance d between the atoms. Image and caption taken from Ref. [1].

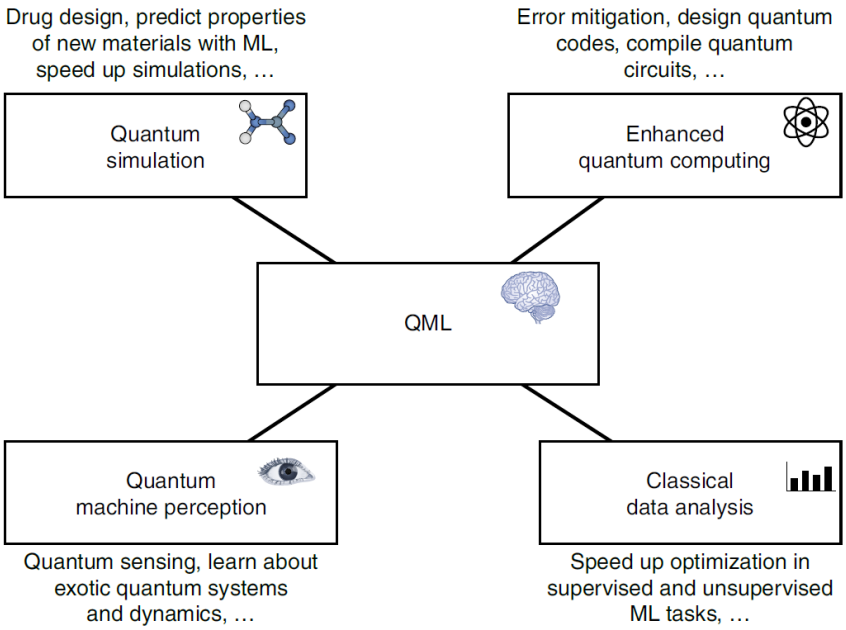


Figure 1.2: Key applications for QML. QML has been envisioned to bring a computational advantage in many applications. QML can enhance quantum simulation for chemistry (for example, molecular ground states, equilibrium states and time evolution) and materials science (for example, quantum phase recognition and generative design with a target property in mind). QML can enhance quantum computing by learning quantum error correction codes and syndrome decoders, performing quantum control, learning to mitigate errors and compiling and optimizing quantum circuits. QML can enhance sensing and metrology and extract hidden parameters from quantum systems. Finally, QML may speed up classical data analysis, including clustering and classification. Image and caption taken from Ref. [1].

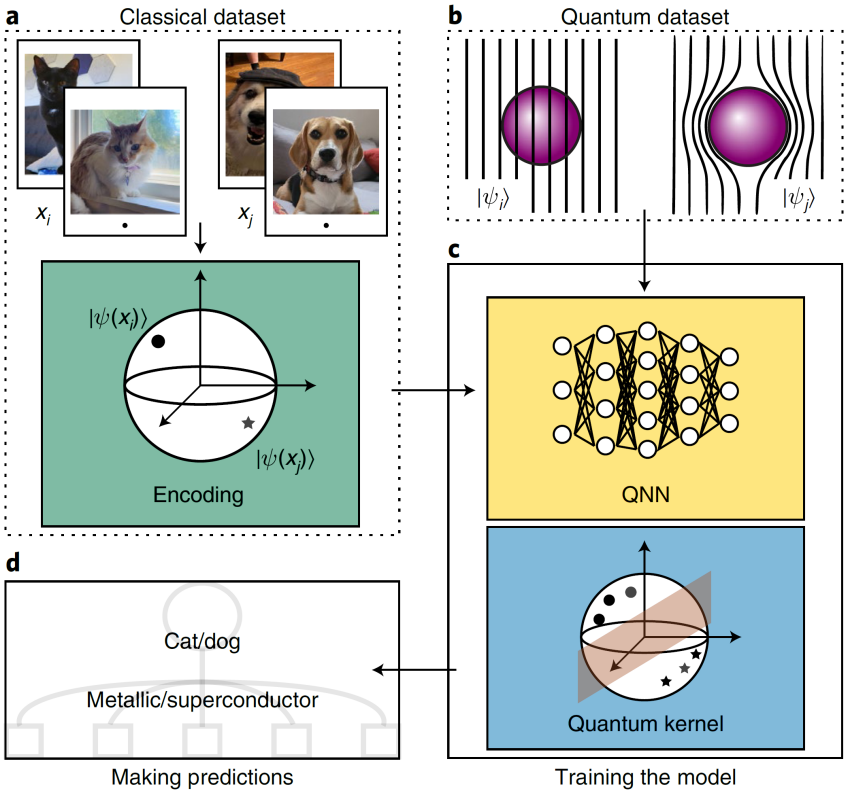


Figure 1.3: Classification with QML. (a) The classical data x, that is, images of cats and images of dogs, is encoded into a Hilbert space via some map $x \to |\psi(x)\rangle$. Ideally, data from different classes (here represented by dots and stars) are mapped to different regions of the Hilbert space. (b) Quantum data $|\psi(x)\rangle$ can be directly analyzed on a quantum device. Here the dataset is composed of states representing metallic or superconducting systems. (c) The dataset is used to train a QML model. Two common paradigms in QML are QNNs and quantum kernels, both of which allow for classification of either classical or quantum data. In kernel methods we fit a decision hyperplane that separates the classes. (d) Once the model is trained, it can be used to make predictions. Image and caption taken from Ref. [1].

2. A Toy Example

In this chapter, we will discuss a toy example of a quantum algorithm for classification. It will give us an intuition of how a quantum computer can help us to learn from classical data (bottom left task in Fig. 1.1 above). In particular, we will implement a nearest neighbor method with the use of quantum interference. We follow Ref. [5].

Let's first fix our notation: We are given a data set with feature vectors $\mathbf{x}_i$ and binary labels $y_i \in \{0,1\}$, where the label $i = 1,\ldots,M$ runs over all M samples.

In the k-nearest neighbor algorithm, a new feature vector $\mathbf{x}$ is assigned the same label as given by the majority vote of the k data points that are closest to it. The notion of "closeness" mathematically requires a distance measure. Here, we will consider a modified version of the algorithm in which all data points vote, but the vote of each sample $\mathbf{x}_i$ is weighted by a factor that depends on the squared distance to the new data point $\mathbf{x}$:

$$\gamma_i = 1 - \frac{1}{c}\,||\mathbf{x}_i - \mathbf{x}||^2. \tag{2.1}$$

When the new input $\mathbf{x}$ is close to a feature $\mathbf{x}_i$, then the (square of the norm of the) difference will be small and γ_i will be close to 1. If, however, $\mathbf{x}$ and $\mathbf{x}_i$ are fairly distant, then γ_i will become small. The parameter c makes sure that γ_i can't fall below 0. (You will notice the connection to kernel methods here.)

Now we define the probability to predict label $y = b$, given a new input $\mathbf{x}$, as the sum over the weights from all those M_b feature vectors that are labeled with $b \in \{0,1\}$. We call this the *squared-distance classifier*:

$$p(y=b|\mathbf{x}) = \frac{1}{\kappa}\frac{1}{M_b}\sum_{i|y_i=b}^{M_b}\gamma_i. \tag{2.2}$$

Here, the parameter κ ensures that $p(y=0|\mathbf{x}) + p(y=1|\mathbf{x}) = 1$, i.e. that the probabilities are properly normalized. Moreover, we have $M_0 + M_1 = M$, i.e. the sum of the samples of the two classes equals the total number of samples.

At this point, we switch to the quantum world. We start by quickly revisiting the (unitary) Hadamard gate $H = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$, which has the following properties:

$$H|0\rangle = |+\rangle, \;\; H|1\rangle = |-\rangle. \tag{2.3}$$

Here, $|0\rangle = \left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$ and $|1\rangle = \left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)$ are the computational (z) basis 1-qubit quantum states, while $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 \\ -1 \end{smallmatrix}\right)$ are the diagonal (x) basis quantum states. And $\mathbb{1}$ is the 2×2 identity matrix.

Let us consider n qubits in the generic state

$$|\psi\rangle = (\alpha_1, \alpha_2, \ldots, \alpha_{2^n})^T. \tag{2.4}$$

This vector has (Hilbert space) dimension $N = 2^n$ and is properly normalized as the modulus squared (complex) amplitudes sum up to 1, i.e. $|\alpha_1|^2 + |\alpha_2|^2 + \ldots + |\alpha_{2^n}|^2 = 1$.

Let us apply the Hadamard gate only to the first qubit. To all other qubits, we do nothing, i.e. apply the identity $\mathbb{1}$. The total unitary transformation then reads

$$U = H \otimes \mathbb{1} \otimes \mathbb{1} \otimes \cdots \otimes \mathbb{1} = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbb{1}_{N/2} & \mathbb{1}_{N/2} \\ \mathbb{1}_{N/2} & -\mathbb{1}_{N/2} \end{pmatrix}, \tag{2.5}$$

where $\mathbb{1}_{N/2}$ is the $\frac{N}{2} \times \frac{N}{2}$ identity matrix. This unitary transforms our initial state as follows:

$$|\psi\rangle = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_{\frac{N}{2}} \\ \alpha_{\frac{N}{2}+1} \\ \vdots \\ \alpha_N \end{pmatrix} \;\rightarrow\; U|\psi\rangle = |\psi'\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} \alpha_1 + \alpha_{\frac{N}{2}+1} \\ \vdots \\ \alpha_{\frac{N}{2}} + \alpha_N \\ \alpha_1 - \alpha_{\frac{N}{2}+1} \\ \vdots \\ \alpha_{\frac{N}{2}} - \alpha_N \end{pmatrix}. \tag{2.6}$$

If we write the quantum state as $|\psi\rangle = (\boldsymbol{\alpha}, \boldsymbol{\beta})^T$, where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_{\frac{N}{2}})$ and $\boldsymbol{\beta} = (\alpha_{\frac{N}{2}+1}, \alpha_{\frac{N}{2}+2}, \ldots, \alpha_N)$, then $|\psi'\rangle = (\boldsymbol{\alpha} + \boldsymbol{\beta}, \boldsymbol{\alpha} - \boldsymbol{\beta})^T/\sqrt{2}$.

Most importantly, although the Hadamard transformation was applied only to the first qubit, all 2^n amplitudes have changed. This gives us some insight into the power of quantum computation: one single operation can change exponentially many probabilities if the initial state was highly entangled, i.e. had exponentially many amplitudes. And it also shows us why it is hard to design quantum algorithms: one needs to find sequences of gates such that amplitudes interfere (add and cancel) in a very specific way to solve a useful task.

Now we are in the position to go through the quantum machine learning algorithm that solves the classification task by computing the squared-distance classifier. For simplicity and didactical purposes, we assume that we have a data set with only $M = 2$ samples,

	price	room	survival
Passenger 1	0.921	0.390	yes (1)
Passenger 2	0.141	0.990	no (0)
Passenger 3	0.866	0.500	?

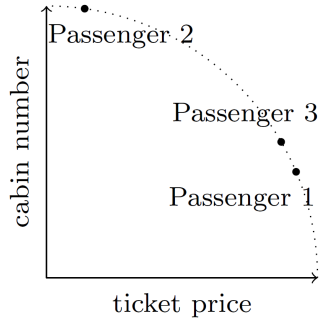


Figure 2.1: Left: Data after preprocessing. Each feature was first normalized to a number between 0 and 1 such that features are of similar importance. Then, each vector was normalized to unit length. Right: The points now lie on a unit circle. Image taken from Ref. [5].

where each feature vector has length 2 and each label appears once:

$$\mathbf{z}_1 = \{\mathbf{x}_1 = (x_1^{(1)}, x_1^{(2)}), y_1 = 1\}, \tag{2.7}$$

$$\mathbf{z}_2 = \{\mathbf{x}_2 = (x_2^{(1)}, x_2^{(2)}), y_2 = 0\}. \tag{2.8}$$

We want to classify a new sample $\mathbf{x} = (x^{(1)}, x^{(2)})$ whose true label y we don't know.

If you want an example in mind, take a simplified version of the Titanic dataset, where the first numerical feature is the ticket price, and second is the cabin number, and the label indicates survival ($y = 1$) or death ($y = 0$).

Step 1: Data Preprocessing

Data preprocessing happens in two rounds: The first is quite common: All features are individually normalized to numbers between 0 and 1. In the second round, we normalize all vectors to length 1, i.e. $\mathbf{x}_i := \mathbf{x}_i / ||\mathbf{x}_i||$. This projects the data onto a unit circle. For some data sets, this second normalization is reasonable as only the angle is relevant. For others, the loss of information may be problematic. Figure 2.1 illustrates our toy data of Titanic passengers after preprocessing.

Step 2: Data Encoding

We need to encode 6 features and 2 labels. We do this by a method called *amplitude encoding*, i.e. we write the features into the amplitudes of our initial state. Concretely, the state vector is a concatenation of the 2 feature vectors of the data set, followed by 2 copies of the feature vector whose label we want to predict:

$$|\psi\rangle = \frac{1}{\sqrt{4}} (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}, \mathbf{x})^T = \frac{1}{\sqrt{4}} \left(x_1^{(1)}, x_1^{(2)}, x_2^{(1)}, x_2^{(2)}, x^{(1)}, x^{(2)}, x^{(1)}, x^{(2)}\right)^T. \tag{2.9}$$

This is a 3-qubit state vector. As the individual feature vectors were previously normalized to length 1, we now need a factor $\frac{1}{\sqrt{4}}$ for quantum state normalization, as there are 4 vectors in the state.

Next, we have to encode also the two known labels. We thus need to extend our state to

q_1	q_2	q_3	q_4	$\|\psi\rangle_{\text{init}}$	$\|\psi\rangle_{\text{inter}}$	$\|\psi\rangle_{\text{final}}$
0	0	0	0	0	0	0
0	0	0	1	$\frac{1}{\sqrt{4}}x_1^{(1)}$	$\frac{1}{\sqrt{8}}(x_1^{(1)}+x^{(1)})$	$\frac{1}{\sqrt{8\chi}}(x_1^{(1)}+x^{(1)})$
0	0	1	0	0	0	0
0	0	1	1	$\frac{1}{\sqrt{4}}x_1^{(2)}$	$\frac{1}{\sqrt{8}}(x_1^{(2)}+x^{(2)})$	$\frac{1}{\sqrt{8\chi}}(x_1^{(2)}+x^{(2)})$
0	1	0	0	$\frac{1}{\sqrt{4}}x_2^{(1)}$	$\frac{1}{\sqrt{8}}(x_2^{(1)}+x^{(1)})$	$\frac{1}{\sqrt{8\chi}}(x_2^{(1)}+x^{(1)})$
0	1	0	1	0	0	0
0	1	1	0	$\frac{1}{\sqrt{4}}x_2^{(2)}$	$\frac{1}{\sqrt{8}}(x_2^{(2)}+x^{(2)})$	$\frac{1}{\sqrt{8\chi}}(x_2^{(2)}+x^{(2)})$
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	$\frac{1}{\sqrt{4}}x^{(1)}$	$\frac{1}{\sqrt{8}}(x_1^{(1)}-x^{(1)})$	0
1	0	1	0	0	0	0
1	0	1	1	$\frac{1}{\sqrt{4}}x^{(2)}$	$\frac{1}{\sqrt{8}}(x_1^{(2)}-x^{(2)})$	0
1	1	0	0	$\frac{1}{\sqrt{4}}x^{(1)}$	$\frac{1}{\sqrt{8}}(x_2^{(1)}-x^{(1)})$	0
1	1	0	1	0	0	0
1	1	1	0	$\frac{1}{\sqrt{4}}x^{(2)}$	$\frac{1}{\sqrt{8}}(x_2^{(2)}-x^{(2)})$	0
1	1	1	1	0	0	0

Table 2.1: Steps of the quantum machine learning algorithm solving a classification task by computing the squared-distance classifier. The table shows the amplitudes of each of the 4-qubit (computational basis) states $|q_1\rangle|q_2\rangle|q_3\rangle|q_4\rangle$ after (step 2) initialization, (step 3) Hadamard transform, and (step 4) measurement.

a 4-qubit state to allow for additional amplitudes. The 4^{th} qubit is in a state that corresponds to the label of the feature vector. As we don't have a label for the new input vector $\mathbf{x}$, we associate the first copy of $\mathbf{x}$ with the label of passenger 1 (i.e. 1) and the second copy with the label of passenger 2 (i.e. 0). Table 2.1 illustrates the encoding. We hence reach the initial quantum state, which encodes all the given data:

$$|\psi\rangle_{\text{init}} = \frac{1}{\sqrt{4}}\left(0,x_1^{(1)},0,x_1^{(2)},x_2^{(1)},0,x_2^{(2)},0,0,x^{(1)},0,x^{(2)},x^{(1)},0,x^{(2)},0\right)^T. \tag{2.10}$$

Step 3: Hadamard Transformation

The Hadamard gate is applied onto the first qubit. This leads to the state

$$|\psi\rangle_{\text{inter}} = (H\otimes\mathbb{1}\otimes\mathbb{1}\otimes\mathbb{1})\,|\psi\rangle_{\text{init}}. \tag{2.11}$$

The explicit form of this state can be read from Table 2.1. The Hadamard transform computes the sums and differences between blocks of amplitudes, namely between the training samples on the one hand and the new input on the other.

Step 4: Measurement of First Qubit

We measure the first qubit. If the result is $q_1 = 0$, the algorithm continues; otherwise we abort and go back to step 1. The accepted 3-qubit post-measurement state

$$|\psi\rangle_{\text{final}} = \frac{1}{\chi}\langle 0|_1 |\psi\rangle_{\text{inter}} \tag{2.12}$$

only has amplitudes represented by the first 8 entries in Table 2.1 where $q_1 = 0$. The parameter $\chi = \frac{1}{8}\left(|x_1^{(1)} + x^{(1)}|^2 + |x_1^{(2)} + x^{(2)}|^2 + |x_2^{(1)} + x^{(1)}|^2 + |x_2^{(2)} + x^{(2)}|^2\right)$ takes care of the normalization.

Step 5: Measurement of Last Qubit

We measure the last qubit. In fact, we repeat all steps 1 to 5 often enough, to estimate the probability for the last qubit to be in state $q_4 = 0$, which is given by Born's rule:

$$\begin{aligned} p(q_4 = 0) = |\langle 0|_4 |\psi\rangle_{\text{final}}|^2 &= \frac{1}{8\chi}\left(|x_2^{(1)} + x^{(1)}|^2 + |x_2^{(2)} + x^{(2)}|^2\right) \\ &= \frac{1}{8\chi}\,||\mathbf{x}_2 + \mathbf{x}||^2. \end{aligned} \tag{2.13}$$

This probability is the output of our machine learning model, i.e. it represents the probability that the classifier predicts the label 0 for the new input. The probability to predict label 1 is, of course, $p(q_4 = 1) = |\langle 1|_4 |\psi\rangle_{\text{final}}|^2 = 1 - p(q_4 = 0)$. Using the numbers from Table 2.1, we get $p(q_4 = 0) \approx 0.448$ and $p(q_4 = 1) \approx 0.552$.

Finally, we compare this to the classical squared-distance classifier (2.2). For normalized vectors (as we decided to use), the constant c in (2.1) equals to 4 as two normalized vectors can subtract to a vector of at most length 2, whose squared norm is 4. Also, for two unit vectors $\mathbf{a}$ and $\mathbf{b}$, it always holds that

$$||\mathbf{a} + \mathbf{b}||^2 + ||\mathbf{a} - \mathbf{b}||^2 = 4. \tag{2.14}$$

Hence, we get

$$\begin{aligned} p(y = 0|\mathbf{x}) = \frac{1}{\kappa}\frac{1}{M_0}\sum_{i|y_i=0}^{M_0}\gamma_i &= \frac{1}{\kappa}\left(1 - \tfrac{1}{4}\,||\mathbf{x}_2 - \mathbf{x}||^2\right) \\ &= \frac{1}{4\kappa}\,||\mathbf{x}_2 + \mathbf{x}||^2. \end{aligned} \tag{2.15}$$

This is the same as (2.13) since the different constants need to guarantee normalization, i.e. $\kappa = 2\chi$. Hence, our quantum model makes the same prediction as the classical algorithm.

What are some take-home messages from this toy example?

- This quantum algorithm is independent of the data size, which is a very remarkable

and promising feature. The Hadamard transform needs to be performed only once, and only on the first qubit.

- However, the data encoding needs to prepare an entangled input state which requires another algorithm whose operations scale linearly with the data size.
- Then again, in a QQ setting, the data would be given in quantum form already such that no encoding needs to be performed.
- The Hadamard transform is in the Clifford group. So a classical simulation of the quantum algorithm with only polynomial time overhead may be possible, depending on the data encoding operations.

We can also draw some general conclusions about quantum machine learning:

- Data encoding is sometimes the most important step when dealing with classical data. It decides which quantum algorithm can be used and therefore determines the runtime and potential quantum advantage.
- Before encoding, classical preprocessing may be required.
- The result of a quantum machine learning method is always gathered by measurements, typically by many repetitions in order to gather reliable statistics.
- Quantum machine learning algorithms are often inspired by classical algorithms.

3. Variational Quantum Algorithms

There are well-known quantum algorithms such as Shor's or Grover's algorithm, which can solve interesting tasks such as factoring or database search. However, as of today, quantum computers are not yet developed enough to run these algorithms for a large enough number of qubits such that classical computers could be beaten.

An alternative class of quantum algorithms, which can be run on small-scale quantum computers, are *Variational Quantum Algorithms* (VQAs). VQAs are a central paradigm in near-term quantum computing, designed to leverage the capabilities of noisy intermediate-scale quantum (NISQ) devices. Unlike fault-tolerant quantum algorithms, which require large error-corrected quantum computers, VQAs combine classical optimization methods with parametrized quantum circuits to solve problems approximately, while tolerating hardware imperfections. Such problems often stem from material science, chemistry, or combinatorial optimization.

In VQAs, a quantum processor prepares a quantum state using a parameterized quantum circuit, also called an ansatz, $W(\boldsymbol{\theta})$ which depends on a list of tunable gate parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots)$. The quantum computer then measures expectation values of relevant observables. These measurement results are fed into a classical optimizer, which updates the parameters $\boldsymbol{\theta}$ to minimize a cost function $C(\boldsymbol{\theta})$. This hybrid quantum-classical loop continues until convergence. Similar to training a (classical) neural network, a variational quantum algorithm is therefore in fact a family of algorithms, from which the training picks a (hopefully) good candidate (see Fig. 3.1).

The ansatz circuit $W(\boldsymbol{\theta})$ typically consists of 2-qubit CNOT gates and single-qubit Pauli rotation gates:[1]

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{3.1}$$

[1] Note that the $\boldsymbol{\theta}$ in $W(\boldsymbol{\theta})$ is a (large) vector of parameters, while the θ in the rotation gates is a single real number.

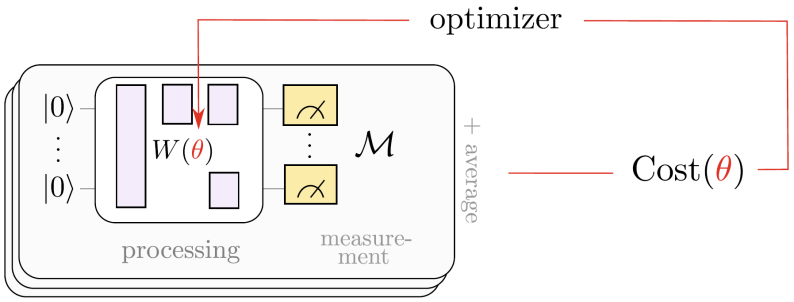


Figure 3.1: Working scheme of a variational quantum algorithm. An initial product state is transformed via a parametrized sequence of gates $W(\boldsymbol{\theta})$ and finally measured. A cost function, which represents the given computational problem, evaluates the expected measurement for a given set of parameters, and an optimizer adapts the parameters accordingly to lower the future cost. The training happens iteratively until some convergence criterion is met. The inputs of the algorithm are: The ansatz $W(\boldsymbol{\theta})$, the cost function, and the training data. Depending on the formulation of the problem, the possible outputs (solutions) are: the final quantum state, a probability distribution, a bitstring, or the final (trained) gate sequence $W(\boldsymbol{\theta})$. Image taken from Ref. [5].

$$R_X(\theta) = e^{-i\frac{\theta}{2}X} = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \tag{3.2}$$

$$R_Y(\theta) = e^{-i\frac{\theta}{2}Y} = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \tag{3.3}$$

$$R_Z(\theta) = e^{-i\frac{\theta}{2}Z} = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}. \tag{3.4}$$

Here, X, Y, and Z are the Pauli matrices. As the name suggests, the Pauli rotation gates $R_X(\theta)$, $R_Y(\theta)$, and $R_Z(\theta)$ perform single-qubit rotations by an angle θ about the x, y, and z axis, respectively.[2]

An ansatz typically consists of several layers $k = 1, ..., p$ with the same structure. As a concrete example, a layer may have the following form: First, there is an R_Y gate for every qubit, each with its individual θ parameter. And then there is a sequence of CNOTs, connecting adjacent pairs, and finally connecting the last to the first qubit. Such a layer has n parameters. The full ansatz $W(\boldsymbol{\theta})$ thus consists of pn parameters, i.e. $\boldsymbol{\theta} \in \mathbb{R}^{pn}$.

This relatively decent scaling in quantum resources stands in stark contrast with the ressources that are necessary to simulate such circuits on classical hardware. For n qubits, the Hilbert space has dimension 2^n, and the entangling gates (i.e. CNOTs) in each variational layer guarantee that these exponentially many degrees of freedom indeed become used. There is no known classical algorithm that can simulate VQAs with more than a few dozen qubits and layers.[3]

Several well-known VQAs have been developed for different applications. The Variational Quantum Eigensolver (VQE) aims to approximate ground state energies of molecules

[2] One can also write: $R_X(\theta) = \cos(\frac{\theta}{2})\mathbb{1} - i\sin(\frac{\theta}{2})X$, $R_Y(\theta) = \cos(\frac{\theta}{2})\mathbb{1} - i\sin(\frac{\theta}{2})Y$, and $R_Z(\theta) = \cos(\frac{\theta}{2})\mathbb{1} - i\sin(\frac{\theta}{2})Z$.

[3] For $n = 300$, the Hilbert space dimension is larger than the number of atoms in the universe.

and materials, making it promising for quantum chemistry and materials science. The Quantum Approximate Optimization Algorithm (QAOA) is tailored to combinatorial optimization problems, using shallow circuits and a parameterized structure inspired by adiabatic evolution. Extensions of these methods also target machine learning, quantum control, and error mitigation.

The key advantages of VQAs include their relatively low circuit depth, adaptability to noisy hardware, and flexibility in problem formulation. However, challenges remain. These include barren plateaus (flat optimization landscapes that hinder training), the need for efficient classical optimizers, and measurement overhead due to repeated sampling. Despite these obstacles, VQAs remain a promising strategy for extracting useful results from NISQ devices, bridging the gap between current quantum hardware and future fault-tolerant quantum computing.

3.1 Variational Quantum Eigensolver

The Variational Quantum Eigensolver (VQE) is motivated by the problem of finding ground states of physical systems, which is important not only in physics but also in chemistry, materials science, and even parts of computer science. Ground states are the most stable, lowest-energy configuration of a system. Their understanding can provide key insights into a system's structure, stability, and behavior.

In physics, the energy observable is the Hamiltonian (operator) H.[4] The eigenstates $|\psi_i\rangle$ of the Hamiltonian correspond to energy eigenvalues E_i:[5]

$$H|\psi_i\rangle = E_i|\psi_i\rangle \tag{3.5}$$

The ground state $|\psi_0\rangle$ of a system is the eigenstate with the lowest eigenvalue, called the ground state energy:

$$E_0 = \langle\psi_0|H|\psi_0\rangle = \min_\psi \langle\psi|H|\psi\rangle. \tag{3.6}$$

VQEs employ a parametrised ansatz $W(\boldsymbol{\theta})$ which prepares a state $|\psi(\boldsymbol{\theta})\rangle$. From this state, the cost (= energy)

$$C(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle \tag{3.7}$$

is computed. The value of the cost function is the error signal which is fed back – using classical algorithms – and thus changes the parameters $\boldsymbol{\theta}$. This is repeated until the minimum of the cost function (or a good approximation thereof) is reached:

$$\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle = E_0. \tag{3.8}$$

A crucial problem for variational quantum eigensolvers is the implementation of the

[4] There is an unfortunate collision with the same notation H for the Hadamard gate.

[5] In this eigenvalue equation, the Hamiltonian H is a Hermitian operator (or matrix), while the energy E_i is a real number.

Hamiltonian as an observable. For general problem settings, a prohibitively large number of measurements would be necessary. Consider the example of n-qubit system with 2^n dimensional Hilbert space with a Hamiltonian which has a -1 at the, say, k-th diagonal entry, with k some number between 1 and 2^n. The ground state of the Hamiltonian is the k-th basis state. It has the eigenvalue (i.e. energy) -1, while all other basis states have eigenvalue 0. If we measure in the computational basis, the expectation value $\langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$ is estimated by the number of times we find the k-th state, divided by all runs, and everything multiplied by -1. If the state before measurement is highly entangled, it is somewhat close to uniform superposition of all basis states, i.e. all 2^n amplitudes are "populated". Hence, we require $\mathcal{O}(2^n)$ measurements for the estimate of the cost function, which we need for a single update step. Even for moderate n, this quickly becomes unfeasible.

Fortunately, in many interesting problems, the Hamiltonian is a sum of *local* observables H_j, i.e. acting on only 1 or 2 qubits:

$$H = \sum_{j=1}^{J} h_j H_j, \tag{3.9}$$

where the h_j are some real numbers. In particular, these operators H_j often are products of (a small number of) Pauli operators such as X_i or $Z_i Z_j$. This is, e.g., the case for Ising and Heisenberg models, which play a fundamental role in many-body physics. With this decomposition (3.9), the overall cost is simply the sum of individual cost terms, which are all tractable:

$$C(\boldsymbol{\theta}) = \sum_{j=1}^{J} h_j \langle\psi(\boldsymbol{\theta})|H_j|\psi(\boldsymbol{\theta})\rangle. \tag{3.10}$$

The procedure is shown in Fig. 3.2. If the number of terms J grows only polynomially with the number of qubits n, the variational quantum eigensolver can estimate the energy efficiently.

Let's sum up the steps of the VQE algorithm, which proceeds as a hybrid quantum–classical loop:

1. Prepare the state $|\psi(\boldsymbol{\theta})\rangle = W(\boldsymbol{\theta})|0\rangle^{\otimes n}$ on a quantum computer.
2. Measure the expectation values of the Pauli terms in H.
3. Compute the total energy $C(\boldsymbol{\theta})$, which serves as a cost function.
4. Update the parameters $\boldsymbol{\theta}$ using a classical optimizer (such as gradient descent[6]) to minimize $C(\boldsymbol{\theta})$.

Example: One-Dimensional Transverse-Field Ising Model

As a simple and illustrative example of the Variational Quantum Eigensolver (VQE), we consider the one-dimensional transverse-field Ising model (TFIM) consisting of three interacting spin-$\frac{1}{2}$ particles (qubits) with open boundary conditions, i.e. the spins are in a

[6]We will discuss optimization later in detail, in particular the parameter shift rule.

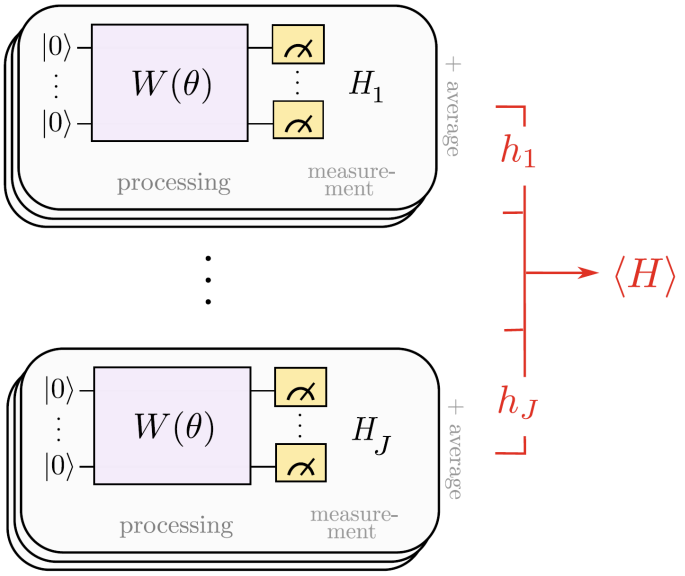


Figure 3.2: Working scheme of the variational quantum eigensolver. The cost function which is minimized is the expectation value of the system Hamiltonian. For many interesting systems, the Hamiltonian can be decomposed into a linear combination of a sufficiently small number of local terms, which can be measured separately and then added classically. Image taken from Ref. [5].

chain, not in a circle.[7] The Hamiltonian of the transverse-field Ising model is given by

$$H = -J \sum_{i=1}^{N-1} Z_i Z_{i+1} - h \sum_{i=1}^{N} X_i, \tag{3.11}$$

where Z_i and X_i are the Pauli matrices acting on qubit i, J is the interaction strength between neighboring spins[8], and $h > 0$ is the strength of the transverse magnetic field.

The first term is called the interaction term (or Ising term) between nearest neighbors. If $J > 0$, we call the coupling *ferromagnetic*, and spins prefer to align, i.e. all qubits are in the state up $|\uparrow\rangle = |0\rangle$ or all are in the state down $|\downarrow\rangle = |1\rangle$. If $J < 0$, the coupling is called *antiferromagnetic*, and spins prefer to anti-align (up-down-up-down pattern).

The second term is the transverse field term and represents the interaction of each spin with an external magnetic field in the x-direction, i.e. orthogonal (transverse) to the z-axis. The operator X_i flips the spin at site i between up and down in the z-basis. Hence, this term introduces quantum fluctuations. It causes the spins to superpose between the up and down states.

The Ising term and the field term compete. The first wants to align (or anti-align) all spins along z. The second wants to align all spins along x, which are superpositions in the z-basis. Both is not possible simultaneously, which makes the system non-trivial and interesting. The specific values of J and $h > 0$ determine the ground state and its energy.

[7] Spin up along z-axis is the qubit state $|0\rangle$, and spin down is the state $|1\rangle$, just as visualized on the Bloch sphere.

[8] Again, an unfortunate notational collision, as we had J denoting the number of terms in the decomposition of the Hamiltonian (3.9).

There are three regimes:

- $|J| \gg h$: Ising interaction dominates: (Anti-)Ferromagnetic regime: spins (anti-)align along z (classical order)
- $|J| \ll h$: Transverse field dominates: Paramagnetic regime: spins align along x, i.e. quantum superposition of z-states
- $|J| \approx h$: No term dominates: Quantum critical point, non-tivial correlations and strong entanglement

In summary, the one-dimensional transverse-field Ising model describes a chain of quantum spins that try to align with each other along the z-axis (Ising coupling), but are simultaneously being flipped between up and down along z by a magnetic field along the x-axis (transverse field). It is one of the simplest models showing a quantum phase transition between classical and quantum ordered phases.

For three qubits, i.e. $N = 3$, the Hamiltonian becomes[9]

$$H = -J\,(Z_1Z_2 + Z_2Z_3) - h\,(X_1 + X_2 + X_3). \tag{3.12}$$

The goal of the VQE algorithm is to find the ground-state energy

$$E_0 = \min_{\boldsymbol{\theta}} \langle \psi(\boldsymbol{\theta}) | H | \psi(\boldsymbol{\theta}) \rangle, \tag{3.13}$$

where $|\psi(\boldsymbol{\theta})\rangle$ is a parameterized quantum state prepared by a quantum circuit with parameters $\boldsymbol{\theta}$.

A simple hardware-efficient ansatz for three qubits can be constructed using single-qubit rotations followed by entangling gates. For example,

$$W(\boldsymbol{\theta}) = \mathrm{CNOT}_{3,1}\,\mathrm{CNOT}_{2,3}\,\mathrm{CNOT}_{1,2} \prod_{i=1}^{3} R_y^{(i)}(\theta_i), \tag{3.14}$$

where $R_Y^{(i)}$ is a single-qubit rotation on qubit i, and the CNOT gates introduce entanglement between neighboring qubits.

Starting from a product state, where all qubits are in the 0 state, the trial (i.e. variational) state is then

$$|\psi(\boldsymbol{\theta})\rangle = W(\boldsymbol{\theta})\,|000\rangle. \tag{3.15}$$

The expectation value of the Hamiltonian (3.12) in the variational state is

$$C(\boldsymbol{\theta}) = \langle \psi(\boldsymbol{\theta}) | H | \psi(\boldsymbol{\theta}) \rangle. \tag{3.16}$$

Since H is a sum of Pauli strings, this energy can be evaluated as

$$C(\boldsymbol{\theta}) = -J\,(\langle Z_1Z_2 \rangle_{\boldsymbol{\theta}} + \langle Z_2Z_3 \rangle_{\boldsymbol{\theta}}) - h\,(\langle X_1 \rangle_{\boldsymbol{\theta}} + \langle X_2 \rangle_{\boldsymbol{\theta}} + \langle X_3 \rangle_{\boldsymbol{\theta}}), \tag{3.17}$$

[9] In full notation, which almost nobody ever uses, the Hamiltonian reads $H = -J\,(Z_1 \otimes Z_2 \otimes \mathbb{1} + \mathbb{1} \otimes Z_2 \otimes Z_3) - h\,(X_1 \otimes \mathbb{1} \otimes \mathbb{1} + \mathbb{1} \otimes X_2 \otimes \mathbb{1} + \mathbb{1} \otimes \mathbb{1} \otimes X_3)$.

where each expectation value $\langle P \rangle_{\boldsymbol{\theta}}$ is measured on the quantum device.

With this decomposition, the VQE algorithm runs its hybrid quantum–classical loop until convergence. The resulting minimum energy E_{min} approximates the true ground state energy E_0, and the corresponding circuit parameters $\boldsymbol{\theta}_{\text{min}}$ define a quantum state that approximates the ground state.

For this small three-qubit system, the Hamiltonian acts on a Hilbert space of dimension $2^3 = 8$, so the exact solution can be computed classically for benchmarking. However, the VQE approach illustrates how quantum and classical resources can be combined to estimate ground-state properties of more complex systems that would be intractable for a purely classical simulation.

3.2 Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA) is very similar to the VQE. However, instead of a quantum Hamiltonian a classical Hamiltonian is used, i.e. a matrix which is diagonal (in the computational basis representation). Such Hamiltonians can be used to encode interesting problems in the field of combinatorial optimization.

The combinatorial problem at hand is represented by an objective function C, which is represented by a cost Hamiltonian H_C. (We will discuss an example of such a cost function and its Hamiltonian shortly.) This Hamiltonian gives rise to a unitary evolution operator $U(C,\gamma)$:

$$H_C \;\rightarrow\; U(C,\gamma) = e^{-i\gamma H_C}, \tag{3.18}$$

where γ represents the time (or strength) for (or with) which the Hamiltionian is acting onto the qubits. We also need Pauli X operators, which introduce transitions between computational basis states, “mixing” them so the algorithm can explore the search space. The standard choice is the mixer Hamiltonian H_B which gives rise to the mixing evolution $U(B,\beta)$

$$H_B = \sum_{i=1}^{n} X_i \;\rightarrow\; U(B,\beta) = e^{-i\beta H_B}. \tag{3.19}$$

If you only had the cost Hamiltonian, which is diagonal in the computational basis, we would get only changes of phases of basis states, but no changes of their amplitudes. Therefore, we could never increase the amplitude (and hence the probability) of the optimal bit string. The algorithm would be completely static in terms of probabilities, and the optimization would fail. Think of H_C as giving each state a “score” (or energy). The mixer B lets you move between states. Without the mixer, you can only look at the scores, but never move towards a better one.[10]

The QAOA procedure is illustrated in Fig. 3.3 and reads as follows:

[10] In the later discussion on quantum annealing, we will further motivate this procedure.

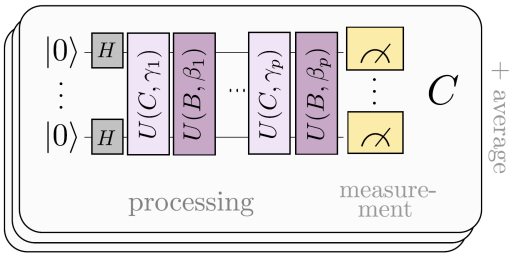


Figure 3.3: Working scheme of the quantum approximate optimization algorithm. A cost function C of a combinatorial optimization problem is encoded into a Hamiltonian, giving rise to the unitary evolution $U(C,\gamma)$. The ansatz further uses a mixing evolution $U(B,\beta)$. Multiple layers of these two evolutions are applied. The task is to maximize the expectation value of the Hamiltonian. Image taken from Ref. [5].

1. Prepare the uniform n-qubit superposition state

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{z\in\{0,1\}^n} |z\rangle = |+\rangle^{\otimes n}. \tag{3.20}$$

   This state can be prepared by initializing the product state $|0\rangle^{\otimes n}$ and applying a Hadamard gate to each qubit: $H^{\otimes n}|0\rangle^{\otimes n} = |+\rangle^{\otimes n}$.
2. Apply p layers of the cost and mixer unitaries to obtain the variational (parametrized) state

$$|\psi(\boldsymbol{\gamma},\boldsymbol{\beta})\rangle = U_B(\beta_p)\,U_C(\gamma_p)\,\ldots\,U_B(\beta_1)\,U_C(\gamma_1)\,|s\rangle. \tag{3.21}$$

   Here, $\boldsymbol{\gamma} = (\gamma_1,\ldots,\gamma_p)$ and $\boldsymbol{\beta} = (\beta_1,\ldots,\beta_p)$ are the vectors summarizing all parameters.
3. The parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are optimized to maximize

$$C(\boldsymbol{\gamma},\boldsymbol{\beta}) = \langle\psi(\boldsymbol{\gamma},\boldsymbol{\beta})|H_C|\psi(\boldsymbol{\gamma},\boldsymbol{\beta})\rangle, \tag{3.22}$$

   or minimize $-C(\boldsymbol{\gamma},\boldsymbol{\beta})$, respectively.

Example: Max-Cut on a Square Graph

Consider an undirected graph $G = (V,E)$, where set of vertices V, and the set of edges E form a square:

$$V = \{0,1,2,3\}, \quad E = \{(0,1),(1,2),(2,3),(3,0)\}. \tag{3.23}$$

The goal of the maximum cut (Max-Cut) problem is to partition the vertices into two sets A and B so that the number of edges between the two sets is maximized (see Fig. 3.4).

Finding the best cut is an NP-complete problem.[11] It is very unlikely that the solution

[11] In computational complexity theory, P is the class of problems which can be solved efficiently (i.e. quickly, i.e. in polynomial time). NP is the class of problems that can be verified efficiently. It is widely

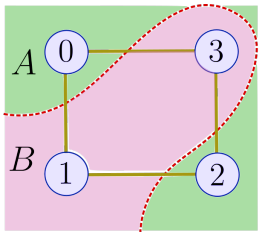


Figure 3.4: The task of maximum cut is to find the maximum number of edges (yellow lines) in a graph which are "cut" by a given partition of the vertices (blue circles) into two sets A and B, i.e. where one vertex of an edge belongs to set A, and the other vertex of the same edge belongs to set B. The maximum cut for a square graph is 4. Image taken from *here*.

can be found in polynomial time. But one can hope for a polynomial-time algorithm that finds a good approximation.

Classical formulation: Each vertex i is assigned a bit $z_i \in \{0,1\}$, or equivalently, a spin variable $s_i = (-1)^{z_i} \in \{-1,+1\}$. Spins that are up ($z_i = 0$) belong to set A, and spins that are down ($z_i = 1$) belong to set B. An edge (i,j) contributes the value 1 to the cut if $z_i \neq z_j$, or equivalently if $s_i s_j = -1$. Otherwise, i.e. if $s_i s_j = +1$, the contribution is 0. Hence, the objective function to maximize is

$$C(z) = \sum_{(i,j)\in E} \frac{1 - s_i s_j}{2}. \tag{3.24}$$

For the square graph, this becomes

$$C(z) = \frac{1}{2}\big[(1 - s_0 s_1) + (1 - s_1 s_2) + (1 - s_2 s_3) + (1 - s_3 s_0)\big]. \tag{3.25}$$

The maximum cut value for this graph is $C_{\max} = 4$, obtained when alternating vertices are assigned opposite bit values (e.g. 0101 or 1010). To find this assignment for such a small graph is easily doable, even by hand. But for a large graph with n vertices, you need to go through 2^n possible configurations, which becomes infeasible already for moderately large n.

Quantum formulation: In QAOA, we map the cost function to a classical (i.e. diagonal) Ising-type Hamiltonian by replacing the spin variables by spin operators:

$$s_i \;\to\; Z_i \tag{3.26}$$

Multiplication is replaced by the tensor product. Of course, these symbols are usually

believed, though not known, that many problems in NP cannot be solved efficiently (i.e. need exponential time). This is the conjecture that P $\neq$ NP. (A proof or disproof of this "P versus NP" problem would get you a million dollars from the Clay Mathematics Institute.) NP-complete problems are the most difficult problems in NP, as every problem in NP can be reduced efficiently to every NP-complete problem. If any NP-complete problem could be solved efficiently, then all problems in NP could be solved efficiently. See Figure 4.2 in the next chapter.

suppressed anyway:

$$s_i \times s_j = s_i s_j \;\rightarrow\; Z_i \otimes Z_j = Z_i Z_j \tag{3.27}$$

The cost Hamiltonian then reads:

$$\begin{aligned} H_C &= \frac{1}{2}\left(4\,\mathbb{1} - Z_0 Z_1 - Z_1 Z_2 - Z_2 Z_3 - Z_3 Z_0\right) \\ &= \mathrm{diag}(0,2,2,2,2,4,2,2,2,2,4,2,2,2,2,0). \end{aligned} \tag{3.28}$$

The goal is to prepare a quantum state $|\psi(\boldsymbol{\gamma},\boldsymbol{\beta})\rangle$ that maximizes the expectation value

$$\langle H_C \rangle_{\boldsymbol{\gamma},\boldsymbol{\beta}} = \langle \psi(\boldsymbol{\gamma},\boldsymbol{\beta}) | H_C | \psi(\boldsymbol{\gamma},\boldsymbol{\beta}) \rangle. \tag{3.29}$$

Let us enumerate the states by the natural numbers that correspond to their binary representation, $|0\rangle = |0000\rangle$, $|1\rangle = |0001\rangle$, …, $|15\rangle = |1111\rangle$. We can see from the Hamiltonian that the states $|5\rangle = |0101\rangle$ and state $|10\rangle = |1010\rangle$ lead to the best cut, as expected.

The QAOA algorithm with depth $p = 1$ can already produce a high-quality approximation. For $p = 2$, the expectation $\langle H_C \rangle_{\boldsymbol{\gamma},\boldsymbol{\beta}}$ can reach the optimal cut value $C_{\max} = 4$ with almost certainty.

4. Adiabatic Quantum Computation

Adiabatic Quantum Computation is a quantum computational technique designed to solve optimization problems by exploiting the principles of quantum mechanics, particularly via the quantum tunneling mechanism. It provides an alternative to gate-based quantum computation and is especially suited for problems that can be formulated as finding the global minimum of an energy landscape. While the quantum circuit model is often described as "digital", adiabatic quantum computation can be seen as "analog".

The method is based on the adiabatic theorem of quantum mechanics, which states that a system will remain in its ground state if its Hamiltonian changes sufficiently slowly.[1] The system starts in the easily prepared ground state of an initial Hamiltonian H_0 and gradually evolves to a problem Hamiltonian H_P whose ground state encodes the optimal solution to a given problem. Mathematically, the total Hamiltonian can be written as

$$H(t) = (1 - s(t))\, H_0 + s(t)\, H_P \tag{4.1}$$

where $s(t)$ varies smoothly from $s(0) = 0$ to $s(\tau) = 1$ while the annealing time proceeds from $t = 0$ to $t = \tau$. If the evolution is slow enough, the system remains in the ground state of $H(t)$ throughout the process, yielding the solution, i.e. the ground state of H_P, when $s(\tau) = 1$. The Hamiltonian H_0 is also called driver Hamiltonian as it drives the system toward the ground state of the separate problem Hamiltonian.

4.1 Quantum Annealing

While adiabatic quantum computation is a theoretical idealization which guarantees to find the ground state if the evolution is slow enough, *quantum annealing* is a heuristic realization which works in noisy systems and approximates the ground state with current hardware

[1] The adiabatic theorem (due to Max Born and Vladimir Fock) from the year 1928 states: If a quantum system starts in the ground state of some Hamiltonian and evolves slowly enough to another Hamiltonian, it will remain in the ground state of the instantaneous Hamiltonian throughout the evolution, thus ending in the ground state of the final Hamiltonian.

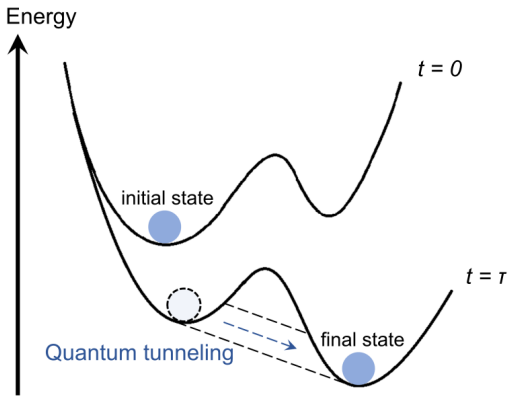


Figure 4.1: Quantum annealing: A system is initially, at time $t = 0$, in the ground state of a given Hamiltonian, i.e. in the minimum of its energy landscape. The Hamiltonian is then slowly changed to a problem (or target) Hamiltonian which is reached at a final time $t = \tau$. If the change happens slow enough, i.e. adiabatically, the system remains – potentially due to quantum tunneling – in the ground state throughout the whole evolution. Image taken from the Supplementary Information of Ref. [2].

restrictions. Unlike classical simulated annealing, which relies on thermal fluctuations to escape local minima, quantum annealing leverages quantum tunneling. This allows the system to penetrate through energy barriers rather than climb over them, potentially leading to faster convergence to the global minimum in certain energy landscapes (see Fig. 4.1. While the probability of quantum tunneling decreases exponentially with the barrier width, it only scales subexponentially with its height. That makes quantum annealing especially useful when the objective function has sharp and narrow structures.

Quantum annealing represents a promising approach to solving hard optimization problems by harnessing quantum mechanical effects. It has been applied to diverse fields including logistics, machine learning, material science, and financial modeling. However, it faces challenges such as decoherence, control errors, and the limited connectivity of physical qubit architectures. Additionally, the extent of its speedup over classical algorithms remains an active area of research. While current hardware is still limited in scale and performance, continuous progress in qubit technology and error mitigation techniques may enable quantum annealers to outperform classical optimization methods in the future.

4.2 Trotterized Quantum Annealing

As mentioned above, in quantum annealing, a system evolves under a continuous, time-dependent Hamiltonian

$$H(t) = A(t)\,H_0 + B(t)\,H_P, \tag{4.2}$$

where the initial (or driver) Hamiltonian is H_0 (called mixer Hamiltonian H_B above) and the problem Hamiltonian is H_P (called cost Hamiltonian H_C above). The function $A(t)$

starts at value 1 for time $t = 0$ and steadily decreases until it ends at value 0 for the final time $t = \tau$. And $B(t)$ starts at value 0 and increased to 1. The corresponding time-evolution operator for this full process is

$$U(\tau) = \mathcal{T} \exp\left(-i \int_0^\tau H(t)\, dt \right), \tag{4.3}$$

where $\mathcal{T}$ denotes time ordering.[2]

To simulate this evolution on a gate-based quantum computer, the total time τ is divided into p discrete steps of duration $\Delta t = \tau / p$:

$$\int_0^\tau H(t)\, dt \approx \sum_{k=1}^{p} H(k\Delta t)\Delta t. \tag{4.4}$$

During each short interval $k = 1, 2, \ldots, p$, the Hamiltonian is approximately constant, so that[3]

$$e^{-i\Delta t[A_k H_0 + B_k H_P]} \approx e^{-i\Delta t A_k H_0}\, e^{-i\Delta t B_k H_P}, \tag{4.5}$$

where $A_k = A(k\Delta t)$, $B_k = B(k\Delta t)$. This approximation follows from the first-order Trotter–Suzuki expansion and is valid when Δt is small. Defining

$$\beta_k = \Delta t\, A_k, \quad \gamma_k = \Delta t\, B_k, \tag{4.6}$$

the discretized evolution becomes

$$U(\tau) \approx \prod_{k=1}^{p} e^{-i\beta_k H_0}\, e^{-i\gamma_k H_P}. \tag{4.7}$$

This is precisely the structure of the QAOA ansatz,

$$|\psi_p(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = \prod_{k=1}^{p} e^{-i\beta_k H_0}\, e^{-i\gamma_k H_P} |\psi_0\rangle, \tag{4.8}$$

where $|\psi_0\rangle$ is the initial ground state of H_0, and $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$ are the parameter vectors.

Hence, the alternating application of the problem and mixer layers in QAOA can be viewed as a Trotterized (digitized) version of the continuous quantum annealing process. In the limit $p \to \infty$ and $\Delta t \to 0$, the QAOA evolution converges to the adiabatic evolution of quantum annealing.

---

[2] This is the solution to the time-dependent Schrödinger equation $i\hbar \frac{\mathrm{d}|\psi\rangle}{\mathrm{d}t} = H|\psi\rangle$, where we choose physical units such that $\hbar = 1$.

[3] Note that the matrices H_0 and H_P generally do not commute, i.e. $[H_0, H_P] = H_0 H_P - H_P H_0 \neq 0$. Then $e^{H_0 + H_P} = e^{H_P + H_0} \neq e^{H_0} e^{H_P}$.

4.3 Quadratic Unconstrained Binary Optimization

Many combinatorial optimization problems can be mapped onto the Ising Hamiltonian

$$H_P = -\sum_{i<j} J_{ij} Z_i Z_j - \sum_i h_i Z_i. \tag{4.9}$$

It is often assumed that the coupling J_{ij} is non-zero only for neighboring spins. If (almost) all the couplings are positive, we get a ferromagnet. If it is (almost) always negative, the coupling term strives to anti-align the spins (anti-ferromagnet), while the field term tries to align them. If positive and negative couplings appear (approximately) equally often, we speak of a spin glass.

The goal of a quantum annealer is to find the spin configuration that minimizes H_P by starting from the initial Hamiltonian $H_0 = \sum_i X_i$, for which the ground state solution is known, and then following the annealing schedule (4.1). Quantum hardware physically implements such Ising models using, e.g. superconducting qubits, coupled through programmable interactions J_{ij}.

In Quadratic Unconstrained Binary Optimization (QUBO), we seek a binary vector $\mathbf{x} \in \{0,1\}^n$ that minimizes a quadratic objective function:

$$\min_{\mathbf{x}\in\{0,1\}^n} f(\mathbf{x}), \quad f(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} = \sum_i Q_{ii} x_i + \sum_{i\neq j} Q_{ij} x_i x_j, \tag{4.10}$$

where $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is a (not necessarily symmetric) weight matrix. Typically, a symmetrized version $\frac{\mathbf{Q}+\mathbf{Q}^T}{2}$ is used, so that the terms $x_i x_j$ appear with weights $Q_{ij} = Q_{ji}$. Diagonal entries Q_{ii} represent linear costs associated with $x_i = 1$.

QUBO is a very expressive framework: many combinatorial optimization problems (e.g., maximum cut, partition, or binary programming with parity constraints) can be reformulated in this way, often by encoding the constraints as penalty terms inside $\mathbf{Q}$. Areas of application reach from finance and economics to machine learning. QUBO is an NP-hard problem. All NP-complete problems are NP-hard. But there are NP-hard problems that are not NP-complete and thus not in NP (see Fig. 4.2).

To link QUBO to physical spin systems, one maps binary variables $x_i \in \{0,1\}$ to spin variables $s_i \in \{-1,+1\}$ using

$$s_i = 2x_i - 1 \quad \text{or equivalently} \quad x_i = \frac{1+s_i}{2}. \tag{4.11}$$

Substituting this into (4.10) and rearranging constants gives the Ising energy function

$$E(\boldsymbol{s}) = -\sum_{i<j} J_{ij} s_i s_j - \sum_i h_i s_i + \text{const.}, \tag{4.12}$$

with parameters J_{ij} (couplings) and h_i (local fields). Thus, every QUBO problem is equivalent to an Ising problem up to additive constants.

Physical quantum annealers (such as those built by D-Wave) implement H_P in the form of an Ising Hamiltonian. A QUBO problem must first be translated into the equivalent set

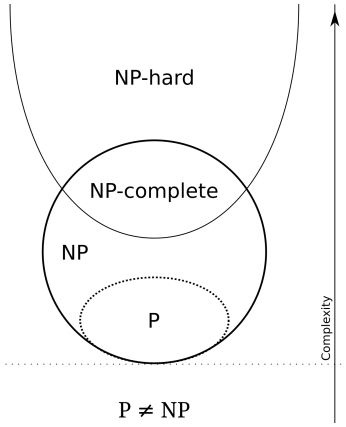


Figure 4.2: Euler diagram for P, NP, NP-complete, and NP-hard sets of problems under the assumption that P $\neq$ NP. Image taken from Wikipedia.

of J_{ij} and h_i and then implemented physically in the quantum hardware. Typically, spins can only be coupled to their next neighbors but not to all other spins, which can limit the computational capacity.

In summary, QUBO provides a compact and general framework for binary nonlinear optimization problems. Because it maps directly onto the Ising Hamiltonian, QUBO forms the natural bridge to quantum annealing hardware: problems are encoded as energy landscapes, and quantum annealing heuristically searches for their lowest valleys, i.e. good or even optimal solutions of the original QUBO problem.

5. Implementation Techniques

In this chapter, we will review some important techniques concerning the implementation of quantum circuits.

5.1 Hadamard Test

In the variational algorithms discussed above, measurements of (tensor products of) Pauli operators are made. Pauli Z operators can be measured directly since they are diagonal in the computational basis. Pauli X operators can be measured after a suitable basis change, i.e. a Hadamard gate.

In some quantum algorithms, however, we may be interested in measuring non-Pauli operators or measuring overlaps (i.e. inner products) between two quantum states (e.g. for fidelity estimation, kernel evaluation, or complex amplitudes). Consider the expectation value

$$\langle\psi|U|\psi\rangle, \tag{5.1}$$

where U is a unitary operator. This quantity contains information about the overlap between the original state $|\psi\rangle$ and the evolved state $U|\psi\rangle$. It is, in general, complex and thus cannot be directly measured in a single quantum experiment, because quantum measurements yield probabilities (real numbers between 0 and 1), not complex amplitudes.

The *Hadamard test* provides a way to estimate either the real or imaginary part of $\langle\psi|U|\psi\rangle$ by using an ancilla qubit and a controlled-unitary operation (see Figure 5.1).

The procedure reads as follows:

1. The initial state is $|0\rangle_a|\psi\rangle$, where the index a denotes the ancilla qubit.
2. The ancilla qubit is evolved with a Hadamard gate: $\frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a)|\psi\rangle$.
3. A controlled-U gate applies U to $|\psi\rangle$ if the ancilla is in state $|1\rangle_a$, and otherwise the identity. The resulting state is $\frac{1}{\sqrt{2}}(|0\rangle_a|\psi\rangle + |1\rangle_a U|\psi\rangle)$.
4. A second Hadamard is applied to the ancilla: $|\Psi\rangle = \frac{1}{\sqrt{2}}\left(\frac{|0\rangle_a+|1\rangle_a}{\sqrt{2}}|\psi\rangle + \frac{|0\rangle_a-|1\rangle_a}{\sqrt{2}}U|\psi\rangle\right) =$

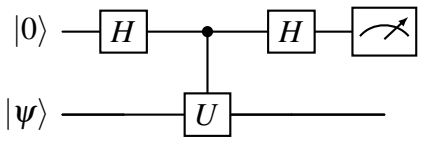


Figure 5.1: Hadamard test: The expectation value $\langle\psi|U|\psi\rangle$ can be measured with the help on an ancilla qubit, Hadamard gates, and a controlled-U operation.

$\frac{1}{2}\left[|0\rangle_a(\mathbb{1}+U)|\psi\rangle+|1\rangle_a(\mathbb{1}-U)|\psi\rangle\right]$

5. Measurement of the ancilla in the computational basis.

The probability of measuring the ancilla in state $|0\rangle$ (outcome $+1$) is given by Born's rule:

$$\begin{aligned}P(0) &= |\langle 0|_a|\Psi\rangle|^2 = |\tfrac{1}{2}(\mathbb{1}+U)|\psi\rangle|^2 \\ &= \tfrac{1}{4}\langle\psi|(\mathbb{1}+U^\dagger)(\mathbb{1}+U)|\psi\rangle = \tfrac{1}{4}\langle\psi|(\mathbb{1}+U^\dagger+U+U^\dagger U)|\psi\rangle \\ &= \tfrac{1}{4}(2+\overline{\langle\psi|U|\psi\rangle})+\langle\psi|U|\psi\rangle) \\ &= \tfrac{1}{2}(1+\mathrm{Re}\langle\psi|U|\psi\rangle).\end{aligned} \tag{5.2}$$

The probability for measuring $|1\rangle_a$ (outcome -1) is $P(1) = 1-P(0)$. The expected value of the measurement result of the ancilla is therefore

$$P(0)-P(1) = P(0)-(1-P(0)) = 2P(0)-1 = \mathrm{Re}\langle\psi|U|\psi\rangle. \tag{5.3}$$

To obtain the expectation value corresponding to the imaginary part, $\mathrm{Im}\langle\psi|U|\psi\rangle$, the procedure above needs to be followed, but in step 2, the state $\frac{1}{\sqrt{2}}(|0\rangle_a - i|1\rangle_a|\psi\rangle$ should be prepared with the help of an S gate.

Finally, the desired expectation value (5.1) is given by the sum of the real and the imaginary part:

$$\langle\psi|U|\psi\rangle = \mathrm{Re}\langle\psi|U|\psi\rangle + i\,\mathrm{Im}\langle\psi|U|\psi\rangle. \tag{5.4}$$

We can conclude: The Hadamard test transforms the problem of estimating a complex amplitude into the problem of measuring a classical probability difference of the ancilla outcomes. By using interference between the branches where U is and is not applied, the real and imaginary parts of $\langle\psi|U|\psi\rangle$ are encoded in the bias of the ancilla measurement statistics.

5.2 Parameter Shift Rule

The output of a variational circuit, i.e. the expectation of an observable A, can be written as a parametrized "quantum function"

$$f(\boldsymbol{\theta}) = \langle A\rangle_{\boldsymbol{\theta}}, \tag{5.5}$$

where $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots)$ is a vector of parameters. The training process typically uses gradient descent and therefore requires to calculate the derivative of this function. One straightforward way is to use the method of finite difference:

$$\frac{\partial f}{\partial \theta_j} \approx \frac{f(\theta_1, \theta_2, \ldots, \theta_j + \varepsilon, \ldots) - f(\theta_1, \theta_2, \ldots, \theta_j - \varepsilon, \ldots)}{2\,\varepsilon}. \tag{5.6}$$

Here, the partial derivative of f with respect to a θ_j is calculated via evaluating the function a small step to the left and to the right of the parameter value. In noisy quantum hardware, however, it is often problematic to rely on this method. Too large values of ε typically lead to inaccurate results. And if ε is too small, the two shifted evaluations of the circuit are overcome by noise.

The *parameter-shift rule* is a solution to this problem. In many cases, the derivative can be expressed as a linear combination of the quantum function f itself. Thus, notably, the computation of gradients in variational quantum circuits can be achieved without modifying the circuit architecture itself.

To illustrate this with a classically computable function, let us consider the following function and its first derivative:

$$f(\theta) = \sin\theta, \quad \frac{\mathrm{d}f(\theta)}{\mathrm{d}\theta} = \cos\theta. \tag{5.7}$$

Now let us entertain the thought that for some reason we only can compute the sine function itself but don't have easy access to the cosine function. Fortunately, the derivative can also be computed by only using f, namely via

$$\frac{\mathrm{d}f(\theta)}{\mathrm{d}\theta} = \tfrac{1}{2}\big[\sin(x+\tfrac{\pi}{2}) - \sin(x-\tfrac{\pi}{2})\big]. \tag{5.8}$$

This resembles the finite-difference method for computing derivatives. But here, the shift is finite, and not small, let alone not infinitesimal. And the derivative is exact, not approximated.

This useful characteristic holds for many quantum functions, i.e. the same circuit can be used to compute both the quantum function itself and its gradient. Consider a quantum circuit in which multiple gates depend on distinct parameters, i.e.

$$U(\boldsymbol{\theta}) = \prod_{j=1}^{m} U_j(\theta_j), \tag{5.9}$$

The expectation value of an observable A is given by

$$f(\boldsymbol{\theta}) = \langle A \rangle_{\boldsymbol{\theta}} = \langle 0 | U^\dagger(\boldsymbol{\theta}) A\, U(\boldsymbol{\theta}) | 0 \rangle. \tag{5.10}$$

Under appropriate conditions, e.g. when each generator has only two distinct eigenvalues, the partial derivative of f with respect to one parameter θ_j can be evaluated via the parameter-shift rule by keeping all other parameters fixed while shifting θ_j to the left and

to the right:

$$\frac{\partial}{\partial\theta_j}f(\boldsymbol{\theta}) = c\left[f(\theta_1,\ldots,\theta_j+s,\ldots,\theta_m) - f(\theta_1,\ldots,\theta_j-s,\ldots,\theta_m)\right], \tag{5.11}$$

where the constant c and shift amount s are determined by the spectral properties of the generator. The key point is that the circuit architecture remains unchanged; only the value of θ_j is modified, and no additional gates are introduced.

As a concrete example, let us assume that the j-th gate is single-qubit rotation

$$U_j(\theta_j) = e^{-i\frac{\theta_j}{2}P_j}, \tag{5.12}$$

where $P_j \in \{X,Y,Z\}$ is a Pauli gate and called the generator of the evolution.[1] We have $P_j^2 = \mathbb{1}$, i.e. the Pauli matrices all have eigenvalues $+1$ and -1.[2] The derivative of this unitary is

$$\frac{\partial}{\partial\theta_j}U_j(\theta_j) = -\tfrac{i}{2}P_jU_j(\theta_j) = -\tfrac{i}{2}U_j(\theta_j)P_j. \tag{5.13}$$

The last equality sign follows from the fact that any Pauli matrix P_j commutes with its own unitary evolution as it commutes with itself.

We are interested only in θ_j. So, let us write our unitary as

$$U(\boldsymbol{\theta}) = V(\boldsymbol{\theta}_V)U_j(\theta_j)W(\boldsymbol{\theta}_W). \tag{5.14}$$

Here, $W(\boldsymbol{\theta}_W) = U_{j-1}(\theta_{j-1})\ldots U_1(\theta_1)$ and $V(\boldsymbol{\theta}_V) = U_m(\theta_m)\ldots U_{j+1}(\theta_{j+1})$ represent all prior and later gates, respectively, and $\boldsymbol{\theta}_W = (\theta_1,\ldots,\theta_{j-1})$ and $\boldsymbol{\theta}_V = (\theta_{j+1},\ldots,\theta_m)$ are their respective parameter lists.

In the following, we will suppress all other parameters except θ_j in the notation, as they are held constant. Our quantum function hence is

$$\begin{aligned} f(\boldsymbol{\theta}) &= f(\boldsymbol{\theta}_W,\theta_j,\boldsymbol{\theta}_V) = \langle 0|U^\dagger(\boldsymbol{\theta})AU(\boldsymbol{\theta})|0\rangle \\ &=: f(\theta_j) = \langle 0|W^\dagger U_j^\dagger(\theta_j)V^\dagger AVU_j(\theta_j)W|0\rangle. \end{aligned} \tag{5.15}$$

Let us absorb W into the state and V into the observable:

$$|\psi\rangle := W|0\rangle, \tag{5.16}$$

$$B := V^\dagger AV. \tag{5.17}$$

Then, we can write

$$f(\theta_j) = \langle\psi|U_j^\dagger(\theta_j)BU_j(\theta_j)|\psi\rangle. \tag{5.18}$$

[1] The index j does not necessarily mean that the gate acts on the j-th qubit, but rather that it is the j-th gate in the circuit.

[2] Due to $P_j^2 = \mathbb{1}$ the unitary can be written as $U_j(\theta_j) = \exp(-i\frac{\theta_j}{2}P_j) = \cos(\theta_j)\mathbb{1} - i\sin(\theta_j)P_j$.

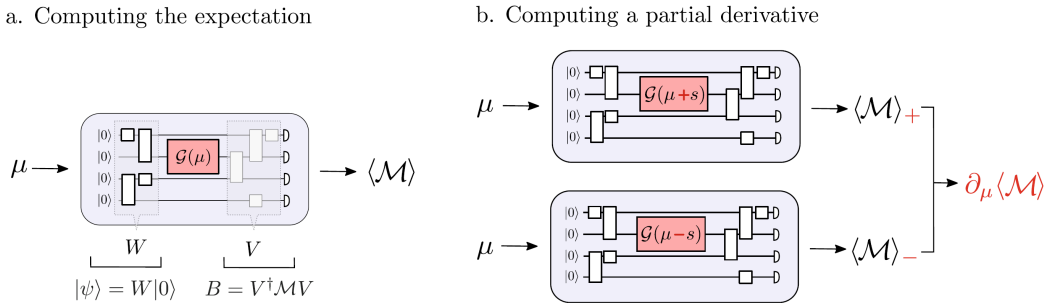


Figure 5.2: Illustration of the parameter-shift rule. Accordance with the main text is achieved via $A \equiv \mathcal{M}$ and $U_j(\theta_j) \equiv \mathcal{G}(\mu)$. The partial derivative of the expectation value can be calculated by a linear combination of two expectation values, in each of which the original gate parameter is shifted by a constant amount s. Image taken from Ref. [5].

Now we can calculate the derivative, using the product rule of differentiation:

$$\begin{aligned}
\frac{\partial}{\partial\theta_j} f(\theta_j) &= \frac{\partial}{\partial\theta_j} \langle\psi| U_j^\dagger(\theta_j) B U_j(\theta_j) |\psi\rangle \\
&= \langle\psi| [\tfrac{\partial}{\partial\theta_j} U_j^\dagger(\theta_j)] B U_j(\theta_j) |\psi\rangle + \langle\psi| U_j^\dagger(\theta_j) B [\tfrac{\partial}{\partial\theta_j} U_j(\theta_j)] |\psi\rangle \\
&= \langle\psi| \tfrac{i}{2} U_j^\dagger(\theta_j) P_j B U_j(\theta_j) |\psi\rangle + \langle\psi| U_j^\dagger(\theta_j) B \tfrac{-i}{2} P_j U_j(\theta_j) |\psi\rangle \\
&= \tfrac{i}{2} \langle\psi| U_j^\dagger(\theta_j) [P_j, B] U_j(\theta_j) |\psi\rangle .
\end{aligned} \tag{5.19}$$

Here, $[P_j, B] = P_j B - B P_j$ denotes the commutator. For Pauli operators P_j, the following identity holds:

$$[P_j, B] = -i\big[U_j^\dagger(\tfrac{\pi}{2}) B U_j(\tfrac{\pi}{2}) - U_j^\dagger(-\tfrac{\pi}{2}) B U_j(-\tfrac{\pi}{2})\big]. \tag{5.20}$$

If we substitute this into (5.19), we get

$$\begin{aligned}
\frac{\partial}{\partial\theta_j} f(\theta_j) &= \tfrac{1}{2} \langle\psi| U_j^\dagger(\theta_j) [U_j^\dagger(\tfrac{\pi}{2}) B U_j(\tfrac{\pi}{2}) - U_j^\dagger(-\tfrac{\pi}{2}) B U(-\tfrac{\pi}{2})] U_j(\theta_j) |\psi\rangle \\
&= \tfrac{1}{2} \langle\psi| U_j^\dagger(\theta_j + \tfrac{\pi}{2}) B U_j(\theta_j + \tfrac{\pi}{2}) |\psi\rangle - \tfrac{1}{2} \langle\psi| U_j^\dagger(\theta_j - \tfrac{\pi}{2}) B U_j(\theta_j - \tfrac{\pi}{2}) |\psi\rangle .
\end{aligned} \tag{5.21}$$

Using (5.18), this can be written in terms of quantum functions:

$$\frac{\partial}{\partial\theta_j} f(\theta_j) = \tfrac{1}{2} \big[f(\theta_j + \tfrac{\pi}{2}) - f(\theta_j - \tfrac{\pi}{2})\big]. \tag{5.22}$$

The parameter-shift rule is illustrated in Figure 5.2. It turns the derivative of an expectation value with respect to a circuit parameter into a finite difference of expectation values at shifted parameters. For Pauli generators P with $P^2 = \mathbb{1}$ the shifts are $s = \pm\pi/2$, and the derivative is the simple half-difference. This derivative is not an approximation but exact, and it is straightforward to implement on quantum hardware.

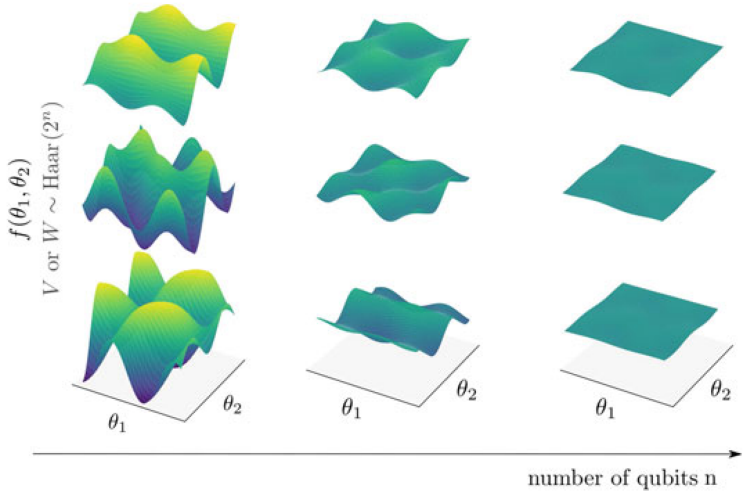


Figure 5.3: Illustration of the problem of barren plateaus. A gate with two parameters θ_1 and θ_2 is applied in the middle of a circuit. All prior gates are represented by W, all later gates by V. The gates in W and V are sampled uniformly from the set of all quantum gates (Haar measure), and all their parameters are held constant and are suppressed in the notation. The function $f(\theta_1, \theta_2)$ is the expectation value which should be optimized and whose gradient must therefore be computed. With growing number n of qubits, the function becomes flatter and flatter. If the cost f depends on these parameters, these vanishing gradients make the optimization infeasible. Image taken from Ref. [5].

5.3 Barren Plateaus

A *barren plateau* is a region of parameter space where the gradients of the quantum function are in expected value exponentially small and have exponentially small variance with respect to the system size, as measured by the number of qubits n:

$$\mathbb{E}\left[\frac{\partial f(\boldsymbol{\theta})}{\partial \theta_j}\right] = \mathcal{O}\left(\frac{1}{2^n}\right), \quad \mathrm{Var}\left[\frac{\partial f(\boldsymbol{\theta})}{\partial \theta_j}\right] = \mathcal{O}\left(\frac{1}{2^n}\right). \tag{5.23}$$

In such regions, the gradient signal is effectively lost in noise and numerical precision, making gradient-based training infeasible. Figure 5.3 illustrates this phenomenon.

Intuitively, as the quantum circuit becomes more complex (many highly entangled qubits), it starts to behave like a random unitary drawn from the Haar measure. For a random unitary, the expectation value of an observable is typically close to its average over the entire Hilbert space, and the gradients tend to cancel out. When the circuit is shallow or structured, parameter changes have a noticeable effect on the cost. As the number of qubits increases, the state space becomes exponentially large, and random directions in this space almost cancel out on average.

In fact, there are various reasons why the problem of barren plateaus arises:

- Expressible / random circuits: If parts of the circuit are close to random circuits, expectation values of local observables concentrate near their global averages. Small changes in parameters then produce extremely small changes in the cost.
- Global cost functions: When the observable A acts non-trivially on many qubits

(a global cost), the gradients typically average to zero and their variance scales exponentially poorly in n.

- Noise and decoherence: Real-device noise (depolarizing, amplitude damping, etc.) tends to shrink Bloch vectors toward the maximally mixed state, further flattening the landscape and accelerating the decay of gradient magnitude with circuit depth and system size.
- Initialization choice: Random initializations that effectively sample parameters from distributions that make the circuit highly mixing – meaning that the whole Hilbert space is explored, not that mixed states are produced – lead to barren plateaus at initialization.

Several techniques have been proposed and empirically validated to reduce the risk of barren plateaus:

- Use local cost functions: Define the cost to depend only on few-qubit observables or on sums of local terms. Local costs typically show much milder scaling.
- Problem-inspired ansätze: Construct ansätze that encode problem symmetries, conservation laws, or known structure, rather than using highly expressive random circuits.
- Layerwise / greedy training: Grow and train the circuit in stages (train a small circuit, then add layers and continue), which can prevent parameters from starting already in a problematic regime.
- Smart initialization: Initialize parameters near identity or use initializations that preserve gradients (e.g. narrow parameter distributions).
- Symmetry-preserving parameterization: Restrict parameter updates to subspaces consistent with symmetries to reduce expressibility and variance concentration.
- Gradient-free / hybrid optimizers: Use sampling-based or heuristic optimizers (e.g., Bayesian optimization) that can, in some cases, navigate flat landscapes better than simple gradient descent.
- Regularization and architecture tweaks: Add skip connections, shorter-depth blocks, or ancilla strategies to change expressibility and local correlations.
- Noise-aware strategies: Design circuits to be shallow and robust to noise; perform error mitigation to partially recover the gradient signal.

In classical deep learning, the most familiar form of vanishing gradients arises from the vanishing or exploding norm with depth, where gradients shrink or grow uncontrollably due to the compounding effect of activation functions and weights. This behavior contrasts with barren plateaus in variational quantum computing, where (in the absence of noise) the gradients typically vanish as the number of qubits increases rather than with circuit depth. In other words, classical networks suffer from a “curse of depth”, whereas quantum circuits face a “curse of dimension”. Moreover, in the quantum setting, the occurrence of barren plateaus is closely tied to the choice of the quantum state and the observable; different initializations and measurement settings can determine whether the loss landscape becomes exponentially flat or remains trainable

Barren plateaus are a significant practical obstacle to training parametrized quantum circuits. They arise when the training landscape becomes extremely flat due to express-

ibility, global cost choices, or noise. Understanding their origin and applying mitigation strategies (local costs, problem-informed ansätze, layerwise training, etc.) is essential for feasible variational quantum algorithms on both near-term and future quantum hardware.

6. Data Encoding

In quantum computing, one of the first steps in many algorithms is the encoding of classical data into quantum states. Several encoding strategies have been developed, each with its own advantages and trade-offs. We go through the most important in the following sections.

Figure 6.1 illustrates the important role of encoding and read-out in quantum algorithms.

6.1 Basis Encoding

A classical n-bit string $\mathbf{x} \in \{0,1\}^n$ is encoded into the computational basis state of n qubits:

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \;\rightarrow\; |\psi\rangle = |x_1 x_2 \ldots x_n\rangle. \tag{6.1}$$

This encoding requires one qubit per classical bit and is conceptually simple.

The classical bit string may represent a natural number, e.g. $7 = 0111$. Or it may encode a real number via its binary floating point representation.

One way to encode real numbers is the binary fraction representation. There, a real number $x \in]-1,1[$ is represented by $K+1$ bits b_k up to precision $\frac{1}{2^K}$ as follows:

$$x = (-1)^{b_0} \sum_{k=1}^{K} \frac{b_k}{2^k}. \tag{6.2}$$

The bit b_0 encodes the sign.

For instance, the real number $x = -0.33$ can be approximated with $K = 4$ to precision $\frac{1}{16}$ as

$$-0.33 \approx (-1)^1 \left(0\tfrac{1}{2} + 1\tfrac{1}{4} + 0\tfrac{1}{8} + 1\tfrac{1}{16}\right) = -0.3125. \tag{6.3}$$

Hence, the bit sequence for x is $(b_0 b_1 b_2 b_3 b_4) = (10101)$. And it is encoded via the action

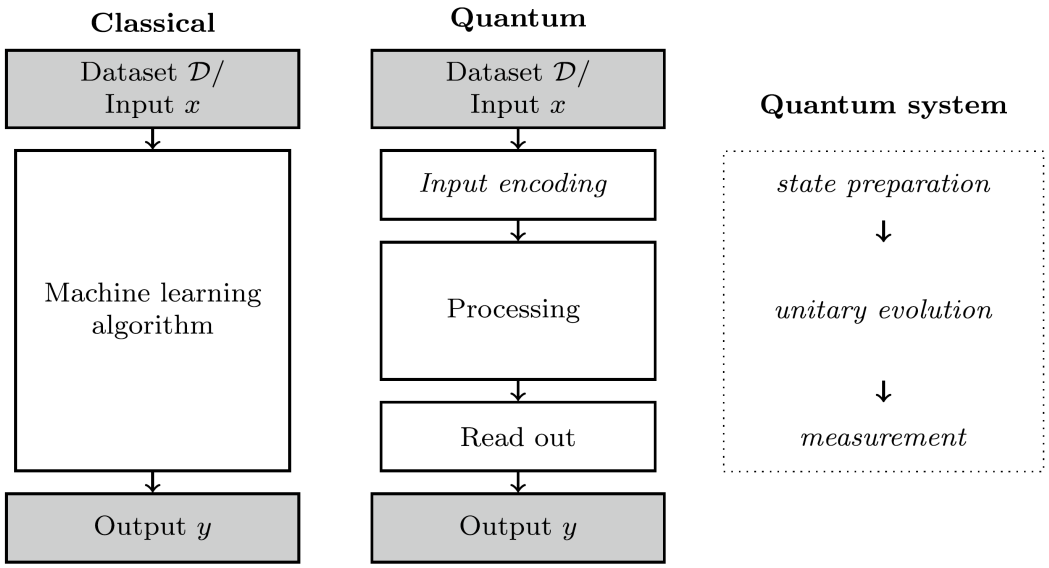


Figure 6.1: In contrast to classical machine learning methods, quantum algorithms often require special encoding and read-out schemes which are important for the success and runtime. Image and table taken from Ref. [5].

of $X\mathbb{1}X\mathbb{1}X$ onto the state $|00000\rangle$.

A vector of real numbers can be represented by the concatenation of their bit sequences.

6.2 Amplitude Encoding

Amplitude encoding embeds classical data of 2^n real numbers into the amplitudes of an n-qubit quantum state:

$$\mathbf{x} = (x_0, x_1, \ldots, x_{2^n-1})^T \;\rightarrow\; |\psi_\mathbf{x}\rangle = \sum_{i=0}^{2^n-1} x_i |i\rangle = \mathbf{x}. \tag{6.4}$$

Here, each state $|i\rangle$ denotes the i-th computational basis state corresponding to the binary representation of i. For this encoding to work, the vector $\mathbf{x}$ must be normalized. Amplitude encoding is highly efficient in terms of qubit usage, as n qubits can represent 2^n classical values simultaneously. However, preparing such a state can be computationally expensive.

Similarly, a matrix $\mathbf{A} \in \mathbb{C}^{2^n \times 2^m}$ with entries a_{ij} that obey $\sum_{i,j} |a_{ij}|^2 = 1$ can be encoded by using $m+n$ qubits:

$$\mathbf{A} \;\rightarrow\; |\psi_\mathbf{A}\rangle = \sum_{i=0}^{2^m-1} \sum_{j=0}^{2^n-1} a_{ij} |i\rangle |j\rangle . \tag{6.5}$$

If $\mathbf{A} \in \mathbb{C}^{2^n \times 2^n}$ is Hermitian and positive and has trace 1, then one can associate its entries

with the elements of an n-qubit density matrix:

$$\mathbf{A} \;\rightarrow\; \rho_{\mathbf{A}} = \sum_{i=0}^{2^n-1} \sum_{j=0}^{2^n-1} a_{ij} \left|i\right\rangle \left\langle j\right| . \tag{6.6}$$

6.3 Time-Evolution/Angle/Rotation Encoding

In time-evolution encoding, a real-valued parameter $x \in \mathbb{R}$ is associated with the evolution time t of a unitary operator generated by a Hamiltonian H as defined by

$$U(x) = e^{-ixH}. \tag{6.7}$$

The resulting quantum state after the evolution, $\left|\psi(x)\right\rangle = U(x)\left|\psi_0\right\rangle$ depends on the parameter x through the Hamiltonian H.

In quantum machine learning, this type of encoding is frequently employed to map classical, trainable parameters onto a quantum circuit. A common example is given by the Pauli rotation gates, where

$$H = \tfrac{1}{2}P, \quad P \in \{X, Y, Z\}. \tag{6.8}$$

If $P = Y$, then an initial state $\left|0\right\rangle$ is altered to

$$\left|\psi(x)\right\rangle = \cos\tfrac{x}{2}\left|0\right\rangle + \sin\tfrac{x}{2}\left|1\right\rangle . \tag{6.9}$$

This is why, this encoding is also called angle or rotation encoding.

By applying multiple time evolutions of the form $U(x)$, one can encode an entire real-valued vector $\mathbf{x} \in \mathbb{R}^N$ into the quantum system.

6.4 Hamiltonian Encoding

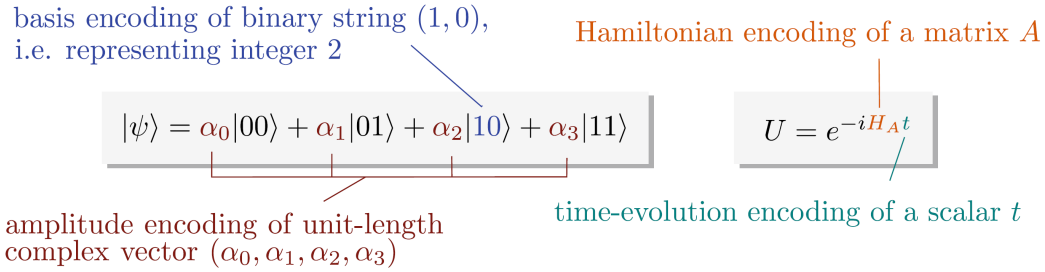
For certain applications, it is advantageous to represent matrices within the Hamiltonian governing a time evolution. This approach is, for instance, employed in the well-known HHL algorithm for matrix inversion. The main concept is to associate a Hamiltonian H with a given square matrix $\mathbf{A}$. If the matrix $\mathbf{A}$ is not Hermitian, one can embed $\mathbf{A}$ into a larger Hermitian matrix of the form

$$H_A = \begin{pmatrix} 0 & \mathbf{A} \\ \mathbf{A}^\dagger & 0 \end{pmatrix}, \tag{6.10}$$

and then perform the required operations within two corresponding subspaces of the Hilbert space.

Encoding a matrix into a Hamiltonian enables one to extract and manipulate the eigenvalues of $\mathbf{A}$, which can be utilized, for instance, to multiply $\mathbf{A}$ or its inverse $\mathbf{A}^{-1}$ by an amplitude-encoded vector.

We distinguish between analog and digital Hamiltonian simulation. The former refers



Classical data	Requirements	Quantum state
Basis encoding		
$\mathbf{x} \in \{0,1\}^{\otimes n}$	–	$\lvert\psi\rangle = \lvert x_1, \ldots, x_n\rangle$
Amplitude encoding		
$\mathbf{x} \in \mathbb{R}^{2^n}$	$\sum_i \lvert x_i\rvert^2 = 1$	$\lvert\psi_\mathbf{x}\rangle = \sum_i x_i \lvert i\rangle$
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^m}$	$\sum_{i,j} \lvert a_{ij}\rvert^2 = 1$	$\lvert\psi_\mathbf{A}\rangle = \sum_{i,j} a_{ij} \lvert i\rangle\lvert j\rangle$
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^n}$	$\sum_i a_{ii} = 1, a_{ij} = a^*_{ji}$, A pos.	$\rho_\mathbf{A} = \sum_{ij} a_{ij} \lvert i\rangle\langle j\rvert$
Time-evolution encoding		
$x \in \mathbb{R}$	$x \in [0, 2\pi[$	$U(x) = e^{-ixH}$
Hamiltonian encoding		
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^n}$	A Hermitian	$H_\mathbf{A} = \mathbf{A}$
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^n}$	–	$H_\mathbf{A} = \begin{pmatrix} 0 & \mathbf{A} \\ \mathbf{A}^\dagger & 0 \end{pmatrix}$

Figure 6.2: Summary of different encoding methods. Image and table taken from Ref. [5].

to situations in which quantum systems natively simulate or implement a Hamiltonian, while in the latter the time evolution simulation is decomposed into quantum gates via Trotterization.

Figures 6.2 summarizes the encodings that we have discussed in this chapter.

6.5 Data Encoding as a Feature Map

After having visited specific encoding schemes, let us again take a more conceptual position.

In quantum machine learning, a central conceptual step is the encoding of classical data into quantum states. Given an input space X (typically a subset of $\mathbb{R}^d$), a data-encoding strategy amounts to specifying a map

$$\phi : X \longrightarrow \mathcal{H}, \tag{6.11}$$

where $\mathcal{H}$ denotes the Hilbert space, i.e. the state space of a quantum system. Since $\mathcal{H}$ is equipped with an inner product, this mapping can naturally be interpreted as a *feature map*, analogous to those used in classical kernel methods.

In kernel-based learning, data is implicitly embedded into a high-dimensional feature space, and learning algorithms operate by evaluating inner products between feature

vectors. The quantum setting mirrors this idea. There are two principal ways to define (quantum) feature maps and inner products:

The first approach is to map the classical data $\mathbf{x} \in X$ into parametrized pure quantum states:

$$\phi_1 : \mathbf{x} \mapsto |\phi(\mathbf{x})\rangle. \tag{6.12}$$

The kernel is the standard bra-ket inner product:

$$k_1(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}) | \phi(\mathbf{x}') \rangle. \tag{6.13}$$

This viewpoint is well aligned with variational circuits, where the state $|\phi(\mathbf{x})\rangle$ is prepared via a data-dependent unitary acting on an initial state.

The second approach is to use density operators to encode features. In this case we define

$$\phi_2 : \mathbf{x} \mapsto \rho(\mathbf{x}). \tag{6.14}$$

The kernel is given by the overlap of two states which is given by the Hilbert-Schmidt inner product:

$$\langle \rho(\mathbf{x}), \rho(\mathbf{x}') \rangle_{\text{HS}} = \text{Tr}[\rho(\mathbf{x})\rho(\mathbf{x}')]. \tag{6.15}$$

This formulation is more general and naturally accommodates mixed states, noise, and ensemble encodings. Pure-state maps appear as a special case via $\rho(\mathbf{x}) = |\phi(\mathbf{x})\rangle\langle\phi(\mathbf{x})|$. The inner product induces the Hilbert-Schmidt norm: $||\rho||_{\text{HS}} = \sqrt{\langle \rho, \rho \rangle_{\text{HS}}} = \sqrt{\text{Tr}[\rho^2]}$. The Hilbert-Schmidt distance between two density matrices is defined as $d_{\text{HS}}(\rho, \rho') = ||\rho - \rho'||_{\text{HS}} = \sqrt{\text{Tr}[(\rho - \rho')^2]}$.

Describing data encoding as a feature map frames the process as an *embedding* of classical information into a quantum metric space. This terminology is common in other areas such as natural language processing, where objects like words are embedded into continuous vector spaces. In the quantum case, the embedding space is the state manifold of the quantum system, equipped with an inner product that defines the geometry relevant for learning.

Interpreting data encoding as a feature map provides

- a geometric picture of how classical data is represented in quantum form,
- a direct link to kernel-based methods via quantum state inner products,
- a unifying framework for comparing different encoding strategies.

This perspective highlights that the choice of encoding, i.e. the definition of ϕ, plays a crucial role in determining the expressivity, inductive bias, and computational structure of a quantum machine learning model.

Interpreting data encoding as a feature map highlights that it is far more than a technical preprocessing step in a quantum machine learning pipeline. In many cases, it is the *central design choice* of the model. The key reason is that a feature map performs a nontrivial

transformation of the data. Except for amplitude encoding, most encoding schemes introduce nonlinear functions of the input variables. For instance, rotation encoding produces states whose amplitudes contain trigonometric terms such as $\sin(x_i)$ and $\cos(x_i)$ derived from the original features.

Nonlinear feature transformations can significantly modify pairwise distances between data points. This may either simplify or complicate the learning task: a dataset that is linearly separable in its original form might lose this property after encoding, while in other cases the feature map may render previously inseparable classes separable, thereby solving the learning problem. Certain quantum feature maps can even induce separations that are believed to be classically intractable.

It is also crucial to recognize that, once the data has been embedded into quantum states, the quantum algorithm has limited ability to reshape it. Quantum gates are unitary and therefore implement linear transformations that preserve distances between states. Although some limited nonlinearity can be introduced via measurements, linear operations remain the natural mode of quantum processing. Thus, after the final feature of the input is encoded, the subsequent quantum computation consists mainly of linear operations, with the final measurement supplying at most a mild quadratic nonlinearity. This is in stark contrast to classical machine learning where many non-linearities operations occur in every forward pass.

Examples of Feature Maps

Basis encoding is a non-linear feature map. It maps an input $\mathbf{x} \in \mathbb{R}^n$ to one of the 2^n computational basis states

$$\phi(\mathbf{x}) = (0,0,\ldots,1_{\mathbf{x}},0,0,\ldots 0)^T. \tag{6.16}$$

We have $\dim(X) = n$ and $\dim(\mathcal{H}) = 2^n$. These points lie on the corners of the state-space hyper-cube. And all states are orthogonal. This means that on the one hand, the embedding separates all data. On the other, any new test data point will certainly be embedded into a state that is orthogonal to all training data points. Hence, more powerful tools than an inner product of feature states are required for building algorithms.

Amplitude encoding is a linear feature map:

$$\phi(\mathbf{x}) = \mathbf{x}. \tag{6.17}$$

We have $\dim(X) = \dim(\mathcal{H}) = 2^n$. The feature map thus preserves the distance between states. The quantum circuit to implement this quantum feature map is highly entangling.

Rotation encoding is again non-linear. If we take the Pauli-Y rotation gate, we get

$$\phi(x_1) = (\cos\tfrac{x_1}{2}, \sin\tfrac{x_1}{2})^T. \tag{6.18}$$

For two real numbers, we get

$$\phi(x_1,x_2) = (\cos\tfrac{x_1}{2}, \sin\tfrac{x_1}{2})^T \otimes (\cos\tfrac{x_2}{2}, \sin\tfrac{x_2}{2})$$

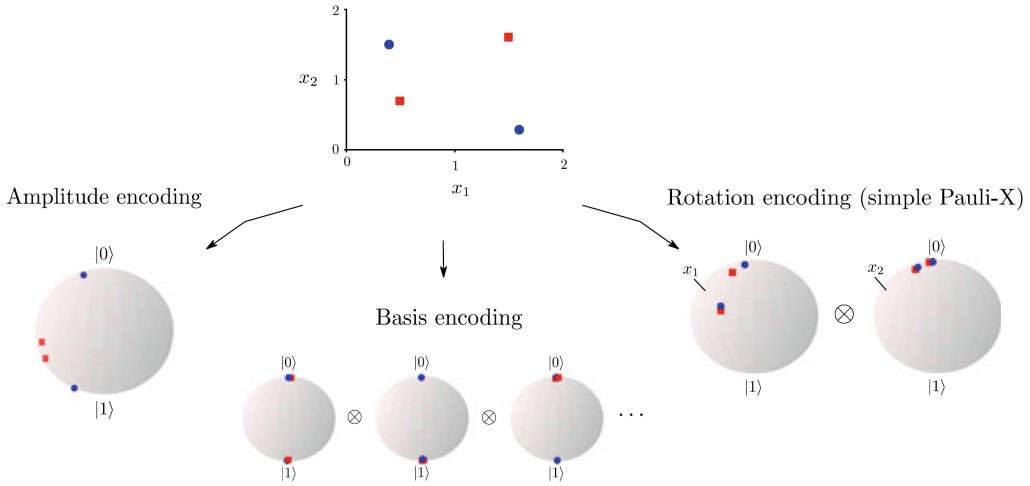


Figure 6.3: Summary of different encoding methods. In amplitude encoding, normalization of the feature vector needs to be performed before embedding. In basis encoding, 3-bit accuacy was used. Image taken from Ref. [5].

$$= (\cos\tfrac{x_1}{2}\cos\tfrac{x_2}{2}, \cos\tfrac{x_1}{2}\sin\tfrac{x_2}{2}, \sin\tfrac{x_1}{2}\cos\tfrac{x_2}{2}, \sin\tfrac{x_1}{2}\sin\tfrac{x_2}{2})^T. \tag{6.19}$$

For $\mathbf{x} \in \mathbb{R}^n$, we have

$$\phi(\mathbf{x}) = \begin{pmatrix} \cos\frac{x_1}{2}\cos\frac{x_2}{2}\ldots\cos\frac{x_{n-1}}{2}\cos\frac{x_n}{2} \\ \cos\frac{x_1}{2}\cos\frac{x_2}{2}\ldots\cos\frac{x_{n-1}}{2}\sin\frac{x_n}{2} \\ \vdots \\ \sin\frac{x_1}{2}\sin\frac{x_2}{2}\ldots\sin\frac{x_{n-1}}{2}\cos\frac{x_n}{2} \\ \sin\frac{x_1}{2}\sin\frac{x_2}{2}\ldots\sin\frac{x_{n-1}}{2}\sin\frac{x_n}{2} \end{pmatrix}. \tag{6.20}$$

We have $\dim(X) = n$ and $\dim(\mathcal{H}) = 2^n$.

Figure 6.3 illustrates these feature maps via the Bloch sphere representation.

7. Quantum Neural Networks

Modern machine learning is largely empirical: algorithms are evaluated through practical benchmarks, and many phenomena in deep learning still lack solid theoretical explanations. Classical computational complexity, the traditional tool for assessing quantum algorithms, often provides limited insight in this context. Problems encountered during training, such as non-convex optimization in neural networks, are computationally hard in general, yet heuristics solve them efficiently in practice. This creates challenges for both fault-tolerant and near-term quantum approaches to machine learning. Also, promises of exponential speedups require careful scrutiny, especially when classical routines already achieve polynomial runtimes or when data-loading costs dominate. Finally, while near-term quantum devices permit more practical experimentation, algorithms use only a few or few dozens of qubits and must be resilient to hardware noise.

These constraints motivated two major design decisions in quantum machine learning. First, *hybrid quantum-classical* schemes outsource only small quantum subroutines to a quantum processor, while classical hardware performs the training. Second, rather than directly quantizing classical models, researchers design hardware-efficient circuits, also called *quantum models*, tailored to available devices. Variational quantum circuits, in the context of machine learning also called *quantum neural networks*, are a central example. They consist of an ansatz of fixed and parametrized gates whose architecture resembles the layered structure of neural networks. Parameters are optimized by minimizing a cost function in a classical feedback loop, using either black-box evaluations or gradient information.

7.1 Deterministic Quantum Models

A deterministic quantum machine learning model consists of a data embedding block $S(\mathbf{x})$ of inputs $\mathbf{x}$ and a variational quantum circuit $W(\boldsymbol{\theta})$ parametrized by a set of variables $\boldsymbol{\theta}$:

$$U(\mathbf{x}, \boldsymbol{\theta}) = W(\boldsymbol{\theta}) S(\mathbf{x}). \tag{7.1}$$

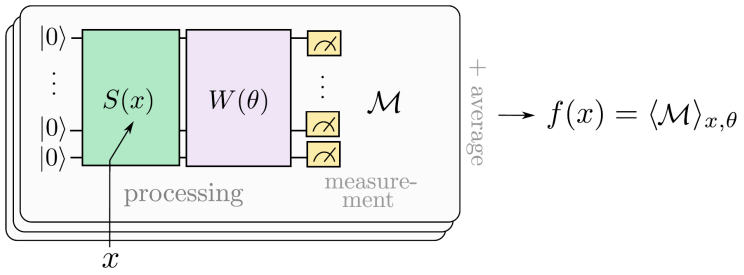


Figure 7.1: Deterministic quantum model. An input $\mathbf{x}$ is encoded via the unitary $S(\mathbf{x})$ and processed by a variational quantum circuit $W(\boldsymbol{\theta})$ parametrized by $\boldsymbol{\theta}$. The model output $f(\mathbf{x})$ is the expectation value of the observable $\mathcal{M}$. Image taken from Ref. [5].

This unitary acts on an initial state as: $U(\mathbf{x}, \boldsymbol{\theta})|0\rangle = |\psi(\mathbf{x}, \boldsymbol{\theta})\rangle$. The expectation value of an observable $\mathcal{M}$ in that state is the model output (see Figure 7.1):

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \langle \psi(\mathbf{x}, \boldsymbol{\theta}) | \mathcal{M} | \psi(\mathbf{x}, \boldsymbol{\theta}) \rangle. \tag{7.2}$$

There are also more general approaches, in which the embedding itself is trainable by mixing the embedding and parametrized blocks.

If we write the measurement operator in its diagonal basis,

$$\mathcal{M} = \sum_i \mu_i |\mu_i\rangle\langle\mu_i|, \tag{7.3}$$

with μ_i the measurement outcomes and $|\mu_i\rangle$ the corresponding eigenvectors, then the model output is

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_i \mu_i |\langle \mu_i | \psi(\mathbf{x}, \boldsymbol{\theta}) \rangle|^2 = \sum_i \mu_i \, p(\mu_i), \tag{7.4}$$

with $p(\mu_i) = |\langle \mu_i | \psi(\mathbf{x}, \boldsymbol{\theta}) \rangle|^2$ the probability to observe the outcome μ_i. As an example, let us take the single-qubit computational basis measurement $\mathcal{M} = Z$. Then,

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = |\langle 0 | \psi(\mathbf{x}, \boldsymbol{\theta}) \rangle|^2 - |\langle 1 | \psi(\mathbf{x}, \boldsymbol{\theta}) \rangle|^2 = p(0) - p(1). \tag{7.5}$$

In the lab, we can estimate (7.4) by sampling over K measurements:

$$\hat{f}_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} \mu^{(k)}, \tag{7.6}$$

where $\mu^{(k)}$ is the measurement result for sample k. In order to estimate $f_{\boldsymbol{\theta}}(\mathbf{x})$ with accuracy ε, one requires $K = \mathcal{O}(\frac{1}{\varepsilon^2})$ samples. For every sample $\mathbf{x}$, the whole algorithm – initialization, embedding, parametrized circuit, and measurement – has to be run again.

7.2 Probabilistic Quantum Models

Due to the probabilistic nature of quantum theory, variational circuits can naturally implement probabilistic quantum models. In the supervised setting, one considers a conditional distribution $p_{\boldsymbol{\theta}}(y|\mathbf{x})$ over outputs y given inputs $\mathbf{x}$. Again, we have the input- and parameter-dependent unitary $U(\mathbf{x},\boldsymbol{\theta})$. Now we associate each eigenvalue of the measurement observable

$$\mathcal{M} = \sum_{y\in Y} y\,|y\rangle\langle y| \tag{7.7}$$

with a possible prediction y. The supervised probabilistic quantum model is then

$$p_{\boldsymbol{\theta}}(y|\mathbf{x}) = |\langle y|\psi(\mathbf{x},\boldsymbol{\theta})\rangle|^2. \tag{7.8}$$

Hence, we have the same picture as in Figure 7.1, but with $f(\mathbf{x})$ replaced by $\hat{y}\sim p_{\boldsymbol{\theta}}(y|\mathbf{x})$, i.e. the output $\hat{y}$ is sampled from the distribution $p_{\boldsymbol{\theta}}(y|\mathbf{x})$.

For binary classification, one may use a Pauli-Z measurement with labels $y=\pm 1$ corresponding to the projectors $|0\rangle\langle 0|$ and $|1\rangle\langle 1|$. For $D=2^n$ classes, the computational basis $\{|d\rangle\}_{d=0}^{D-1}$ of n qubits can encode the D outcomes.

Probabilistic quantum models are inherently *generative*, as executing a quantum circuit yields samples directly from the modeled distribution. In practice, however, computing explicit probabilities for given data points can be challenging: estimating probabilities on a quantum device typically requires a number of measurement shots that grows exponentially with the number of qubits. For this reason, the literature on quantum machine learning often refers to such probabilistic quantum models simply as *quantum generative models*.

7.3 Variational Quantum Classifiers

Variational quantum classifiers (VQCs) constitute one of the central applications of near-term quantum computing within the broader field of quantum machine learning. They combine parameterized quantum circuits with classical optimization loops to perform supervised learning tasks such as binary or multi-class classification. The key idea is to exploit the expressive power of quantum states and the nonlinearity introduced by measurement to construct flexible decision boundaries that can, in principle, surpass those attainable by shallow classical models.

From a deterministic or probabilistic quantum model, we can easily build a variational quantum classifier. In the deterministic case, we can apply a discretization function onto our estimator $\hat{f}_{\boldsymbol{\theta}}(\mathbf{x})$. For instance, if we measure $\mathcal{M}=Z$, we can take the sign function as our class prediction:

$$\hat{y} = \mathrm{sgn}(\hat{f}_{\boldsymbol{\theta}}(\mathbf{x})). \tag{7.9}$$

In the probabilistic case, we can take directly take the sampled output $\hat{y}$ itself.

Training proceeds by minimizing a classical loss

$$L_{\boldsymbol{\theta}} = \frac{1}{M}\sum_{i=1}^{M}\mathcal{L}(y_i,\hat{y}_i), \tag{7.10}$$

where $\mathcal{L}$ is, for example, the cross-entropy or mean squared error and y_i is the true label for sample $\mathbf{x}_i$. The average is computed over a labeled dataset $(\{(\mathbf{x}_i,y_i)\})_{i=1}^{M}$. Gradients with respect to the parameters θ_j can be estimated using parameter-shift rules or finite differences and fed to a classical optimizer.

VQCs can exploit quantum feature maps that produce highly expressive feature representations and highly non-linear decision boundaries, and they naturally produce probabilistic outputs via repeated measurements. Their potential stems from the possibility that certain data distributions or feature maps may be encoded more efficiently or more naturally in quantum Hilbert space than in classical models. Practical challenges include shot noise (finite measurement samples), noise in near-term devices, and optimization issues such as barren plateaus where gradients vanish exponentially with system size. Despite these limitations, VQCs are a flexible framework for exploring quantum advantage in classification, especially when combined with careful encoding design and noise-aware training strategies.

7.4 Quantum Circuits and Neural Networks

Note that the term "quantum neural network" for variational circuits is somewhat misleading in the sense that quantum networks do not have the element-wise non-linear behavior which multi-layer perceptrons exhibit using activation functions. Non-linearity can only be brought in by encoding and by measurement.

A simple (non-linear) activation function is the step function:

$$\varphi(z) = \begin{cases} 0, & z < 0, \\ 1, & z \geq 0. \end{cases} \tag{7.11}$$

In basis encoding, it is straightforward to implement it. One simply uses the flipped sign-qubit as the output: $|x\rangle = |b_0 b_1 b_2 b_3 b_4 \ldots\rangle \to X|b_0\rangle$. If the sign qubit is $b_0 = 0$ ($b_0 = 1$), i.e. $z \geq 0$ ($z < 0$), then the output is $|1\rangle$ ($|0\rangle$).

Another activation function is the the rectified linear unit, which has the form

$$\varphi(z) = \begin{cases} 0, & z < 0, \\ z, & z \geq 0. \end{cases} \tag{7.12}$$

If the sign qubit is $b_0 = 0$, i.e. $z \geq 0$, the input register is copied to the output register. If $b_0 = 1$, i.e. $z < 0$, a sequence of controlled NOT gates sets all output register bits to 0.

It is also possible to implement the sigmoid non-linearity (up to some precision). There, Quine-McCluskey methods allow to reduze the size of look-up tables considerably.

Variational circuits are similar to linear deep neural networks (see Figure 7.2). After

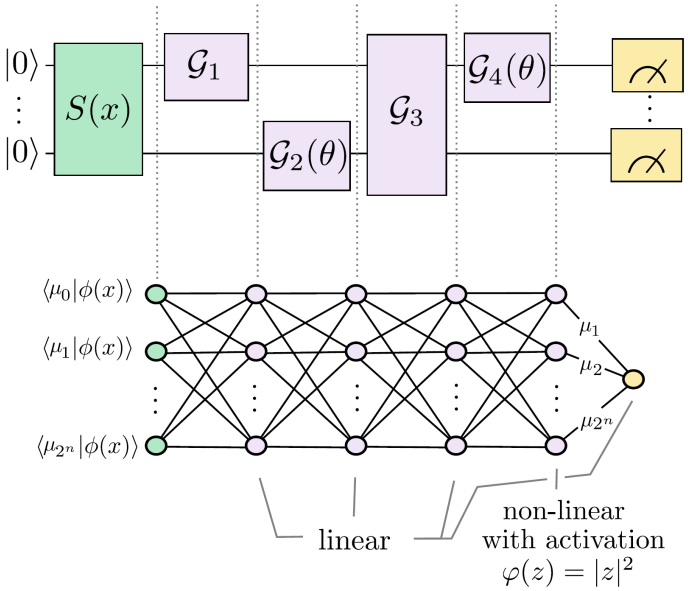


Figure 7.2: Variational circuits as deep linear neural networks. The data encoding prepares the state $|\phi(\mathbf{x})\rangle = S(\mathbf{x})|0\rangle^{\otimes n}$. The quantum circuit can then be viewed as a linear neural network whose inputs (green) are the components $\langle\mu_i|\phi(x)\rangle$ of the data-encoding state $|\phi(\mathbf{x})\rangle$ expressed in the eigenbasis $\{|\mu_i\rangle\}_{i=1}^{2^n}$ of the measurement operator $\mathcal{M} = \sum_i \mu_i |\mu_i\rangle\langle\mu_i|$. The quantum gates act as linear network layers (purple), with some layers fixed and others containing trainable parameters. The final measurement induces a nonlinear output layer (yellow) with an "absolute-square activation" $\varphi(z) = |z|^2$, and the weights of the final layer correspond to the eigenvalues μ_i of the measurement operator. Importantly, while there are n qubits (wires) in the quantum circuit, its neural network analog has 2^n neurons in every layer. Image taken from Ref. [5].

data encoding, the n-qubit feature state is $|\phi(\mathbf{x})\rangle$. After data encoding, each amplitude $\langle\mu_i|\phi(\mathbf{x})\rangle$ of the feature state is the value of an input neuron in the first layer. Here, $|\mu_i\rangle$ are the eigenvectors of the final measurement operator $\mathcal{M} = \sum_i \mu_i |\mu_i\rangle\langle\mu_i|$. For a measurement in the computational basis, $|\mu_i\rangle = (00\ldots01_i0\ldots0)^T$ are the 2^n computational basis states.

From this first layer onward, the quantum circuit applies unitary, i.e. linear, transformations on the quantum state, where the individual gates may or may not have trainable parameters θ_j. It is worth noting that the corresponing weights of the neural network, w_{ij}, are not directly the variational parameters themselves, but rather functions of the set of circuit parameters: $w_{ij} = w_{ij}(\boldsymbol{\theta})$. Let's call the quantum state after the last gate $W(\boldsymbol{\theta})S(\mathbf{x})|0\rangle^{\otimes n} =: |\psi(\mathbf{x},\boldsymbol{\theta})\rangle$.

In the last step, the measurement corresponds to a non-linear output layer with an activation that computes the sum of the absolute squares of all the amplitudes of the previous layer, weighted by the eigenvalues of the measurement operator; see again equation (7.4):

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_i \mu_i |\langle\mu_i|\psi(\mathbf{x},\boldsymbol{\theta})\rangle|^2. \tag{7.13}$$

In summary, variational circuits demonstrate certain similarities to the multilayer-perceptron structure of classical neural networks. The following chapter will introduce

a more direct and intuitive correspondence between quantum models and established machine-learning techniques.

8. Quantum Kernel Methods

In kernel methods, data vectors are mapped to vectors in a higher-dimensional Hilbert space without the need of ever computing the latter explicitly. In many cases, supervised deterministic quantum models can be formulated as classical kernel methods where the kernel is computed by a quantum device. Since unitary evolution is linear, the expressivity and generalization behavior of a quantum model is predominantly defined by the data-encoding (embedding) strategy that fixes the kernel.

Even though the kernel exploits the exponentially high-dimensional quantum state space, the actual training and operation of quantum models can take place within a low-dimensional subspace. Unlike variational approaches, there is no necessity to select an appropriate ansatz or to mitigate barren plateaus. However, one must evaluate inner products, i.e. pairwise distances, between all embedded data points (Figure 8.1).

In a variational quantum circuit, all parametrized gates are summarized by the unitary operator $W(\boldsymbol{\theta})$. This is followed by the measurement operator $\mathcal{M}$. Mathematically, this is equivalent to not applying the variational circuit but "directly" measuring the transformed observable

$$\mathcal{M}'_{\boldsymbol{\theta}} := W(\boldsymbol{\theta})^{\dagger} \mathcal{M} W(\boldsymbol{\theta}). \tag{8.1}$$

Training is the process of finding the optimal measurement which minimizes a cost function.

The embedding $S(\mathbf{x})$ is fixed and not trainable. It maps data $\mathbf{x} \in \mathcal{X}$ from the input space $\mathcal{X}$ onto density matrices

$$\rho(\mathbf{x}) = |\phi(\mathbf{x})\rangle\langle\phi(\mathbf{x})| \tag{8.2}$$

While these are (density) matrices and not (state) vectors, they are still *feature vectors* in the sense that they belong to a vector space. This feature space $\mathcal{F}$ of complex matrices

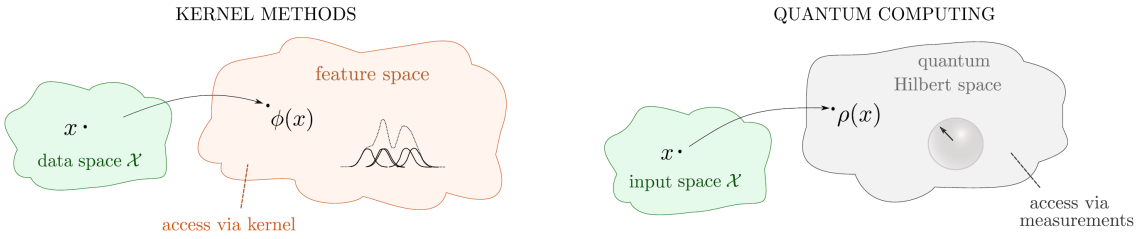


Figure 8.1: Quantum computing and kernel methods are based on a similar principle. Both have mathematical frameworks in which information is mapped into and then processed in high-dimensional spaces to which we have only limited access. In kernel methods, the access to the feature space is facilitated through kernels or inner products of feature vectors. In quantum computing, access to the Hilbert space of quantum states is given by measurements, which can also be expressed through inner products of quantum states. Image and caption taken from Ref. [5].

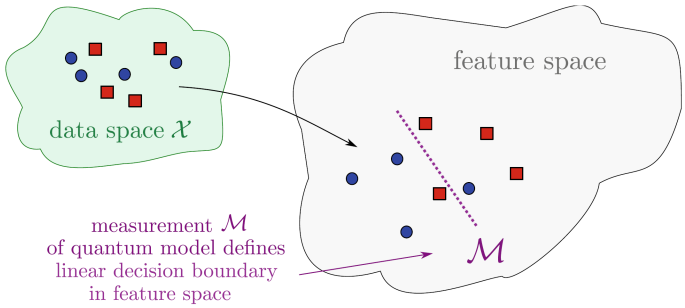


Figure 8.2: Quantum models as linear models in a feature space. A quantum model can be understood as a model that maps data into a feature space in which the measurement defines a linear decision boundary. This feature space is not identical to the Hilbert space of the quantum system. Instead, we can define it as the space of complex matrices enriched with the Hilbert-Schmidt inner product. Image and caption taken from Ref. [5].

$\rho(\mathbf{x})$ has an inner product given by the Hilbert-Schmidt inner product (6.15):[1]

$$\kappa(\mathbf{x},\mathbf{x}') = \langle \rho(\mathbf{x}), \rho(\mathbf{x}') \rangle_{\text{HS}} = \text{Tr}[\rho(\mathbf{x})\rho(\mathbf{x}')]. \tag{8.3}$$

This inner product is called *quantum kernel*. Quantum models are linear in the feature vectors $\rho(\mathbf{x})$. The (potentially trainable) measurement observable corresponds to the weight vector and defines a linear decision boundary in feature space (Figure 8.2).

Our original Hilbert space is the feature space $\mathcal{F}$ of density matrices ρ. A Reproducing Kernel Hilbert Space (RKHS) is an alternative feature space F of functions $f(.)$ that are constructed from the kernel κ:

$$\mathbf{x} \mapsto f_{\mathbf{x}}(.) = \kappa(\mathbf{x},.) \tag{8.4}$$

For every input $\mathbf{x} \in \mathcal{X}$, there exists one such function $f_{\mathbf{x}}(.)$ in the RKHS F. Moreover, F

[1]The product $\rho(\mathbf{x})\rho(\mathbf{x}')$ is really the standard matrix product, not the tensor product $\rho(\mathbf{x}) \otimes \rho(\mathbf{x}')$.

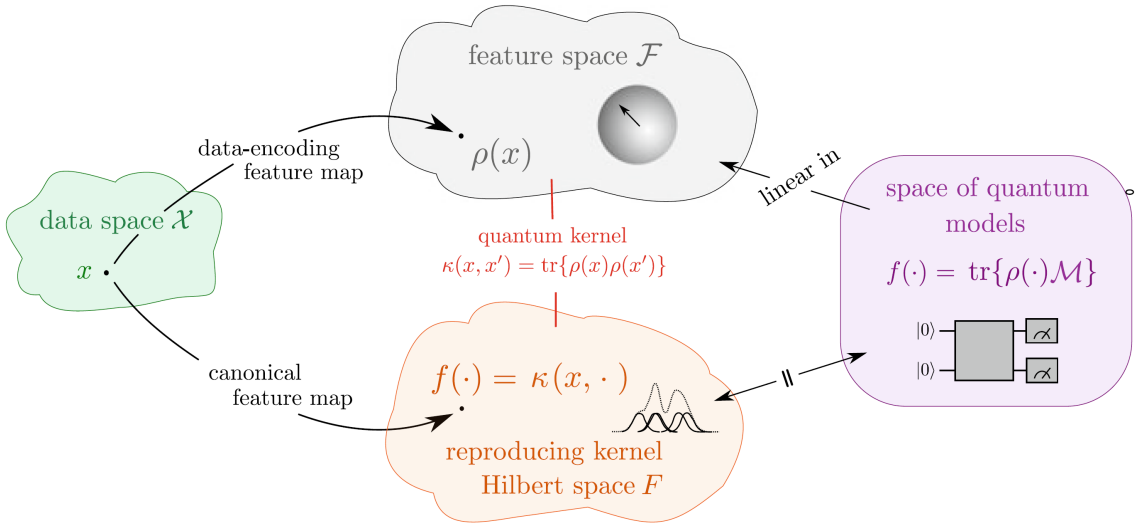


Figure 8.3: Overview of the link between quantum models and kernel methods. The strategy with which data is encoded into quantum states is a feature map from the space of data to the feature space $\mathcal{F}$ of density matrices ρ. In this space, quantum models can be expressed as linear models whose decision boundary is defined by the measurement. According to kernel theory, an alternative feature space with the same kernel is the RKHS F, whose vectors are functions arising from fixing one entry of the kernel (i.e., the inner product of data-encoding density matrices). The RKHS is equivalent to the space of quantum models, which are linear models in the data-encoding feature space. These connections can be used to study the properties of quantum models as learners, which turn out to be largely determined by the kernel, and therefore, by the data-encoding strategy. Image and caption taken from Ref. [5].

contains all their linear combinations. For instance, let us take the Gaussian kernel

$$\kappa_{\mathrm{G}}(\mathbf{x},\mathbf{x}') = e^{-\frac{||\mathbf{x}-\mathbf{x}'||^2}{2\sigma^2}}. \tag{8.5}$$

Then also all the sums of Gaussians centered in the individual data points are in F:

$$\kappa_{\mathrm{G}}(\mathbf{x}_1,.) + \kappa_{\mathrm{G}}(\mathbf{x}_2,.) + \kappa_{\mathrm{G}}(\mathbf{x}_3,.) + \ldots \tag{8.6}$$

Importantly, the kernel functions correspond to linear models in the feature space $\mathcal{F}$. The space of quantum models $\mathcal{Q}$ contains the "quantum functions" f, which are expectation values of our measurement operators $\mathcal{M}$. In the density matrix formalism, these expectation values have the form

$$f(.) = \mathrm{Tr}[\rho(.)\,\mathcal{M}]. \tag{8.7}$$

Thus, the space of quantum models $\mathcal{Q}$ and the RKHS F contain exactly the same functions $f(.)$. This means that we have found an alternative way to describe quantum models, namely by only using the quantum kernel (8.3), see Figure 8.3.

We can now use this representation for optimization: Minimizing cost functions $f(.)$ over the space of quantum models $\mathcal{Q}$ is equivalent to minimizing the same cost function

over the the reproducing kernel Hilbert Space F.

The *representer theorem* is a fundamental result which states that for a broad class of regularized empirical risk minimization problems, the optimal solution does not lie in an infinite-dimensional function space, but instead in the finite-dimensional span of kernel evaluations on the M training samples $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$. Concretely, if we minimize a cost function of the form

$$L(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_M)) + \lambda \|f\|^2 \tag{8.8}$$

over all functions $f \in F$, where $\lambda \|f\|^2$ is a regularization term, then the minimizer admits the representation

$$f_{\text{opt}}(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m \, \kappa(\mathbf{x}_m, \mathbf{x}). \tag{8.9}$$

Here, κ is the reproducing kernel (quantum kernel) associated with the RKHS F. This result shows that, despite the potentially infinite dimensionality of the RKHS, learning reduces to finding a finite set of real-valued coefficients α_m. Using (8.3), we can rewrite (8.9) as follows:

$$f_{\text{opt}}(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m \, \text{Tr}[\rho(\mathbf{x}_m)\rho(\mathbf{x})] = \text{Tr}\left[\sum_{m=1}^{M} \alpha_m \, \rho(\mathbf{x}_m) \, \rho(\mathbf{x})\right]. \tag{8.10}$$

If we define the measurement operator as

$$\mathcal{M} = \sum_{m=1}^{M} \alpha_m \, \rho(\mathbf{x}_m), \tag{8.11}$$

the optimal quantum model takes the form

$$f_{\text{opt}}(\mathbf{x}) = \text{Tr}[\mathcal{M}\rho(\mathbf{x})], \tag{8.12}$$

which is (8.7) again, i.e. the expectation value of $\mathcal{M}$ in the state $\rho(\mathbf{x})$. Therefore, the optimal quantum model f_{opt}, minimizing the cost function 8.8, uses the measurement (8.11), i.e. measuring only M observables with M degrees of freedom α_m, namely the density matrices of the M data points. Typically, $M \ll 2^n$. (Note that while density matrices are not physical observables whose eigenvalues represent the possible measurement outcomes, they are nonetheless Hermitian operators.) With the regularization term in (8.8) included, the kernel – which only depends on the data encoding – entirely determines the landscape of the cost function and thus which models are optimal.

Finding the optimal optimal measurement basis via variational training is infeasible. A general measurement operator has $\mathcal{O}(2^{2n})$ degrees of freedom, and one would thus require (at least) as many variational parameters. As soon as the number K of parameters is smaller, not the whole space can be explored. Fortunately, however, finding the optimal measurement for typical cost functions via the kernel-based method is an only M-dimensional

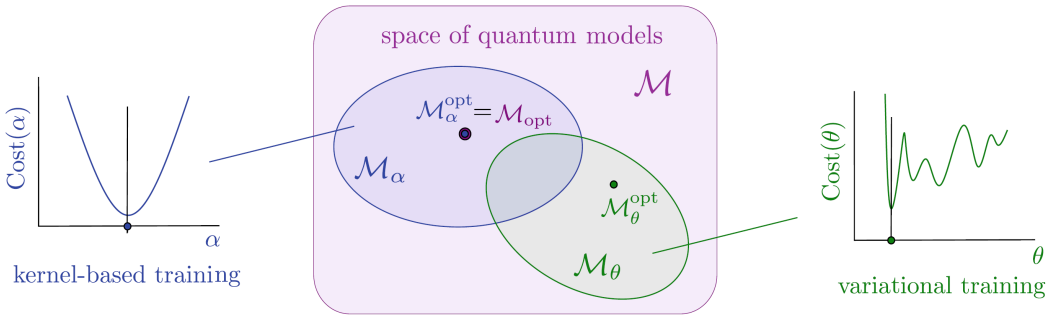


Figure 8.4: Kernel-based training versus variational training. Training a quantum model as defined in this chapter tries to find the optimal measurement $\mathcal{M}_{\text{opt}}$ over all possible quantum measurements. Kernel theory guarantees that in most cases this optimal measurement will have a representation that is a linear combination in the training data with coefficients $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)^T$. Kernel-based training, therefore, optimises over the parameters α directly, effectively searching for the best model in an M-dimensional subspace spanned by the training data (blue). We are guaranteed that $\mathcal{M}_{\boldsymbol{\alpha}}^{\text{opt}} = \mathcal{M}_{\text{opt}}$, and if the loss is convex, this is the only minimum, which means that kernel-based training will find the best measurement out of all measurements. Variational training parametrises the measurement instead by a general ansatz that depends on K parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_K)$, and tries to find the optimal measurement $\mathcal{M}_{\boldsymbol{\theta}}^{\text{opt}}$ in the subspace explored by the ansatz. This $\boldsymbol{\theta}$-subspace is not guaranteed to contain the globally optimal measurement $\mathcal{M}_{\text{opt}}$, and optimisation is usually non-convex. We are, therefore, guaranteed that kernel-based training finds better or the same minima to variational training, but at the expense of having to compute pairwise distances of data points for training and classification. Image and caption taken from Ref. [5].

optimization problem. Figure 8.4 illustrates kernel-based and variational training.

Loss functions are often convex, and then the optimization problem becomes particularly simple and trainable with a guaranteed global optimal solution that can be found numerically without barren plateaus.

8.1 Quantum Kernel Support Vector Machines

A particularly interesting situation arises when we use the hinge loss

$$\mathcal{L}(y, f(\mathbf{x})) = \max[0, 1 - y\, f(\mathbf{x})] \tag{8.13}$$

together with L_2 regularization. This leads to the optimization problem

$$\min_{\mathcal{M}, b} \tfrac{1}{2} \|\mathcal{M}\|_{\text{HS}}^2 + C \sum_{m=1}^{M} \max\left[0, 1 - y\,(\text{Tr}[\mathcal{M}\,\rho(\mathbf{x}_m)] + b)\right], \tag{8.14}$$

which has the same form as a support vector machine (SVM) with slack variables. Here, $\mathcal{M}$ is the normal "vector" of the separating hyperplane, b the distance of the hyperplane from the origin, and C a hyperparameter controlling the tradeoff between margin size and classification errors. The hinge loss of every data sample $\mathbf{x}_m$ corresponds to its slack variable. The optimal solution is fully characterized by the optimal Lagrange multipliers

$\alpha_m^\star$. The corresponding optimal classifier operator is

$$\mathcal{M}_{\text{opt}} = \sum_{m=1}^{M} \alpha_m^\star y_m \rho(\mathbf{x}_m), \tag{8.15}$$

which shows that the solution lies in the span of the training density matrices (representer theorem). The optimal Lagrange parameters $\alpha_m^\star$ have a closed-form solution, e.g. via solving the dual problem. This can be done on a classical computer. The bias term can be computed from any support vector $\mathbf{x}_j$ with $0 < \alpha_j^\star < C$ via

$$b^\star = y_j - \sum_{m=1}^{M} \alpha_m^\star y_m \kappa(\mathbf{x}_m, \mathbf{x}_j). \tag{8.16}$$

The final decision function is

$$f_{\text{opt}}(\mathbf{x}) = \text{Tr}[\mathcal{M}_{\text{opt}} \rho(\mathbf{x})] + b^\star = \sum_{m=1}^{M} \alpha_m^\star y_m \kappa(\mathbf{x}_m, \mathbf{x}) + b^\star. \tag{8.17}$$

Hence, minimizing hinge loss yields a support vector machine (SVM) whose kernel is given by quantum state overlaps, i.e. a *quantum kernel SVM*. Quantum kernel methods retain the optimal solutions and convex structure of classical kernel learning, while potentially benefiting from quantum-enhanced feature maps. As a result, any potential quantum advantages enter exclusively through the quantumly computed kernel, while training and optimality remain classically efficient and well understood.

The key element is thus to estimate the quantum kernel matrix

$$\begin{aligned} K_{ij} &= \text{Tr}[\rho(\mathbf{x}_i)\, \rho(\mathbf{x}_j)] \\ &= \left|\langle \phi(\mathbf{x}_i) | \phi(\mathbf{x}_j) \rangle\right|^2, \end{aligned} \tag{8.18}$$

where the last equality holds, if the states are pure. One method to estimate inner products is a slightly modified version of the already discussed Hadamard test. We will now learn other techniques.

8.2 Kernel Estimation Methods

Inversion Test

The canonical way to access the inner product $\kappa(\mathbf{x}, \mathbf{x}') = |\langle \phi(\mathbf{x}) | \phi(\mathbf{x}') \rangle|^2$ is via the inversion test shown in Figure 8.5. The unitary $U(\mathbf{x})$ is embedding the instance $\mathbf{x}$ in a quantum state $|\phi(\mathbf{x})\rangle = U(\mathbf{x})\,|0\rangle^{\otimes n}$. This quantum state should then be compared to the state $|\phi(\mathbf{x}')\rangle = U(\mathbf{x}')\,|0\rangle^{\otimes n}$ of another data point embedded by $U(\mathbf{x}')$. The probability to measure all qubits in the $|0\rangle$ state is then given by

$$\begin{aligned} P(0,\ldots,0) &= |\langle 0,\ldots,0 | U^\dagger(\mathbf{x}') U(\mathbf{x}) | 0,\ldots,0 \rangle|^2 \\ &= |\langle \phi(\mathbf{x}') | \phi(\mathbf{x}) \rangle|^2 = |\langle \phi(\mathbf{x}) | \phi(\mathbf{x}') \rangle|^2. \end{aligned} \tag{8.19}$$

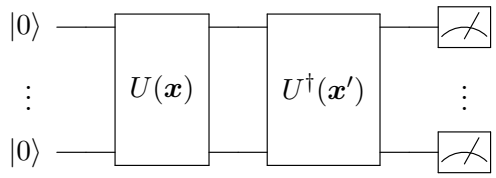


Figure 8.5: Circuit of the inversion test.

Swap Test

The swap test is another fundamental and efficient quantum subroutine that estimates the overlap between two quantum states. It makes use of the controlled-SWAP gate. Before we can understand the controlled-SWAP gate, we need to first discuss the SWAP gate itself. The SWAP gate is a two-qubit unitary operator that exchanges the states of two quantum systems with identical Hilbert spaces $\mathcal{H}$. It is defined by its action on product states:

$$\text{SWAP}\,|a\rangle\,|b\rangle = |b\rangle\,|a\rangle\,. \tag{8.20}$$

Equivalently, in operator form, the SWAP operator can be written as

$$\text{SWAP} = \sum_{i,j} |i\rangle\langle j| \otimes |j\rangle\langle i|\,, \tag{8.21}$$

where $\{|i\rangle\}$ is an orthonormal basis of $\mathcal{H}$. The SWAP operator is Hermitian and unitary:

$$\text{SWAP}^\dagger = \text{SWAP}, \quad \text{SWAP}^\dagger = \text{SWAP}^{-1}. \tag{8.22}$$

For two qubits, its matrix representation is

$$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{8.23}$$

The controlled-SWAP gate is called CSWAP and acts on an ancilla qubit (control) and two target registers. Conditioned on the control qubit being in state $|1\rangle$, the SWAP operation is applied to the two target systems. If the control is in state $|0\rangle$, the targets are left unchanged. Its unitary (and Hermitian) operator therefore is

$$\text{CSWAP} = |0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes \text{SWAP}. \tag{8.24}$$

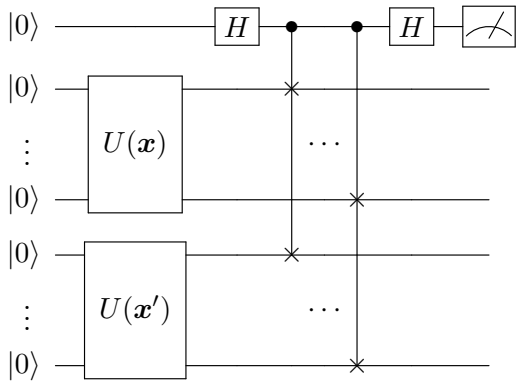


Figure 8.6: Swap test: We are given two state registers $|\psi\rangle = U(\mathbf{x})\,|0\rangle^{\otimes n}$ and $|\phi\rangle = U(\mathbf{x}')\,|0\rangle^{\otimes n}$, each consisting of n qubits. Their (absolute square) overlap can be computed with the help of a single ancilla qubit (first wire, initialized in $|0\rangle$), two Hadamard gates H and n intermediary controlled-SWAP gates (indicated by the $\bullet$ and $\times$ symbols).

Acting on basis states, this gives

$$\text{CSWAP}\,|c\rangle\,|a\rangle\,|b\rangle = \begin{cases} |0\rangle\,|a\rangle\,|b\rangle\,, & c=0, \\ |1\rangle\,|b\rangle\,|a\rangle\,, & c=1. \end{cases} \tag{8.25}$$

The CSWAP gate is commonly referred to as the Fredkin gate. It is a universal, reversible 3-(qu)bit logic gate that swaps two target bits if and only if a control bit has value 1.

We now go through the steps of the swap test. The circuit is shown in Figure 8.6. It uses a single ancilla qubit and n controlled-SWAP gates acting on the two registers that store the n-qubit states $|\psi\rangle = U(\mathbf{x})\,|0\rangle^{\otimes n}$ and $|\phi\rangle = U(\mathbf{x}')\,|0\rangle^{\otimes n}$. The state of the joint system of $2n+1$ qubits then is

$$|\Psi_0\rangle = |0\rangle\,|\psi\rangle\,|\phi\rangle\,. \tag{8.26}$$

After applying the first Hadamard gate to the ancilla, we obtain

$$|\Psi_1\rangle = \tfrac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big)\,|\psi\rangle\,|\phi\rangle\,. \tag{8.27}$$

Next, the n CSWAP gates exchange – qubit pair by qubit pair – the two data registers conditioned on the ancilla being in state $|1\rangle$, yielding

$$|\Psi_2\rangle = \tfrac{1}{\sqrt{2}}\big(|0\rangle|\psi\rangle\,|\phi\rangle + |1\rangle|\phi\rangle\,|\psi\rangle\big). \tag{8.28}$$

Applying the second Hadamard gate to the ancilla gives

$$|\Psi_3\rangle = \tfrac{1}{2}\big[\,|0\rangle\,\big(|\psi\rangle\,|\phi\rangle + |\phi\rangle\,|\psi\rangle\big) + |1\rangle\,\big(|\psi\rangle\,|\phi\rangle - |\phi\rangle\,|\psi\rangle\big)\big]. \tag{8.29}$$

The probability of measuring the ancilla in state $|0\rangle$ is thus

$$\begin{aligned} P(0) &= |\langle 0|\Psi_3\rangle|^2 = \tfrac{1}{4}\left\| |\psi\rangle|\phi\rangle + |\phi\rangle|\psi\rangle \right\|^2 \\ &= \tfrac{1}{4}\big(\langle\psi|\langle\phi| + \langle\phi|\langle\psi|\big)\big(|\psi\rangle|\phi\rangle + |\phi\rangle|\psi\rangle\big) \\ &= \tfrac{1}{4}\big(2+2|\langle\psi|\phi\rangle|^2\big) = \tfrac{1}{2}\big(1+|\langle\psi|\phi\rangle|^2\big). \end{aligned} \tag{8.30}$$

Therefore, the swap test allows to obtain the absolute squared state overlap from the measurement probability $P(0)$:

$$|\langle\psi|\phi\rangle|^2 = 2P(0) - 1. \tag{8.31}$$

In conclusion, the swap test allows us to estimate state overlap without performing full quantum state tomography and thus to efficiently evaluate quantum kernels (8.18).

Comparison of Inversion and Swap Test

The inversion test uses only n qubits, but it needs to apply the unitaries $U(\mathbf{x})$ and $U(\mathbf{x}')$ sequentially, i.e. we denote its depth by $2u$, where u is the number of sequential gates required to implement $U(\mathbf{x})$ or $U(\mathbf{x}')$. The swap test requires $2n+1$ qubits, but has the advantage of a smaller circuit depth, because the unitaries can be run in parallel. Its depth is only $u+1$, assuming that the n CSWAP gates can also be run in parallel. In certain architectures with small coherence times, this may be relevant. The price to pay, however, is the need of n CSWAP gates.

A SWAP gate between two qubits a and b can be written as a product of three CNOT gates:

$$\mathrm{SWAP}(a,b) = \mathrm{CNOT}_{a\to b}\,\mathrm{CNOT}_{b\to a}\,\mathrm{CNOT}_{a\to b}. \tag{8.32}$$

If the SWAP operation is conditioned on a control qubit c, each CNOT in the above decomposition is replaced by a controlled-CNOT = CCNOT = Toffoli gate. This yields the controlled-SWAP = CSWAP = Fredkin gate decomposition:

$$\mathrm{CSWAP}(c;a,b) = \mathrm{CCNOT}_{c,a\to b}\,\mathrm{CCNOT}_{c,b\to a}\,\mathrm{CCNOT}_{c,a\to b}. \tag{8.33}$$

Each CCNOT gate admits a standard decomposition into single-qubit gates and six CNOT gates. Consequently, this construction of the CSWAP gate requires a total of 18 CNOT gates, in addition to single-qubit rotations. (While this decomposition is not CNOT-optimal, it provides a conceptually simple and systematic method to realize the CSWAP gate.) Hence, the swap test (usually) requires $18n$ CNOT gates.

Destructive Swap Test

The destructive swap test projects all qubit pairs between two input states $|\psi\rangle$ and $|\phi\rangle$ into the Bell basis. The circuit is shown in Figure 8.7. The absolute squared state overlap is

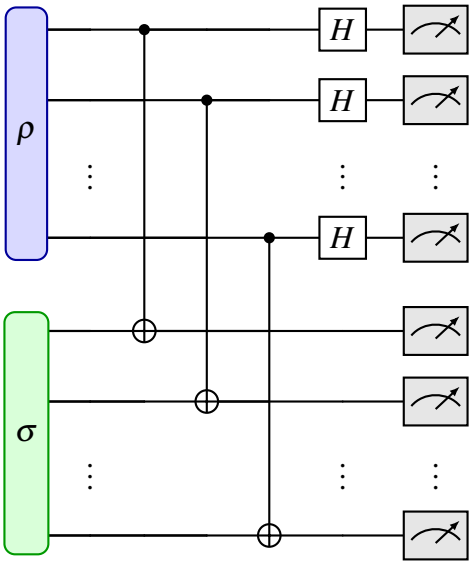


Figure 8.7: Destructive swap test: We are given two state registers $\rho = |\psi\rangle\langle\psi|$ and $\sigma = |\phi\rangle\langle\phi|$, each consisting of n qubits. For each pair of qubits, a Bell measurement is performed by first applying a CNOT, then a Hadamard, followed by a measurement in the computational basis.

given by

$$|\langle\psi|\phi\rangle|^2 = 2P_{\text{same}} - 1. \tag{8.34}$$

Here, P_{same} is the probability that an even number of pairs shows the result $|1,1\rangle$. Calculation in the assignment.

While the original swap test only measures an ancilla qubit and leaves the embedded states intact, the destructive swap test destroys all the qubits with final measurements in the computational basis.

The destructive swap test uses $2n$ qubits (compared to $2n+1$ in the swap test), has a circuit depth of $u+2$ (compared to $u+1$), but uses only n CNOT gates (compared to $18n$). Hence, it is much less resource demanding in terms of CNOT gates than the swap test but equally shallow.

8.3 Quantum Kernel Functions

Quantum devices can embed data into quantum states $\mathbf{x} \mapsto |\phi(\mathbf{x})\rangle$ living in high-dimensional spaces. There, we can access the inner products by measuring their overlaps $\kappa(\mathbf{x},\mathbf{x}') = |\langle\phi(\mathbf{x})|\phi(\mathbf{x}')\rangle|^2$. For these approaches to offer an advantage, they need to have the following properties:

(1) The feature transformation needs to increase the separability of the problem.

(2) The kernel $\kappa(\mathbf{x},\mathbf{x}')$ cannot be efficiently computed classically.

Point (1) is difficult to prove or disprove in general, but can be shown for artificial problems. Point (2) depends a lot on the way we embed the data vectors $\mathbf{x} = (x_1, x_2, \ldots)$ into a quantum

state of n qubits.

Basis Encoding

We recall: Each data point $\mathbf{x}$ has n binary features, i.e. $\mathbf{x} \in \{0,1\}^n$. We use n qubits, and this time we index them with $k = 0,\ldots,n-1$ (and not from 1 to n). If $x_k = 0$ we do nothing on qubit k, and if $x_k = 1$ we apply an X gate. In other words we use those features as bits in the binary representation of an integer $i(\mathbf{x}) = \sum_{k=0}^{n-1} 2^k x_k$ and prepare the computational basis state $\mathbf{x} \mapsto |i(\mathbf{x})\rangle = |x_0, x_1, \ldots, x_{n-1}\rangle$.

Since all basis states are orthonormal, we have

$$\kappa(\mathbf{x},\mathbf{x}') = \left|\langle i(\mathbf{x})|i(\mathbf{x}')\rangle\right|^2 = \delta_{\mathbf{x},\mathbf{x}'}. \tag{8.35}$$

This is a very bad kernel with basically no generalization capabilities, and it can also be easily computed classically. No quantum advantage can be expected.

Amplitude Encoding

We recall: Each data point $\mathbf{x}$ consists of real features, and this time we choose the convention to have N such features $x_0, x_1, \ldots, x_{N-1}$, i.e. $\mathbf{x} \in \mathbb{R}^N$. Normalization $\|\mathbf{x}\| = 1$ is assumed. We use $n = \lceil \log_2(N) \rceil$ qubits, as they have at least N computational basis states $|i\rangle$, $i = 0,\ldots,N-1$. Now we prepare a state where the amplitude of the basis state $|i\rangle$ is the value of the i-th feature of $\mathbf{x}$, i.e. $\mathbf{x} \mapsto |a(\mathbf{x})\rangle = \sum_{i=0}^{N-1} x_i\,|i\rangle$. Note: Finding the circuit that prepares these states can be very difficult.

Again, with the orthonormality of basis states $\langle j|i\rangle = \delta_{i,j}$:

$$\begin{aligned}\kappa(x,x') &= \left|\langle a(\mathbf{x})|a(\mathbf{x}')\rangle\right|^2 \\ &= \left|\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} x_i x'_j \langle j|i\rangle\right|^2 = \left|\sum_{i=0}^{N-1} x_i x'_i\right|^2 = \left|\mathbf{x}^T\mathbf{x}'\right|^2\end{aligned} \tag{8.36}$$

This is simply the squared linear kernel and can also be computed classically. It is certainly not worth going through the extremely difficult amplitude preparation procedure for this kernel.

Rotational Encoding

We recall: Each datapoint $\mathbf{x}$ has n real features, i.e. $\mathbf{x} \in \mathbb{R}^n$, and we use n qubits. We embed the data as $\mathbf{x} \mapsto |r(\mathbf{x})\rangle$. Here, $|r(\mathbf{x})\rangle$ is the quantum state initialized in $|0\rangle^{\otimes n}$ where on each qubit k a Pauli-Y rotation $R_Y(\theta) = \exp(-i\frac{\theta}{2}Yt) = \cos(-\frac{\theta}{2})\mathbb{1} + i\sin(-\frac{\theta}{2})Y$ with angle $\theta = x_k$ was applied: $|r(\mathbf{x})\rangle = \prod_{k=0}^{n-1} R_Y^{(k)}(x_k)|0\rangle^{\otimes n}$. The covector is obtained by rotating in the negative direction.

Since the states separable, we can decompose the scalar product of the n-qubit states into n individual products for the individual qubits:

$$\begin{aligned}\kappa(\mathbf{x},\mathbf{x}') &= \left|\langle r(\mathbf{x})|r(\mathbf{x}')\rangle\right|^2 \\ &= \left|\langle 0|^{\otimes n} \prod_{k=0}^{n-1} R_Y^{(k)}(-x_k) \prod_{k'=0}^{n-1} R_Y^{(k')}(x'_{k'})|0\rangle^{\otimes n}\right|^2\end{aligned}$$

$$= \prod_{k=0}^{n-1} \left| \langle 0 | R_Y(-x_k) R_Y(x'_k) | 0 \rangle \right|^2 = \prod_{k=0}^{n-1} \left| \langle 0 | R_Y(x'_k - x_k) | 0 \rangle \right|^2$$
$$= \prod_{k=0}^{n-1} \left| \langle 0 | \left[\cos(\tfrac{x'_k - x_k}{2}) \mathbb{1} - i \sin(\tfrac{x'_k - x_k}{2}) Y \right] | 0 \rangle \right|^2 = \prod_{k=0}^{n-1} \left| \cos(\tfrac{x'_k - x_k}{2}) \right|^2. \quad (8.37)$$

In the third line, we stopped to index onto which qubit the rotation gates are acting. This kernel can also be easily calculated on a classical computer.

Instantaneous Quantum Polynomial Encoding

What is commonly done is to use the given data to parameterize a non-trivial Hamiltonian under which we then evolve some fixed initial state. A widely used example is the Instantaneous Quantum Polynomial (IQP) embedding. When restricted to 1- and 2-qubits interactions, it embeds the data point $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ as follows:[2]

$$\mathbf{x} \mapsto |\phi(\mathbf{x})\rangle = U_{\text{IQP}}(\mathbf{x}) H^{\otimes n} |0\rangle^{\otimes n} \quad (8.38)$$

with

$$U_{\text{IQP}} = \exp\left(i \sum_{j=1}^{n} x_j Z_j + i \sum_{j<k} x_j x_k Z_j Z_k \right). \quad (8.39)$$

The quantum kernel associated with the IQP embedding is

$$\kappa(\mathbf{x}, \mathbf{x}') = \left| \langle \phi(\mathbf{x}) | \phi(\mathbf{x}') \rangle \right|^2$$
$$= \left| \langle + |^{\otimes n} U_{\text{IQP}}^{\dagger}(\mathbf{x}) U_{\text{IQP}}(\mathbf{x}') | + \rangle^{\otimes n} \right|^2$$
$$= \left| \langle + |^{\otimes n} \exp\left(i \sum_{j=1}^{n} (x'_j - x_j) Z_j + i \sum_{j<k} (x'_j x'_k - x_j x_k) Z_j Z_k \right) | + \rangle^{\otimes n} \right|^2. \quad (8.40)$$

This expression shows that the IQP kernel depends on both linear and quadratic differences of the input features and implicitly encodes high-dimensional polynomial correlations through the quantum feature map. The exact classical evaluation of the IQP kernel scales exponentially in the number of qubits for generic inputs and is thus not possible efficiently. The kernel is, however, efficiently accessible quantumly, i.e. it does not require an exponential overhead (in particular: number of gates) in the number of qubits.

8.4 Quantum Advantage

While quantum computers can efficiently compute inner products of exponentially large density operators that are classically hard, such large feature spaces (i) make generalization difficult and (ii) do not by themselves guarantee a quantum advantage, since even classically tractable kernels can correspond to high- or infinite-dimensional RKHSs. It was shown [4] that a quantum advantage is possible only when the RKHS is effectively low dimensional and contains functions that are hard to compute classically, allowing a quantum model

[2] Often, two layers are used: $\mathbf{x} \mapsto |\phi(\mathbf{x})\rangle = U_{\text{IQP}}(\mathbf{x}) H^{\otimes n} U_{\text{IQP}}(\mathbf{x}) H^{\otimes n} |0\rangle^{\otimes n}$.

to encode a useful inductive bias that cannot be efficiently imposed classically. However, identifying such kernels is challenging, as their evaluation may require exponentially many measurements, leading to the conclusion that quantum speed-ups are likely only when strong problem-specific structure can be encoded into quantum circuits, a scenario more plausible for quantum-generated than for classical data.

Projected kernels address a key problem in standard quantum kernels where, in high-dimensional Hilbert spaces, data points tend to appear nearly orthogonal so the kernel matrix becomes almost identity and offers little useful structure for learning [3]. By projecting quantum states onto a lower-dimensional classical representation (e.g. via reduced observables), they reduce the effective dimension and improve the geometric distinction between data, enabling better generalization and stronger empirical performance than unprojected quantum kernels.

Bibliography

Articles

[1] M. Cerezo, G. Verdon, H.-Y. Huang, L. Cincio, and P. J. Coles. "Challenges and opportunities in quantum machine learning". In: *Nature Computational Science* 2 (2022), page 567. DOI: https://doi.org/10.1038/s43588-022-00311-3 (cited on pages 6–8).

[2] Y. Chen, J-H. Huang, Y. Sun, Y. Zhang, Y. Li, and X. Xu. "Haplotype-resolved assembly of diploid and polyploid genomes using quantum computing". In: *Cell Reports Methods* 4 (2024), page 100754. DOI: https://doi.org/10.1016/j.crmeth.2024.100754 (cited on page 26).

[3] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, H. Neven, and J. R. McClean. "Power of data in quantum machine learning". In: *Nature Communications* 12 (2021), page 2631. DOI: https://doi.org/10.1038/s41467-021-22539-9 (cited on page 65).

[4] M. K.. Kübler, S. Buchholz, and B. Schölkopf. "The Inductive Bias of Quantum Kernels". In: *arXiv:2106.03747* (2021). DOI: https://doi.org/10.48550/arXiv.2106.03747 (cited on page 64).

Books

[5] M. Schuld and F. Petruccione. *Machine Learning with Quantum Computers*. 2nd Edition. Springer, 2021 (cited on pages 9, 11, 16, 19, 22, 35, 36, 40, 42, 45, 48, 51, 54, 55, 57).