Karlsruhe Institute of Technology

Training Quantum Kernels for Clustering Algorithms

Master's Thesis of

Niklas Metz

at the Department of Informatics
Steinbuch Centre for Computing

Reviewer:	Prof. Dr.-Ing. Achim Streit
Second reviewer:	Prof. Dr.-Ing. Bernhard Neumair
Advisor:	Dr.-Ing. Eileen Kühn
Second advisor:	Christof Wendenius, M.Sc.

02. November 2021 – 02. May 2022

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 02. May 2022

. . . . . .

(Niklas Metz)

Abstract

Clustering is an important machine learning (ML) task, especially for applications dealing with large amounts of data. In these cases, clustering is used to detect structure in the data by grouping together similar datapoints. In this context, similarity is no strictly defined measure. Similarity is rather defined for each individual situation and dataset. This is the reason why there is no single clustering algorithm that is applicable to all situations, as each algorithm assumes a different cluster model. Furthermore, many clustering algorithms use interchangeable kernel functions that define the similarity between two samples.

These kernel functions can not only be calculated on classical computers, but recent research has shown that quantum computers are capable of computing them as well. Quantum computers are generally known to be able to solve specific problems, which are intractable on classical computers. The hope is that there are quantum kernel functions, which are classically intractable, and pose an advantage over classical kernel functions.

Furthermore, these quantum kernel functions can be defined in combination with variational quantum circuits (VQCs). VQCs are similar to classical neural networks (NNs) in that they have free parameters that can be optimized to represent a target function. Combining quantum kernels with VQCs makes the kernel function adjustable to reflect the actual structure of the provided data. This idea was first proposed in [Hub+21] and combined with a support vector machine (SVM).

This thesis instead proposes a combination of variational quantum kernels (VQKs) with classical clustering algorithms. While clustering is generally defined as a task of unsupervised learning, optimizing the parameters of a VQK is difficult without any labeled training data. Therefore this thesis considers two ways of training the VQK, completely unsupervised, where no labeled training data is available, and semi-supervised, where a small amount of labeled training data is available. The trained VQKs are then combined with various clustering algorithms that can utilize kernel functions.

In order to evaluate the proposed approach, a training and testing pipeline was implemented using a simulated quantum computer on classical hardware.

Extensive experiments show that the VQKs can be optimized using different semi-supervised cost functions. Additionally, the trained quantum kernel can be used with many different classical clustering algorithms and perform well. In most cases the quantum kernel performs as well as or even better than the popular classical radial basis function (RBF) kernel. However, the unsupervised learning approaches proposed in this thesis were shown to not be able to optimize the kernel function.

Zusammenfassung

Clustering ist eine außerordentlich wichtige Sparte des Maschinellen Lernens von der besonders Anwendungen, die sich mit großen Datenmengen beschäftigen, profitieren können. In diesen Anwendungen wird Clustering verwendet um Strukturen in Daten zu finden und ähnliche Datenpunkte zu gruppieren. Allerdings ist Ähnlichkeit in diesem Kontext kein wohldefiniertes Maß und hängt stark von der jeweiligen Situation und dem Datensatz ab. Diese Situationsabhängigkeit ist auch der Grund weshalb es nicht nur einen Clustering Algorithmus gibt. Vielmehr gibt es unterschiedliche Modelle dafür wie ein Cluster definiert sein kann und entsprechend unterschiedliche Algorithmen, die Cluster basierend auf diesen Modellen finden können. Außerdem verwenden viele dieser Algorithmen austauschbare Kernel-Funktionen, welche die Ähnlichkeit zwischen zwei Datenpunkten definiert.

Diese Kernel-Funktionen können allerdings nicht ausschließlich auf klassischen Computern berechnet werden, denn jüngste Forschung im Bereich des Quanten Maschinellen Lernens zeigt, dass auch Quantencomputer verwendet werden können um Kernel-Funktionen zu berechnen. Quantencomputer sind hauptsächlich dafür bekannt bestimmte Probleme, die im Klassischen unlösbar oder nur schwer lösbar sind, effizient berechnen zu können. Die Hoffnung ist, dass es bestimmte Quanten Kernel-Funktionen gibt die klassisch nicht berechnet werden können, aber einen Vorteil gegenüber klassischen Kernel-Funktionen darstellen.

Quanten Kernel-Funktionen können des weiteren mit variierbaren Quanten Schaltkreisen kombiniert werden. Diese Quanten Schaltkreise sind sehr ähnlich zu klassischen Neuronalen Netzen, denn beide besitzen variierbare Parameter die optimiert werden können, um eine bestimmte Zielfunktion zu realisieren. Die Kombination einer Quanten Kernel-Funktion und einem variierbaren Quanten Schaltkreis, auch variierbarer Quanten Kernel genannt, sorgt dafür, dass die Kernel-Funktion durch Training an den vorhandenen Datensatz angepasst werden kann. Diese Idee wurde zuerst von den Autoren von [Hub+21] eingeführt, die die Kernel-Funktion mit einer Support Vector Machine kombiniert haben.

Im Vergleich dazu schlägt diese Arbeit eine Kombination von variierbaren Quanten Kernels mit klassischen Clustering Algorithmen vor. Da Clustering generell im Bereich des unüberwachten Maschinellen Lernens angesiedelt ist, wird die Optimierung der Parameter ohne klassifizierte Trainingsdaten schwierig. Deshalb betrachtet diese Arbeit zwei unterschiedliche Wege die Parameter zu trainieren. Einerseits vollständig unüberwacht, wodurch keine klassifizierten Trainingsdaten vorhanden sind, andererseits semi-überwacht, bei dem kleine Mengen von klassifizierten Daten für das Training verwendet werden können. Die trainierten Quanten Kernels werden anschließend mit unterschiedlichen klassischen Clustering Algorithmen, die Kernel-Funktionen nutzen können, kombiniert.

Um diesen Ansatz zu evaluieren wurde eine Trainings- und Testpipeline implementiert, die simulierte Quantencomputer auf klassischer Hardware verwendet.

Umfangreiche Experimente zeigen, dass variierbare Quanten Kernels mit unterschiedlichen semi-überwachten Kostenfunktionen trainiert werden können. Desweiteren können diese trainierten Quanten Kernels in klassischen Clustering Verfahren gute Ergebnisse erzielen. In den meisten Fällen sind die Clustering Ergebnisse mit trainierten Quanten Kernels besser als mit dem häufig benutzten, klassischen radial basis (rbf) Kernel. Allerdings konnten für die vorgestellten unüberwachten Kostenfunktionen keine guten Ergebnisse erreicht werden, weshalb diese momentan nicht geeignet sind, um einen variierbaren Quanten Kernel zu trainieren.

Contents

List of Figures

1. Introduction

Many applications rely on grouping similar data together. Especially when dealing with large amounts of data it is in many cases interesting to separate the data into groups and discover patterns. This grouping of similar samples is the machine learning task of clustering. However, there is not one specific algorithm that is able to solve the clustering task in all situations. This is due to the fact that there is not one consistent definition of what constitutes a good grouping. This led to the development of many different clustering algorithms, each with a unique definition of a cluster model. Many of these algorithms rely on a similarity measure to decide if two samples are similar to each other and if they should be grouped together. One type of possible similarity measures are kernel functions, which are predominantly known from support vector machines (SVMs) and can calculate similarities in implicit, high-dimensional spaces.

Recent advances in building quantum computers have sparked an interest in designing adequate algorithms. Quantum computing (QC) is based on the well-known theories of quantum mechanics. The core concepts of computations are very similar to classical computers. Quantum bits (qubits), the quantum equivalent of classical bits, are processed by quantum gates, which are similar to classical logic gates, in order to implement some algorithm. These sequences of gates are referred to as quantum circuits, analogously to classical circuits. There are multiple different quantum circuits and algorithms that were shown to outperform their best classical counterpart.

Furthermore, QC can be combined with classical machine learning (ML) to a new field of study called quantum machine learning (QML). There are some proposals of transferring classical clustering algorithms to quantum computers or even designing completely new quantum clustering algorithms. It can be shown that some of these algorithms in theory perform better than classical clustering. However, many of these algorithms require a large number of qubits. This is problematic in the current era of noisy intermediate scale quantum (NISQ) devices, as real quantum hardware is heavily affected by noise, which currently prohibits building quantum computers with many qubits.

With these problems in mind, the attention turned to replacing smaller parts of classical ML algorithms with quantum subroutines. As it turns out, the kernel functions used in clustering algorithms, can also be calculated using a quantum computer by designing specific quantum circuits. As kernel functions calculate a similarity between two samples at a time, quantum kernels are possible in the current NISQ era. The hope is that there are specific quantum kernels that are classically intractable and pose an advantage over common classical kernel functions.

Furthermore, quantum kernels can not only be calculated by fixed quantum circuits, but also through the usage of variational quantum circuits (VQCs). These circuits are similar to classical neural networks (NNs), in that they take some parameters that can be optimized to represent a target function. This combination leads to the introduction of variational

quantum kernels (VQKs), where the kernel function can be optimized to represent the given data. This idea was first introduced in [Hub+21] and used in combination with a support vector machine (SVM).

This thesis proposes a combination of VQKs with classical clustering algorithms. The goal is to increase the clustering performance by training the quantum circuit to reflect the similarities in the dataset. While clustering is generally considered an unsupervised task, optimizing parameters without labeled training data is difficult. This is why this thesis introduces different approaches of unsupervised training as well as semi-supervised training, where typically a small amount of labeled training data is available.

In order to show the usability of VQKs in clustering, a pipeline for training and testing was implemented in the context of this thesis. This implementation is used with different toy and real-world datasets to evaluate its performance.

Chapter 2 introduces the necessary basics, which are important for the remainder of this thesis. Starting with an explanation of some classical clustering algorithms that can utilize kernel functions. Afterwards some necessary basic concepts from QC, as well as more advanced topics from QML are introduced.

The approach of this thesis, including the pipeline for training and testing, is shown in Chapter 3. This section introduces some possible cost functions used for optimizing the quantum kernel for unsupervised and semi-supervised learning.

Chapter 4 evaluates the general clustering performance compared to classical kernel functions. Furthermore, this chapter also investigates which approaches from Chapter 3 are capable of optimizing the kernel function. Important parameters that influence the clustering performance are detailed as well.

Lastly, Chapter 5 summarizes the important results of this thesis and discusses some possible further research topics.

2. Foundations

The popularity of machine learning in science and the general media has skyrocketed in recent years. However the concepts and fundamentals behind this topic are nothing new. The recent popularity can be attributed to the fact that computing resources are finally affordable and able to satisfy the performance demands of complex machine learning algorithms.

The same goes for quantum computers. The general theory of quantum mechanics is known since the 20th century. But only recently the first bigger quantum computers were introduced with up to 127 qubits [Tak+21]. These advances have led to much research into the possible advantages of quantum computers for computer science and other fields. One topic that quickly came into focus for a possible combination with quantum computers was machine learning. Many classical algorithms were and are currently being changed to work with quantum computers, possibly leading to advances compared to classical computers [SP18].

The following chapters introduce the necessary foundations for classical machine learning and quantum computing. Section 2.1 details some concepts and algorithms from machine learning. The ideas of quantum computing and quantum machine learning are introduced in Section 2.2.

2.1. Machine Learning

Machine learning can be defined as a group of algorithms that use experience to improve their performance. This experience generally comes through learning from available data, often called training data. Using the training data the algorithm is able to make predictions on unseen data, which is similar to the provided training data.

This rather general definition of machine learning can be further refined into some subcategories, which are presented in the following. While there are more possible categories [MRT12], supervised, unsupervised and semi-supervised learning are the important ones in the scope of this thesis.

Supervised Learning The algorithm receives training data containing the datapoints and their assigned labels. Using this data the algorithm has to learn from it and predict the labels for new and unseen data. This approach is most commonly used for assigning discrete or continuous labels to the data, also called classification and regression.

Unsupervised Learning Only unlabeled training data is provided to the algorithm and can be used for training. Afterwards the algorithm has to assign labels to all unlabeled datapoints. Common scenarios are *clustering*, where unlabeled data is assigned to similar groups, or dimensionality reduction used to transform data into lower dimensional spaces.

Semi-supervised Learning This type is the middleground between supervised and unsupervised learning. The algorithm receives labeled and unlabeled training data and predicts labels for unseen data. Most commonly the training data only consists of a small amount of labeled data. Many algorithms typically associated with supervised and unsupervised learning can also be phrased as semi-supervised learning. It is especially interesting how good their performance is considering the small amount of available labeled training data.

The next sections introduce some concepts from machine learning that are important for understanding this thesis. Namely kernel methods and some fundamental algorithms for clustering.

2.1.1. Linear Separability

Many machine learning algorithms, including classification and clustering, rely on separating data. The easiest way to separate data is by using a hyperplane whose dimension is one less than that of its euclidean space. If the data can be perfectly separated by a hyperplane, it is called *linear separable*. This can be seen in Figure 2.1 for a 2-dimensional example. However, this is not possible in most cases. As can be seen in Figure 2.2, there are cases where using a hyperplane does not separate the data perfectly into the two classes defined by green and blue datapoints. This is called *non-linearly separable*. The alternative for non-linearly separable datasets is using non-linear functions.

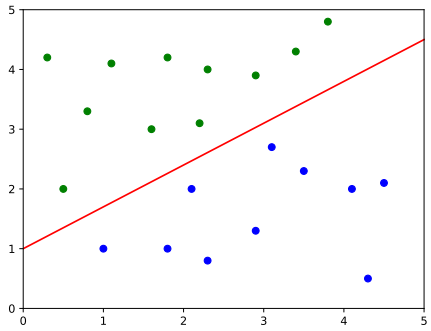


Figure 2.1.: Samples can be perfectly separated by a linear function

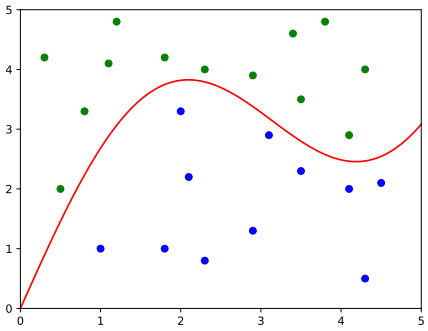


Figure 2.2.: Samples that are non-linearly separable

2.1.2. Kernels

The previous section introduced the concept of linear separability and the fact that more complex functions are necessary for data that is non-linearly separable. Necessarily using complex functions can be prevented by non-linearly mapping data from an input space X into higher dimensional spaces H, in which the data becomes linearly separable. Such a function $\phi : X \longrightarrow H$ that maps data from the input space X into a feature Hilbert space H is called a *feature map*. However, just mapping the data into a higher dimensional feature space and using this transformed data in algorithms is not always feasible. This is due

to a very high or even infinite dimension of the feature space, thus making the explicit mapping very calculation intense or even impossible.

This is where *kernels* come into play. A kernel $k : X \times X \rightarrow \mathbb{R}$ is a function that maps two datapoints from the input set $X = \{x_1, \ldots, x_m\}$ to a real number [MRT12]. Furthermore the matrix

$$K = [k(x_i, x_j)]_{ij} \in \mathbb{R}^{mxm} \qquad 1 \leq i, j \leq m,$$

called the kernel- or Gram matrix, has to be positive semidefinite [MRT12]. This requirement is equivalent to

$$\sum_{i,j=1}^{N} c_i c_j^* k(x_i, x_j) \geq 0 \tag{2.1}$$

for any subset $\{x_1, \ldots, x_N\} \subseteq X$ and $c_1, \ldots, c_N \in \mathbb{R}$ [MRT12].

Using a feature map $\phi : X \rightarrow H$, a kernel function can be defined as

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_H,$$

using the inner product $\langle \cdot, \cdot \rangle_H$ defined on the Hilbert space H. To proof that this is a valid kernel function, one has to show that the condition in Equation (2.1) is fulfilled. For any $\{x_1, \ldots, x_N\} \subseteq X$ and $c_1, \ldots, c_N \in \mathbb{R}$ one can show that

$$\begin{aligned}
\sum_{i,j=1}^{N} c_i c_j^* k(x_i, x_j) &= \sum_{i,j=1}^{N} c_i c_j^* \langle \phi(x_i), \phi(x_j) \rangle_H \\
&= \langle \sum_{i=1}^{N} c_i \phi(x_i), \sum_{j=1}^{N} c_j \phi(x_j) \rangle_H \\
&= || \sum_{i=1}^{N} c_i \phi(x_i) ||^2 \geq 0 \qquad \square
\end{aligned}$$

This shows that every feature map induces a kernel function using the inner product between the results of the feature mapping, also called *feature vectors*. Also, as these kernels are defined through the inner product, they can be interpreted as similarity measures between the datapoints.

The advantage of feature maps is the possibly non-linear mapping to higher dimensional spaces, where the data is linearly separable. However, explicitly mapping each datapoint into this space is expensive or impossible depending on its dimension. This is where kernel functions can become useful. If the algorithm in question can be rewritten to use the inner product between feature vectors $\langle \phi(x_i), \phi(x_j) \rangle$, this inner product can be replaced by a kernel function. Instead of explicitly mapping the input vectors into higher dimensional spaces and calculating their inner product, the kernel functions offer an equation without explicitly doing the transformation into higher dimensions. This practice is known as the *kernel trick* [HSS08].

An example for kernel functions are polynomial kernels defined as follows [MRT12]:

$$k(x,y) = (x^T y + c)^d \qquad x, y \in \mathbb{R}^m, d \in \mathbb{N}, c \in \mathbb{R} > 0.$$

For $m = 2$, $d = 3$ and $c = 1$ this leads to

$$\begin{aligned} k(x,y) &= (\begin{pmatrix} x_1, & x_2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} + 1)^3 \\ &= (x_1 y_1 + x_2 y_2 + 1)^3 \\ &= 1 + x_1^3 y_1^3 + x_2^3 y_2^3 + 3x_1 y_1 + 3x_2 y_2 + 3x_1^2 y_1^2 x_2 y_2 \\ &+ 3x_1^2 y_1^2 + 3x_2^2 y_2^2 + 3x_1 y_1 x_2^2 y_2^2 + 6x_1 y_1 x_2 y_2. \end{aligned}$$

This is the same as the inner product between

$$\begin{pmatrix} 1, & x_1^3, & x_2^3, & \sqrt{3}x_1, & \sqrt{3}x_2, & \sqrt{3}x_1^2 x_2, & \sqrt{3}x_1^2, & \sqrt{3}x_2^2, & \sqrt{3}x_1 x_2^2, & \sqrt{6}x_1 x_2 \end{pmatrix},$$

and

$$\begin{pmatrix} 1, & y_1^3, & y_2^3, & \sqrt{3}y_1, & \sqrt{3}y_2, & \sqrt{3}y_1^2 y_2, & \sqrt{3}y_1^2, & \sqrt{3}y_2^2, & \sqrt{3}y_1 y_2^2, & \sqrt{6}y_1 y_2 \end{pmatrix},$$

which leads to the feature map

$$\begin{aligned} \phi(x) &= \phi(x_1, x_2) \\ &= \begin{pmatrix} 1, & x_1^3, & x_2^3, & \sqrt{3}x_1, & \sqrt{3}x_2, & \sqrt{3}x_1^2 x_2, & \sqrt{3}x_1^2, & \sqrt{3}x_2^2, & \sqrt{3}x_1 x_2^2, & \sqrt{6}x_1 x_2 \end{pmatrix}. \end{aligned}$$

In conclusion, the feature map ϕ maps datapoints to a 10-dimensional space. The kernel $(x^T y + 1)^3$ calculates an inner product between $\phi(x)$ and $\phi(y)$ without explicitly visiting this 10-dimensional space, which shows the advantage of using kernels.

2.1.3. Clustering

Clustering is the task of grouping together datapoints so that samples from one group, or cluster, are more similar to samples from the same cluster than from other clusters. However, there is not only one reference algorithm for clustering, because the notion of clusters and similarity is not perfectly defined [Est02a]. Depending on the use case there are many different ways of defining a cluster and again multiple different algorithms that are able to find these clusters. In the following some of these cluster-definitions are presented.

Centroid based clustering defines a cluster through a mean vector of all samples assigned to the cluster. A popular algorithm for this class is the k-means algorithm [Mac67]. Even though weighted kernel k-means [DGK04] does not represent a cluster internally as a mean vector, it can be assigned to this class of clustering. This is due to the fact that it is an extension of the classical k-means algorithm and predominantly uses the mean in its calculations. Spectral clustering [Lux07] can also be seen as a centroid based clustering algorithm, as it uses k-means after a preprocessing of the input data.

Hierarchical clustering methods define clusters through a hierarchical structure [Nie16]. There are two different strategies of building this hierarchy. Bottom-up, or agglomerative clustering, starts with each datasample in its own cluster and iteratively merges them to bigger clusters. Top-down, or divisive clustering, starts with one cluster of all samples and iteratively separates it into smaller clusters. A linkage criteria determines the dissimilarity between sets of datapoints, while a dissimilarity functions calculates pairwise dissimilarities. Linkage criteria and dissimilarity function combined are used to determine which clusters are merged or separated.

Density based clustering defines a cluster as a volume with many datapoints or a high density of datapoints [Kri+11]. To detect these areas of higher densities, a distance or dissimilarity measure is required to group the samples. One of the most popular density based algorithms is DBSCAN [Est+96]. Common alternatives are OPTICS [Ank+99] or the mean-shift algorithm [Che95].

In the following sections some of the mentioned algorithms are explained in greater detail.

2.1.3.1. k-means

Standard k-means [Mac67] is based on finding k clusters, which are represented by k different mean vectors $m_1, \ldots, m_k$, in a set of N samples. The algorithm to find these cluster centroids is shown in Algorithm 1.

Algorithm 1 k-means

Require: Dataset D, number of clusters k, number of $steps$

1: Initialize k clusters:
2: Sets of points for clusters $S_1, \ldots, S_k$
3: Cluster centers $m_1, \ldots, m_k$
4: **for** $steps$ **do**
5: **for all** Points $p \in D$ **do**
6: Assign p to the cluster S_i with the closest cluster center $m_i \rightarrow argmin_i ||p - m_i||^2$
7: **end for**
8: **for all** Clusters S_i **do**
9: Update the cluster center m_i as the mean over all points in S_i
10: **end for**
11: **if** the assignment has not changed since the last step **then**
12: k-means has converged → Stop
13: **end if**
14: **end for**

This algorithm can be reduced to two steps, the assignment and update step. In the assignment step (line 6), each sample is assigned to the cluster with the closest cluster center. In the update step (line 9), the cluster centers are recalculated as the mean over all samples that are assigned to the cluster. Once the assignment of samples to a cluster does

not change in the next step, the algorithm has converged, as further steps also would not change the assignment.

Note that k-means produces circular clusters due to the assignment step using the euclidean distance. These circular clusters are one restriction of k-means, as datasets with clusters in other shapes, which are not linearly separable, cannot be clustered correctly.

2.1.3.2. Weighted kernel k-means

Kernel k-means [DGK04] is a variant of the standard algorithm shown in Section 2.1.3.1. It tries to solve the problem of non-linearly separable data, by using a kernel function to transform the input data into a higher dimensional space. Furthermore this version adds the option to assign a weight w_i to each input sample a_i.

Kernel k-means tries to find k clusters $C_1, \ldots, C_k$, which are defined by the set of samples that belong to the given cluster. The objective function is given by:

$$D(C_i) = \sum_{i=1}^{k} \sum_{a \in C_i} w(a)||\phi(a) - m_i||^2 \qquad m_i = \frac{\sum_{b \in C_i} w(b)\phi(b)}{\sum_{b \in C_i} w(b)}, \tag{2.2}$$

with m_i being the cluster center for cluster C_i and ϕ being a feature map [DGK04]. The objective function is intuitively explained as the sum over the distances between each embedded sample and all cluster centers. After minimizing this function, each sample is assigned to the nearest cluster center.

However, the objective function as defined in Equation (2.2) cannot be optimized, as ϕ may be projecting a into an infinite dimensional space in which case $\phi(a)$ cannot be explicitly calculated [DGK04]. To resolve this, the euclidean distance between $\phi(a)$ and m_i in Equation (2.2) is rewritten as [DGK04]:

$$||\phi(a) - m_i||^2 = \left\| \phi(a) - \frac{\sum_{b \in C_i} w(b)\phi(b)}{\sum_{b \in C_i} w(b)} \right\|^2 \tag{2.3}$$

$$= \phi(a)\phi(a) - \frac{2\sum_{b \in C_i} w(b)\phi(a)\phi(b)}{\sum_{b \in C_i} w(b)} + \frac{\sum_{b,c \in C_i} w(b)w(c)\phi(b)\phi(c)}{(\sum_{b \in C_i} w(b))^2}. \tag{2.4}$$

Note that Equation (2.4) only uses products between two feature vectors, which allows us to apply the kernel trick and calculate these products using a kernel function. This leads to the final Equation (2.5) to calculate the distance between $\phi(a)$ and m_i [DGK04].

$$||\phi(a) - m_i||^2 = k(a,a) - \frac{2\sum_{b \in C_i} w(b)k(a,b)}{\sum_{b \in C_i} w(b)} + \frac{\sum_{b,c \in C_i} w(b)w(c)k(b,c)}{(\sum_{b \in C_i} w(b))^2}. \tag{2.5}$$

Using Equation (2.5), the weighted kernel k-means algorithm can be defined as seen in Algorithm 2.

Note that this algorithm is very similar to the standard k-means algorithm, as each sample is assigned to a specific cluster. The differences are that clusters are defined over a set of points instead of a mean, and that distances between a sample and the clusters are not calculated using euclidean distance but a kernel function. This kernel function allows weighted kernel k-means to find clusters for not linearly separable datasets. However, this feature is heavily confined to the quality of the kernel.

Algorithm 2 Weighted kernel k-means

Require: Dataset D, number of clusters k, kernel function $k(\cdot,\cdot)$, weights w

Initialize k clusters $C_1, \ldots, C_k$
while True **do**
    for all Points $p \in D$ **do**
        Find the new cluster index $j(p)$ for p by solving $j(p) = argmin_i ||\phi(a) - m_i||^2$ using Equation (2.5)
    end for
    for all Clusters C_i **do**
        Compute new clusters $C_i = \{p | j(p) = i\}$
    end for
    if the assignment has not changed since the last step **then**
        Algorithm has converged → Stop
    end if
end while

2.1.3.3. DBSCAN

DBSCAN is short for density-based spatial clustering of applications with noise [Est+96]. As the name suggests, it is based on finding clusters in regions with a high density of data samples. To specify how dense an area is, DBSCAN uses a distance metric.

DBSCAN distinguishes between three different types of samples. Samples inside of a cluster (*core points*), samples on the border of a cluster (*border points*) and outliers, which are not part of any cluster (*noise points*).

The ϵ-neighbourhood of a point p is defined by all points q that lie within a distance ϵ from p [Est+96]. The mathematical definition for this is $N_\epsilon(p) = \{q \in D | dist(p,q) \leq \epsilon\}$, with D being the set of points and $dist(\cdot,\cdot)$ being a distance measure. With that a core point p is defined as a point with an ϵ-neighbourhood of at least $MinPts$ other points ($|N_\epsilon(p)| \geq MinPts$). Border points do not fulfill the core point condition, but lie within the ϵ-neighbourhood of a core point.

The cluster model of DBSCAN can be described as follows. A point p is called *directly density-reachable* from another point q, if q is a core point and p is in the ϵ-neighbourhood of q [Est+96]. Furthermore, if a sequence of points $p = p_1, \ldots, p_n = q$ exists, with p_{i+1} being directly density-reachable from p_i, then q is called *density-reachable* from p [Est+96].

Directly density-reachability adds all points p in the ϵ-neighbourhood of a core point q to the cluster of the core point. These points p are either core points or border points. If any point is again a core point, their ϵ-neighbourhood is also added to the cluster. This is described by density-reachability. Border points are density-reachable from a core point of the cluster. All points which are not density-reachable from any core point are classified as noise.

This mathematical definition of a cluster model in DBSCAN can be translated into pseudo-code [Sch+17] as seen in Algorithm 3. This algorithm gradually expands the neighbourhood around core-points to include all neighbouring core- and border-points.

Algorithm 3 DBSCAN

Require: Dataset D
Require: $MinPts$ to be a core point
Require: ϵ to define neighbourhood
Require: $dist$ as distance measure

for all $p \in D$ **do**
  if $label(p) \neq undefined$ **then**
    continue ▷ Point was already processed
  end if
  Neighbours $N = \{\}$
  for all $q \in D$ **do**
    if $dist(p, q) \leq \epsilon$ **then**
      $N = N \cup \{q\}$ ▷ Get ϵ-neighbourhood of q
    end if
  end for
  if $|N| < MinPts$ **then**
    $label(p) = Noise$ ▷ Point is noise
    continue
  end if
  c = next cluster label ▷ Start new cluster
  $label(p) = c$
  for all $q \in N$ **do**
    if $label(q) = Noise$ **then**
      $label(q) = c$ ▷ Point was previously classified as noise
    end if
    if $label(q) \neq undefined$ **then**
      continue ▷ Point already has a cluster label
    end if
    $label(q) = c$
    for all $k \in D$ **do**
      if $dist(q, k) \leq \epsilon$ **then**
        $N = N \cup \{k\}$ ▷ Expand neighbourhood with all points in neighbourhood of q
      end if
    end for
  end for
end for

The advantages of DBSCAN are that firstly, it can detect clusters of any shape. Secondly it can detect outliers in the data and classify them as such. And lastly DBSCAN can find clusters without having to specify in advance how many clusters there are.

Problems of DBSCAN are that it is not deterministic, as a border point can potentially be part of multiple clusters, if it is reachable from multiple clusters [Sch+17]. Furthermore the algorithm relies on a distance metric and is therefore reliant on the quality of said distance metric.

2.1.3.4. Spectral Clustering

Spectral clustering uses dimensionality reduction of the dataset $X = \{x_1, \ldots, x_n\}$ before clustering the reduced dataset using standard clustering algorithms to find k different clusters [Lux07].

Spectral clustering requires the definition of a similarity function $s(x_i, x_j)$. Using this function, the similarity matrix $[S]_{i,j} \in \mathbb{R}^{n \times n}$ can be defined as the similarity between all pairs of samples (x_i, x_j). This matrix can be interpreted as a similarity graph. Each sample is seen as a vertex in the graph and two vertices are connected by an edge if they are similar to each other. The edges are typically weighted by the similarity of the two connected samples. Note that there are many different ways of constructing this similarity graph. Some examples are thresholding, where two vertices are only connected if their similarity is above a certain threshold value. Another way is to connect a vertex only to k other vertices with the highest similarity [Lux07].

The similarity graph can be represented by a weighted *adjacency matrix* $[W]_{i,j}$ that contains the weight associated with an edge between two vertices or 0 if the two vertices are not connected by an edge. Using the adjacency matrix W a Laplacian matrix can be defined as a representation of the similarity graph. Note that there are again many different ways of defining this *Laplacian matrix*. One way is to define the Laplacian matrix as [Lux07]:

$$L = D - W,$$

where D is defined as a diagonal matrix containing the number of connected vertices:

$$D_{ii} = \sum_j W_{ij}.$$

Using the Laplacian L, the first k eigenvectors $e_1, \ldots, e_k$ can be computed. They are used to form the matrix $U \in \mathbb{R}^{n \times k}$ that contains the eigenvectors e_i as columns. This is effectively a dimensionality reduction from $n \times n$ to $n \times k$, as k is typically much smaller than n. To calculate the final clusters, the n row vectors of U, each of dimension k, are clustered using standard clustering algorithms like k-means. The pseudocode [Lux07] for spectral clustering is shown in Algorithm 4.

Note that there are many different ways of specifying this clustering algorithm. Especially, there are many different ways of building the similarity graph and defining the Laplacian matrix. A good overview is provided in [Lux07].

Algorithm 4 Spectral Clustering

Require: Dataset $X = \{(x_i)\}_{i=1}^{n}$, similarity measure $s(x_i, x_j)$, number of clusters k

for all pairs (x_i, x_j) **do**
    $S_{ij} = s(x_i, x_j)$
end for
Construct a similarity graph and its adjacency matrix W
for $i = 1$ to n **do**
    $D_{ii} = \sum_j W_{ij}$
end for
Laplacian matrix $L = D - W$
Compute first k eigenvectors of L as $e_1, \ldots, e_k$
Define $U \in \mathbb{R}^{n \times k}$ as the matrix containing the eigenvectors e_i as columns
Define $y_i \in \mathbb{R}^k$ as the n row vectors of U
Cluster $\{(y_i)\}_{i=1}^{n}$ into k clusters using k-means

2.1.3.5. Hierarchical Clustering

Hierarchical clustering builds a hierarchy of clusters [Nie16]. Either bottom-up, also called *agglomerative*, through assigning each sample to its own cluster and iteratively merging these clusters. Or top-down, also called *divisive*, by creating one cluster of all samples and splitting it into smaller clusters.

For both variants two functions are required: a distance metric and a linkage criteria. The distance metric $dist(x, y)$ measures the distance between two samples x and y. There is a multitude of different distance metrics that can be used. Some of the more popular ones are the euclidean distance ($||a - b|| = \sqrt{\sum_i (a_i - b_i)^2}$) or manhattan distance ($||a - b|| = \sum_i |a_i - b_i|$). However, every function that is a metric can be used.

The linkage criteria is used to define the distance between two clusters, which possibly contain more than one sample. Typically a linkage criteria uses the distance metric to calculate the distance for all pairwise samples in these clusters. Equally as for the distance metric, there are many different ways of defining a linkage criteria. Some of the popular ones are maximum linkage or single linkage. Maximum linkage is the largest distance $d(x, y)$ between two samples from the clusters C_1 and C_2. Single linkage is analogously the smallest distance between two samples from both clusters.

$$\text{Maximum linkage: } max(\{d(x, y) | x \in C_1, y \in C_2\})$$
$$\text{Single linkage: } min(\{d(x, y) | x \in C_1, y \in C_2\})$$

The linkage criteria can then be used to merge of split clusters as long as the linkage stays under some threshold $distance\_threshold$.

2.2. Quantum Computing

This section explains some of the basic concepts from quantum computing. From qubits in Section 2.2.1 and gates in Section 2.2.2 over entanglement in Section 2.2.5 up to the effects

of quantum noise in Section 2.2.7. Afterwards the ideas of quantum machine learning, variational circuits and quantum kernels are explained.

2.2.1. Qubits

In classical computing everything is built up from the smallest amount of information, a bit. Each bit is either 0 or 1 at all times. Quantum computing has a similar concept called a quantum bit (qubit). Furthermore there are also similar basis states for qubits, $|0\rangle$ and $|1\rangle$, which are defined as follows [NC10]:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$
$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

The $|\cdot\rangle$ notation is called a *ket-vector* in the Dirac-notation [Dir39], which is used to represent quantum states. The Dirac notation also defines *bra-vectors* ($\langle\cdot|$), which are the complex conjugate and transposed of ket-vectors $\langle\cdot| = |\cdot\rangle^\dagger$. The basis states as bra-vectors look like:

$$\langle 0| = |0\rangle^\dagger = \begin{pmatrix} 1, & 0 \end{pmatrix}$$
$$\langle 1| = |1\rangle^\dagger = \begin{pmatrix} 0, & 1 \end{pmatrix}$$

In contrast to a classical bit, a qubit does not have to be in either of the two basis states $|0\rangle$ or $|1\rangle$ at all times. It can be a linear combination of these two basis states. This linear combination is called a *superposition* in quantum computing and leads to the general definition of a qubit [NC10]:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \qquad \alpha, \beta \in \mathbb{C}. \tag{2.6}$$

Equation (2.6) shows a qubit in ket-state as a superposition of the basis states $|0\rangle$ and $|1\rangle$ using the factors α and β, which are often called the amplitudes.

The analogous bra-state to (2.6) is

$$\langle\psi| = \alpha^*\,\langle 0| + \beta^*\,\langle 1| = \begin{pmatrix} \alpha^*, & \beta^* \end{pmatrix} \qquad \alpha, \beta \in \mathbb{C},$$

with $*$ being the complex conjugate.

Each state vector $|\phi\rangle$ belongs to a two-dimensional complex vector space with an inner product, also called the *Hilbert space* [NC10]. This inner product is defined by:

$$\langle a|b\rangle = \langle a| \cdot |b\rangle = \begin{pmatrix} a_1^*, a_2^*, \ldots, a_n^* \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_n \end{pmatrix} = a_1^* b_1 + a_2^* b_2 + \ldots + a_n^* b_n. \tag{2.7}$$

Even though the general qubit state can be mathematically expressed as defined in Equation (2.6), the superposition cannot be obtained through a measurement. The result of a measurement is always either $|0\rangle$ with probability $|\alpha|^2$ or $|1\rangle$ with probability $|\beta|^2$ [NC10]. This interpretation of α and β as probabilities adds another constraint from probability theory to their possible values:

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2.8}$$

Furthermore, the definition of qubits from Equation (2.6) can be extended from one qubit to n-qubit systems. In this case there are 2^n basis states and a quantum state is a superpostion of these basis states, leading to 2^n complex amplitudes. The general definition for a 2-qubit quantum systems is

$$|\psi\rangle = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle .$$

Note that the combined quantum state (e.g. $|00\rangle$) of two separate qubits (e.g. $|0\rangle$ and $|0\rangle$) is described by the tensor product (i.e. $|00\rangle = |0\rangle \otimes |0\rangle$) [NC10].

Again the α_i coefficients are interpreted as probabilities, so the constraint from Equation (2.8) also extends to n-qubit systems:

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$$

In Equation (2.6) vectors were used to define a quantum state $|\psi\rangle$. An alternative way of describing quantum states is achieved by using matrices, often called the density matrix [ANI+21]. Assuming a quantum state $|\psi\rangle$ in vector notation, the corresponding density matrix is shown in Equation (2.9), defined through the dyadic product of the vectors. The density matrix for a quantum state in an n-qubit system generally is $\mathbb{C}^{2^n x 2^n}$.

$$\rho = |\psi\rangle \langle\psi| \qquad |\psi\rangle \in \mathbb{C}^{2^n} \tag{2.9}$$

A way of representing quantum states is using a graphical representation. As one qubit with both complex amplitudes α and β has four degrees of freedom, which is difficult to visualize, the general qubit definition has to be rewritten. Using the definition for complex numbers leads to:

$$|\phi\rangle = \alpha |0\rangle + \beta |1\rangle = r_0 e^{i\varphi_0} |0\rangle + r_1 e^{i\varphi_1} |1\rangle . \tag{2.10}$$

The next step requires the basics about measurements, which will be further explained in 2.2.4. The measurement rule in Equation (2.11) returns the probability of finding a quantum state $|\phi\rangle$ in state $|\psi\rangle$ [SP18].

$$p(|\psi\rangle) = |\langle\psi|\phi\rangle|^2. \tag{2.11}$$

However, for an overall factor x with $|x| = 1$ multiplied to a quantum state ϕ, the measurement rule leads to

$$|\langle\psi|x|\phi\rangle|^2 = |x\langle\psi|\phi\rangle|^2 = |\langle\psi|\phi\rangle|^2. \tag{2.12}$$

x is called the global phase and does not make a difference when measuring a quantum state(see Equation (2.12)). The relative phase in turn is the difference between the amplitudes of a quantum state and can indeed be measured.

With this knowledge, that the global phase cannot be measured, but only the relative phase $\varphi_1 - \varphi_0$ (see Equation 2.10), the four parameters of a quantum state reduce to three real valued parameters [ANI+21]

$$|\phi\rangle = \gamma\,|0\rangle + \delta e^{i\vartheta}\,|1\rangle \qquad \gamma, \delta, \vartheta \in \mathbb{R}.$$

Furthermore, the normalization in Equation (2.8) leads to

$$\sqrt{\gamma^2 + \delta^2} = 1$$

and in combination with the trigonometric identity

$$\sqrt{sin^2(x) + cos^2(x)} = 1$$

γ and δ can be rewritten as

$$\gamma = cos(\frac{\varphi}{2}),\ \delta = sin(\frac{\varphi}{2}) \qquad \varphi \in \mathbb{R}.$$

This leads to the final equation for a quantum state using only two variables:

$$|\phi\rangle = cos(\frac{\varphi}{2})\,|0\rangle + e^{i\vartheta} sin(\frac{\varphi}{2})\,|1\rangle \qquad \varphi, \vartheta \in \mathbb{R}.$$

Finally, rewriting any quantum state to the previously shown form and interpreting φ and ϑ as spherical coordinates allows plotting the quantum state using the so called *Bloch sphere* [Blo46]. An example of the Bloch sphere is shown in Figure 2.3.

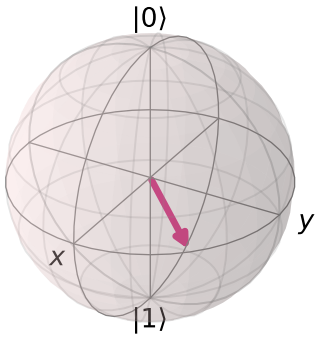


Figure 2.3.: Example of a quantum state visualized in the Bloch sphere

2.2.2. Gates

Analogous to operations on classical bits, which change the state of a bit, there are operations, so called *gates*, which change the state of qubits. Gates that work on $\log_2 N$ qubits are mathematically represented by complex unitary matrices $U \in \mathbb{C}^{NxN}$ [NC10]. The application of gate operations to a quantum state is a matrix-vector multiplication between the corresponding gate matrix and the quantum state, as seen in Equation (2.13).

$$|\psi\rangle = U\,|\phi\rangle \qquad U \in \mathbb{C}^{NxN}, \psi, \phi \in \mathbb{C}^N \tag{2.13}$$

Furthermore, there is a way to visually represent qubits and the gates which are applied to them as shown in Figure 2.4, which is often referred to as a *circuit diagram*. Each line in this figure represents an individual qubit and how its state changes over time depending on the gates which are applied to it. Figure 2.4 shows how a H and X gate are applied to a qubit.

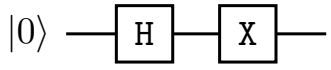


Figure 2.4.: Example of a cicuit diagram using the H gate and X gate

The following paragraphs introduce some different gate types, which are important for the remainder of this thesis. If not states otherwise, all of these gates were introduced in [NC10].

Basic Gates The most basic gate is the I gate, also known as the identity. The matrix is shown in (2.14). However, as the name suggests, it does not change the state of a qubit, as seen in (2.15).

$$\mathrm{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.14}$$

$$\mathrm{I}\,|\psi\rangle = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2.15}$$

A more interesting and important type of gate is the Hadamard gate, or H gate for short in (2.16). It is one of the important gates, because it turns a qubit from the default state $|0\rangle$ into an equal superposition of the basis states $|0\rangle$ and $|1\rangle$, as seen in (2.17). The circuit notation is shown in Figure 2.5. Figure 2.6 shows the $|0\rangle$ state in a Bloch sphere. The resulting state after applying a H to the $|0\rangle$ state is shown in Figure 2.7.

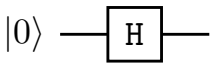


Figure 2.5.: Circuit diagram with an H gate applied to the $|0\rangle$ state

$$\mathrm{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.16}$$

$$\mathrm{H}|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \tag{2.17}$$

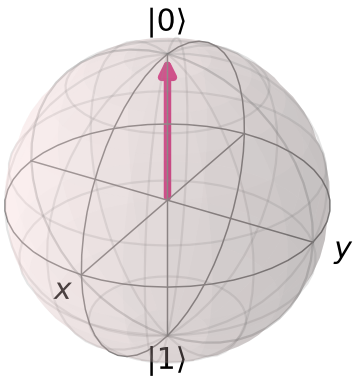


Figure 2.6.: Bloch sphere showing the $|0\rangle$ state

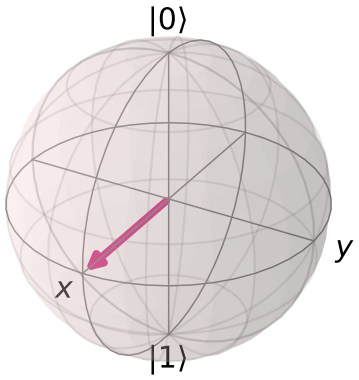


Figure 2.7.: Bloch sphere showing the result of the H gate on the $|0\rangle$ state

Pauli Gates The Pauli gates are the most basic rotational gates used in quantum computing.

The first type of Pauli gates is the X gate in (2.18). From the example in (2.19) it can be examined that the X gate switches the amplitudes of the provided qubit. This is commonly referred to as a negation of the quantum state. Examining the Bloch spheres in Figure 2.8 and Figure 2.9 it can be seen that the X gate also represents a rotation around the x-axis by π radians.

$$\mathrm{X} = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix} \tag{2.18}$$

$$\mathrm{X}|\psi\rangle = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \begin{pmatrix}\beta\\ \alpha\end{pmatrix} \tag{2.19}$$

Analogous to the X gate there are also Y gates and Z gates, which correspond to rotations around the y-axis and z-axis of the Bloch sphere by π radians respectively. Their matrices are shown in (2.20) and (2.21).

$$\mathrm{Y} = \begin{pmatrix}0 & -i\\ i & 0\end{pmatrix} \tag{2.20}$$

$$\mathrm{Z} = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix} \tag{2.21}$$

The circuit representations for the X, Y and Z gates are shown in Figure 2.10.

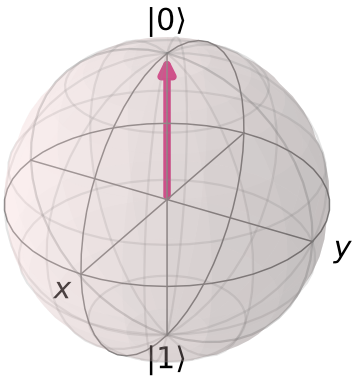


Figure 2.8.: Bloch sphere showing the $|0\rangle$ state

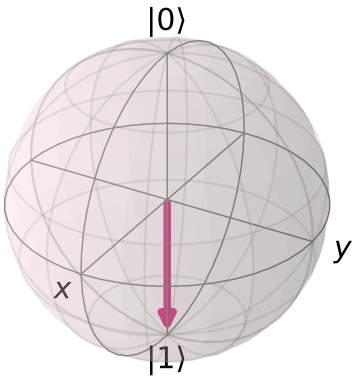


Figure 2.9.: Bloch sphere showing the result of the X on the $|0\rangle$ state

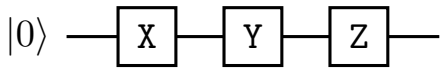


Figure 2.10.: Circuit diagram showing the X, Y and Z-gate

General Rotation Gates There are also generalized versions of the previously introduced Pauli X, Y and Z gates. The matrices for these gates are shown in (2.22), (2.23) and (2.24). These gates are parameterized by a value $\phi \in [0, 2\pi)$, which represents the factor to rotate around the given axis. Their gate representations are shown in the circuit diagram in Figure 2.11.

$$\mathrm{R_X}(\phi) = \begin{pmatrix} cos(\phi/2) & -i\ sin(\phi/2) \\ -i\ sin(\phi/2) & cos(\phi/2) \end{pmatrix} \tag{2.22}$$

$$\mathrm{R_Y}(\phi) = \begin{pmatrix} cos(\phi/2) & -sin(\phi/2) \\ sin(\phi/2) & cos(\phi/2) \end{pmatrix} \tag{2.23}$$

$$\mathrm{R_Z}(\phi) = \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix} \tag{2.24}$$

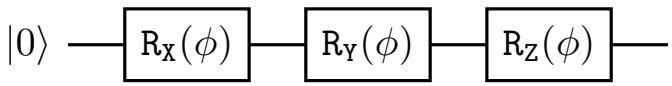


Figure 2.11.: Circuit diagram showing the parameterized rotational gates

There are also controlled versions of these three gates, namely $\mathrm{CR_X}$, $\mathrm{CR_Y}$ and $\mathrm{CR_Z}$. Each of these gates work with two qubits, the control and target qubit. If the control qubit is in

state $|1\rangle$, the parameterized rotation is applied to the target qubit. If the control qubit is in state $|0\rangle$, the rotation is not applied. Their matrix representations are shown in (2.25), (2.26) and (2.27). The circuit representations are shown in Figure 2.12, where the black dots represent the control qubit.

$$\mathrm{CR_X}(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & cos(\phi/2) & -i\ sin(\phi/2) \\ 0 & 0 & -i\ sin(\phi/2) & cos(\phi/2) \end{pmatrix} \tag{2.25}$$

$$\mathrm{CR_Y}(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & cos(\phi/2) & sin(\phi/2) \\ 0 & 0 & sin(\phi/2) & cos(\phi/2) \end{pmatrix} \tag{2.26}$$

$$\mathrm{CR_Z}(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{-i\phi/2} & 0 \\ 0 & 0 & 0 & e^{i\phi/2} \end{pmatrix} \tag{2.27}$$

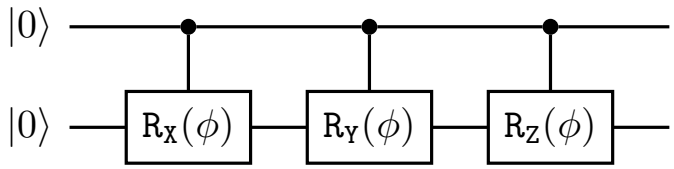


Figure 2.12.: Circuit diagram showing the controlled rotational gates

CNOT The CNOT gate is a two qubit gate, with one being the target and the other one being the control qubit. If the control qubit is found in state $|1\rangle$, a negation (X gate) is performed on the target qubit. If the control qubit is in state $|0\rangle$, nothing is done to the target qubit. The matrix represenation for the CNOT gate is shown in (2.28), if the first qubit is considered the control and the second qubit the target. The reverse case can be specified analogously.

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{2.28}$$

The circuit diagram representation is shown in Figure 2.13, with the first qubit being the control and the second being the target qubit.

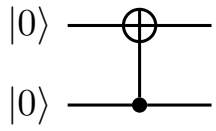


Figure 2.13.: Representation of a CNOT gate with the first qubit as the target and the second qubit the control

Table 2.1.: Input and output states of the CNOT

Input (target, control)	Output (target, control)
$\|0\rangle, \|0\rangle$	$\|0\rangle, \|0\rangle$
$\|0\rangle, \|1\rangle$	$\|1\rangle, \|1\rangle$
$\|1\rangle, \|0\rangle$	$\|1\rangle, \|0\rangle$
$\|1\rangle, \|1\rangle$	$\|0\rangle, \|1\rangle$

When ignoring quantum states in a superposition of $|0\rangle$ and $|1\rangle$, the CNOT gate can be understood through a truth table from classical computing as seen in Tab. 2.1.

When accounting for general two-qubit quantum states $|\alpha\rangle$, Equation (2.29) shows that the CNOT gate switches the amplitudes for the two basis states $|01\rangle$ and $|11\rangle$.

$$\text{CNOT}\,|\alpha\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} = \begin{pmatrix} \alpha_{00} \\ \alpha_{11} \\ \alpha_{10} \\ \alpha_{01} \end{pmatrix} \tag{2.29}$$

SWAP Gate The SWAP gate is a two-qubit gate, which is used to swap the states of two qubits. Its matrix representation is shown in (2.30) and the corresponding circuit in Figure 2.14.

$$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.30}$$

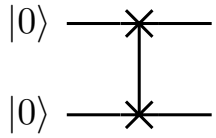


Figure 2.14.: Circuit diagram showing the SWAP gate affecting two qubits

Table 2.2.: Input and output states of the SWAP

Input ($qubit_1$, $qubit_2$)	Output ($qubit_1$, $qubit_2$)
$\|0\rangle$, $\|0\rangle$	$\|0\rangle$, $\|0\rangle$
$\|0\rangle$, $\|1\rangle$	$\|1\rangle$, $\|0\rangle$
$\|1\rangle$, $\|0\rangle$	$\|0\rangle$, $\|1\rangle$
$\|1\rangle$, $\|1\rangle$	$\|1\rangle$, $\|1\rangle$

The following example shows why the states are swapped. Starting with two qubits defined as:

$$|\phi\rangle = a\,|0\rangle + b\,|1\rangle = \begin{pmatrix} a \\ b \end{pmatrix}$$
$$|\psi\rangle = c\,|0\rangle + d\,|1\rangle = \begin{pmatrix} c \\ d \end{pmatrix}.$$

The two separate states lead to a combined quantum state of

$$|\psi\phi\rangle = |\psi\rangle \otimes |\phi\rangle = ca\,|00\rangle + cb\,|01\rangle + da\,|10\rangle + db\,|11\rangle = \begin{pmatrix} ca \\ cb \\ da \\ db \end{pmatrix}.$$

Now, applying the SWAP gate to this quantum state yields

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} ca \\ cb \\ da \\ db \end{pmatrix} = \begin{pmatrix} ca \\ da \\ cb \\ db \end{pmatrix}.$$

Writing this state vector in terms of its basis states leads to

$$\begin{aligned} ca\,|00\rangle + da\,|01\rangle + cb\,|10\rangle + db\,|11\rangle &= (a\,|0\rangle + b\,|1\rangle) \otimes (c\,|0\rangle + d\,|1\rangle) \\ &= |\phi\rangle \otimes |\psi\rangle \\ &= |\phi\psi\rangle, \end{aligned}$$

which shows that the SWAP gate has actually swapped the amplitudes and therefore the states.

The SWAP gate can be visualized with the truth table in Tab. 2.2.

An extension of this is the Fredkin gate, sometimes called the controlled SWAP gate or CSWAP gate [Pat+16]. It uses an additional control qubit, that decides whether or not the two target qubits are swapped. If the control qubit is found in state $|1\rangle$, the two qubits are swapped. If the control qubit is $|0\rangle$, the target qubits are not swapped. This can be seen in the truth table in Tab. 2.3.

The circuit representation for the CSWAP gate is shown in Figure 2.15.

Table 2.3.: Input and output states of the CSWAP

Input (control, $target_1$, $target_2$)	Output (control, $target_1$, $target_2$)
$\|0\rangle, \|0\rangle, \|0\rangle$	$\|0\rangle, \|0\rangle, \|0\rangle$
$\|0\rangle, \|0\rangle, \|1\rangle$	$\|0\rangle, \|0\rangle, \|1\rangle$
$\|0\rangle, \|1\rangle, \|0\rangle$	$\|0\rangle, \|1\rangle, \|0\rangle$
$\|0\rangle, \|1\rangle, \|1\rangle$	$\|0\rangle, \|1\rangle, \|1\rangle$
$\|1\rangle, \|0\rangle, \|0\rangle$	$\|1\rangle, \|0\rangle, \|0\rangle$
$\|1\rangle, \|0\rangle, \|1\rangle$	$\|1\rangle, \|1\rangle, \|0\rangle$
$\|1\rangle, \|1\rangle, \|0\rangle$	$\|1\rangle, \|0\rangle, \|1\rangle$
$\|1\rangle, \|1\rangle, \|1\rangle$	$\|1\rangle, \|1\rangle, \|1\rangle$

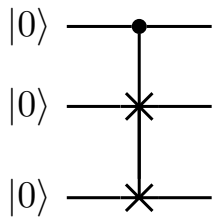


Figure 2.15.: Circuit diagram showing the CSWAP gate with the first qubit being the control qubit and the last two qubits being the targets

2.2.3. Basis states

Note that up until now, quantum states were always described as superpositions of the *basis states* $|0\rangle$ and $|1\rangle$. However, these two states are not the only possible basis states. In fact, any two states $|x\rangle$ and $|y\rangle$ could form a basis for quantum states, as long as they are orthonormal [ANI+21]. A resulting quantum state in this arbitrary basis would be

$$|\psi\rangle = \alpha\,|x\rangle + \beta\,|y\rangle\,.$$

The earlier introduced definition of a qubit in (2.6) is just a special case of this general definition, with $|x\rangle = |0\rangle$ and $|y\rangle = |1\rangle$.

One other popular basis is the $\{|+\rangle, |-\rangle\}$ basis, also called Hadamard-basis, defined in Equation (2.31) and Equation (2.32) [ANI+21].

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \tag{2.31}$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix} \tag{2.32}$$

While there are in theory infinitely many different bases that could be used to define a quantum state, in practice the $\{|0\rangle, |1\rangle\}$ basis is used most of the time.

2.2.4. Measurement

As mentioned earlier, a quantum state cannot just be read from a quantum computer. Instead a quantum state has to be *measured* to get information about it. This is done through the rule for quantum measurement [SP18] in Equation (2.11), which returns the probability of measuring a quantum state $|\psi\rangle$ in a state $|\phi\rangle$.

Let us assume a quantum state as follows:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix}.$$

To get the probability of measuring the basis state $|0\rangle$ the rule of measurement would be applied like:

$$|\,\langle 0|\psi\rangle\,|^2 = |\alpha\,\langle 0|0\rangle + \beta\,\langle 0|1\rangle\,|^2 = |\alpha|^2.$$

Note that the measurement rule can use any arbitrary state $|\psi\rangle$ for the measurement and is not confined to the basis states $|0\rangle$ and $|1\rangle$.

However, when measuring in a real quantum computer once, neither the superposition nor the probabilities for specific states are acquired directly. The result of measuring a qubit is always either $|0\rangle$ or $|1\rangle$. To get the probabilities that a qubit is found in one of the basis states, the same qubit has to be measured multiple times. One such evaluation of the circuit is referred to as a shot.

The result of many shots is typically plotted in histograms like in Figure 2.17. This plot was generated by measuring the state $|\psi\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$, which is build by the circuit shown in Figure 2.16, 50 times. In theory, both $|0\rangle$ and $|1\rangle$ should be measured with 50% probability according to Equation (2.11). However, the plot does not reflect this equal probability. This is due to the fact that the plot shows the probabilities from only 50 shots, which is not enough to approximate the theoretical probabilities correctly.

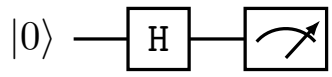


Figure 2.16.: Circuit diagram with a H-gate and the measurement operator applied to the qubit

2.2.5. Entanglement

In Section 2.2.1 the notion of multi-qubit systems and their state description was introduced. Multiple qubits can either be seen as separate quantum states ($|\psi\rangle$ and $|\phi\rangle$) or as a combined quantum state for all qubits ($|\psi\phi\rangle = |\psi\rangle \otimes |\phi\rangle$) using the tensor product. This definition means that multiple qubits can be combined to one quantum state and vice versa. However, there are quantum states which can only be expressed as combined quantum states and cannot be decomposed into separate states. This is known as *entanglement* between the concerning qubits [NC10].

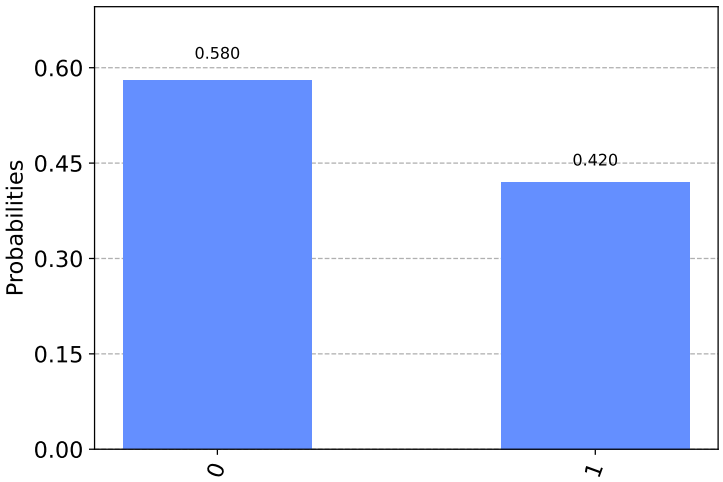


Figure 2.17.: Histogram showing results of a real measurement

One example of such an entangled state is $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. This state can be generated by using a H and CNOT gate on two qubits as seen in Figure 2.18. This state cannot be decomposed into two separate quantum states [ANI+21]. The state has a chance of 50% of being $|00\rangle$ or $|11\rangle$, but it can never be $|01\rangle$ nor $|10\rangle$. This presents the interesting fact that measuring only one of the qubits and finding it to be $|0\rangle$ ($|1\rangle$) means that the other qubit also has to be in state $|0\rangle$ ($|1\rangle$).

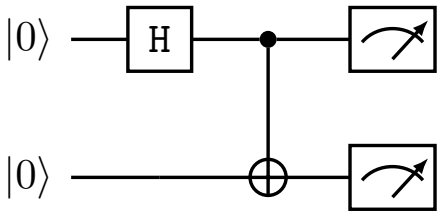


Figure 2.18.: Example of two qubits being entangled by the CNOT-gate

In summary, measuring only one qubit of an entangled state also tells us the state of the second qubit, without having to measure it.

2.2.6. Data Encoding

Working with classical datasets requires additional steps compared to data which is already in a quantum state. To process classical data on a quantum computer, it needs to be *encoded* into a quantum state. This is often referred to as a *quantum feature map*, because the data is mapped from its input space into a Hilbert space [SK19].

There are multiple different strategies to achieve an encoding, but they all share some fundamental strategies. Assume some arbitrary datapoint x needs to be encoded into a

quantum state. As qubits commonly start in the $|0\rangle$ state, there needs to be some quantum circuit U that uses x as an input and applies gates to change the initial state [SK19]. This is shown in Equation (2.33).

$$|\phi\rangle = U(x)\,|0\rangle \tag{2.33}$$

The following paragraphs show some specific strategies for building this circuit $U(x)$, to encode classical data onto a quantum computer.

Basis Encoding Basis encoding is the most intuitive encoding strategy, as it associates each classical bit of information with a corresponding qubit [SP18]. For this, the data has to be present in the form of a bit string x^n with $x \in \{0, 1\}$. This bit string is then encoded into a quantum state $|x\rangle$. As an example, the datapoint $x = (110)$ would correspond to a quantum state of $|\phi\rangle = |110\rangle$.

In a circuit diagram, this can be achieved by adding an X gate to each qubit that is supposed to be $|1\rangle$ and leaving the qubits that are supposed to be $|0\rangle$ as is. A circuit diagram representing the basis encoding of the previously mentioned example of $x = (110)$ is shown in Figure 2.19.

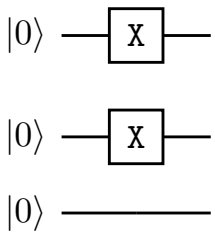


Figure 2.19.: Circuit showing the basis encoding of the vector $x = (110)$ using X gates

When encoding multiple bit-strings of the same length, for example $x_1 = (110)$ and $x_2 = (111)$, both samples can be encoded in a superposition with 3 qubits as follows:

$$|\phi\rangle = \frac{1}{\sqrt{2}}\,|110\rangle + \frac{1}{\sqrt{2}}\,|111\rangle\,.$$

In general, basis encoding requires n qubits to encode bit-strings of length n.

Angle Encoding Angle encoding utilizes the general rotation gates introduced in Section 2.2.2 to encode data into a quantum state [Ber+20]. An input vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ can therefore be encoded in an n-qubit system. Each $x_i \in x$ is used as the rotational parameter for a rotational gate, which is applied to the ith qubit. As the x_i are now rotational parameters, they should be limited to $[0, 2\pi)$. Which one of the three gates is used is generally determined by the user.

An example with input vector $x = (1.2, 2.6, 4.2)$ and R_X gates is shown in Figure 2.20.

The advantage of angle encoding over basis encoding is that the input vector $x \in \mathbb{R}^n$ can be encoded, without having to convert it to a binary string first.

Angle encoding requires n qubits to encode a vector $x \in \mathbb{R}^n$. While it requires the same amount of qubits as basis encoding, angle encoding can encode a real value, instead of just one bit of information, in each qubit.

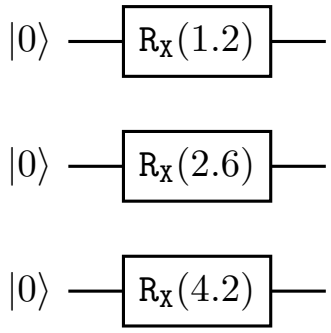


Figure 2.20.: Angle encoding circuit for the vector $x = (1.2, 2.6, 4.2)$ using R_X gates

Amplitude Encoding Amplitude encoding uses the amplitudes of a quantum state to encode the values of an input vector $x = (x_1, \ldots, x_{2^n}) \in \mathbb{R}^{2^n}$ [SK19]. The resulting quantum state would look like

$$|\phi\rangle = \sum_{j=1}^{2^n} x_j |j\rangle ,$$

with $|j\rangle$ being the j-th computational basis state.

Using this method a 2^n-dimensional vector can be encoded in 2^n amplitudes, which requires $n = log(2^n)$ qubits. Note that the input vector has to be padded with extra values, if the size n of the vector x is smaller than the total number of available amplitudes 2^n.

Furthermore, as defined in Section 2.2.1, the fact that amplitudes correspond to probabilities leads to a normalization constraint for the input vector:

$$\sum_{i}^{n} |x_i|^2 = 1.$$

When encoding vectors with many features, this normalization can be problematic as the differences between features are decreased. This can lead to features being undistinguishable.

2.2.7. Noise

Up until now, this introduction of quantum computing assumed perfect conditions with all gates performing exactly as intended. However, with quantum hardware this is not feasible. The current era of quantum computers is called the noisy intermediate scale quantum (NISQ) era [Pre18]. The name already suggests that current quantum computers are affected by *quantum noise*, which prohibits the idea of perfect quantum computers and leads to errors in the calculations. This noise needs to be accounted for when simulating or using quantum hardware.

This section introduces some common sources of noise and their effect on calculations.

2.2.7.1. Sampling Noise

In Section 2.2.4 the concept of perfect and real measurements on quantum computers was introduced. The mathematical definition of a measurement was shown in Equation (2.11) as the probability of finding a qubit in a specific state. On a real quantum computer, a measurement only tells us if the qubit is 0 or 1. To approximate the perfect probability given by the mathematical definition, the expectation value over many shots is used.

However, the quality of this approximation depends on the number of shots. This is visualized in Figure 2.21 and Figure 2.22. The same experiment with only one qubit and a H gate is repeated multiple times with a different number of shots. The figure shows that the more shots are used, the closer the approximation is to the calculated probability of 0.5 for 0 and 1.

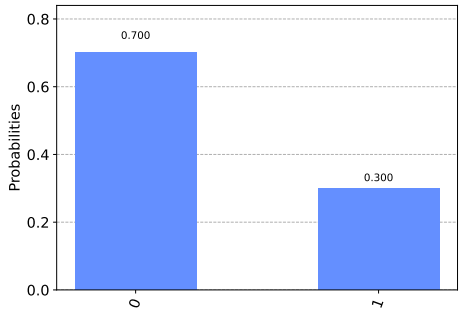


Figure 2.21.: Histogram for 10 measurements

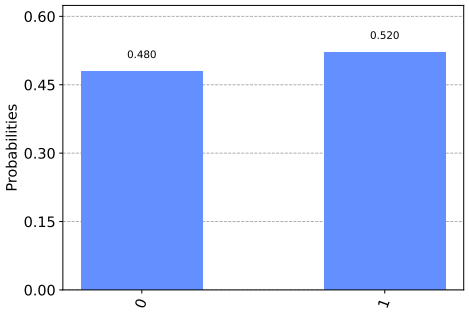


Figure 2.22.: Histogram for 100 measurements

This leads to a trade-off between accuracy and time complexity. Doing many shots will increase the accuracy but also the execution time, as the same circuit has to be evaluated many times. Using less shots leads to shorter execution times, but comes at the cost of less accuracy in the result.

2.2.7.2. Coherent Noise

Coherent noise can be described as noisy gates that differ from the perfect gate by some unitary [Bou+04], sometimes called a systematic error. This noise generally originates from miscalibrated gates, which do not perform perfect operations.

One example for coherent noise is the noisy X gate, denoted by $\widetilde{\mathrm{X}}$. The perfect X gate would rotate around the x-axis by π. The noisy version $\widetilde{\mathrm{X}}$ instead rotates by $\pi + \epsilon$, with ϵ being the error term.

It can be shown, that this noisy gate $\widetilde{\mathrm{X}}$ can be decomposed into two separate gates, which are applied one after another. The first gate rotates around the x-axis by the error term ϵ, which is equivalent to a $\mathrm{R_X}$ with parameter ϵ. The second gate is the ideal X, which rotates around the x-axis by π. In sum, this leads to a total rotation of $\pi + \epsilon$. This decomposition shows that the error is in fact only a unitary difference from the perfect gate. The circuit for this decomposition can be seen in Figure 2.23.

Figure 2.23.: Decomposition of the noisy $\widetilde{\mathrm{X}}$ gate into an error gate and a perfect X gate

Note that the $\widetilde{\mathrm{X}}$ gate is only one example. The idea of coherent noise as some unitary difference can be extended to other gates as well.

2.2.7.3. Incoherent Noise

Incoherent noise is the idea that there is the probability of the environment influencing the state of a quantum system [Neu27]. As an example, Figure 2.24 shows an X gate which is affected by one type of incoherent noise. Instead of simply applying the gate to a qubit, there are now two cases. In the ideal case the X gate is preceded by the I gate, which does not change the desired operation. This case occurs with a probability of $1 - p$. In the second case, occurring with probability p, the X gate is preceded by another X gate, which is not the intended operation and therefore an error.

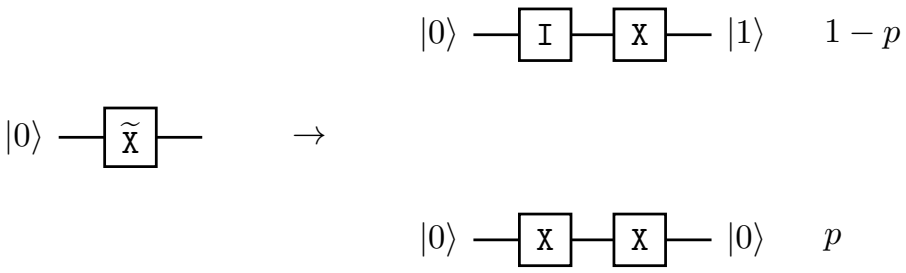


Figure 2.24.: Decomposition of incoherent noise in the ideal case with probability $1 - p$ and the not-ideal case with probability p

To model this probabilistic behaviour, a new representation for quantum states is needed, as the bra-ket notation does not have a convenient way of representing an ensemble of quantum states. This is achieved using density matrices and mixed states. Density matrices were first introduced in Equation (2.9), as an alternative way of representing a quantum state $|\phi\rangle$ as a matrix. This can be further generalized to an ensemble of multiple states as follows [NC10]:

$$\rho = \sum_i p_i \left|\phi_i\right\rangle \left\langle\phi_i\right| = \sum_i p_i \rho_i. \tag{2.34}$$

Note that ρ now is a sum over many different quantum states $|\phi_i\rangle \langle\phi_i| = \rho_i$ with p_i being the probability that the system is in one specific state $|\phi_i\rangle \langle\phi_i| = \rho_i$.

If a quantum state represented by a density matrix ρ only contains one state $|\phi\rangle \langle\phi|$, the state is said to be a pure state. Otherwise, if the state is an ensemble of multiple pure states, it is called a mixed state [NC10]. Mixed states can therefore encode the probabilistic behaviour as required by the example in Figure 2.24.

The ideal case of I and X leads to the quantum state $|1\rangle$. Written as a density matrix this is:

$$|1\rangle\langle 1| = \begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}0 & 1\end{pmatrix} = \begin{pmatrix}0 & 0\\0 & 1\end{pmatrix}.$$

In turn the non-ideal case with two X gates leads to the quantum state $|0\rangle$ and the following density matrix:

$$|0\rangle\langle 0| = \begin{pmatrix}1\\0\end{pmatrix}\begin{pmatrix}1 & 0\end{pmatrix} = \begin{pmatrix}1 & 0\\0 & 0\end{pmatrix}.$$

The combined quantum state for the system in Figure 2.24 would therefore be the combination of both density matrices:

$$\rho = p\begin{pmatrix}1 & 0\\0 & 0\end{pmatrix} + (1-p)\begin{pmatrix}0 & 0\\0 & 1\end{pmatrix}.$$

With this motivation, incoherent noise can be modelled as a map ϕ that maps a density matrix ρ to a new density matrix $\phi(\rho)$, which is also called a quantum channel [NC10]. This map can be defined as

$$\phi(\rho) = \sum_i K_i \rho K_i^\dagger, \tag{2.35}$$

if $\sum_i K_i^\dagger K_i = I$ for a set of operators $\{K_i\}$ [NC10]. The K_i are also called Kraus operators [Kra+83]. Following Equation (2.35), incoherent noise can be defined through a set of Kraus operators.

The next paragraphs introduce some well known types of incoherent noise, their Kraus operators and their effects on quantum states.

Bit-flip noise corresponds to noise as seen in the previous introduction and Figure 2.24. It is defined by the two following Kraus operators:

$$K_0 = \sqrt{p}\mathrm{X} = \sqrt{p}\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$$
$$K_1 = \sqrt{1-p}\,\mathrm{I} = \sqrt{1-p}\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}.$$

The operators show that with a probability of p a X gate is applied, which flips the amplitudes of a quantum state. This is similar to flipping a bit in classical computing. The alternative case with probability $1-p$ applies the identity, which does not change the quantum state.

Amplitude damping models effects that arise from energy losses in a quantum system. In real quantum computers the $|0\rangle$ state is represented through a low energy level, while the $|1\rangle$ state equals a high energy level [NC10]. The process of energy relaxation describes the fact that a physical system like a quantum computer always tends to a state with low

energy [NC10]. This means that over time a qubit in the $|1\rangle$ state will loose its energy and turn into a $|0\rangle$ state.

This can be modelled with the two Kraus operators [NC10]:

$$K_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-y} \end{pmatrix}$$
$$K_1 = \begin{pmatrix} 0 & \sqrt{y} \\ 0 & 0 \end{pmatrix},$$

with $y \in [0, 1]$ being the amplitude damping probability.

Phase damping models physical processes that lead to a loss of quantum information without losing energy [NC10]. This can generally be seen as a change in the relative phase between basis states.

Phase damping can be expressed using the two following Kraus operators:

$$K_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-y} \end{pmatrix}$$
$$K_1 = \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{y} \end{pmatrix}.$$

$y \in [0, 1]$ is the probability of damping the phase.

Depolarizing noise has the probability of completly destroying all available information on a single qubit [NC10]. This is modelled as the following map:

$$\phi(\rho) = p\frac{I}{2} + (1-p)\rho.$$

With a probability of $1-p$ the quantum state is stays the same. However, with a probability of p, the state is replaced by the completly mixed state $\frac{I}{2}$, which leaves the qubit with no meaningfull information.

This can be generalized to n qubits as follows:

$$\phi(\rho) = p\frac{I}{2^n} + (1-p)\rho.$$

The depolarizing channel for a single qubit can be described through the four following Kraus operators:

$$K_0 = \sqrt{1-\frac{3p}{4}}\,\mathrm{I} = \sqrt{1-\frac{3p}{4}}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$
$$K_1 = \sqrt{\frac{p}{4}}\mathrm{X} = \sqrt{\frac{p}{4}}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$K_2 = \sqrt{\frac{p}{4}}\mathrm{Y} = \sqrt{\frac{p}{4}}\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$
$$K_3 = \sqrt{\frac{p}{4}}\mathrm{Z} = \sqrt{\frac{p}{4}}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

In this case $p \in [0, 1]$ is the depolarizing probability.

2.2.8. Quantum Machine Learning

Quantum machine learning (QML) is the general theory of using advantages from quantum computing to solve typical problems from classical machine learning. This is generally split into four different categories defined in [ABG06] and shown in Figure 2.25.

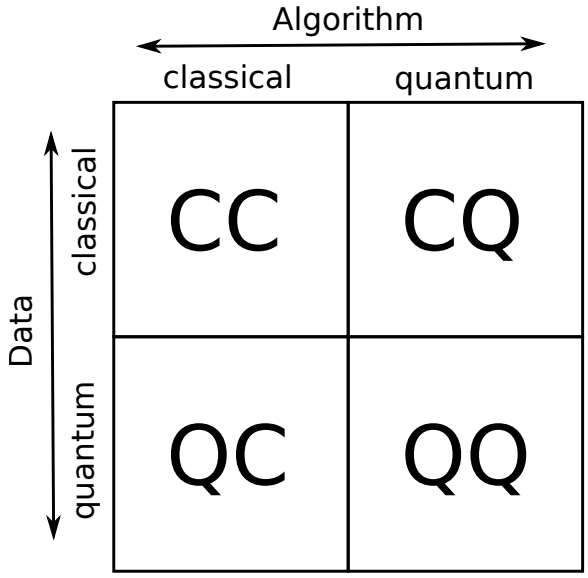


Figure 2.25.: Combination of classical/quantum data and algorithms [DTB16]

These four categories are defined in two different dimensions. Firstly if the data to be processed is generated classically (C) or by a quantum system (Q). And secondly the way the data is processed, again either classically (C) or through a quantum system (Q).

The first category is CC, which is classical data classically processed. This is the standard approach for classical machine learning without any involvement from a quantum system. The second approach is using data generated by quantum systems and process it using classical methods (QC). The QQ approach is focused on generating quantum data and processing it with quantum computers.

Lastly the CQ, or classical data processed with quantum systems, is the most interesting category in the scope of this thesis. There are several different problems associated with this category. The classical datasets have to be transferred onto the quantum computer. This is generally achieved by some data encoding strategy as discussed in Section 2.2.6. There are several different ways the quantum computation in this category can be interpreted. Firstly, classical machine learning methods can be transferred onto quantum computers to replicate the classical algorithms as close as possible. Secondly, new specific quantum algorithms could be designed to solve typical machine learning problems. And lastly the quantum computer could only be used for small portions or subtasks of the classical algorithms, which are especially computational expensive on classical computers.

2.2.9. Variational Quantum Circuits

Variational quantum circuits (VQCs) are one of the first ideas of incorporating machine learning ideas into quantum circuits. These variational circuits extensively use parameterized gates introduced in Section 2.2.2. The idea is to optimize the parameters for these gates so the circuit reflects some function [McC+16].

Figure 2.26 gives a general overview of the approach, with the green block being calculated on a quantum computer and the red block on a classical computer. The quantum device implements some circuit $U(\theta)$, where θ is a set of parameters used as inputs for parameterized gates. The output of the measurement is then used in a cost function $C(\theta)$ to decide how good the current parameters θ_t are representing the desired function. This cost function is then typically used in classical optimization algorithms. The optimization iteratively updates the current parameters θ_t with new parameters θ_{t+1}, to minimize the cost function $C(\theta)$. This combination of quantum devices and classical optimization is commonly referred to as hybrid training of variational circuits [SP18].

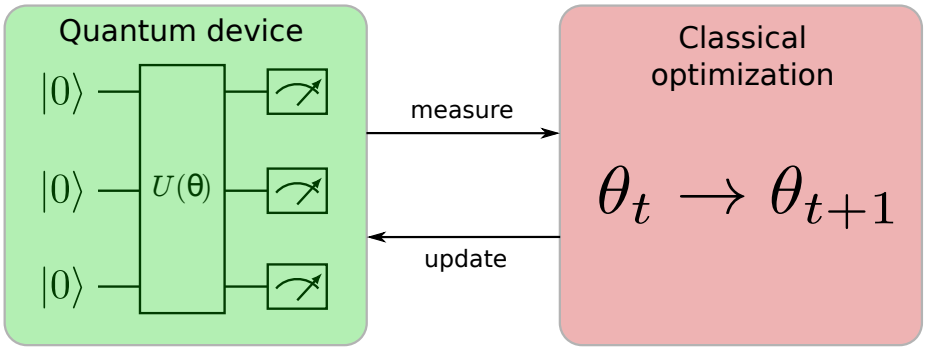


Figure 2.26.: Overview of variational quantum circuits and their optimization

Typically the ciruit $U(\theta)$ is built as a layered architecture [Ber+20]. Each layer is generally defined by a sequence of gates applied to a set of qubits. This sequence of gates is typically referred to as the *ansatz*. For the complete circuit, the ansatz is repeated multiple times with different paramters θ. This is shown in Figure 2.27.

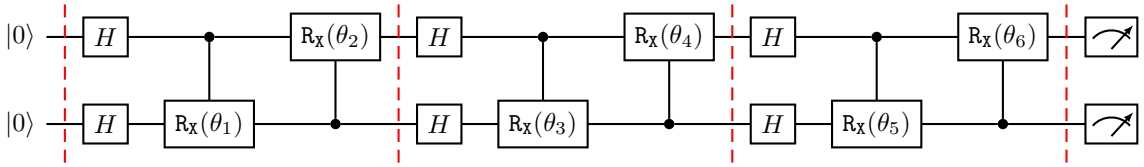


Figure 2.27.: Layered architecture with multiple repetitions of the same ansatz

The advantage of VQCs is the fact that they are very robust against noise [SP18]. As the parameters θ are optimized depending on the output of the circuit, a gate error caused by noise in the quantum device can be compensated by adjusting the affected parameter.

The overall idea of variational circuits is very general and applicable to different scenarios. There is one popular example for the usage of variational circuits, namely variational

classifiers [SP18]. It is used similar to classical classifiers to assign a class label to a datapoint x. Figure 2.28 shows the typical layout for a variational classifier. Note that this figure and therefore the process is very similar to Figure 2.26, with only one necessary extension. The circuit U not only depends on the set of parameters θ, but also on the datapoint x which has to be classified. The measured output of this circuit can then be interpreted as the assigned label for x.

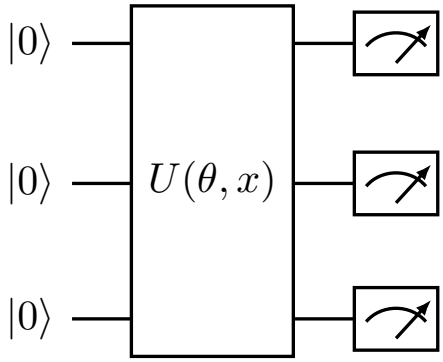


Figure 2.28.: Circuit for a classifier using a variational quantum circuit

2.2.10. Quantum Kernels

In Section 2.1.2 the idea of feature maps and kernel methods was introduced. Furthermore it was shown in Section 2.1 that many machine learning algorithms can benefit from these kernel methods. As it turns out, the idea of kernel methods can be translated to quantum theory.

In Section 2.1.2 it was established that every feature map $\phi : X \to H$, with an inner product $\langle \cdot, \cdot \rangle_H$ defined on H, defines a kernel function as:

$$k(x_i, x_j) := \langle \phi(x_i), \phi(x_j) \rangle_H.$$

Furthermore, in Section 2.2.6 the notion of data encoding in quantum states was introduced as one type of feature map. A quantum feature map ϕ maps input data into a Hilbert space.

Using these two insights and the Hilbert Schmidt inner product $\langle \rho, \sigma \rangle_H = tr\{\rho\sigma\}$ [Sch21] for mixed states $\rho, \sigma \in \mathbb{C}^{2^n \times 2^n}$ a quantum kernel function can be defined as the overlap of two quantum states:

$$k(x, x') = tr\{\rho(x_i)\rho(x_j)\} \qquad \rho(x_i), \rho(x_j) \in \mathbb{C}^{2^n x 2^n}, \tag{2.36}$$

with $tr\{\cdot\}$ being the *trace* of a matrix.

For the case of pure quantum states (i.e. $\rho(x_i) = |\phi(x_i)\rangle \langle\phi(x_i)|$ and $\rho(x_j) = |\phi(x_j)\rangle \langle\phi(x_j)|$) Equation (2.36) reduces to

$$k(x_i, x_j) = tr\{\rho(x_i)\rho(x_j)\} = |\langle\phi(x_i)|\phi(x_j)\rangle|^2, \tag{2.37}$$

as $tr\{\rho(x_i)\rho(x_j)\} = tr\{|x_i\rangle \langle x_i| |x_j\rangle \langle x_j|\} = |\langle x_i|x_j\rangle|^2$ [PBP20].

There are different possible routines to calculate the overlap of two quantum states on a quantum computer. The next two sections describe the SWAP test and the adjoint method to compute the overlap. The last section provides a comparison between the two methods.

2.2.10.1. SWAP test

The SWAP test is one way of calculating the overlap of two quantum states $|\phi\rangle$ and $|\psi\rangle$ [Buh+01]. It uses one ancilla qubit that stores the overlap of the two states in the end. Furthermore it assumes that $|\phi\rangle$ and $|\psi\rangle$ were previously prepared by some circuit U. The circuit diagram for the SWAP test is shown in Figure 2.29.

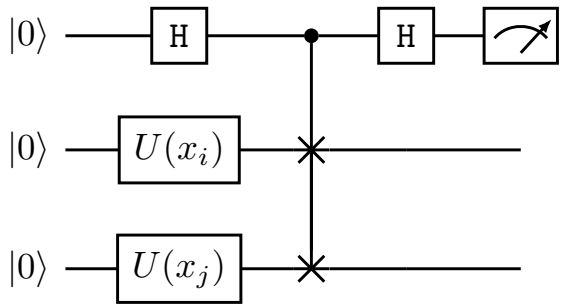


Figure 2.29.: Circuit showing the SWAP test using an ancilla qubit as an output and two qubits being prepared by their respective transformation $U(x_i)$ and $U(x_j)$

The circuit starts in the state $|000\rangle$. The first H gate transforms it to $\frac{1}{\sqrt{2}}(|0,\phi,\psi\rangle+|1,\phi,\psi\rangle)$. The CSWAP gate only swaps ϕ and ψ, if the ancilla qubit is in state $|1\rangle$, resulting in the state $\frac{1}{\sqrt{2}}(|0,\phi,\psi\rangle + |1,\psi,\phi\rangle)$. The last H gate leads to the final quantum state of $\frac{1}{2}(|0,\phi,\psi\rangle + |1,\phi,\psi\rangle + |0,\psi,\phi\rangle - |1,\psi,\phi\rangle)$. Measuring the ancilla qubit after the last H gate leads to the two following probabilities:

$$P(ancilla = 0) = \frac{1}{2} + \frac{1}{2}|\langle\psi|\phi\rangle|^2$$
$$P(ancilla = 1) = \frac{1}{2} - \frac{1}{2}|\langle\psi|\phi\rangle|^2.$$

Using the equation for $P(ancilla = 1)$, the overlap of the quantum states ϕ and ψ can be rewritten to the following equation:

$$|\langle\psi|\phi\rangle|^2 = 1 - 2 \cdot P(ancilla = 1). \tag{2.38}$$

2.2.10.2. Adjoint method

The adjoint method uses the fact that the overlap of two quantum states in Equation (2.37) can be rewritten as [Hub+21]

$$|\langle\phi(x_j)|\phi(x_i)\rangle|^2 = |\langle 0|U^\dagger(x_j)U(x_i)|0\rangle|^2. \tag{2.39}$$

This rewrite is possible because $\langle\phi(x')|$ is prepared on the quantum computer with $\langle 0|\, U^\dagger(x_j)$ and $|\phi(x_i)\rangle$ with $U(x_i)\,|0\rangle$.

Using this circuit to calculate the overlap means applying the data encoding circuit $U(x_i)$ and its adjoint $U^\dagger(x_j)$ to $|0\rangle$ and measuring the probability of finding the system in the state $|0\rangle$.

A circuit diagram which depicts this method is shown in Figure 2.30.

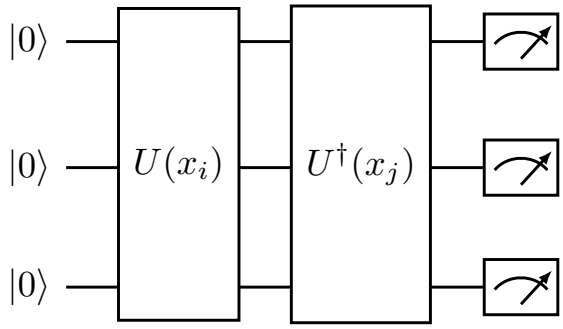


Figure 2.30.: Adjoint method where two samples are encoded on the same qubits using $U(x_i)$ and $U^\dagger(x_j)$

2.2.10.3. Comparison

Both, the SWAP test and adjoint method, can be used to calculate the overlap of two quantum states. But there are some key differences that have to be considered.

In terms of circuit size, if the adjoint method requires n qubits, an analogous SWAP test requires $2 \cdot n + 1$ qubits instead. This is simply due to the fact that the adjoint method prepares both quantum states on the same qubits, while the SWAP test uses a set of n qubits for each quantum state and an additional ancilla qubit to store the overlap.

For the amount of gates both methods are quite similar. If the state preparation circuit needs m gates, the adjoint method uses $2 \cdot m$ gates and the SWAP test $2 \cdot m + 3$. The additional gates for the SWAP test are the two H gates and the CSWAP. There is however a difference in the depth of the circuit. Again, the adjoint method uses the same qubits for both states, which leads to a double of the circuit depth compared to the SWAP test.

The biggest difference between the two methods comes through noise, where gates are not perfectly unitary. This leads to a big problem for the adjoint method, as it requires adjoint of the state preparation circuit. However, it is not possible to construct the adjoint circuit in case of noisy gates, as the noise part is typically unknown [Hub+21]. The SWAP test in turn can be used with noisy gates, since no adjoint circuit has to be constructed.

2.2.11. Variational Quantum Kernels

Instead of only parameterizing the circuit U, used for state preparation in quantum kernels, with the input data x, the circuit can also use additional adjustable parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_k)$ with $\theta_i \in \mathbb{R}$. These θ_i can be used for example as parameters for additional rotational gates. This constitutes a combination of VQCs and data encoding [Llo+20], commonly referred to as a variational quantum kernel (VQK) [Hub+21]. These circuits $U_\theta(x)$ can then be used analogously to compute a quantum kernel function as seen in Section 2.2.10.

The parameters θ can be optimized similar to parameters for variational circuits introduced in Section 2.2.9. This optimization requires a cost or loss function $C : \mathbb{R}^k \to \mathbb{R}$, which determines a quality for the given parameters $\theta \in \mathbb{R}^k$. What constitutes a good or bad selection of parameters θ depends on the application.

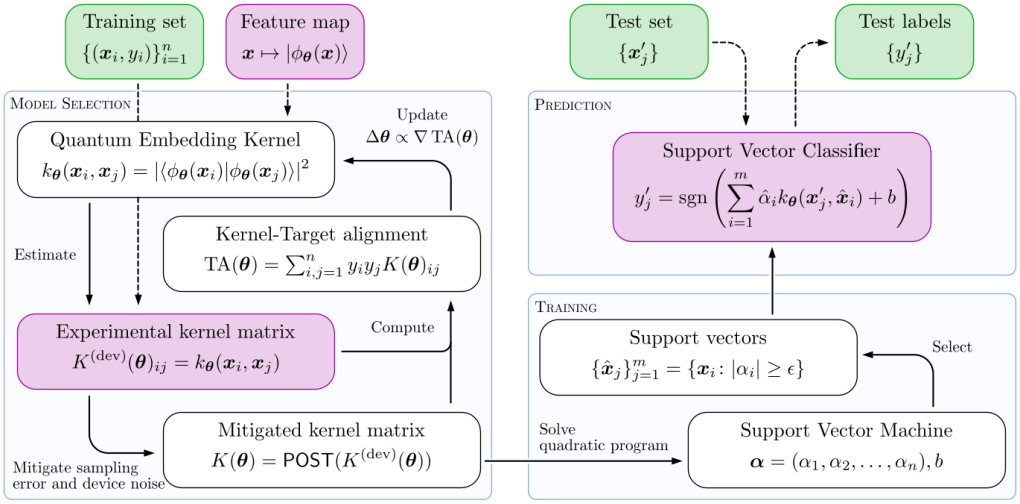


Figure 2.31.: Pipeline for learning a kernel and using it for a SVM [Hub+21]

Once a loss function is defined, it can be used in various classical optimization algorithms to minimize the loss by optimizing the parameters θ. Some popular algorithm choices are gradient descent [Lem12] and its variants like the Adam optimizer [KB15].

2.3. Related Work

Trainable quantum data encodings were first introduced in [Llo+20] as a type of quantum metric learning. This idea was combined with quantum kernels in [Hub+21] to produce trainable quantum kernels. [Hub+21] combined trainable quantum kernels with classical support vector machines to a hybrid quantum classical algorithm. The proposed pipeline for this algorithm from [Hub+21] is shown in Figure 2.31, where the kernel can be trained separately and used as an input for the support vector machine. This pipeline of trainable quantum kernels in combination with SVMs is very similar to the approach of this thesis. However, instead of being concerned with a classification problem, this thesis is focused on the application of clustering.

There are also ideas for bringing classical clustering algorithms fully onto quantum computers, like for example q-means [Ker+18]. The idea is to adapt the popular k-means algorithm (see Section 2.1.3.1) to utilize advantages of quantum mechanics. Q-means follows the same procedure as k-means by calculating distances and assigning data to clusters accordingly. It promises similar results but better performance compared to the classical algorithm. However, it requires the whole dataset to be stored in QRAM [LST10]. At the current state and size of quantum simulators and computers this requirement is not fulfillable, but possibly will be in the future. Kernel methods used as a subroutine in a bigger algorithm only query the quantum computer with two samples at a time and are therefore usable with the current generation of quantum computers.

Trainable quantum kernels are also related to the classical idea of metric/similarity learning as well as kernel learning. This is due to the fact that kernels are inherently similarity measures between samples. Similarity and metric learning provide some inter-

esting cost functions, which could be used to train the parameters for a quantum kernel. One example for this is the triplet loss function used for learning similarities between images [Che+10], which will be used as one loss function in this thesis. Triplet loss has been shown to be usable for training variational quantum circuits [WKS22].

The closest classical approach to the topic of this thesis is "Deep Kernel Learning for Clustering" [Wu+19]. Their approach is to learn two objectives at the same time. Firstly, an encoding of the samples into a space, where the data is linearly separable. This is achieved using a deep neural network. And secondly a spectral embedding is learned, that reduces the dimension of all samples similar to the original spectral clustering algorithm (see Section 2.1.3.4). This learned embedding is then used in standard clustering algorithms such as k-means. In the worst case, the approach promises to be as good as spectral clustering. But due to the optimization with real data, in many cases the clustering outperforms the spectral clustering algorithm.

3. Approach

This section introduces the main idea of this thesis, using variational quantum kernels (VQKs), introduced in Section 2.2.11, in combination with classical clustering algorithms. The goal is to increase the clustering accuracy especially for datasets that are not-linearly separable. The first section showcases the pipeline and process for this combination. Afterwards some options for possible cost functions for semi-supervised and unsupervised learning are presented.

3.1. Variational Quantum Kernels for Clustering

The combination of VQKs and classical machine learning algorithms was first introduced in [Hub+21]. They used the kernel target alignment (KTA) (see Section 3.3.1) to train the quantum kernel function to represent the training data well. This kernel function was then used in combination with a support vector machine (SVM) to assign class labels to unseen data.

There are however other machine learning applications that can benefit from kernel functions as well, namely clustering (see Section 2.1.3), where kernels are used to group similar samples. The idea is to train a variational quantum kernel similar to [Hub+21], but combine it with popular clustering algorithms.

One challenge of this approach is training the quantum kernel. SVMs are generally considered a supervised learning task, where large training datasets with labeled samples can be used for training. Clustering on the other hand is considered unsupervised, meaning that no labeled data is accessible. This is problematic for optimizing the parameters of a quantum kernel, as many classical cost functions as well as KTA from [Hub+21] require labeled training data. To circumvent these issues and use already established cost functions for training, clustering can be described as a semi-supervised learning task. This redefinition makes small amounts of labeled training data accessible.

3.2. Training and Testing Pipeline

The idea presented in this thesis for clustering with a variational quantum kernel (VQK) is separated into two steps, training the VQK and testing it in combination with a clustering algorithm. The pipeline for training is shown in Figure 3.4, the one for testing in Figure 3.5. Section 3.2.1 details the circuit layout of the variational quantum kernel. Afterwards, the training and testing pipelines are detailed in Section 3.2.2 and Section 3.2.3.

3.2.1. Circuit Layout

As outlined in Section 2.2.11, variational quantum kernels apply the concept of variational circuits to encode data and use trainable parameters. Therefore a basic circuit ansatz has to be defined, which is repeated multiple times and used for calculating the kernel function. To focus on the transfer of VQK to clustering, this thesis uses an ansatz introduced by [Hub+21] and shown in Figure 3.1.

The ansatz starts with H gates on each qubit followed by R_Z gates that encode the data. The data is encoded cyclically. If the dimension of the data x is larger than the number of qubits N, the first N parts of x are encoded in the first layer and the rest in the following layers. Furthermore, if the data can be fully encoded in a subset of gates, the remaining gates are used to encode the data again. The next gates in the ansatz are R_Y gates that are parameterized by the variational parameters. And lastly, CR_Z gates which are also parameterized by the variational parameters. For the CR_Z gates a qubit is the control bit for the rotation on the following qubit.

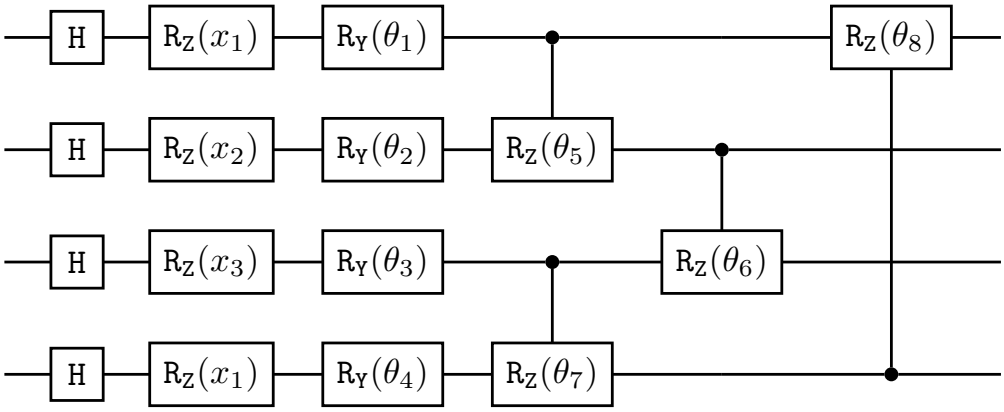


Figure 3.1.: Used ansatz for 4 qubits and data of dimension 3

In general, there is no intuition whether a specific ansatz is a good fit for a specific application [SP18]. However, the ansatz in Figure 3.1 has two advantages concerning the data encoding. Firstly, the data is repeated if there are more qubits than features or split with the next layer if there are more features in the data than qubits. This allows for a flexible scaling of the number of qubits. Secondly, the data encoding is repeated in every layer. This technique is known as data reuploading and was shown to have a positive effect on quantum classifiers [Pér+20].

Note that when designing a circuit layout to use with the adjoint method (see Section 2.2.10) of computing a quantum kernel, it is important to use a separating data gate. The ensuing problem, when not following this, is shown in Figure 3.2. The shown ansatz consists of a R_X gate to encode the data and a R_Y gate with a free parameter. When designing the ansatz in this order and its adjoint is added on the same qubits, the effects of the parameterized R_Y gates will cancel each other out. A better design is to turn around the ansatz and firstly use the free parameter and encode the data afterwards as shown in Figure 3.3. When the adjoint is added, the data encoding acts as a barrier and the parameterized gates will not cancel each other.

If the circuit layout is built up of a layered architecture with many layers, where the data is encoded in every layer, the problem is not as predominant, as only the last layer is affected.

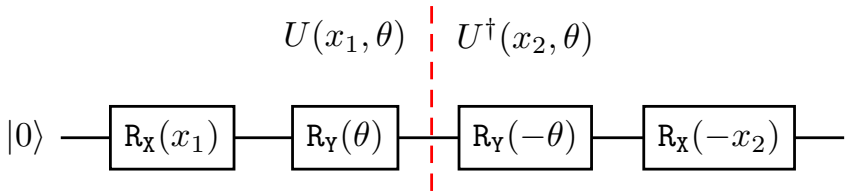


Figure 3.2.: Adjoint ansatz which leads to parameterized gates cancelling each other.

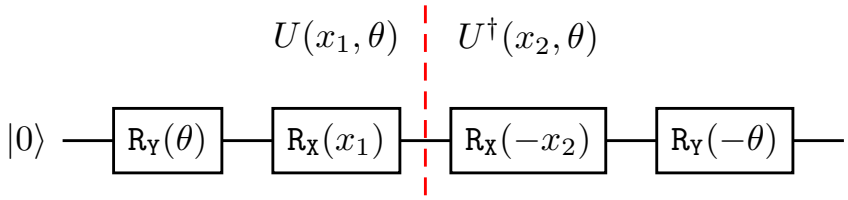


Figure 3.3.: Adjoint ansatz where a data encoding gate is used as a barrier so that the parameterized gates do not cancel each other.

3.2.2. Training Pipeline

The training pipeline in Figure 3.4 starts with the necessary input, training data D and initial parameters θ_0. Whether the training dataset contains labeled $\{(x_i, y_i)\}_{i=1}^k$ or unlabeled $\{(x_i)\}_{i=1}^k$ data is determined by whether the task is considered semi-supervised or unsupervised. In the case of semi-supervised it is generally assumed that only few labeled training samples are available (k typically small).

The pipeline itself consists of several steps, each of which is computed either classically or on a quantum computer. In Figure 3.4 this is highlighted by the used coloring scheme. The parameterized quantum kernel $k_\theta(x_i, x_j)$ is used as a subroutine to calculate the full kernel matrix $[K]_{i,j}$ for all possible sample combinations (x_i, x_j). This kernel matrix is then used in a cost function $C(\theta)$ to determine the quality of the parameters θ. The cost function is then used in a classical optimization algorithm, which finds new parameters θ_{t+1} that minimize the cost function. This cycle is repeated numerous times and the result is a set of optimized parameters θ_t.

3.2.3. Testing Pipeline

The testing pipeline shown in Figure 3.5 uses a combination of classical and quantum computation, similar to the training pipeline in Figure 3.4. The testing data is always a set of unlabeled data $\{(x_i)\}_{i=1}^l$ and the parameters θ_t were determined by the training pipeline

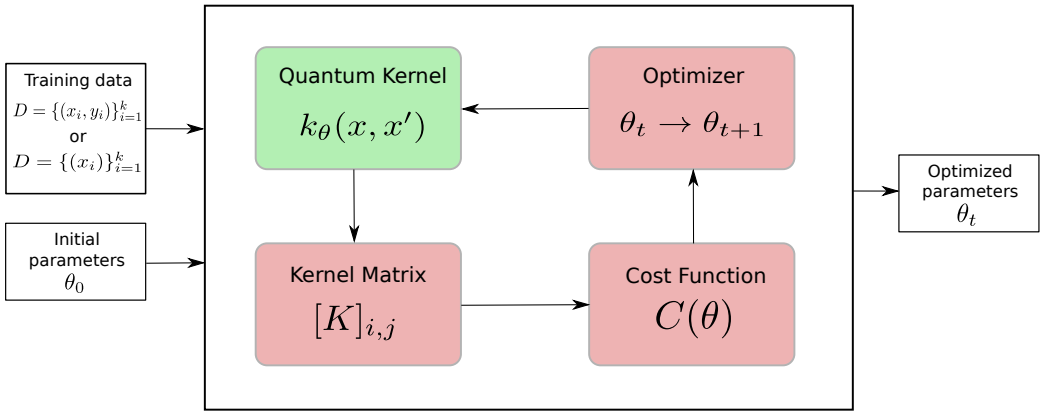


Figure 3.4.: Pipeline for training a variational quantum kernel as a combination of quantum computations (green) and classical optimization (red)

in Figure 3.4. Based on this, the kernel matrix can be calculated for each pair of samples (x_i, x_j) using the quantum kernel function $k_{\theta_t}(x_i, x_j)$. On the one hand, the kernel matrix represents a measurement of the quantum kernel and on the other hand it can be used as a core part of classical clustering algorithms. Therefore, it allows the combination of quantum kernels and classical clustering algorithms to cluster the test dataset.

Some possible clustering algorithms were detailed in Section 2.1.3. Kernel k-means and spectral clustering can use the kernel matrix directly. DBSCAN and hierarchical clustering both rely on a distance measure instead. The kernel function k is inherently a similarity measure and a corresponding distance function can be calculated as $1 - k$.

The performance of the overall pipeline is evaluated by comparing the clusters found by the algorithm and the real labels of the test data.

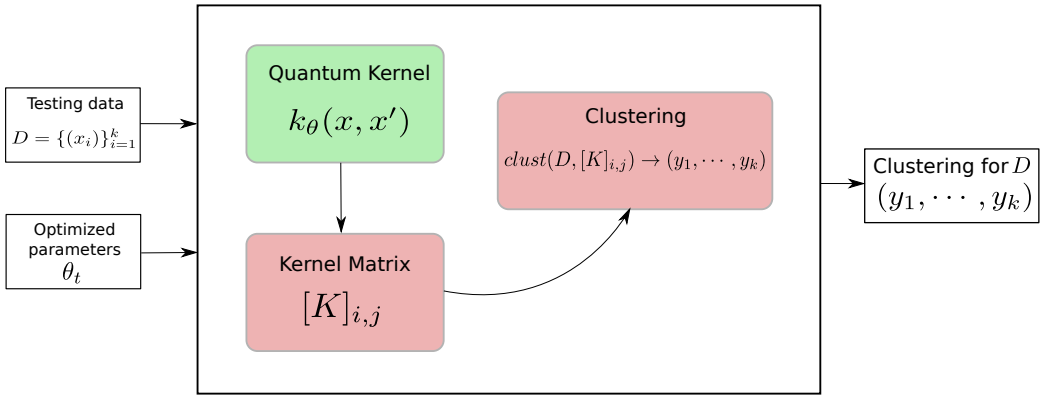


Figure 3.5.: Pipeline for combining a trained quantum kernel (green) with classical clustering algorithms (red)

3.3. Semi-Supervised Cost Functions

While clustering is typically defined as an unsupervised learning problem, this condition can be softened to a semi-supervised learning problem. This makes a training dataset D with labeled samples (x_i, y_i) available. In comparison to fully supervised learning, semi-supervised learning assumes that this dataset D is rather small, as labeled samples are hard to acquire.

This section introduces some possible cost functions that can utilize this semi-supervised learning approach to optimize a variational quantum kernel, namely kernel target alignment and triplet loss.

3.3.1. Kernel Target Alignment

Kernel target alignment (KTA) is a measure to compare the differences between two kernel matrices and can be used as a cost function to train a quantum kernel function [Cri+06].

Given a set of labeled samples $D = \{(x_i, y_i)\}_{i=1}^{n}$, with $x_i \in X$ being the input space and $y_i \in \{\pm 1\}$ being the labels. Assuming two kernel functions k_1 and k_2 and their according kernel matrices K and M, the KTA is defined as:

$$A(K, M) = \frac{\langle K, M\rangle_F}{\sqrt{\langle K, K\rangle_F \langle M, M\rangle_F}}. \tag{3.1}$$

$\langle A, B\rangle_F$ is the Frobenius inner product between two general matrices $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{n\times m}$ as defined by:

$$\langle A, B\rangle_F = \sum_{i=1}^{n}\sum_{j=1}^{m} A_{ij}B_{ij}.$$

For two kernel matrices $K \in \mathbb{R}^{n\times n}$ and $M \in \mathbb{R}^{n\times n}$ the Frobenius inner product can be described by:

$$\langle K, M\rangle_F = \sum_{i,j=1}^{n} K_{ij}M_{ij}.$$

The value of the KTA for two arbitrary matrices is in the range of $[-1, 1]$ [WZT12]. For two kernel matrices the range reduces to $[0, 1]$, as kernel matrices are always positive semidefinite (see Section 2.1.2). A KTA close to 1 means the kernel matrices are very similar, while a KTA closer to 0 shows a significant difference between the two matrices.

To use the kernel target alignment as a cost function to optimize a variational quantum kernel, the current kernel function k_θ and its kernel matrix K have to be compared to a target matrix. This is done by defining an ideal kernel matrix

$$K^* = yy^T, \tag{3.2}$$

with $y = (y_1, \ldots, y_k)$ being the vector of labels from the dataset D. This definition can be reformulated to

$$[K^*]_{i,j} = \begin{cases} 1, & \text{if } y_i = y_j \\ -1, & \text{if } y_i \neq y_j. \end{cases}$$

If both samples x_i and x_j are from the same class, the multiplication of their labels y_i and y_j leads to a matrix entry of 1. In the opposite case, if the two samples are from different classes, the multiplication leads to -1. This is the desired behaviour, as samples from the same class should have a high kernel value, as they are very similar to another. Samples from different classes are not similar and therefore should have a low kernel value.

Furthermore, as most optimization algorithms try to minimize the cost function, but higher KTA values denote a better alignment, an adequate cost function for minimization is the negative KTA.

The resulting cost function to be optimized by a minimization algorithm for a kernel matrix K is:

$$\begin{aligned} cost(K, K^*) = -A(K, K^*) &= -\frac{\langle K, K^* \rangle_F}{\sqrt{\langle K, K \rangle_F \langle K^*, K^* \rangle_F}} \\ &= -\frac{\langle K, yy^T \rangle_F}{\sqrt{\langle K, K \rangle_F \langle yy^T, yy^T \rangle_F}} \end{aligned}$$

3.3.1.1. Multiclass datasets

The definition of kernel target alignment used so far assumes that the data is separated into two classes, +1 and -1. This is especially important for the definition of the ideal target matrix in Equation (3.2).

However, many datasets contain more than two classes. This requires a new definition of the ideal matrix K^*. Matching classical approaches [GLV04], this thesis proposes the solution:

$$[K^*]_{i,j} = \begin{cases} 1, & \text{if } y_i = y_j \\ \frac{-1}{m-1}, & \text{if } y_i \neq y_j, \end{cases}$$

with m the number of different classes in the dataset, for quantum kernels as well. For $m = 2$ this reduces to the definition of $K^* = yy^T$ as seen in Equation (3.2).

3.3.1.2. Unbalanced datasets

Balanced datasets have equal amounts of samples from each possible class. However, in most real-world cases datasets are not balanced but have an equal amount of samples from each class. These unbalanced datasets haven been shown to not work well with the standard definition of kernel target alignment in the classical case, as the accuracy of the measure drops significantly [CMR12].

This problem can be addressed by centering the kernel function k by subtracting the expected value [CMR12]. This leads to the definition of a centered kernel function:

$$k_c(x_i, x_j) = k(x_i, x_j) - E_{x_i \in D}[k(x_i, x_j)] - E_{x_j \in D}[k(x_i, x_j)] + E_{x_i, x_j \in D}[k(x_i, x_j)],$$

where $E_{x \in D}$ is the expected value of the kernel function for a $x \in D$.

The associated centered kernel matrix K_c to a non-centered matrix K can be equally defined by subtracting the expectation value:

$$[K_c]_{ij} = K_{ij} - \frac{1}{n}\sum_{i=1}^{n} K_{ij} - \frac{1}{n}\sum_{j=1}^{n} K_{ij} + \frac{1}{n^2}\sum_{i,j=1}^{n} K_{ij}.$$

This centered kernel matrix can be calculated using [CMR12]:

$$K_c = (I - \frac{\mathbf{1}\mathbf{1}^T}{n})K(I - \frac{\mathbf{1}\mathbf{1}^T}{n}),$$

with $\mathbf{1} \in \mathbb{R}^{n\times 1}$ being the vector of all ones.

The kernel target alignment for centered matrices is equivalent to the standard definition in Equation (3.1), only replacing matrices with their centered alternative [CMR12]:

$$A(K, K^*) = \frac{\langle K_c, K_c^* \rangle_F}{\sqrt{\langle K_c, K_c \rangle_F \langle K_c^*, K_c^* \rangle_F}}.$$

3.3.2. Triplet Loss

Triplet loss is the general idea of constructing a loss function as a comparison between three samples, also called a triplet. The triplet contains a reference sample (anchor a), a sample from the same class as the anchor (positive p) and a sample from a different class (negative n). The assumption is that the anchor should be closer to the positive example than to the negative example [SKP15].

This idea of triplets is used in cost functions for learning an embedding of images [SKP15], but also for learning a similarity measure [Che+10]. The loss for a single triplet (a, p, n) and a parameterized similarity measure S_θ can be defined as:

$$l(a, p, n) = \max\{0, 1 - S_\theta(a, p) + S_\theta(a, n)\}. \tag{3.3}$$

When minimizing this loss, $S_\theta(a, p)$ is maximized and $S_\theta(a, n)$ is minimized, which conforms to the intended purpose of the anchor and positive being similar and the anchor and negative dissimilar.

The combined loss function for all triplets in a dataset D is then defined as the sum over all possible triplets:

$$L = \sum_{(a,p,n)\in D} l(a, p, n). \tag{3.4}$$

This cost function for similarity learning can be transferred to kernel learning, as kernels are inherently similarity measures. For this, only the similarity function S_θ in Equation (3.3) is replaced by a parameterized kernel function k_θ, which leads to:

$$l(a, p, n) = max\{0, 1 - k_\theta(a, p) + k_\theta(a, n)\}.$$

The overall loss function in Equation (3.4) is defined over all possible triplets (a, p, n) in a dataset. However in most cases there are too many triplets for this approach to be

feasible. A possible alternative is using only a fixed number of triplets, which reduces the complexity. These triplets could be generated by randomly sampling an anchor. The positive example is then randomly sampled from the set of samples with the same class as the anchor. The negative example equivalently from the set of samples with a different class [Che+10].

3.4. Unsupervised Cost Functions

When staying strictly with an unsupervised learning approach for clustering, the dataset D does not contain any label information, but only the data samples itself $D = \{(x_i)\}_{i=1}^{k}, x_i \in X$. In this case, it is hard to use the kernel values directly for optimization, as there is no target value available to compare them against. It is however possible to use the kernel values to cluster the training data D and use the resulting clusters for optimization. This can be done, for example, with well known *clustering metrics* that evaluate the performance of a clustering for a given dataset D.

The following two sections introduce two clustering metrics that evaluate performance solely based on the dataset D and a clustering and could be used as a cost function for training variational quantum kernels.

3.4.1. Davies-Bouldin Index

The Davies-Bouldin index (DBI) measures good clustering as clusters that are dense and well separated from each other [DB79]. The density of a cluster i is defined as its internal dispersion B_i, which could be calculated as the distance of all samples in the cluster to its center. The importance is that B_i is rather small for very dense clusters.

The separation between different clusters i and j is defined as $M_{i,j}$, which can be calculated as the distance between the cluster centers of i and j. Well separated clusters are indicated by a high value for $M_{i,j}$.

Density and separation can be combined to a single measure:

$$R_{i,j} = \frac{B_i + B_j}{M_{i,j}},$$

which is small for dense and well separated clusters i and j.

The Davies-Bouldin index can then be defined as:

$$DBI = \frac{1}{k} \sum_{i=1}^{k} max_{i \neq j} R_{i,j},$$

for a dataset of k clusters. The DBI chooses the worst possible $R_{i,j}$ for each cluster. This results in scores closer to 0 indicating dense and well separated clusters.

3.4.2. Calinski-Harabasz Index

The Calinski-Harabasz index (CHI) is a metric to measure the dispersion within and between different clusters [CJ74]. For a dataset D of size n clustered into k clusters the

CHI can be defined by:

$$CHI = \frac{tr\{B_k\}}{tr\{W_k\}} \cdot \frac{n-k}{k-1}.$$

B_k is the matrix describing dispersion between different clusters. With the number of points n_q in cluster q, the center c_q of cluster q and the center c_D of the dataset D, B_k can be defined through:

$$B_k = \sum_{q=1}^{k} n_q (c_q - c_D)(c_q - c_D)^T.$$

Similarly, W_k describes the dispersion within the clusters using the set of points from each cluster C_q and its center c_q:

$$W_k = \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)(x - c_q)^T.$$

The CHI is generally higher when clusters are well separated, as $tr\{B_k\}$ is large, and dense, as $tr\{W_k\}$ is small.

As a high CHI relates to good clustering performance, for usage as a cost function, which will be minimized, the negative CHI has to be used.

4. Evaluation

This chapter presents the implementation and evaluation results of training a variational quantum kernel (VQK) and combining it with classical clustering algorithms. These results are compared to the performance of various clustering algorithms, which fully operate on classical computers. Furthermore, the influence of some important classical and quantum specific parameters on the performance of the VQK is investigated. Lastly, the effects of different types of quantum specific noise are investigated.

Section 4.1 introduces the implementation, used datasets and different clustering algorithms used for the evaluation. The performance of a VQK under perfect conditions is detailed in Section 4.2. Different simulations with real-word conditions, especially concerning the effect of noise, are presented in Section 4.3. Lastly, Section 4.4 discusses the obtained results and gives an outlook into possible future work.

4.1. Setup

This section introduces the main setup used for the following evaluation. Section 4.1.1 details the implementation of a variational quantum kernel for clustering as described in Chapter 3. Section 4.1.2 describes the datasets used for evaluating the implementation and the reference clustering algorithms are explained in Section 4.1.3. Lastly, Section 4.1.4 explains the used evaluation metric.

4.1.1. Implementation

The approach from Chapter 3 was implemented in python 3.8 [VD09]. The main library used for implementing quantum circuits is pennylane version 0.22.1 [Ber+20]. While pennylane allows the usage of real quantum hardware, for example from IBM, all results in this thesis are obtained by simulating a quantum computer on a classical computer. Pennylane also provides implementations of popular optimization algorithms such as gradient descent (GD). The optimization is done in epochs, where in each epoch all samples from the training dataset are processed once. For this the training dataset is split into smaller batches, which are then used for individual optimization steps.

Pennylane has the advantage of having different ways of simulating a quantum computer. The analytical mode does not use shots, but explicitly calculates and saves the quantum states obtained when applying gates to a qubit. The final probabilities of 0 and 1 are then obtained by the measurement rule from Section 2.2.4. This simulation mode is especially useful when the effects of a calculation are important, but not the influence of noise. The alternative way of simulation is more closely related to real quantum computer. The probabilities are approximated by the expectation value over many shots. This mode

is useful, if the simulation should be as closely related to a real quantum computer as possible.

The remaining mathematical calculations are done with numpy [Har+20]. The implementations for spectral clustering, DBSCAN and hierarchical clustering are used from scikit-learn [Ped+11]. An implementation of kernel k-means [Blo] was used with small modifications to fit python 3 and be usable for unsupervised cost functions. All plots are generated using matplotlib [Hun07].

The source code is available on GitHub[1]. All simulations were performed on bwUniCluster 2.0. The author acknowledges support by the state of Baden-Württemberg through bwHPC.

4.1.2. Datasets

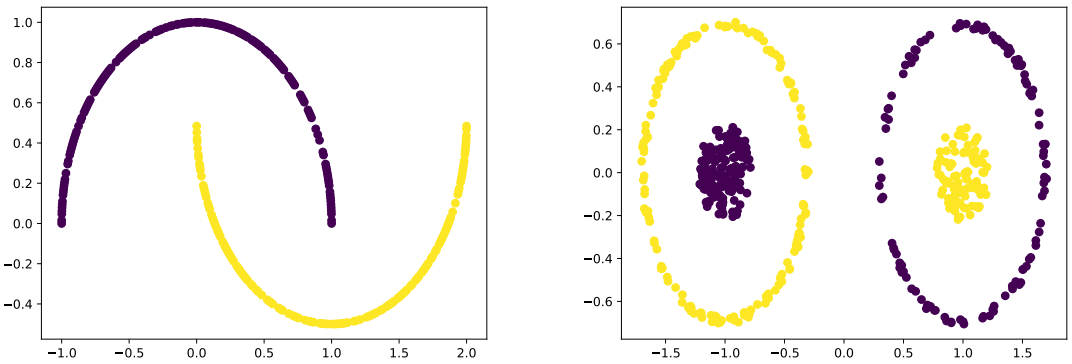
The evaluation is done using three different datasets, namely "moons", "donuts" and "Iris".

Firstly, the moons dataset is provided by scikit-learn [Ped+11]. Moons is a two dimensional toy dataset containing two interleaving half-circles, or "moons". Each moon is assigned its own class label. An example of this dataset is shown in Figure 4.1a.

Secondly, the two dimensional donuts toy dataset introduced in [Hub+21] is used. The dataset contains two shapes that resemble a donut. Both donuts consist of two ovals. The outside oval is assigned to the first class and the inside oval to the second class. For the second donut, the class assignments are switched, meaning the outside oval is assigned to class two and the inside oval to class one. The dataset can be seen in Figure 4.1b.

The last dataset is the real-world Iris dataset introduced in [FIS36] and provided by scikit-learn [Ped+11]. The dataset is three dimensional and contains three different classes for different species of the Iris flower. Iris contains a total of 150 labeled samples.

These datasets are divided into three subsets for training, validation and testing. The training dataset typically has 30 samples. The implications of this size are investigated in Section 4.2.6. The validation set consists of 10 samples and the testing dataset has size 100.



(a) Moons dataset

(b) Donuts dataset

Figure 4.1.: Toy datasets (moons and donuts) used for the evaluation.

[1] https://github.com/nikmetz/quantum-clustering/

4.1.3. Algorithm Selection

The approach of this thesis is compared to multiple different purely classical clustering algorithms, which are explained in this section.

Firstly, the kernel-based algorithms kernel k-means (see Section 2.1.3.2) and spectral clustering (see Section 2.1.3.4). Both are used in combination with the popular radial basis function (RBF) kernel defined as follows [VTS04]:

$$k(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2).$$

The free parameter γ is optimized by hand to achieve a good clustering result.

The KernelNet (KNet) algorithm proposed in [Wu+19] can also be categorised as a kernel-based method. The algorithm can be categorised as a centroid-based algorithm, as it uses k-means after learning a kernel and appropriate embedding. The free parameters are taken from experiments with comparable datasets provided in the implementation of KNet [Wu+].

Secondly, the distance based algorithms DBSCAN (see Section 2.1.3.3) and hierarchical clustering (see Section 2.1.3.5), with both using the euclidean distance. Hierarchical clustering uses the bottom-up approach and the single linkage criteria to combine clusters. The free parameters for both algorithms are optimized by hand to achieve the best clustering.

4.1.4. Clustering Evaluation

There are numerous different evaluation metrics for clustering, but the normalized mutual information (NMI) is most commonly used in the literature and will be used in this thesis to evaluate clustering performance [SG02; VEB10]. For a clustering $U = \{U_1, \ldots, U_R\}$ of n samples into R clusters, the entropy is defined as [VEB10]:

$$H(U) = -\sum_{i=1}^{R} \frac{|U_i|}{n} log\left(\frac{|U_i|}{n}\right).$$

For two clusterings U and V of n samples, the mutual information MI is defined through [VEB10]:

$$MI(U, V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{n} log\left(\frac{n|U_i \cap V_j|}{|U_i||V_j|}\right).$$

This mutual information is then normalized using the entropy of both clusterings to calculate the NMI [VEB10]:

$$NMI = \frac{MI(U, V)}{\sqrt{H(U)H(V)}}. \tag{4.1}$$

The NMI is in the range $[0, 1]$, with 1 representing the perfect clustering result.

Table 4.1.: Default parameters used in all following noiseless simulations

	Dataset		
Parameter	Iris	Donuts	Moons
Training dataset size	30	30	30
Validation dataset size	10	10	10
Testing dataset size	100	100	100
Optimizer	GD	GD	GD
Cost function	KTA	KTA	KTA
Epochs	100	100	100
Batch size	5	5	5
Qubits	3	3	3
Layers	10	20	30

4.2. Noiseless Simulation

This section investigates the application of variational quantum kernels in clustering algorithms under perfect conditions, especially without any noise. This means that all quantum kernel functions can be calculated using the adjoint method.

The next section details the default parameters used in the simulations. These parameters are used as long as nothing else is mentioned in the further sections. Section 4.2.2 gives a general overview into the best results obtained in this thesis compared to standard algorithms and an approach from the literature.

Afterwards, different parameters and strategies are investigated concerning their influence on the clustering result. Section 4.2.3 details the combination of a variational quantum kernel with the four clustering algorithms kernel k-means, spectral clustering, DBSCAN and hierarchical clustering. Section 4.2.4 goes into detail about the semi-supervised and unsupervised cost functions proposed in Chapter 3. Section 4.2.5 examines the effects of quantum circuits with different numbers of qubits and layers. The size of the training dataset and its effects on the clustering performance of a test dataset are shown in Section 4.2.6.

4.2.1. Default parameters

The implementation allows for an extensive configuration using different parameters. In order to not describe these parameters in every section, a default set of parameters is listed in Tab. 4.1. If not stated otherwise, the parameters listed in that table are used in the following experiments.

Many of these parameters were determined by extensive experiments. In fact, the following sections investigate the influence of some of these parameters. The size of the training dataset was a consideration between being too small for training and too large to still count as semi-supervised. The effects of the training dataset size are further investigated in Section 4.2.6. The size of the testing dataset was chosen based on performance criteria,

as the kernel matrix for this dataset has to be computed every time the kernel is used for a clustering algorithm.

The number of epochs was again determined by testing, as after 100 epochs neither the cost nor the clustering results change significantly. The circuit sizes for the three datasets were chosen based on the best achievable results. Further experiments concerning this size are shown in Section 4.2.5.

The initial parameters θ_0 for the VQK are chosen randomly in the range $[0, 2\pi]$.

4.2.2. Overview of Results

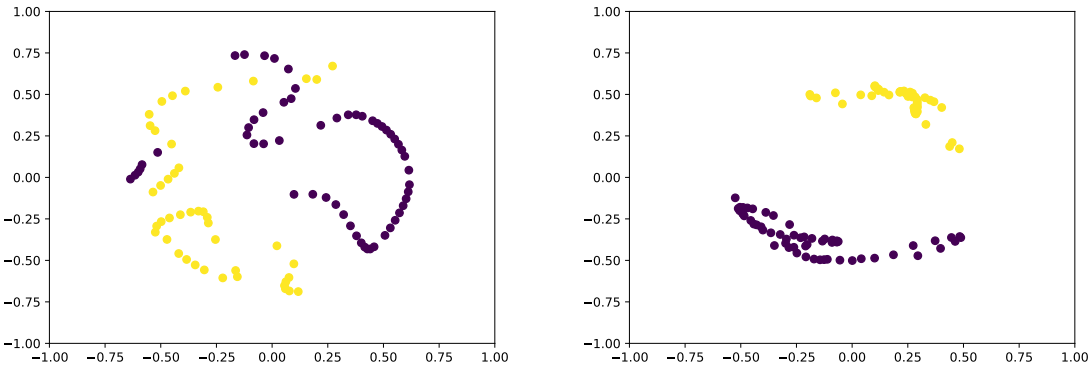
This section serves as an overview of the results and shows that the general approach of this thesis works as intended. The following three questions will be answered in this section:

1. Can the VQK benefit from training?
2. Can the training of a VQK be beneficial for the subsequent clustering?
3. Can VQKs lead to better clustering results than purely classical algorithms?

To visualize that the VQK benefits from training, Figure 4.2 shows the kernel matrix of the test data before the training and after 100 epochs of training for the moons dataset. The figure visualizes the values of a kernel matrix by drawing samples farther apart, if their similarity according to the kernel matrix is small, and closer together, if their similarity is bigger. For this the samples are being reduced to two dimensions based on their similarity. Remember that a good kernel function produces values close to 1, if the two samples are similar, and values close to 0, if the samples are dissimilar. Figure 4.2a shows the kernel matrix before the VQK is being trained. The figure displays samples from different classes close to each other, which means that the kernel assigns them high kernel values. This is unwanted behaviour, because the samples are from different classes and should rather receive kernel values close to 0.

In turn, Figure 4.2b shows the kernel matrix for the same test data after the training of the VQK. It can be seen that the two classes are now separated, meaning that samples from the same class produce kernel values close to 1 and close to 0 if they are from different classes. This is the intended behaviour for a good kernel function and answers the first question of this section, that VQKs can benefit from training. Note that the training in this case was done with the KTA cost function. The effects of different cost functions are further explained in Section 4.2.4.

The second question is concerned with the testing pipeline explained in Section 3.2.3. The VQK is combined with a clustering algorithm during training and every 20 epochs the test data is clustered once using the current VQK. The NMI of this regular testing for the moons dataset is shown in Figure 4.3. It can be seen that training the variational quantum kernel also increases the clustering performance. Before the training starts, the clustering leads to an NMI of 0.34. This improves during training by 0.66 or 194% to 1.00. Note that this result was obtained with spectral clustering, but Section 4.2.3 dives deeper into the usage and characteristics of different clustering algorithms.



Before training After training

Figure 4.2.: Kernel matrices for the moons dataset before training (Figure 4.2a) and after 100 epochs of training (Figure 4.2b) using the KTA cost function. The training leads to a separation of the two classes.

Table 4.2.: Results for clustering using a variational quantum kernel before and after training

	KernelKMeans		Spectral		DBSCAN		Hierarchical	
Dataset	Before	After	Before	After	Before	After	Before	After
Moons	0.73	1.00	0.34	1.00	0.56	1.00	0.58	1.00
Donuts	0.10	0.93	0.12	0.93	0.51	0.97	0.56	0.97
Iris	0.48	0.93	0.71	0.93	0.31	0.82	0.36	0.88

Additionally, Table 4.2 shows that for all datasets and clustering algorithms the NMI increases by training the VQK compared to the NMI before the training. While this performance increase exists for all highlighted clustering algorithms and datasets, the relative increase before and after the optimization is different for all cases. The largest performance increase can be noted for kernel k-means on the donuts dataset. An increase of the NMI from 0.10 to 0.93, i.e. by 0.83 or 830%, can be observed. In turn the smallest improvement can be seen with spectral clustering on the Iris dataset, with an increase from 0.71 to 0.93 or 31%. On average the performance increases by 206% after training the VQK.

Both, Figure 4.3 and Table 4.2, highlight that training a VQK leads to better clustering results compared to untrained VQKs.

The last question of this section is concerned with the performance compared to the purely classical clustering algorithms from Section 4.1.3. Table 4.3 shows the best results achieved by these algorithms on all datasets. Note that the results for the KNet algorithm from [Wu+19] for the Iris and donuts dataset should not be taken as the best achievable results, as the algorithm has many different parameters that can be optimized and influence

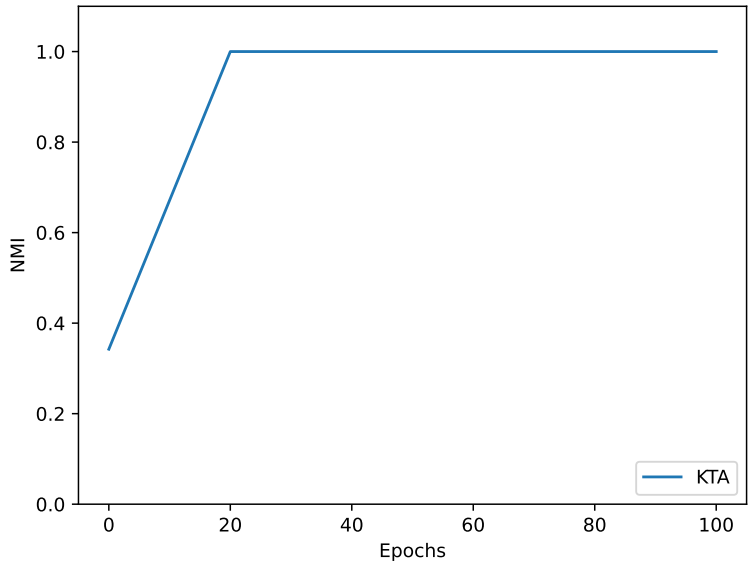


Figure 4.3.: NMI when testing a VQK using spectral clustering on the moons dataset. The clustering changes for the better, while the VQK is being trained.

Table 4.3.: Results for standard clustering algorithms and KNet

Dataset	KernelKMeans	Spectral	DBSCAN	Hierarchical	KNet
Moons	0.48	0.42	1.00	1.00	1.00
Donuts	0.83	0.46	1.00	1.00	0.40
Iris	0.81	0.79	0.74	0.74	0.69

the result heavily. Likewise the results for the standard algorithms can potentially be further optimized.

Comparing the results for standard algorithms in Table 4.3 with the results using the VQK in Table 4.2, it can be noted that the VQK is mostly at least as good and often times even better than the standard algorithms. The largest performance increases can be seen for the kernel k-means algorithm and spectral clustering using the VQK compared to the RBF kernel. On the moons dataset kernel k-means improves by 108% and spectral clustering by 137%. For DBSCAN and hierarchical clustering the performance difference is only marginal. On the moons dataset the NMI is identical at 1.00. The only two instances where a purely classical algorithm is better than the VQK variant is with DBSCAN and hierarchical clustering on the donuts dataset. In both instances the difference is only 3.61%.

Therefore, the answer for the last question is that the approach of this thesis using VQKs can increase the clustering performance compared to classical algorithms.

4.2.3. Combination with Clustering Algorithms

In Section 3.2.3 it was established that a VQK can be combined with different kernel-based algorithms, such as kernel k-means or spectral clustering, as well as distance-based algorithms, such as DBSCAN or hierarchical clustering. This section investigates the practical differences between kernel-based and distance-based algorithms.

4.2.3.1. Kernel-based Clustering Algorithms

Figure 4.4 shows the result of training a VQK and using this kernel function in kernel k-means and spectral clustering. The figure shows the NMI for both algorithms while training the donuts, Iris and moons dataset. It can be seen that both algorithms have a similar performance for all three datasets separately. During the training process there are minimal performance differences between the two clustering algorithms for the Iris (Figure 4.4b) and moons (Figure 4.4c) dataset. However, after the full training of 100 epochs, the NMI is identical for both algorithms. For the donuts dataset (Figure 4.4a) the NMI is identical for kernel k-means and spectral clustering over the complete training process.

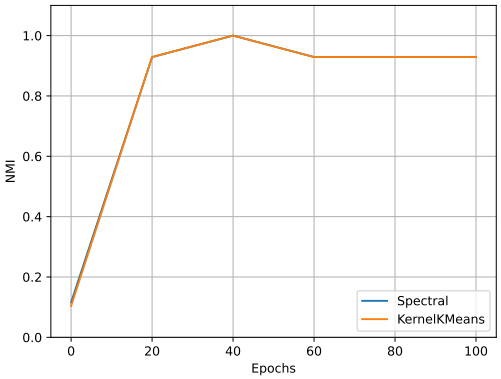
One important advantage for both algorithms, that is not seen in the figure, is that they do not require any additional parameter optimization aside from the kernel function. The algorithms are used "out-of-the-box" in combination with the VQK and perform well, without additional parameter optimization aside from the VQK. This is in contrast to the distance-based algorithms.

4.2.3.2. Distance-based Clustering Algorithms

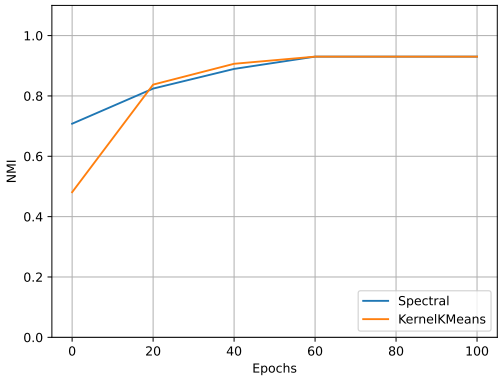
Both, DBSCAN and hierarchical clustering, use a distance measure based on a kernel function. Furthermore, both use parameters that interact with this distance measure. In DBSCAN ϵ defines the neighbourhood of a sample and *min_samples* specifies whether or not a point is a core point (see Section 2.1.3.3). In hierarchical clustering *distance_threshold* defines when to split or merge a cluster (see Section 2.1.3.5). These parameters have to be optimized in addition to the VQK, which adds more overhead.

Figure 4.5 shows the NMI for DBSCAN with different ϵ values for the donuts (Figure 4.5a) and Iris (Figure 4.5b) dataset, while training the VQK with a fixed value for *min_pts*. Both figures show that the performance of DBSCAN heavily depends on the parameter ϵ. This indeed makes sense, as even with a well optimized distance measure, ϵ directly influences how far apart samples can be while still being considered to be in the same cluster. However, ϵ cannot be optimized once for all datasets, but rather for each dataset individually. For example, in case of a fixed value of $min\_samples = 5$, the optimal value of ϵ for the donuts and Iris datasets are 0.4 and 0.3 respectively. Note that $\epsilon \in (0, 1]$, because the used distance measure is computed as $1 - k$, with the kernel function $k \in [0, 1]$. $\epsilon = 0$ is not useful, as the neighbourhood of each sample would be empty, except for identical samples.

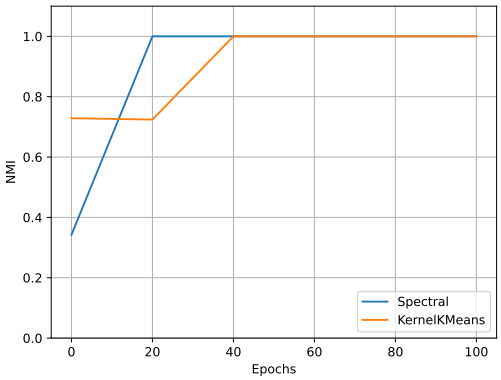
Furthermore, DBSCAN can be parameterized with *min_samples*. The influence of this parameter on DBSCAN for the donuts dataset with $\epsilon = 0.4$ and Iris with $\epsilon = 0.3$ are shown in Figure 4.6a and Figure 4.6b. Both figures show that after optimizing ϵ, different choices for *min_samples* do not make much of a difference to the overall performance.



(a) Donuts dataset

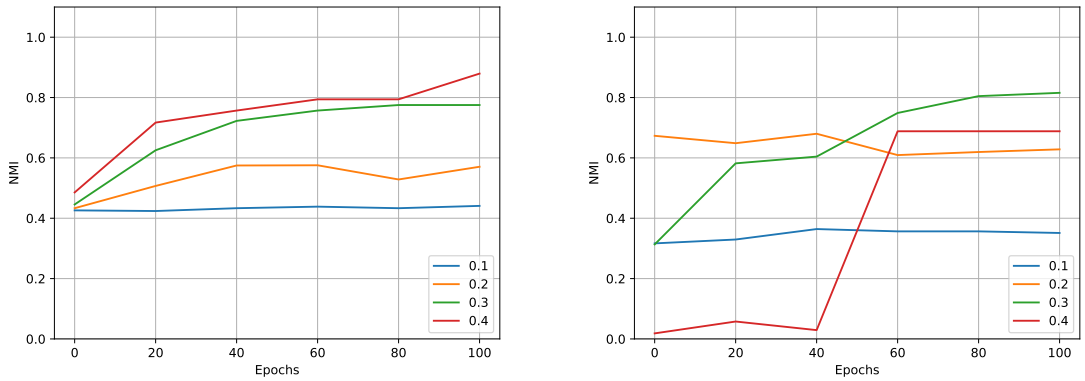


(b) Iris dataset



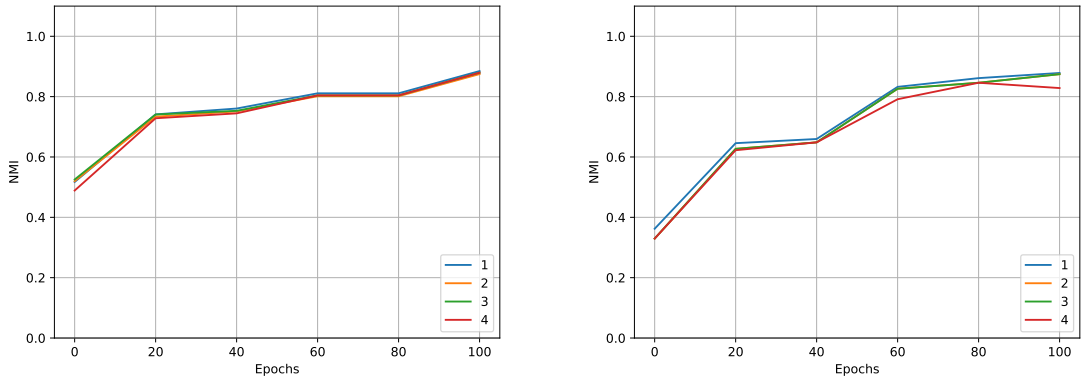
(c) Moons

Figure 4.4.: NMI for testing of kernel k-means and spectral clustering while training a VQK over 100 epochs for the donuts (Figure 4.4a), Iris (Figure 4.4b) and moons (Figure 4.4c) dataset. Both algorithms deliver a nearly identical result on all datasets.



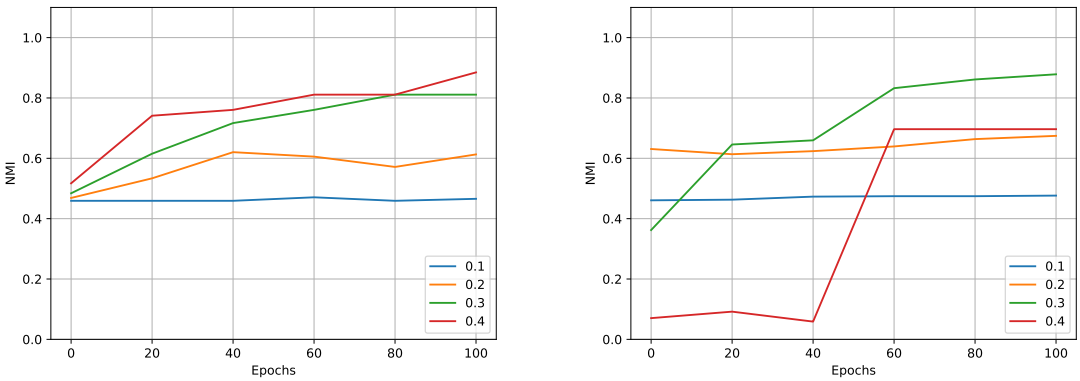
(a) Donuts dataset (b) Iris dataset

Figure 4.5.: NMI for testing of DBSCAN over 100 epochs for different ϵ values for the donuts (Figure 4.5a) and Iris (Figure 4.5b) dataset. For the donuts dataset $\epsilon = 0.4$ and for Iris $\epsilon = 0.3$ is optimal.



(a) Donuts dataset (b) Iris dataset

Figure 4.6.: NMI for testing of DBSCAN over 100 epochs for different *min_samples* values for the donuts (Figure 4.6a) and Iris (Figure 4.6b) dataset. The choice of *min_samples* does not make a big difference after fixating the ϵ parameter.



(a) Donuts dataset

(b) Iris dataset

Figure 4.7.: NMI for testing of hierarchical clustering over 100 epochs for different *distance_threshold* values for the donuts (Figure 4.7a) and Iris (Figure 4.7b) dataset. For the donuts dataset *distance_threshold* = 0.4 and for Iris *distance_threshold* = 0.3 is optimal.

Figure 4.7 shows the performance of hierarchical clustering on the donuts and Iris dataset for different *distance_threshold* values. Similar to DBSCAN, the performance heavily varies for different parameter choices. As the name of the parameter suggests, it is used as a threshold to decide which clusters should be merged or split based on their distance, which has a big impact on the performance no matter how well optimized the used kernel function is. Furthermore, similar to DBSCAN, the parameter is dependant on the dataset in question. For the donuts dataset in Figure 4.7a *distance_threshold* = 0.4 is optimal, while for the Iris dataset in Figure 4.7b *distance_threshold* = 0.3 is the optimal value.

Summarizing it can be seen that optimizing the kernel function alone does not immediately yield good results for distance-based clustering algorithms. They rather require further parameter optimization to fully utilize the trained kernel as a distance measure.

4.2.3.3. Conclusion

In conclusion, variational quantum kernels work well for all mentioned clustering algorithms, kernel-based as well as distance-based. However, distance-based algorithms require further optimization of classical parameters, while kernel-based algorithms can perform well by using solely the provided kernel.

As all algorithms were shown to be working, the remainder of this thesis will focus on spectral clustering.

4.2.4. Cost Functions

This section investigates the usability of different cost functions for optimizing a VQK. The cost function is used to determine the current quality of the parameters θ used in the

VQK. An optimization algorithm uses this cost function to minimize the cost and therefore optimize the parameters.

Chapter 3 introduced two different paradigms of defining such a cost function. Firstly, semi-supervised cost functions as introduced in Section 3.3, where a small set of labeled training data can be used. Secondly, unsupervised cost functions were introduced in Section 3.4 and only use unlabeled training data during the optimization.

The following two sections evaluate the performance of semi-supervised as well as unsupervised cost functions.

4.2.4.1. Semi-Supervised Cost Functions

This section investigates the usability of semi-supervised cost functions as introduced in Section 3.3. Semi-supervised cost functions can use labeled training data for the optimization. This data allows the cost function to compare current kernel values to optimal values obtained from the training labels.

Kernel target alignment (KTA) computes the optimal kernel matrix from the labels in the training dataset and compares this matrix to the current kernel matrix (see Section 3.3.1).

Figure 4.8 shows the cost for the validation data, while training the kernel function on the donuts dataset. For the first 40 epochs the validation cost is optimized from -0.6 down to -0.8. This indicates that the parameters for the VQK are being optimized. However, after 40 epochs, the validation cost starts to rise again, which can be explained by *overfitting*. Overfitting describes the effect that parameters are being optimized to perfectly fit the training data. However, adjusting the parameters to perfectly fit the training data also captures possible noise in the data. This leads to less generalization of the model and therefore worse performance on previously unseen data, i.e. the validation and test data. This could be the case in Figure 4.8 after 40 epochs, where the VQK is trained to perfectly represent the training data but fails to generalize to unseen data from the validation data.

There are multiple different strategies from classical machine learning that can be used to combat overfitting. Especially stopping the training process after 40 epochs, so before the overfitting occurs, can potentially be useful. This practice is known as *early stopping* [YRC07].

Figure 4.9a depicts the similarity induced by the kernel matrix of the testing donuts dataset before the training. Both classes appear to be heavily intertwined, meaning that samples from opposite classes receive kernel values close to 1 according to the untrained VQK. In contrast to this, Figure 4.9b shows the same kernel matrix after 100 epochs of training. The classes are now separated, as the kernel function assigns values close to 1 only to samples from the same class. This again shows that using the KTA cost function, to train the VQK, leads to a kernel that better represents the actual structure of the data.

Lastly, the VQK, that is trained by the KTA cost function, is combined with spectral clustering in the testing pipeline. Figure 4.10 shows the resulting NMI when clustering the testing data in said pipeline. The plot clearly shows that training the VQK using the KTA cost function increases the performance in following clustering steps. The initial NMI before training is at 0.12 and after 100 epochs of optimization it is at 0.93, which is an increase of 675%. The peak performance after 40 epochs is even better with an NMI of 1.00, which relates to an 733% increase.

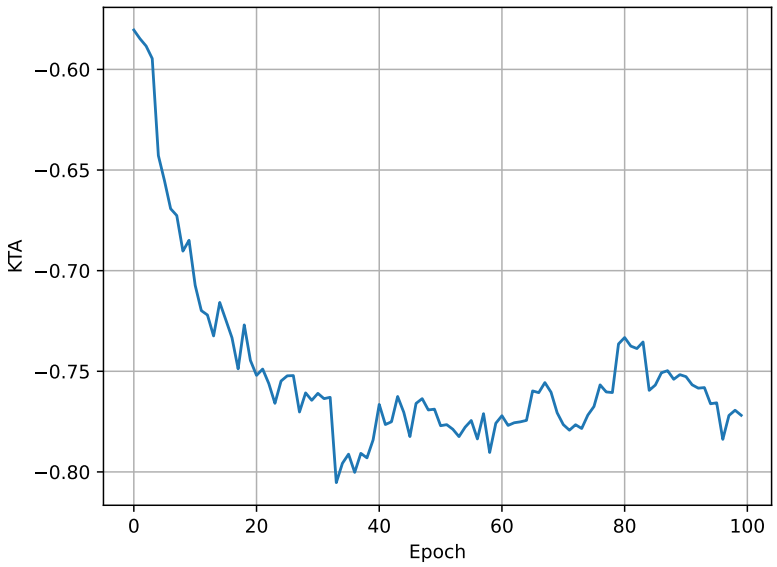
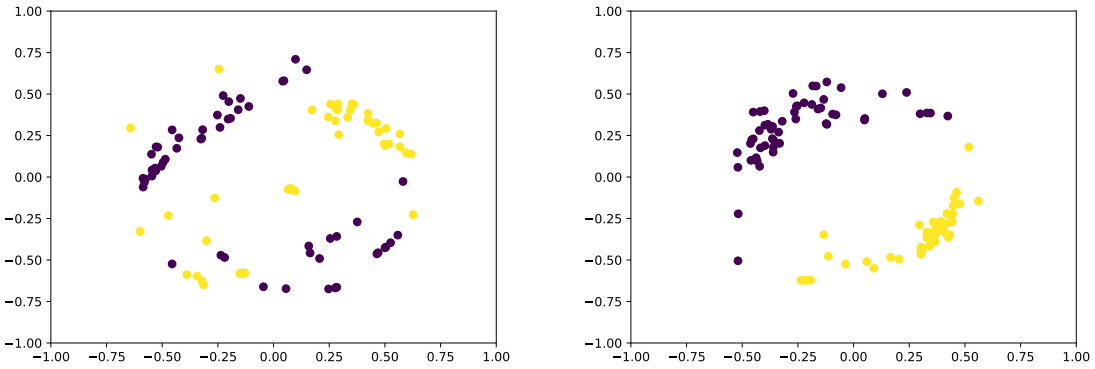


Figure 4.8.: Validation cost while training a VQK with the KTA cost function on the donuts dataset. The first 40 epochs show a good optimization behaviour. After 40 epochs the optimization runs into overfitting.



(a) Before training

(b) After training

Figure 4.9.: Kernel matrices before (Figure 4.9a) and after (Figure 4.9b) optimizing a VQK with KTA on the donuts dataset. Before the training the kernel values do not represent the class structure of the test data. This is only achieved after optimizing the VQK.

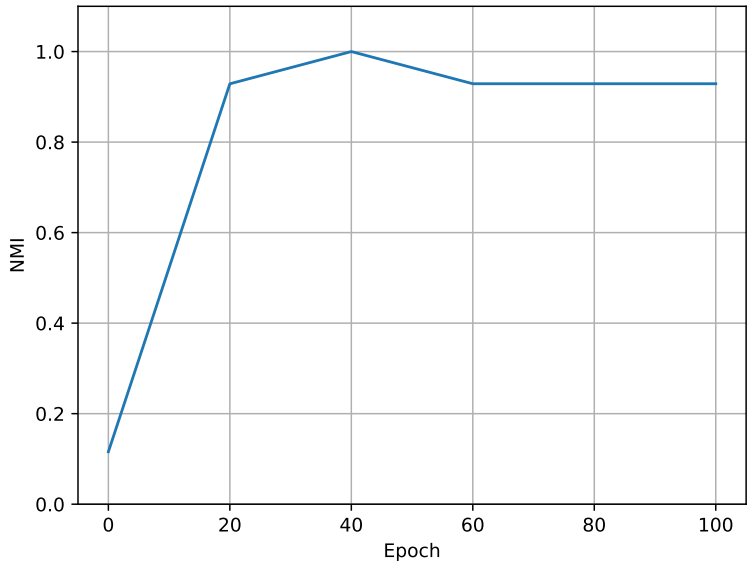


Figure 4.10.: NMI for the testing dataset during training with KTA on the donuts dataset. The behaviour is very similar to the validation cost in Figure 4.8. Until epoch 40 the optimization leads to good clustering performance. However, after epoch 40 the overfitting leads to a decrease in performance.

Furthermore it is also of interest how quickly the KTA cost function improves the kernel function. After 20 epochs the performance is already at an NMI of 0.93 and after 40 epochs at 1.00.

But also the overfitting effect observed in the validation cost in Figure 4.8 is visible in the clustering result in Figure 4.10. After 40 epochs the NMI starts to decrease again, similar to the validation cost which starts to increase after 40 epochs due to overfitting.

The KTA cost function not only works for datasets containing two classes, but also multiclass datasets such as Iris. The kernel matrices before and after the training are shown in Figure A.1 and show a good separation of the classses after training. The NMI after using the VQK with spectral clustering is shown in Figure A.2 and suggests that training the kernel function also increases the subsequent clustering performance.

Triplet loss as defined in Section 3.3.2, describes a cost function based on the kernel values of multiple triplets of samples. A triplet (a, p, n) is defined through an anchor a, a positive sample p of the same class as the anchor, and a negative sample n from a different class than the anchor. The class labels are available, because triplet loss was defined as a cost function for semi-supervised learning. The main idea for optimization is that the kernel value between a and p should be close to 1, as they are from the same class, and the kernel value between a and n should be nearly 0, because they are from different classes.

The validation cost during training on the donuts dataset is shown in Figure 4.11. In the plot, a downwards trend from a value of 6 to about 4.5 can be observed. The heavy fluctuations of the triplet loss value can be explained through the triplet loss definition.

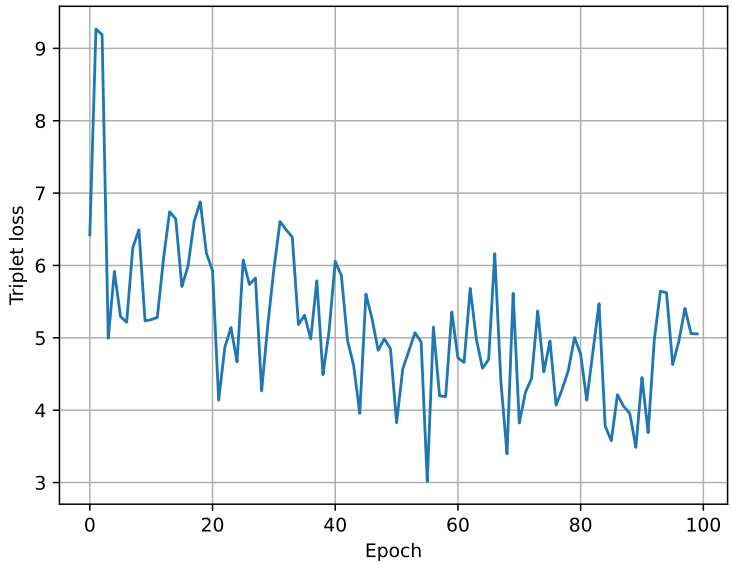


Figure 4.11.: Validation cost during training with triplet loss. Before epoch 60 a decreasing cost value is visible. After epoch 60 the cost increases again due to overfitting.

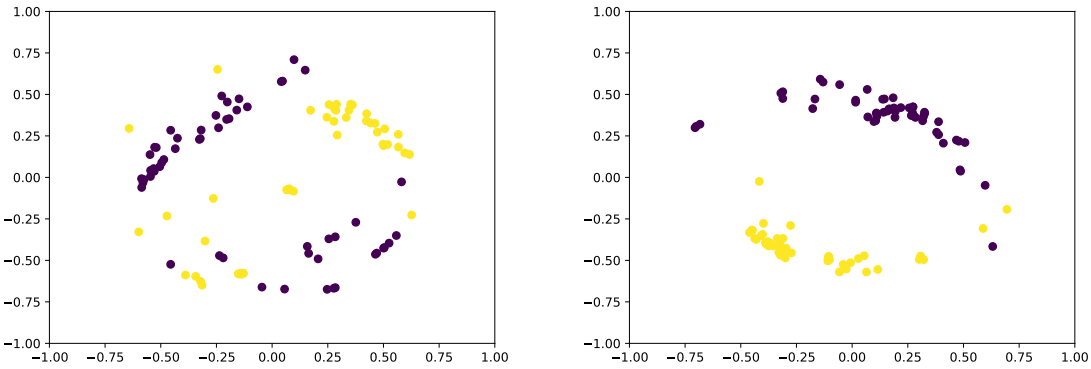
Instead of calculating the sum over all possible triplets, only a subset of all possible triplets is taken into account. In this case, each sample of the training dataset is considered as an anchor once and the positive and negative samples are chosen randomly from the corresponding classes. This randomness is the reason for the observed fluctuations in Figure 4.11.

Another effect observed in Figure 4.11 is the already known overfitting. After epoch 60 the validation cost does not decrease anymore, but rather increases again. The reasoning is the same as for the KTA cost function. The parameters are trained to perfectly represent the training data, but fail to generalize to unseen data from the validation set. Equally, early stopping after 60 epochs could be used to mitigate this behaviour.

The kernel matrix before the training is shown in Figure 4.12a. Before the training, the VQK is not yet optimized to the given dataset and cannot separate the classes. Figure 4.12b shows the kernel matrix after 100 epochs of training. Except for some outliers, all samples from the same class appear near each other, which indicates that the kernel assigns them a value close to 1. This shows that training a VQK with the triplet loss cost function leads to a kernel which better represents the dataset.

The result of the combination of the VQK, trained with triplet loss, and spectral clustering is shown in Figure 4.13. This plot shows some interesting features.

Firstly, the positive effect on the clustering from training the VQK can be clearly seen. The initial VQK produces an NMI of 0.12. After 100 epochs the NMI increases to 0.88, which is an increase of 633%. At the peak performance after 40 epochs the NMI is at 1.00, which relates to a performance increase of 733%.



(a) Before training

(b) After training

Figure 4.12.: Kernel matrices of the test dataset with an untrained (Figure 4.12a) and trained (Figure 4.12b) VQK using triplet loss. Before training, the kernel matrix does not represent the real structure of the data. However, after the optimization, samples from the same class receive a high kernel value and samples from opposite classes a low kernel value, as intended.

Secondly, the overfitting problem mentioned for Figure 4.11 can also be seen in Figure 4.13. After 60 epochs the clustering performance starts to decrease again, because the VQK is being trained too much.

Lastly, optimizing with triplet loss quickly leads to an increase in clustering performance. After 40 epochs of training the test data is already clustered perfectly.

Triplet loss also performs well for a multiclass dataset. Figure A.3 shows the kernel matrix for the Iris dataset before and after training. After the optimization the three classes are separated nicely. The NMI for the clustering result using this VQK is shown in Figure A.4 and further shows that triplet loss can be effectively used to cluster mutliclass datasets.

Comparison The previous sections have shown that both KTA as well as triplet loss can be used to optimize a VQK. However, there are some small but important differences between the two cost functions.

Figure 4.14 shows the clustering performance comparison for the KTA cost function and triplet loss. The plot shows that KTA has a better performance most of the time compared to triplet loss. Furthermore, KTA can increase the performance quicker in comparison to triplet loss, which is seen in epoch 20. These differences can be attributed to the way KTA and triplet loss operate. KTA compares every kernel value to its optimal value, computed from the real labels of the data, and optimizes the parameters of the VQK accordingly. KTA therefore compares $O(batch\_size^2)$ many kernel values in each step of the optimization. Triplet loss only uses $batch\_size$ many triplets, with 2 kernel values each, leading to $O(batch\_size)$ total kernel values used in each step. Therefore KTA has a more precise measurement of how good the current kernel is.

However, this additional precision comes at a cost. The current speed limitation when training the VQK is simulating the quantum device. KTA therefore has the disadvantage

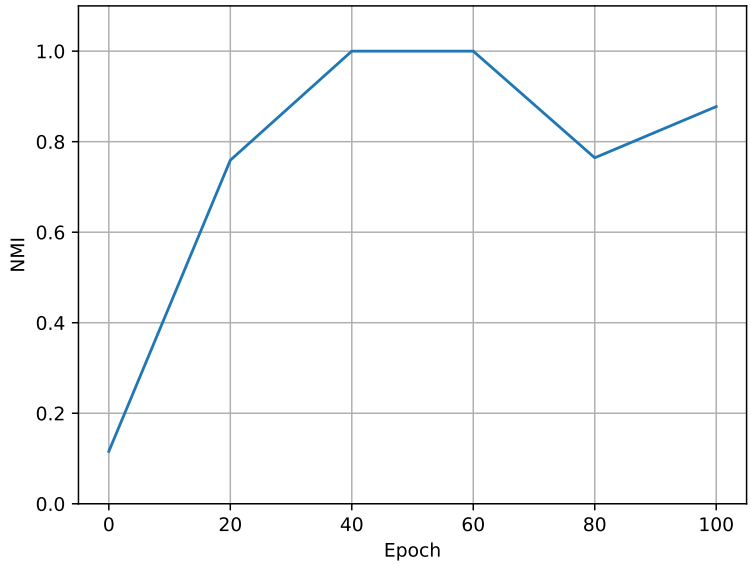


Figure 4.13.: NMI for the test data with the triplet loss function on the donuts dataset. Until epoch 60 the otimization of the VQK leads to an increase in clustering performance. After 60 epochs the overfitting becomes problematic and decreases the performance.

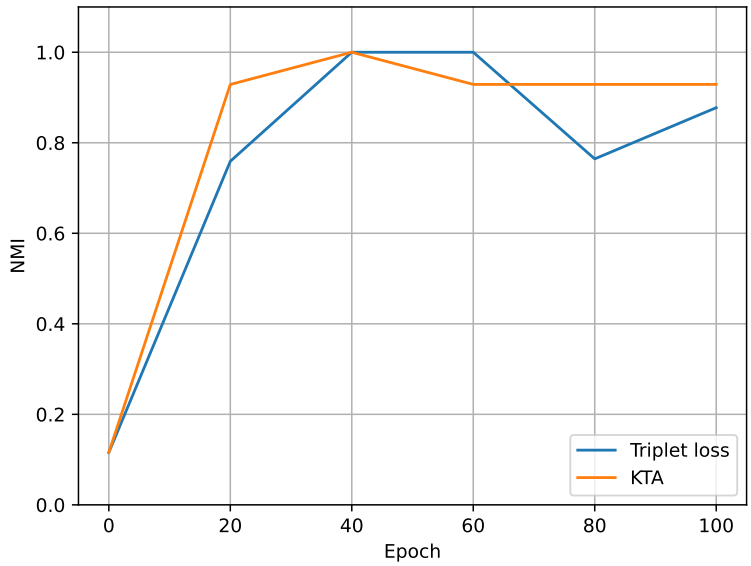
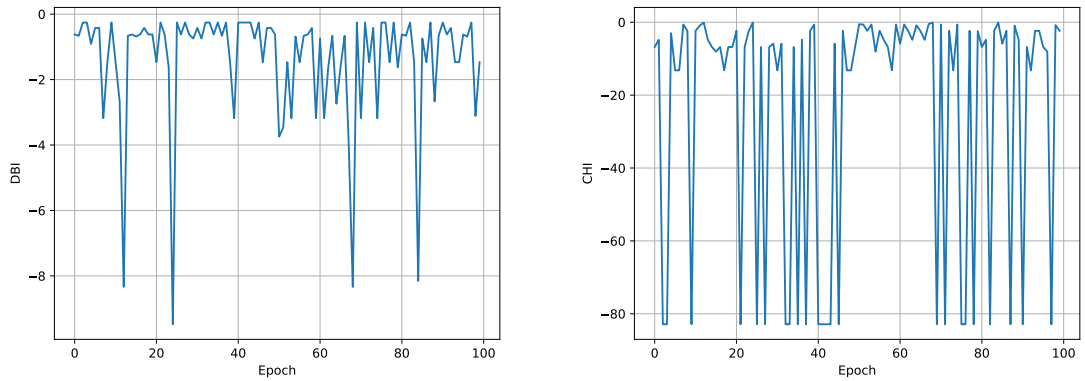


Figure 4.14.: Testing result during training with the KTA and triplet loss cost functions on the donuts dataset. KTA achieves the optimal NMI of 1.00 faster than triplet loss. Both cost functions experience an overfitting, but the general effect is smaller when training with the KTA function.



(a) DBI cost function

(b) CHI cost function

Figure 4.15.: Validation cost during the training of a VQK with the DBI (Figure 4.15a) and CHI (Figure 4.15b) cost functions. Both cost functions fail to optimize the parameters of the VQK to even recognize a trend on the validation data.

of having to query the quantum simulation $O(batch\_size^2)$ times, while triplet loss only requires $O(batch\_size)$. This can also be translated to the usage of current NISQ era quantum devices. Because only few of these quantum computers exist, a cost function that is able to train the VQK, but uses the quantum computer very little, is preferrable. This leads to a possible advantage of triplet loss over KTA for current NISQ devices.

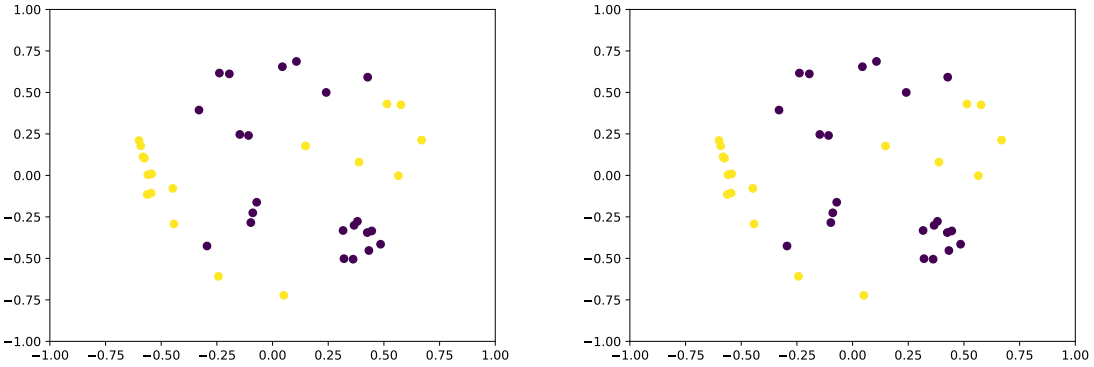
A possible solution to this consideration between computational cost and precision would be a combination of triplet loss and KTA. The first training epochs could use triplet loss to quickly adjust the parameters to an acceptable level. After this initial training, KTA could be used to slowly optimize the parameters to a perfect level. Note that this is only an idea for future work and is not tested in this thesis.

4.2.4.2. Unsupervised Cost Functions

The second type of cost functions are unsupervised cost functions, which only use unlabeled training data. As there are no class labels available, the kernel values cannot be compared to their optimal value. Instead, the kernel is used e.g. in the kernel k-means algorithm (see Section 2.1.3.2) to cluster the training dataset. These predicted clusters are then evaluated using the clustering metrics Davies-Bouldin index (DBI) (see Section 3.4.1) and Calinski-Harabasz index (CHI) (see Section 3.4.2). Optimizing these clustering metrics is therefore expected to also optimize the parameters of the used VQK.

Figure 4.15 shows the validation cost for both cost functions during the optimization on the donuts dataset. Note that with the CHI a good clustering is determined by a higher value, which means that the negative CHI has to be minimized. It can be seen that for both cost functions no real optimization is happening.

Figure 4.16 shows the kernel matrix of the training data after training with the DBI cost function (Figure 4.16a) or the CHI cost function (Figure 4.16b). It can be seen that even



(a) DBI cost function

(b) CHI cost function

Figure 4.16.: Kernel matrix of the donuts dataset after optimizing with the DBI (Figure 4.16a) and CHI (Figure 4.16b) cost functions. Even after training the two classes of the donuts dataset are still heavily intertwined, which means high kernel values for samples from opposite classes.

after training, the two classes are not separated at all. This means that the trained kernel function returns high values for samples that are from different classes.

The problem that the kernel values do not represent the structure of the data can also be seen when combining the VQK with spectral clustering to cluster the training data as visualized in Figure 4.17. Neither the DBI nor the CHI cost function adjust the parameters enough to have a noticeable influence on the clustering result.

As not even the training dataset can be clustered correctly using either cost function, the trained VQK is not be able to generalize on unseen data. With this result, both cost functions can be deemed unusable for optimization purposes in clustering scenarios. There are a few reasons as to why they do not work as intended.

Firstly, both cost functions are too insensitive to be an adequate indication for a good kernel. Both functions are defined as clustering metrics, which only change if the predicted labels change. However, a change in the predicted labels requires quite a large change in the used kernel function, which most implementations of common optimization algorithms cannot cope with.

And secondly, even if these cost functions could be optimized, neither DBI nor CHI are perfect clustering metrics for all situations. This is because there is a large number of different clustering algorithms and many of those algorithms have their own definition of what is considered a cluster [Est02b]. With these many different cluster models it is impossible to define a single metric that can accurately evaluate all of them.

4.2.4.3. Conclusion

In conclusion it can be stated that both semi-supervised cost functions work well for training a variational quantum kernel (VQK). There are advantages and disadvantages to

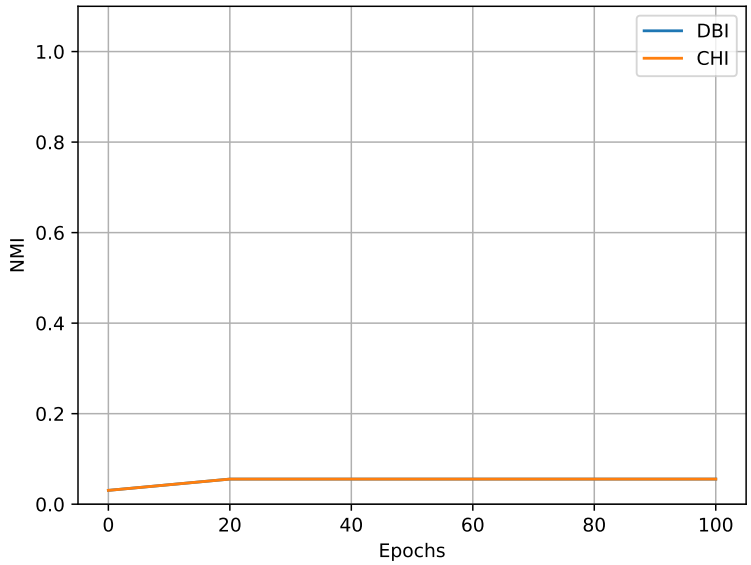


Figure 4.17.: Spectral clustering of the donut training dataset during the optimization process using the DBI and CHI cost functions. Both cost functions fail to optimize the VQK enough to cluster the training dataset correctly.

both functions, especially concerning precision and speed. However, a combination of both cost functions could potentially combine the advantages.

While the semi-supervised cost functions worked quite well, the proposed unsupervised cost functions are not usable for optimization in the current state. They are both too insensitive to be effectively used for parameter optimization. Unsupervised cost functions enable ensuing future work. One possibility could be to use an unsupervised version of KTA and triplet loss. All pairs of samples that have a kernel value above a certain threshold (for example > 0.95) could be deemed to be from the same class, as it is unlikely that their kernel values will change dramatically. With these assumed classes for the unlabeled training data, KTA or triplet loss could be used for the optimization.

All further evaluations use the KTA cost function to optimize the VQK.

4.2.5. Circuit Size

In Section 3.2.1 it was established that the VQK is based on a layered architecture, where the same ansatz is repeated multiple times. Furthermore, the used ansatz allows for a flexible scaling of the number of qubits. The number of layers and number of qubits together defines the size of the used circuit. The following two sections investigate what influence these two parameters have on the VQK and subsequent clustering algorithms.

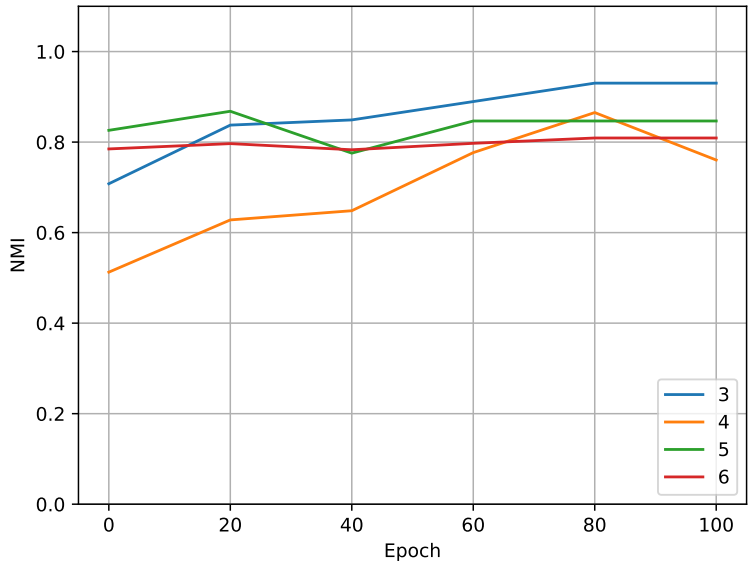


Figure 4.18.: NMI for testing VQKs with different numbers of qubits on the Iris dataset. 3 qubits lead to the best overall performance, while more qubits are worse in terms of clustering performance.

4.2.5.1. Qubits

Figure 4.18 shows the clustering result, if VQKs with different numbers of qubits are used. Note that less than 3 qubits cannot be used with the ansatz defined in Section 3.2.1 due to the structure of the CR_Z gates.

The figure shows the best result for the Iris dataset at an NMI of 0.93 with a VQK that uses 3 qubits. For circuits with increasing number of qubits the clustering performance starts to decrease. Therefore, larger circuits with more qubits do not necessarily increase the performance. This can be described by the already mentioned problem of overfitting. With more qubits, and thus more optimizable parameters for the VQK, the training data can be learned exactly. This however also captures possible noise in the training data and leads to a worse generalization of the learned model, especially for new and unseen samples. Thus, finding the correct number of qubits for a given problem is an optimization task.

However, choosing the correct number of qubits when designing a circuit is not only a question of clustering performance. The decision should also consider feasibility and execution speed. Circuits for real quantum computers are limited in the number of qubits by the amount of qubits the specific quantum computer provides. When the circuits are simulated on classical hardware, both feasibility and execution speed are problematic. Classical hardware can only simulate a limited number of qubits n, because such a simulation has to store all 2^n amplitudes to describe the quantum system. Execution speed becomes a problem, because all of these 2^n amplitudes change, when gates are applied to the qubits.

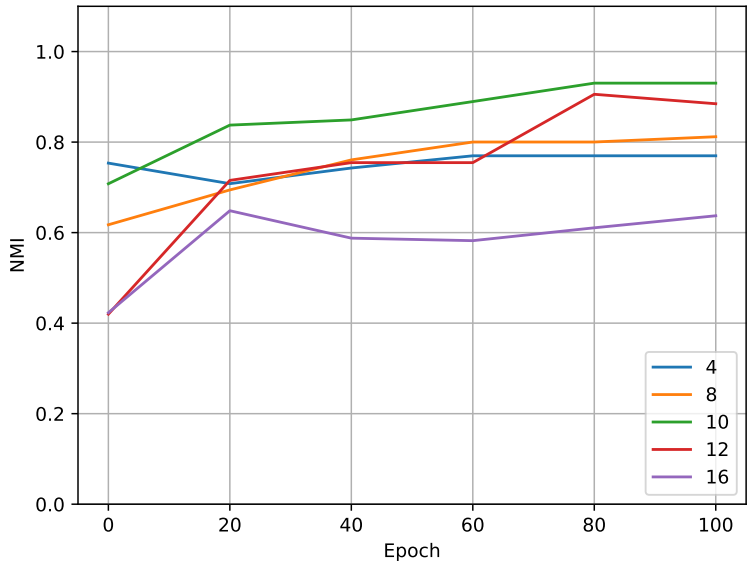


Figure 4.19.: NMI for testing VQKs with different numbers of layers on the Iris dataset. The optimal performance is achieved with 10 layers, while 4 and 8 lead to underfitting and 12 and 16 to overfitting.

Similar figures for the donuts (Figure A.5b) and moons (Figure A.5a) dataset reveal that for both datasets the optimal VQK uses 3 qubits. For more complex datasets, the VQK will likely need more qubits for a good performance.

4.2.5.2. Layers

Similar to the number of qubits, also the number of layers can be adjusted. Figure 4.19 shows the clustering result for VQKs with different numbers of layers. For 10 layers the final clustering result is the best, in the examined range of layers, at an NMI of 0.93. All other tests, with either less or more layers, produce worse results, which is interesting for two reasons.

Firstly, bad results for VQKs with more than 10 layers is again an indication of overfitting. The used circuit is considered again too complex and will fit the training data perfectly, which leads to worse performance on unseen data.

Secondly, the results for less than 10 layers suggest the problem of underfitting, where the used circuit is not large enough to capture the structure of the data. The only real solution to this problem is increasing the model size or in this case the number of layers.

Figure A.6a and Figure A.6b show the clustering results when using VQKs with different numbers of layers on the moons and donuts dataset. The figures indicate that for the donuts dataset the optimum is at 20 layers and for the moons dataset at 30 layers. For larger and more complex datasets, the number of required layers is likely larger.

Identically to the number of qubits, the correct choice of the number of layers not only depends on the clustering performance. The two main additional points to consider are

execution speed and noise. Execution speed is obviously a problem, because more layers lead to more gates that are applied to the qubits, which in turn takes longer to compute in simulations. Noise is especially a problem when using real quantum computers. As introduced in Section 2.2.7, each gate is affected by noise and applying more gates on a qubit will therefore also increase the overall noise level on said qubit.

4.2.5.3. Conclusion

In conclusion, finding the correct number of qubits and layers is a complex optimization problem. Both parameters can lead to under- or overfitting. To solve underfitting, the only real solution is to increase the circuit size in terms of the number of qubits or layers. For overfitting, there are multiple different mitigation strategies from classical machine learning that can potentially also be used for VQKs.

The evaluations in this section also show why the default parameters for the number of qubits and layers in Section 4.1 were chosen this way.

4.2.6. Training Dataset Size

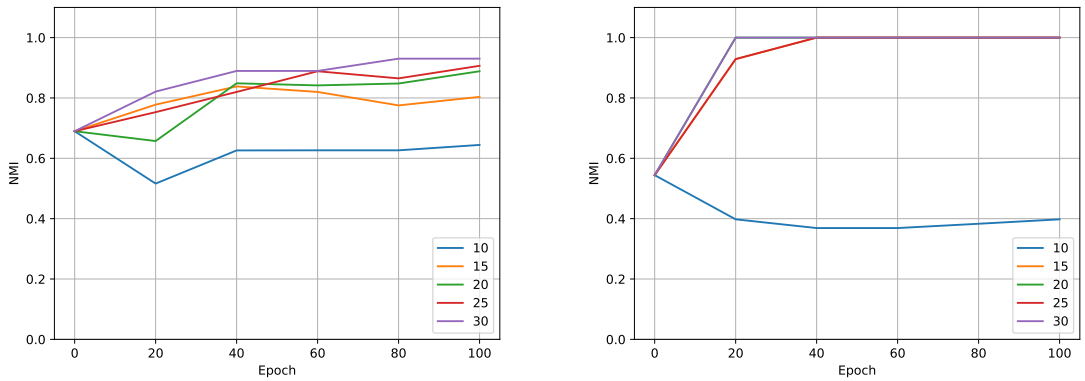
The KTA cost function was introduced as semi-supervised. Semi-supervised learning generally assumes that the size of the labeled training dataset is rather small. Therefore, this section investigates the effects of different amounts of training data on the performance of a VQK.

Figure 4.20 shows the differences in the clustering result for different sizes of training data (Iris dataset in Figure 4.20a and moons dataset in Figure 4.20b). There are a couple of interesting observations that are discussed in the following.

In the case of only 10 training samples, the performance is not increasing but rather decreasing during optimization. For the Iris dataset the clustering performance decreases from 0.69 to 0.64 and for the moons dataset from 0.54 to 0.40. While the VQK is indeed being optimized even with small training datasets, the training set is not a good representation of the overall dataset. This prohibits the trained VQK from making good predictions for previously unseen data. This behaviour is also a known issue in classical machine learning. The first possible solution is obviously increasing the size of the training dataset. However, also a better choice of training data could solve the problem. If the 10 samples would be chosen to be good representations of the real dataset, the VQK can possibly learn a good enough model to predict unseen data.

The second observation in Figure 4.20 is that more training data potentially leads to a better result on previously unseen data. For the moons dataset in Figure 4.20b the increase is fairly obvious. With only 15 training samples, the NMI of the testing set increases to 1.0, which cannot be further improved with more training data.

With the Iris dataset in Figure 4.20a the performance differences for larger training datasets are more nuanced. Generally, the figure shows that each step that increases the training dataset by 5 also increases the performance. For 10 training samples the NMI is at 0.64 and for 30 samples at 0.93. However, the performance improvements get smaller the larger the training dataset gets. For the step from 10 to 15 samples the NMI increases



(a) Iris dataset

(b) Moons dataset

Figure 4.20.: Testing results for the Iris (Figure 4.20a) and donuts (Figure 4.20b) dataset for different training dataset sizes. For the Iris dataset additional training data leads to incremental performance increases. For the moons dataset already 15 training samples achieve the optimal performance.

from 0.64 to 0.80, which equals a growth of 25%. For the step from 25 to 30 samples the performance only increases by 2%, from 0.91 to 0.93.

In summary, the size of the training dataset seems to have a big influence on the performance of the model for unseen data. However, the required size for a good performance is unknown and also depends on the specific dataset. Furthermore the performance increases get smaller, the more training data is added.

4.3. Noisy Simulation

This section investigates the effect of quantum noise (see Section 2.2.7) on the testing of VQKs. With current simulation frameworks, training the VQK under the effects of noise is extremely time intensive. Therefore this thesis focuses on investigating the influence of quantum noise only when the VQK is tested with a clustering algorithm. This means that the optimization pipeline is carried out under perfect conditions, especially without the influence of noise, while the testing pipeline is considered under noisy conditions. Following this evaluation, some possible noise mitigation techniques for VQKs are presented.

4.3.1. Noiseless Training with Noisy Testing

The evaluation in this sections builds on a combination of noiseless training and noisy testing. The parameters of the VQK are optimized under perfect conditions and without any influence from quantum noise. The VQK is combined with spectral clustering in order to test the effects on the clustering. The kernel function for this testing procedure uses the optimized parameters and is affected by the different types of quantum noise as discussed in Section 4.3.1.

Note that all of the following results were obtained by simulating the noise of a quantum computer on a classical computer. Therefore, the results are only as precise as the simulation of real quantum effects. Testing VQKs on real quantum hardware is a topic for future research.

Sampling noise is the type of noise originating from the statistical properties of the quantum measurement. Measuring a real quantum computer only returns either 0 or 1 and the probabilities are obtained as the expectation value over multiple shots.

Figure 4.21 shows the testing results on the Iris dataset for VQKs with different numbers of shots. The result for 0 shots was obtained by the analytical mode and will be used as the baseline comparison. The figure allows the interesting observation that no matter how many shots are used, the results after 100 epochs of training are always identical to the perfect result of the baseline. This can be explained by the differences in the kernel matrices between the baseline (Figure 4.22a) and the simulation with 50 shots (Figure 4.22b). The simulation with 50 shots does add noise to the kernel matrix, due to the approximation of the expectation value. Specifically, the mean difference between the baseline kernel matrix and the one with 50 shots is at 0.10 ± 0.006. While this seems like a huge difference for kernel values that are in the range $[0, 1]$, it does not have an effect on the subsequent clustering. This is due to the usage of the optimized parameters, which maximize the dissimilarity between the three classes.

A second interesting insight can be seen when comparing the kernel matrices for simulations with different numbers of shots. While the mean difference from the baseline kernel matrix for 50 shots is at 0.10 ± 0.006, the mean difference for 150 shots is only at 0.05 ± 0.001. This details the rather obvious idea that increasing the number of shots also increases the precision of the expectation value and therefore the quantum kernel values.

Coherent noise was introduced as miscalibrated gates that lead to operations that differ by a unitary from the intended operation. The predominant example of this are rotation gates that do not rotate by ϕ, but rather by $\phi + \epsilon$, with an error rate of ϵ.

Figure 4.23 shows the effect on the clustering result, when all x, y and z rotations are affected by coherent noise with an error term of $\epsilon = 0.01 \cdot 2 \cdot \pi$. In comparison to this, the target simulation is not affected by any noise. Starting at epoch 40, the performance of the noise affected VQK differs from the target performance by a constant. The target simulation uses the optimized parameters θ, while the noisy simulation uses an offset from the optimal parameters $\theta + \epsilon$. Therefore the noisy simulation constantly rotates further than intended, which leads to the constant difference in performance.

The fact that the noisy simulation performs better at epoch 0 can be explained by the parameter initialization. The target simulation uses randomly initialized parameters θ_0, while the parameters for the noisy simulation offsets these to $\theta_0 + \epsilon$. In this case the noisy parameters are just a better starting point due to the randomness of the initialization.

Incoherent noise is the fact that real quantum hardware constantly interact with its environment, which leads to imprecisions in the state of the quantum computer. There are numerous possible effects, which are explaine in Section 2.2.7.

Figure 4.24 shows the testing result for simulations affected by the different types of incoherent noise compared to the target simulation without noise. All considered noise variants use a noise probability of 1%.

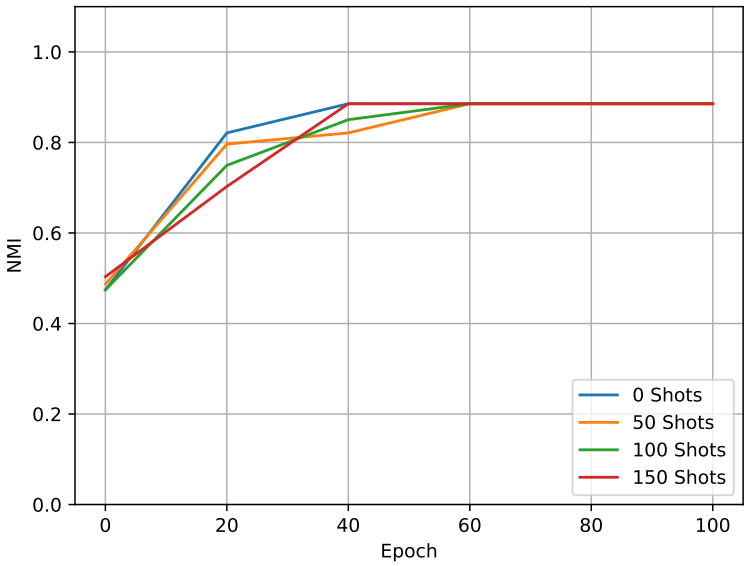
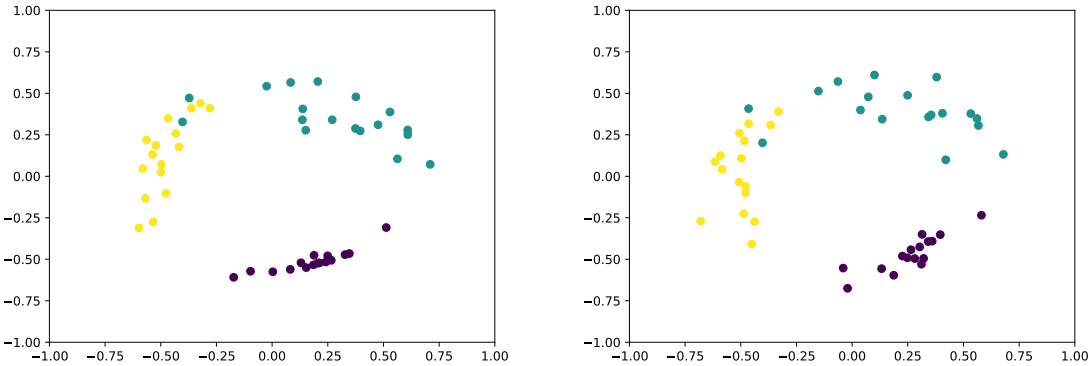


Figure 4.21.: Testing result for simulations with different numbers of shots. All simulations lead to the optimal result due to the otimization of the parameters.



(a) 0 Shots

(b) 50 Shots

Figure 4.22.: Kernel matrices for 0 shots (Figure 4.22a) and 50 shots (Figure 4.22b) after 100 epochs of training. Both kernel matrices separate the three classes. The simulation with 50 shots however adds some noise to the kernel values.

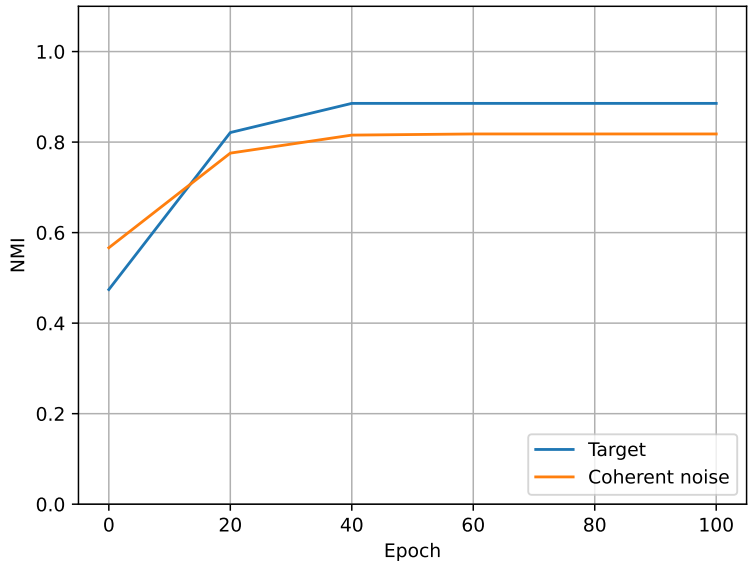


Figure 4.23.: Testing result for simulations with and without the influence of coherent noise. The coherent noise leads to a constant decrease in performance compared to the target simulation.

Phase and amplitude damping are the two noise variants that decrease the clustering performance the least. This is because, as their name suggest, they only damp the phase and amplitude of the quantum states.

In turn both bit-flip and depolarizing noise lead to a clustering performance near an NMI of 0. This leads to the conclusion that both noise variants destroy the optimized performance of the VQK. The bit-flip noise has such a big effect, because even with the small probability of 1%, if such a bit-flip occurs on only 3 qubits, the loss of information in the overall quantum state is quite large. Equally, if the depolarizing error occurs, all information of the quantum state is lost.

4.3.2. Noise Correction

There are many proposed solutions that could be used to reduce the negative effects of quantum noise on VQKs, some of which are explained in this section.

In case of sampling noise, there is no possible way of mitigation other than increasing the number of shots that are used. However, this will also increase the execution time.

The impact of coherent noise can potentially be reduced by the training of the VQK. If the coherent noise factor is constant over time, the training can still learn the target function, as the coherent noise is an offeset to the parameters that can be learned. If the noise factor changes over time, is a bit more complicated as the offset constantly changes. This means that the performance will decrease every time the coherent noise changes and the learning algorithm has to adapt to the changes.

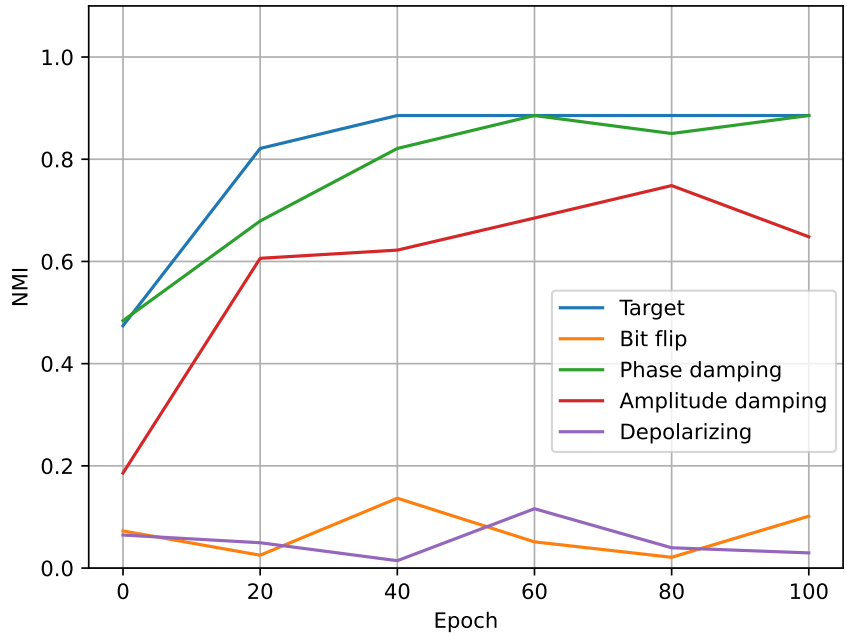


Figure 4.24.: Testing results of simulations affected by different types of incoherent noise compared to a noiseless simulation. Amplitude and phase damping lead to a slight decrease in performance, while both bit-flip as well as depolarizing noise demolish the performance of the optimized parameters.

For incoherent noise there is a wide range of different error correction codes that can be used to make qubits more resilient against the effects of noise [NC10]. There are however also ways of using the special properties of quantum kernel functions to mitigate these effects. A kernel value of a sample with itself has to be equal to 1 at all times ($k(x_i, x_i) = 1$). In general, when computing $k(x_i, x_i)$ on a quantum computer the result will differ from 1 due to the effects of noise. The difference between the real value and 1 can be used to estimate the overall influence of noise on the circuit. With this estimate all other kernel values can be postprocessed accordingly [Hub+21].

4.4. Discussion

This section summarizes the evaluations and subsequent findings of this chapter.

Firstly, the general feasibility of combining a VQK with a clustering algorithm was investigated. It was shown that training and optimizing a VQK has positive effects on the clustering performance. These effects were shown for multiple different clustering algorithms including kernel k-means, spectral clustering, DBSCAN and hierarchical clustering. In comparison to the classical RBF kernel function, the VQK was shown to be advantageous in most cases with performance increases from about 10% to 100%. Only in some instances the RBF kernel resulted in a better performance of about 4%.

The following evaluations considered the combination with different clustering algorithms. Especially kernel-based algorithms, namely kernel k-means and spectral clustering,

were shown to be easily usable with the VQK, as no further parameters have to be optimized. Furthermore, also the distance-based algorithms, DBSCAN and hierarchical clustering, were shown to be working by computing a distance measure from the VQK. However, both distance-based algorithms use further parameters that act as thresholds on the distance metric and have to be optimized in order to achieve the best possible result. In summary, the VQK can be used for various different clustering algorithms that use the kernel function directly or indirectly, through a distance measure.

The proposed semi-supervised and unsupervised cost functions were evaluated next. Both KTA and triplet loss were shown to be able to optimize the VQK. The difference between the two being mainly precision and speed. KTA is more precise as it compares more kernel values in each step, which is also time and compute intensive. Triplet loss in turn compares only a small set of kernel values, which makes the optimization faster. However, both cost functions also lead to problems with overfitting. It was further shown that the two unsupervised cost functions, DBI and CHI, cannot optimize the VQK to have a positive effect on the subsequent clustering. There are two main reasons for this result. Firstly, both are inherently clustering metrics that evaluate whether or not a given clustering fits a specific underlying model of a cluster. The problem is that the notion of a cluster is not well defined, thus the two cost functions try to optimize to a specific cluster model although it does not necessarily fit to the dataset. Secondly, even if the cluster models fit, both measures are extremely insensitive, as their values only change if there are large changes in the kernel function.

Additionally, the size of the VQK circuit, described by the number of qubits and layers, has a big impact on the performance. Underfitting could be the result of a circuit that is too small, where the VQK may not be able to learn the underlying structure of the data. Furthermore, the opposite problem of overfitting is also possible. If the circuit is too big and has too many parameters, the VQK exactly learns the training data, including all possible noise. Both under- and overfitting lead to a decreased performance on unseen data.

The experiments concerning the size of the training dataset have shown that more training data can lead to a better model and therefore better clustering results. However, it was also noted that the performance increases get smaller the more training data is provided.

As training a VQK with simulated quantum noise is currently extremely time intensive, some experiments were conducted to study the effects of quantum noise on the testing pipeline for VQKs. The experiments showed that especially incoherent noise can lead to dramatic decreases in clustering performance. Nevertheless, different mitigation techniques for general quantum circuits and specifically quantum kernels exist, to reduce the impact of quantum noise.

In summary, this thesis is concerned with investigating the usability of a variational quantum kernel (VQK) in combination with classical clustering algorithms. To show the feasibility of this approach two concepts are required. Firstly, it is necessary that there is a way of optimizing the VQK. This thesis provided multiple different cost functions, of which KTA and triplet loss were shown to be usable for training the VQK. The second requirement is that training the VQK should also lead to an increased performance in the combined classical clustering algorithm. This was shown to be adequate for multiple

different kernel-based and distance-based clustering algorithms. Furthermore it was shown that the trained VQK outperforms the classical radial basis function (RBF) kernel function.

5. Conclusion

Quantum machine learning (QML) is a promising new research field combining the theories from classical machine learning with the power of quantum computers.

Motivated by recent work introducing variational quantum kernels (VQKs) for supervised learning like the support vector machine (SVM), this thesis investigates whether VQKs can also be utilized in unsupervised learning algorithms, especially clustering. The questions arise if such an approach can be used computationally efficient and potentially increase the clustering accuracy compared to classical kernel methods. However, solely training a VQK as an unsupervised learning task poses the issue of optimizing parameters only with unlabeled training data. Therefore, this thesis proposes to train the VQK in an unsupervised as well as semi-supervised fashion. To evaluate this approach, a two-step pipeline was introduced. The first step of said pipeline is focused on training the VQK with either unlabeled or labeled training data. The second step tests the trained VQK by combining it with different kernel- and distance-based clustering algorithms.

The methods for unsupervised learning proposed in this thesis use the quality of a clustering to determine the loss of a given VQK. However, this approach of unsupervised learning was shown to have some crucial flaws. Firstly, the two well-known clustering metrics Davies-Bouldin index (DBI) and Calinski-Harabasz index (CHI) were shown to be too insensitive to small changes of the kernel values. Secondly, using clustering metrics for the optimization process leads to an extreme overhead of used computational resources, as every step of the optimization requires a full clustering of the training data. Overall, this thesis deems the proposed clustering metrics unusable for training a VQK, but the task of unsupervised learning of VQKs poses interesting challenges for future work.

A possible alternative to bypass these problems with unsupervised learning is moving to a semi-supervised approach, which provides labeled training data. Consequently, this thesis evaluated the kernel target alignment (KTA) cost function in comparison to a new way of training VQKs using the triplet loss in a hybrid quantum-classical pipeline. As both functions operate directly on the kernel values, the insensitivity and complexity of the unsupervised approaches are no longer a problem. The evaluation showed that both functions can be used to train VQKs, which outperform the purely classical RBF kernel on the chosen toy and real-world datasets. While KTA leads to a slightly better accuracy on the clustering, there are serious differences concerning the computational complexity and usability on current noisy intermediate scale quantum (NISQ) era quantum computers, as triplet loss requires less queries of the quantum computer. Future work could investigate merging the two proposed approaches leading to a possible combination of accuracy and computational complexity.

Furthermore this thesis also investigated the influence of different hyperparameters, especially the size of the chosen circuit. It was shown that both, the number of qubits as well as the number of layers, has a big impact on the performance of the VQK and an

inappropriate selection can lead to over- and underfitting. The applicability of different classical ideas of mitigating over- and underfitting is a possible topic fur future research. While there is no general consensus in the quantum computing literature about what constitutes a good circuit ansatz for a given problem, this thesis has detailed some important characteristics when designing an ansatz specifically for quantum kernels.

Summarizing, this thesis suggests that VQKs, trained in a semi-supervised fashion with either KTA or triplet loss, can be combined with different classical clustering algorithms and even increase their accuracy. While the proposed and evaluated approach is semi-supervised, this thesis also lays the ground work for moving to unsupervised training of VQKs. Due to the complexity of both training and testing a VQK, the current results rely on quantum simulations and their different variants of real quantum noise to study the execution of algorithms under reproducible, well-defined conditions. This creates the foundation for future work to explore both algorithmic improvements for using VQKs in clustering, as well as practical improvements for executing such algorithms on current NISQ and future quantum computing architectures.

Bibliography

[ABG06] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. "Machine Learning in a Quantum World". In: *Advances in Artificial Intelligence*. Ed. by Luc Lamontagne and Mario Marchand. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 431–442. ISBN: 978-3-540-34630-2.

[ANI+21] MD SAJID ANIS et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2021. DOI: 10.5281/zenodo.2573505.

[Ank+99] Mihael Ankerst et al. "OPTICS: Ordering Points To Identify the Clustering Structure". In: ACM Press, 1999, pp. 49–60.

[Ber+20] Ville Bergholm et al. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*. 2020. arXiv: 1811.04968 [quant-ph].

[Blo] Mathieu Blondel. *Source code for kernel k-means*. https://gist.github.com/mblondel/6230787. Accessed: 22.04.2022.

[Blo46] F. Bloch. "Nuclear Induction". In: *Phys. Rev.* 70 (7-8 Oct. 1946), pp. 460–474. DOI: 10.1103/PhysRev.70.460. URL: https://link.aps.org/doi/10.1103/PhysRev.70.460.

[Bou+04] N. Boulant et al. "Incoherent noise and quantum information processing". In: *The Journal of Chemical Physics* 121.7 (Aug. 2004), pp. 2955–2961. DOI: 10.1063/1.1773161. URL: https://doi.org/10.1063%2F1.1773161.

[Buh+01] Harry Buhrman et al. "Quantum Fingerprinting". In: *Physical Review Letters* 87.16 (Sept. 2001). ISSN: 1079-7114. DOI: 10.1103/physrevlett.87.167902. URL: http://dx.doi.org/10.1103/PhysRevLett.87.167902.

[Che+10] Gal Chechik et al. "Large Scale Online Learning of Image Similarity Through Ranking". In: *J. Mach. Learn. Res.* 11 (Mar. 2010), pp. 1109–1135. ISSN: 1532-4435.

[Che95] Yizong Cheng. "Mean shift, mode seeking, and clustering". In: *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE* (1995).

[CJ74] Tadeusz Caliński and Harabasz JA. "A Dendrite Method for Cluster Analysis". In: *Communications in Statistics - Theory and Methods* 3 (Jan. 1974), pp. 1–27. DOI: 10.1080/03610927408827101.

[CMR12] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. "Algorithms for Learning Kernels Based on Centered Alignment". In: (2012). DOI: 10.48550/ARXIV.1203.0550. URL: https://arxiv.org/abs/1203.0550.

[Cri+06] Nello Cristianini et al. "On Kernel Target Alignment". In: *Innovations in Machine Learning: Theory and Applications*. Ed. by Dawn E. Holmes and Lakhmi C. Jain. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 205–256. ISBN: 978-3-540-33486-6. DOI: 10.1007/3-540-33486-6_8. URL: https://doi.org/10.1007/3-540-33486-6_8.

[DB79] David L. Davies and Donald W. Bouldin. "A Cluster Separation Measure". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1.2 (1979), pp. 224–227. DOI: 10.1109/TPAMI.1979.4766909.

[DGK04] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. "Kernel K-Means: Spectral Clustering and Normalized Cuts". In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '04. Seattle, WA, USA: Association for Computing Machinery, 2004, pp. 551–556. ISBN: 1581138881. DOI: 10.1145/1014052.1014118. URL: https://doi.org/10.1145/1014052.1014118.

[Dir39] P. A. M. Dirac. "A new notation for quantum mechanics". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 35.3 (1939), pp. 416–418. DOI: 10.1017/S0305004100021162.

[DTB16] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. "Quantum-Enhanced Machine Learning". In: *Physical Review Letters* 117.13 (Sept. 2016). DOI: 10.1103/physrevlett.117.130501. URL: https://doi.org/10.1103%2Fphysrevlett.117.130501.

[Est+96] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: AAAI Press, 1996, pp. 226–231.

[Est02a] Vladimir Estivill-Castro. "Why so Many Clustering Algorithms: A Position Paper". In: *SIGKDD Explor. Newsl.* 4.1 (June 2002), pp. 65–75. ISSN: 1931-0145. DOI: 10.1145/568574.568575. URL: https://doi.org/10.1145/568574.568575.

[Est02b] Vladimir Estivill-Castro. "Why so Many Clustering Algorithms: A Position Paper". In: *SIGKDD Explor. Newsl.* 4.1 (June 2002), pp. 65–75. ISSN: 1931-0145. DOI: 10.1145/568574.568575. URL: https://doi.org/10.1145/568574.568575.

[FIS36] R. A. FISHER. "THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS". In: *Annals of Eugenics* 7.2 (1936), pp. 179–188. DOI: https://doi.org/10.1111/j.1469-1809.1936.tb02137.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-1809.1936.tb02137.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x.

[GLV04] Yann Guermeur, Alain Lifchitz, and Régis Vert. "A Kernel for Protein Secondary Structure Prediction". In: *Kernel Methods in Computational Biology*. Ed. by Bernhard Schölkopf, Koji Tsuda, and Jean-Philippe Vert. Chap. 9 - ISBN 0-262-19509-7. http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=10338&mode=toc.

The MIT Press, Cambridge, Massachussetts, 2004, pp. 193–206. URL: https://hal.archives-ouvertes.fr/hal-00012701.

[Har+20] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[HSS08] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. "Kernel methods in machine learning". In: *The Annals of Statistics* 36.3 (2008), pp. 1171–1220. DOI: 10.1214/009053607000000677. URL: https://doi.org/10.1214/009053607000000677.

[Hub+21] Thomas Hubregtsen et al. *Training Quantum Embedding Kernels on Near-Term Quantum Computers*. 2021. DOI: 10.48550/ARXIV.2105.02276. URL: https://arxiv.org/abs/2105.02276.

[Hun07] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[KB15] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2015).

[Ker+18] Iordanis Kerenidis et al. "q-means: A quantum algorithm for unsupervised machine learning". In: (2018). DOI: 10.48550/ARXIV.1812.03584. URL: https://arxiv.org/abs/1812.03584.

[Kra+83] Karl Kraus et al., eds. *States, Effects, and Operations Fundamental Notions of Quantum Theory: Lectures in Mathematical Physics at the University of Texas at Austin*. Springer Berlin Heidelberg, 1983. ISBN: 978-3-540-38725-1. DOI: 10.1007/3-540-12732-1. URL: https://doi.org/10.1007/3-540-12732-1.

[Kri+11] Hans-Peter Kriegel et al. "Density-based clustering". In: *WIREs Data Mining and Knowledge Discovery* 1.3 (2011), pp. 231–240. DOI: https://doi.org/10.1002/widm.30. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.30. URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.30.

[Lem12] Claude Lemaréchal. *Cauchy and the Gradient Method*. 2012.

[Llo+20] Seth Lloyd et al. *Quantum embeddings for machine learning*. 2020. DOI: 10.48550/ARXIV.2001.03622. URL: https://arxiv.org/abs/2001.03622.

[LST10] A. I. Lvovsky, B. C. Sanders, and W. Tittel. "Optical quantum memory". In: (2010). DOI: 10.48550/ARXIV.1002.4659. URL: https://arxiv.org/abs/1002.4659.

[Lux07] Ulrike von Luxburg. "A Tutorial on Spectral Clustering". In: (2007). DOI: 10.48550/ARXIV.0711.0189. URL: https://arxiv.org/abs/0711.0189.

[Mac67] J. Macqueen. "Some methods for classification and analysis of multivariate observations". In: *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*. 1967, pp. 281–297.

[McC+16] Jarrod R McClean et al. "The theory of variational hybrid quantum-classical algorithms". In: *New Journal of Physics* 18.2 (Feb. 2016), p. 023023. ISSN: 1367-2630. DOI: 10.1088/1367-2630/18/2/023023. URL: http://dx.doi.org/10.1088/1367-2630/18/2/023023.

[MRT12] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2012. ISBN: 9780262018258. URL: https://books.google.de/books?id=maz6AQAAQBAJ.

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667.

[Neu27] J. von Neumann. "Wahrscheinlichkeitstheoretischer Aufbau der Quantenmechanik". In: *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse* 1927 (1927), pp. 245–272. URL: http://eudml.org/doc/59230.

[Nie16] Frank Nielsen. "Hierarchical Clustering". In: Feb. 2016, pp. 195–211. ISBN: 978-3-319-21902-8. DOI: 10.1007/978-3-319-21903-5_8.

[Pat+16] Raj B. Patel et al. "A quantum Fredkin gate". In: *Science Advances* 2.3 (2016), e1501531. DOI: 10.1126/sciadv.1501531. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.1501531. URL: https://www.science.org/doi/abs/10.1126/sciadv.1501531.

[PBP20] Daniel K. Park, Carsten Blank, and Francesco Petruccione. "The theory of the quantum kernel-based binary classifier". In: *Physics Letters A* 384.21 (July 2020), p. 126422. ISSN: 0375-9601. DOI: 10.1016/j.physleta.2020.126422. URL: http://dx.doi.org/10.1016/j.physleta.2020.126422.

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[Pér+20] Adrián Pérez-Salinas et al. "Data re-uploading for a universal quantum classifier". In: *Quantum* 4 (Feb. 2020), p. 226. DOI: 10.22331/q-2020-02-06-226. URL: https://doi.org/10.22331%2Fq-2020-02-06-226.

[Pre18] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (Aug. 2018), p. 79. DOI: 10.22331/q-2018-08-06-79. URL: https://doi.org/10.22331%2Fq-2018-08-06-79.

[Sch+17] Erich Schubert et al. "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN". In: *ACM Trans. Database Syst.* 42.3 (July 2017). ISSN: 0362-5915. DOI: 10.1145/3068335. URL: https://doi.org/10.1145/3068335.

[Sch21] Maria Schuld. *Supervised quantum machine learning models are kernel methods*. 2021. DOI: 10.48550/ARXIV.2101.11020. URL: https://arxiv.org/abs/2101.11020.

[SG02] Alexander Strehl and Joydeep Ghosh. "Cluster Ensembles - A Knowledge Reuse Framework for Combining Multiple Partitions". In: *Journal of Machine Learning Research* 3 (Jan. 2002), pp. 583–617. DOI: 10.1162/153244303321897735.

[SK19] Maria Schuld and Nathan Killoran. “Quantum Machine Learning in Feature Hilbert Spaces”. In: *Physical Review Letters* 122.4 (Feb. 2019). ISSN: 1079-7114. DOI: 10.1103/physrevlett.122.040504. URL: http://dx.doi.org/10.1103/PhysRevLett.122.040504.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A unified embedding for face recognition and clustering”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015. DOI: 10.1109/cvpr.2015.7298682. URL: https://doi.org/10.1109%2Fcvpr.2015.7298682.

[SP18] Maria Schuld and Francesco Petruccione. *Supervised Learning with Quantum Computers*. Springer, 2018.

[Tak+21] Maika Takita et al. *IBM quantum breaks the 100-qubit processor barrier*. Feb. 2021. URL: https://research.ibm.com/blog/127-qubit-quantum-processor-eagle.

[VD09] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[VEB10] Nguyen Xuan Vinh, Julien Epps, and James Bailey. “Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance”. In: *J. Mach. Learn. Res.* 11 (Dec. 2010), pp. 2837–2854. ISSN: 1532-4435.

[VTS04] J.P. Vert, Koji Tsuda, and Bernhard Schölkopf. “A Primer on Kernel Methods”. In: *Kernel Methods in Computational Biology, 35-70 (2004)* (Jan. 2004).

[WKS22] Christof Wendenius, Eileen Kuehn, and Achim Streit. “Training Parameterized Quantum Circuits with Triplet Loss”. unpublished. 2022.

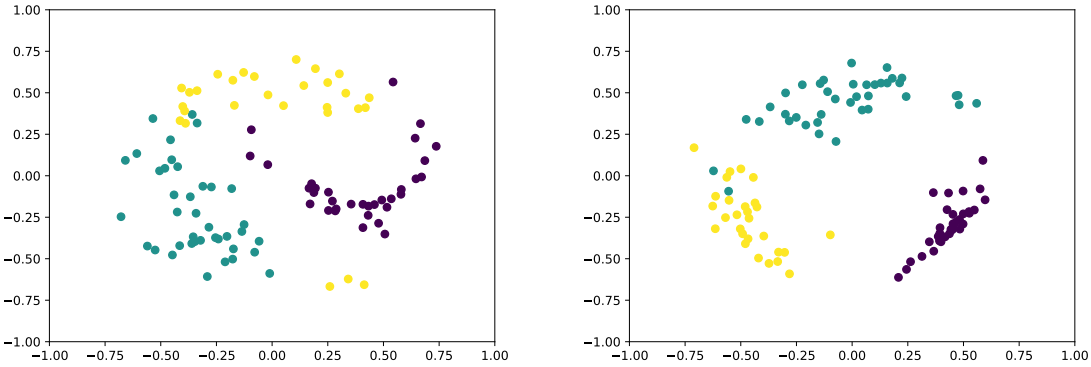
[Wu+] Chieh Wu et al. *KernelNet Implementation*. https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/neu-spiral/kernel_net. Accessed: 22.04.2022. Original github repository not available anymore.

[Wu+19] Chieh Wu et al. *Deep Kernel Learning for Clustering*. 2019. DOI: 10.48550/ARXIV.1908.03515. URL: https://arxiv.org/abs/1908.03515.

[WZT12] Tinghua Wang, Dongyan Zhao, and Shengfeng Tian. “An overview of kernel alignment and its applications”. In: *Artificial Intelligence Review* 43 (Feb. 2012). DOI: 10.1007/s10462-012-9369-4.

[YRC07] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. “On early stopping in gradient descent learning”. In: *Constr. Approx* (2007), pp. 289–315.

A. Appendix



(a) Before training

(b) After training

Figure A.1.: Kernel matrix of the Iris dataset before (Figure A.1a) and after (Figure A.1b) training with the KTA. The three classes are initially intertwined and mostly separated after the optimization.

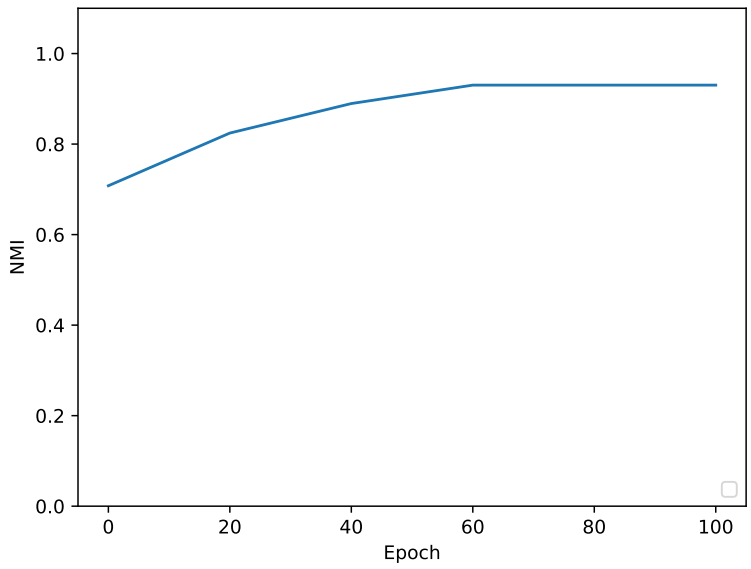
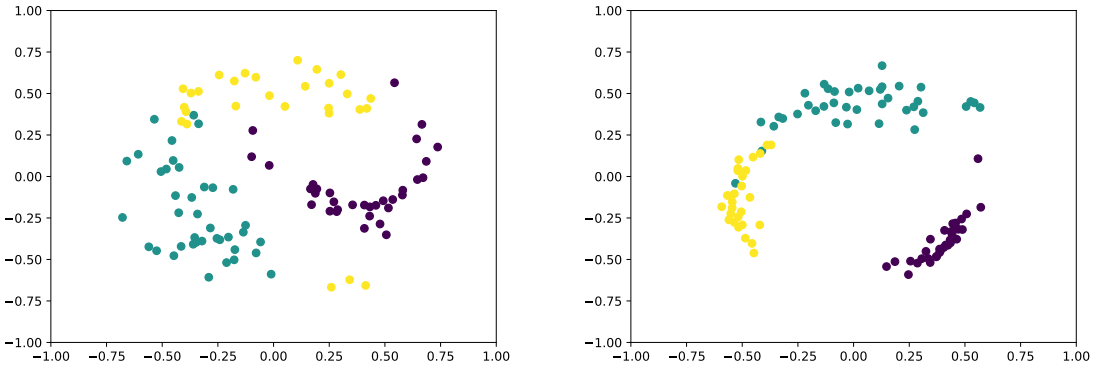


Figure A.2.: Testing result during training on the Iris dataset with KTA. KTA can also be used to train a VQK for a dataset with more than two classes.



(a) Before training

(b) After training

Figure A.3.: Kernel matrix of the Iris dataset before (Figure A.3a) and after (Figure A.3b) training with the triplet loss cost function. The initial kernel matrix shows similarities for samples from opposite classes. After the optimization the tree classes are separated and only samples from the same class appear similar.

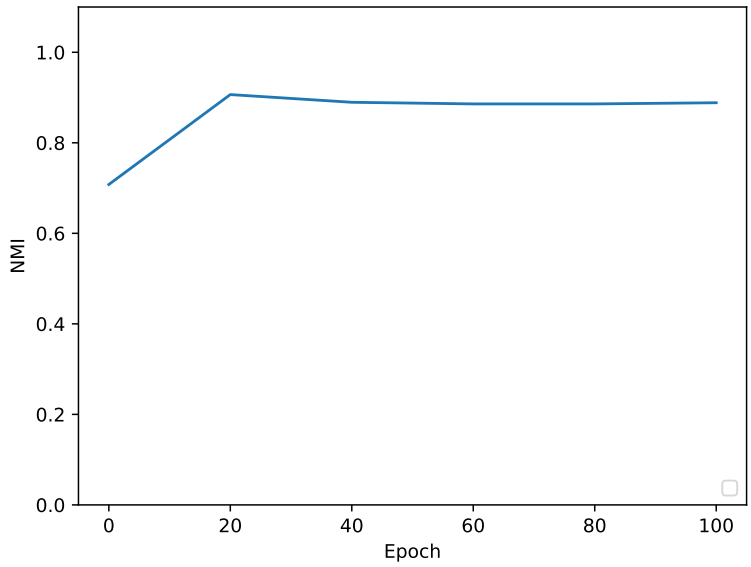
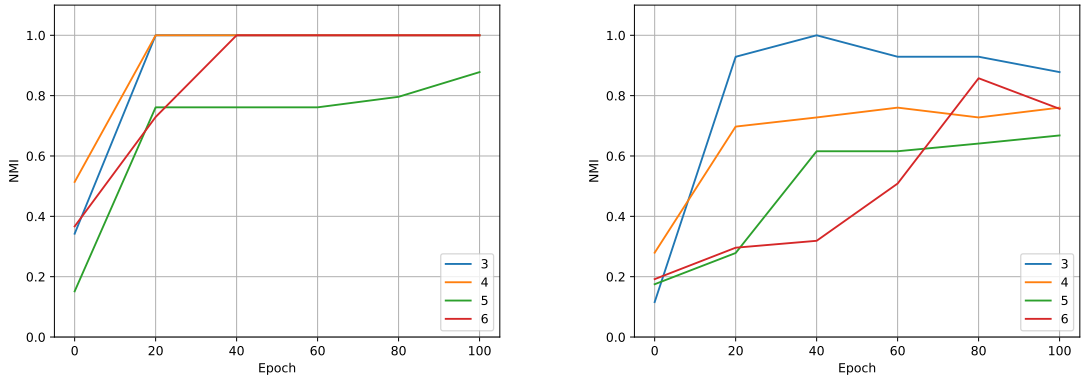
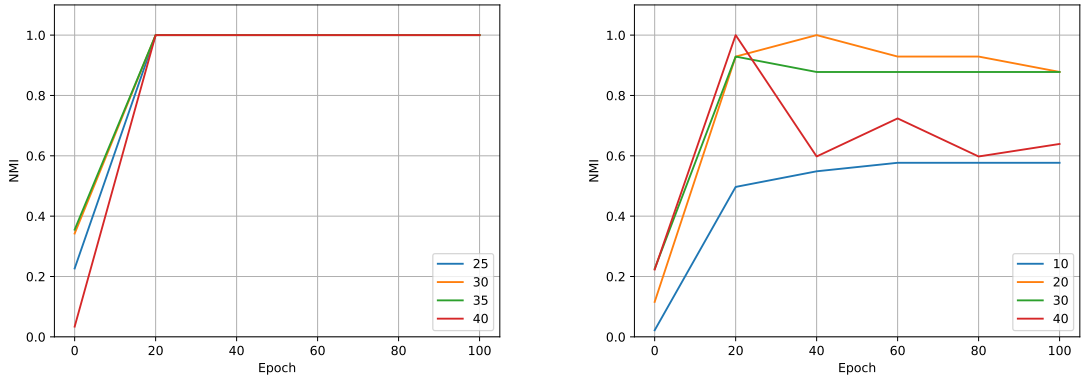


Figure A.4.: Testing result during training on the Iris dataset with triplet loss. The triplet loss cost function can also be used to train a VQK for a dataset with more than two classes.



(a) Moons dataset (b) Donuts dataset

Figure A.5.: Testing result for the moons (Figure A.5a) and donuts (Figure A.5b) dataset for VQKs with different numbers of qubits. For both datasets 3 qubits is the best combination of clustering performance and speed. For the moons dataset, also 4 and 6 qubits lead to good results, but less qubits are generally favorable.



(a) Moons dataset (b) Donuts dataset

Figure A.6.: Testing result for the moons (Figure A.6a) and donuts (Figure A.6b) dataset for VQKs with different numbers of layers. For the moons dataset all of the considered options lead to an optimal performance. For the donuts dataset the optimum is at 20 layers, with 10 layers leading to underfitting and 40 layers to overfitting.

Acronyms

CHI Calinski-Harabasz index.

DBI Davies-Bouldin index.

DBSCAN density-based spatial clustering of applications with noise.

GD gradient descent.

KTA kernel target alignment.

ML machine learning.

NISQ noisy intermediate scale quantum.

NMI normalized mutual information.

NN neural network.

QC quantum computing.

QML quantum machine learning.

qubit quantum bit.

RBF radial basis function.

SVM support vector machine.

VQC variational quantum circuit.

VQK variational quantum kernel.

Glossary

CNOT	Controlled NOT gate.
CR_X	Controlled R_X gate.
CR_Y	Controlled R_Y gate.
CR_Z	Controlled R_Z gate.
CSWAP	Controlled SWAP gate.
H	Hadamard gate.
I	Identity gate.
R_X	R_X gate.
R_Y	R_Y gate.
R_Z	R_Z gate.
SWAP	SWAP gate.
X	Pauli X gate.
Y	Pauli Y gate.
Z	Pauli Z gate.