Comparing Quantum Neural Networks and Quantum Support Vector Machines

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Comparing Quantum Neural Networks and Quantum Support Vector Machines

Master Thesis

Arne Thomsen

September 6, 2021

Advisors:

Dr. David Sutter
Amira Abbas
Dr. Stefan Woerner

IBM Quantum, IBM Research Zurich, Switzerland

Prof. Dr. Renato Renner

Institute for Theoretical Physics, ETH Zurich, Switzerland

Abstract

In the emerging field of quantum machine learning, algorithms are studied which leverage both classical data and quantum computation, inter alia for supervised binary classification. Examples are *quantum support vector machines* (QSVMs) and *quantum neural networks* (QNNs). In this thesis, the relation between these is assessed through theoretical investigations and numerical experiments. As first derived in [1], the classification function employed with QNNs can be rearranged to resemble that of QSVMs; both models in fact define a linear decision boundary in a nonlinearly accessed quantum feature space. Accordingly, heuristic QNNs can be seen as approximations to theoretically tractable QSVMs and the former motivated by the latter. Another point of comparison between these models is their computational complexity in the size of the training set. Central to this analysis is the fact that quantum expectation values can fundamentally only be determined approximately for any finite number of measurement shots. In this work, a polynomial improvement over the result reported in [2] is derived for fitting noisy QSVMs via the dual optimization problem. The Pegasos algorithm [3] is introduced here for the first time to train QSVMs, and under an empirically motivated assumption, a bound independent of the size of the training set is newly established analytically. Numerically, the same independence is found for QNNs. For large training sets, it follows that, despite the speedup, dual trained QSVMs are not viable, while QNNs and Pegasos trained QSVMs can be deployed. Heuristic extensions of the two models with learned feature maps result in classifiers, which show promise both from a theoretical as well as a practical point of view.

Contents

List of Symbols

Spaces

$\mathbb{R}^s$	Space of the classical input data
$\mathcal{H}$	Generic Hilbert space
$\mathcal{S}(2^q)$	Hilbert space of density matrices spanned by q qubits
$\mathcal{U}(2^q)$	Space of unitaries on q qubits

Notation

$\mathbf{x}$	Math boldface for vectors in $\mathbb{R}^s$
$\vec{w}$	Overhead arrow for vectors in $\mathcal{S}(2^q)$
X	Capital math mode letter for random variables

Norms

$\|\cdot\|$	Euclidean norm for real vectors
$\|\cdot\|_2$	Operator norm that is induced by the Euclidean norm

Other Symbols

$\mathcal{E}$	Feature map quantum circuit
$\mathcal{L}$	Loss function
$\mathcal{W}$	Variational form quantum circuit
κ, k	Classical and quantum kernel function

θ	Trainable parameters in a variational form or feature map
φ, ψ	Classical and quantum feature map
c, $\tilde{c}$	Ground truth and modeled classification function
d	Number of trainable parameters in a variational form
M, m	Size of the classical train and test set
q	Number of qubits
R	Number of measurement shots per expectation value
S	Number of support vectors
T, E	Classical train and test set
y	Class membership label in $\{-1, +1\}$

Chapter 1

Introduction

Machine learning is a vibrant field at the intersection of computer science, artificial intelligence and statistics, which studies algorithms that improve their performance by the use of data. In this way, tasks previously solely associated with human intelligence like natural language processing, image classification, medical diagnosis, and many more, can be tackled with computers. Recent successes include beating the world champion in the game of Go [4] and achieving record performance for protein folding [5]. Together with the steadily increasing availability of data, these algorithms hold the promise to solve even more problems in the future.

On an abstract level, machine learning algorithms are grouped into supervised-, unsupervised- and reinforcement learning schemes [6]. Throughout this thesis, the *supervised* setting is considered exclusively. In that approach, the goal is to find a function, which captures a relation that is not explicitly known, by learning from example input-output pairs [6, 7, 8]. In case the output is categorical, like for example whether a given image depicts a dog or a cat, the task is denoted as *classification*.

Quantum information processing promises new directions for the future of computation, as it has been shown that there are problems for which quantum algorithms achieve substantial speedups over classical algorithms [9, 10, 11]. The underlying reason why quantum computation might solve problems more efficiently than classically possible is that it follows a different set of elementary rules, namely those of quantum mechanics.

Now, the young and growing field of *quantum machine learning* (QML) aims to combine these two novel paradigms by simultaneously harnessing data and quantum physics. While all combinations of classical or quantum data and classical or quantum computation fall under the umbrella term QML, in this thesis the discussion is restricted to problems where *quantum computation* is employed to handle *classical data*. For an introduction to the field, see [12].

Throughout this work, the focus is on the two quantum machine learning models *quantum support vector machine* [1, 13] and *quantum neural network* [1, 14], which share a deep mathematical connection.

1.1 Summary of This Work

In this section, a summary of the contents of this thesis is given. The results are organized into two complementary parts, namely Chapter 2 on theory and Chapter 3 on numerical experiments. The aim of the discussion in Chapter 2 is to introduce the quantum machine learning models, derive how they relate and analyze how they are affected by the introduction of statistical uncertainty. The goal of Chapter 3 is twofold. Firstly, the numerical experiments conducted give an empirical justification for some of the statements merely conjectured in the theoretical treatment. Secondly, concrete numerical examples are presented to make the theory more tangible and emphasize some practically relevant implications it poses.

The treatment of theory commences in Section 2.1 with a formal introduction to the machine learning task considered throughout, *supervised binary classification* of classical data. Then, the classical machine learning model *support vector machine* (SVM) [15, 16, 6, Chapter 7, 8, Chapter 5] is introduced. First, as a linear classifier in Section 2.2.1, then as a nonlinear method in Section 2.2.2. In these sections, the focus lies on the different primal and dual optimization problems, which the hard- and soft-margin variants of the classical SVM pose. Transitioning between the two sections, nonlinearity is introduced to the models by the application of a *feature map*, which takes the classical input vectors from their original data space to some Hilbert space denoted the *feature space*. Then, the modified SVM acts in feature space analogously to how it operates in the data space for the original variant and determines a linear decision boundary. However, as the feature map is nonlinear, the hyperplane in feature space corresponds to a nonlinear boundary in data space. A key result for such SVMs is that the dual optimization problem and the classification function can be expressed solely in terms of inner products between feature vectors. Denoting the inner product as the *kernel function* and writing the model in terms of it is then called applying the *kernel trick*. Thereafter, it is evident that feature vectors do not have to be explicitly calculated to employ the models. As a result, only the kernel function has to be tractable and feature space does not need to be directly accessible, such that classical SVMs can operate in infinite dimensional feature spaces.

As a next step, *quantum support vector machines* (QSVMs) [1, 13] are presented in Section 2.2.3. Like classical SVMs, these models come with theoretical guarantees because the optimization problem their training poses is convex. In a specific setting and based on standard complexity theoretic assumptions, an

exponential speedup over any classical machine learning algorithm is proven in [2] and extended in [17]. Taking a step back, a QSVM is a special case of a kernelized SVM, where the feature map assigns the classical data to vectors in the space of density matrices. Concretely, the embedding of the real vectors as density matrices is realized by parameterized quantum circuits. Acting on a previously set state like the all zero state, the parameters in the circuit are fixed by the input data. An example is a *rotation encoding*, where the components of the input vector specify the angles in Pauli rotation gates. The quantum kernel function is then defined in terms of the Hilbert-Schmidt inner product and its evaluation corresponds to determining a quantum expectation value. This has two important implications. Firstly, the only part of the QSVM algorithm that makes use of quantum computation are the quantum kernel evaluations. The optimization itself is handled on a classical computer just like for a classical SVM. Secondly, even on an ideal, noise free quantum computer, the quantum expectation values are not directly accessible by measurement. Instead, Born's rule dictates that they can only be approximated by repeated measurement of definite outcomes. As in practice, the number of measurements that can be performed is always finite, thus, the exact expectation values are never obtained. So in this way, statistical uncertainty is invariably introduced into the algorithm and robustness with respect to it has to be established. This is a fundamental difference between classical kernels and quantum kernels, as only the latter are inherently noisy. Combining previous results from the literature [2, 18] in a novel way, for M the size of the training set and ε the absolute difference between the trained ideal and noisy classification function, it is shown analytically that $R_{\text{train}} = \mathcal{O}\left(M^{4.67}/\varepsilon^2\right)$ quantum circuit evaluations are sufficient to fit a QSVM via the dual optimization problem. This establishes an improvement over the result $R_{\text{train}} = \mathcal{O}\left(M^6/\varepsilon^2\right)$ reported in [2] for the same setting. For context, in the case of an exact classical kernel, $\mathcal{O}\left(M^2\right)$ evaluations are needed for training. Evaluating the quantum kernel function is also required for prediction, and in that setting the result $R_{\text{pred}} = \mathcal{O}\left(S^2/\varepsilon^2\right)$ for $S < M$ the number of *support vectors* is obtained under the assumption that training has completed and its outcome is considered fixed.

Training a QSVM by solving the quadratic program the dual optimization poses is not the only option. In Section 2.2.4, the Pegasos algorithm [3] is introduced, which is a form of stochastic sub-gradient descent. For a δ-accurate approximation to the primal optimization objective, the algorithm takes $\mathcal{O}\left(M/\delta\right)$ evaluations of a classical, noise free kernel. But as previously noted, quantum kernels fundamentally comprise statistical uncertainty. By taking this into account, under an empirically justified assumption, a derivation shows that $R_{\text{train}} = \mathcal{O}\left(M^2/\delta^3\right)$ quantum circuit evaluations are necessary to run the algorithm for a QSVM. Furthermore, the result $R_{\text{train}} = \mathcal{O}\left(1/\delta^5\right)$ is obtained by considering an alternative bound in the derivation, such that the minimum of these complexities holds. Both results are novel and mark a

polynomial speedup over what is found for the QSVM trained via the dual, and the second scaling law is even independent of M. With this, the promise of Pegasos is clear. It enables employment of QSVMs for training set sizes M that are otherwise practically intractable. In accordance with the previous result, the complexity of prediction after training has completed is given by $R_{\text{pred}} = \mathcal{O}\left(S^2/\varepsilon^2\right)$ for these models.

Section 2.3 takes a step away from QSVMs and introduces *quantum neural networks* (QNNs) [1, 14], a type of heuristic quantum machine learning model also capable of binary classification. By definition, QNNs employ a variational quantum circuit that can be divided into two parts, namely a feature map that takes on the same form as for a QSVM and a *variational form* sub-circuit. While all of the parameters in the former are fixed by data, the parameters in the latter are trainable. Classification is performed by obtaining the expectation value of a previously fixed observable with respect to the quantum state, which results from applying the QNN circuit to a predetermined state (for example the all zero state). During training, the variational parameters in the quantum circuit are updated by a classical computer (for example running a gradient descent algorithm), with the goal of minimizing a priorly defined loss function. So similarly to the QSVM, the quantum computation utilized in the model is confined to the evaluation of expectation values and optimization is done classically.

Like for QSVMs, the expectation values in QNNs can also only be determined up to some statistical uncertainty. Analyzed from this point of view, for d the number of parameters in the variational form, the scaling law for training $R_{\text{train}} = \mathcal{O}\left(d/\varepsilon^n\right)$ is motivated and $n = 3$ conjectured. Note that this bound is independent of M. Furthermore, the result $R_{\text{pred}} = \mathcal{O}\left(1/\varepsilon^2\right)$ is derived for the number of quantum circuit evaluations necessary for prediction. Here, ε takes the same role as before and denotes the difference between the noisy and ideal classification function. A recap of the results on computational complexity for the different models is included in Table 1.1.

The other central insight on QNNs is that they are in fact closely related to QSVMs. Following [1], in Section 2.3.1 it is shown that the decision function employed by QNNs can be recast to resemble that of QSVMs. Then, it is apparent that the variational form solely implements a linear decision boundary in the feature space accessed by the quantum feature map. With this, QNNs are kernel methods in the sense of and in agreement with [21]. But while the hyperplane in feature space which a QSVM finds comes with the guarantee to maximize an inter class distance (called the margin), the hyperplanes accessible to QNN classifiers are heuristic and confined to what the variational form can express. Whereas this establishes only the linearity in some feature spaces, in a subsequent step it is derived that when they incorporate identical feature map quantum circuits, QNNs and QSVMs in fact operate in identical feature spaces.

	QSVM (dual)	QSVM (PEGASOS)	QNN
approach	provable	provable	heuristic
optimality	global	global	local
training	$\mathcal{O}\left(M^{4.67}/\varepsilon^2\right)$	$\mathcal{O}\left(\min\left[M^2/\delta^3,\, 1/\delta^5\right]\right)$	$\mathcal{O}\left(d/\varepsilon^n\right)$
prediction	$\mathcal{O}\left(S^2/\varepsilon^2\right)$	$\mathcal{O}\left(S^2/\varepsilon^2\right)$	$\mathcal{O}\left(1/\varepsilon^2\right)$

Table 1.1: Summary of the results in Sections 2.2.3, 2.2.4 and 2.3 regarding how the number of physical quantum circuit evaluations scales for different QML models and optimization algorithms. M is the size of the training set, S the number of support vectors defined by the QSVM and d the number of trainable parameters in the variational form of the QNN. The value $n = 3$ is conjectured in Section 2.3. The quantity ε is defined like in (2.67) and bounds the difference between the ideal and noisy classification function with probability $p > \frac{1}{2}$. Alternatively, for the PEGASOS trained QSVM and the QNN, ε is also equal to the standard deviation of the term inside the sign function in the classification function and the statements can be converted by making use of Chebyshev's inequality (see (2.79)). The accuracy ε is distinct from δ, which is defined in (2.94) as an additive error on the optimization objective. The relation between ε and δ is an open question. As an aside, by employing the quantum amplitude estimation algorithm [19, 20] on fault tolerant hardware, speedups for the QSVM complexities can be achieved. Finally, note that these theoretical results correspond to the worst case and how the models perform in practice for the average case remains an open question.

So overall, it is implied that QNNs can be interpreted as approximations to QSVMs. In that case, they at least partly inherit justifications from QSVMs, like quantum speedups. For a concrete approximation, the QNNs have to be trained with respect to a specific objective function, which is derived in the final part of the section.

In Section 2.4, quantum machine learning models built on top of the QSVM and QNN models analyzed up to that point are introduced and briefly characterized. As an extension of QSVMs, where the circuit parameters in the feature map are exclusively dependent on the input data, trainable parameters are included into the feature map in Section 2.4.1. Then, in addition to the original QSVM optimization problem, a suitable choice for these parameters also has to be found. One method employed to this end is *quantum kernel alignment* [17], where an outer optimization loop is added. Alternatively, the variational kernel can be optimized directly by the *kernel target alignment* procedure suggested in [18]. For both approaches, there is no guarantee for optimality with respect to the variational parameters in the feature map and they are therefore heuristic. However, in any case the trainable parameters afford the models additional flexibility over the standard QSVMs.

In a similar spirit, in Section 2.4.2 the *quantum alternating network* (QAN) (for related models, see [22, 23, 24]) architecture is suggested as an extension of QNNs. There, the restriction that the variational parameters only occur after the feature map is lifted and instead, the familiar feature map- and

variational form sub-circuits are alternated throughout the whole quantum circuit. In such an architecture, only the final variational layer implements a linear decision boundary in feature space, while all prior variational layers together parameterize the feature map. From this, it is clear that compared to QNNs, the parameters in QANs enter in a more complex, nonlinear fashion, which affords additional flexibility in fitting data. For the heuristic QANs, optimization is handled analogously to the QNNs with gradient descent.

The chapter on numerical experiments starts out with Section 3.1 highlighting the central importance of the feature map for both QSVMs and QNNs. First, a comparison of classifiers employing different feature map circuits is presented for two dimensional toy data. In addition to showing what kind of decision boundaries the models implement in practice, these examples illustrate the theoretical result that a perfect classification accuracy on a given training set can only be achieved when the two classes are linearly separable in feature space. From this, it is apparent that the feature map, which determines where the data lie in feature space, has the power to limit the performance of both models. To show how including trainable parameters into the feature map circuit adds versatility, example implementations of kernel alignment and QANs are presented.

As illustrated with numerical examples in Section 3.2, QNN performance can also be confined by the variational form, as it determines which hyperplanes in feature space can be realized. So even when the classes are linearly separable in feature space and a QSVM can therefore achieve perfect accuracy on the training set, the variational form employed in the QNN ought to be expressive enough to parameterize such a hyperplane in order to achieve the same. Furthermore, for the case when the QNN is limited by the feature map instead, it is shown that adding depth to the variational form does not improve the result the model achieves.

In Section 3.3, the effect of introducing statistical uncertainty on the level of the quantum expectation values is explored empirically for QSVM and QNN models. In both cases, the training data are artificially generated to ensure control over the classification accuracy the models can ideally achieve.

For QSVMs, the Pegasos trained variant is considered exclusively and the results are collected in Section 3.3.1. It is found that the performance of the algorithm is largely unaffected when statistical noise modeling $R = 100$ measurement shots is included. Because in that case, the loss incurred and accuracy achieved on the training set do not differ from the noiseless runs in a statistically significant way. This observation gives a justification for an assumption made in the theoretical treatment of Pegasos training in Section 2.2.4. Furthermore, it motivates another experiment where only a single measurement shot is considered for every evaluation of the quantum kernel function. It is observed that even then, the Pegasos algorithm achieves

much better classification accuracy than random guessing. For a fixed total number of measurement shots, this implies an interesting trade-off between the number of measurements performed per expectation value and the number of training steps taken. As another result, it is shown that in the experimental setting under consideration, the algorithm does not scale with the size of the dataset M, which is in line with the theoretical result in Table 1.1. For all experiments conducted, the number of evaluations of the quantum kernel function in the Pegasos runs is orders of magnitude smaller than what is necessary to train QSVMs via the dual.

Section 3.3.2 covers QNN training when two sources of stochasticity are included. Firstly, the aforementioned finite sampling statistics underlying all calculations of quantum expectation values are modeled. For analytical gradients based on parameter shift rules, the QNN training is found to be reasonably robust to this. A second source of uncertainty comes from employing *stochastic gradient descent* (SGD), where the objective function is only evaluated on a random subset of the training set at every step of the algorithm. The experiments show that for an otherwise identical optimizer, SGD converges significantly faster than non-stochastic gradient descent as a function of the number of quantum expectation values employed. Furthermore, for the experimental setting under consideration, the convergence of QNN training is found to be independent of the size of the dataset M, corroborating the result in Table 1.1.

Chapter 2

Theory

This chapter lays the theoretical foundation to understand the two primary machine learning models considered in this thesis, namely *quantum support vector machines* and *quantum neural networks*. Apart from introducing the mathematical structures, the focus is on relating these models.

2.1 Binary Classification

The sole machine learning task considered in this thesis is *supervised binary classification*. In this setting, some *training set* containing data

$$T = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_M\} \quad \text{with} \quad \mathbf{x}_i \in \mathbb{R}^s, \tag{2.1}$$

and corresponding labels determining membership in one of two classes

$$L = \{y_1, y_2, ..., y_M\} \quad \text{with} \quad y_i \in \{-1, +1\}, \tag{2.2}$$

are given. It is assumed that the matching elements in T and L are independent and identically distributed (i.i.d.) samples from a joint probability distribution $P(X, Y)$ with support in $\mathbb{R}^s \times \{-1, +1\}$, which is unknown. The goal is then to accurately predict class membership on a separate *test set*

$$E = \{\mathbf{x}'_1, \mathbf{x}'_2 ..., \mathbf{x}'_m\}, \tag{2.3}$$

where the labels are not known during the learning procedure but assumed to come from the same distribution $P(X, Y)$.

More formally, and in line with the discussion in [1], the task can be framed as a search for some approximate classification function

$$\begin{aligned} \tilde{c}\colon T \cup E &\to \{-1, +1\} \\ \mathbf{x} &\mapsto y \end{aligned} \tag{2.4}$$

that agrees well with the inaccessible ground truth map

$$\begin{aligned} c\colon T \cup E &\to \{-1, +1\} \\ \mathbf{x} &\mapsto \begin{cases} -1 & \text{if } P(\mathbf{x}, -1) > P(\mathbf{x}, +1) \\ +1 & \text{otherwise} \end{cases}, \end{aligned} \tag{2.5}$$

determining all true labels, given only T and the restriction

$$c_{|T}\colon T \to \{-1, +1\}. \tag{2.6}$$

One way to quantify the agreement between the solution found $\tilde{c}$ and the ground truth c, is the average classification accuracy with respect to the test set

$$\text{accuracy}_E = \frac{|\{\mathbf{x} \in E | c(\mathbf{x}) = \tilde{c}(\mathbf{x})\}|}{|E|}. \tag{2.7}$$

In the case where the model not just predicts the binary class membership, but attaches a probability rating to it, the logarithmic or cross entropy loss (see (2.116) on page 35) evaluated on E is another example.

As a side note, a model able to perform binary classification can always be extended to provide classification of any number of classes by considering a *one vs. all* or *one vs. rest* strategy [6, p. 182], so restricting the discussion to binary classification does not imply a loss of generality.

2.2 Support Vector Machines

The *support vector machine* (SVM) is a supervised classical machine learning model first introduced by Vapnik [15, 16], that is broadly applicable to relevant machine learning tasks and simultaneously lends itself to theoretical analysis.

2.2.1 Linear SVMs

In its most basic form, an SVM is a linear model characterized by a specific training procedure which has the geometric interpretation of maximizing an inter-class distance called the margin. In the following, the SVMs are applied to binary classification problems, as defined in Section 2.1.

A dataset is called *linearly separable* if there exists some $\mathbf{w} \in \mathbb{R}^s$ and $b \in \mathbb{R}$, such that data belonging to the two different classes $\{-1, +1\}$ lie on opposite sides of the hyperplane defined by

$$\mathbf{w}^\top \mathbf{x} + b = 0. \tag{2.8}$$

From this, the classification function of a linear SVM is constructed as

$$\tilde{c}_{\text{SVM}}(\mathbf{x}) \coloneqq \text{sign}\left[\mathbf{w}^\top \mathbf{x} + b\right], \tag{2.9}$$

and assigns class membership according to which side of the hyperplane (2.8) the $\mathbf{x}$ falls on. So far, this also holds true for other linear classification models like perceptrons. What sets SVMs apart is the training procedure employed to find $\mathbf{w}$ and b. The following is largely based on [6, Chapter 6, 1, Supplementary Information, 2, Appendix C].

Hard Margin

In the hard margin case, an SVM searches for the (unique up to scaling) linearly separating hyperplane that maximizes the margin, which is defined as the perpendicular distance between two hyperplanes of equal orientation separating the two datasets. See Fig. 2.1 for an illustration. It can be shown that the

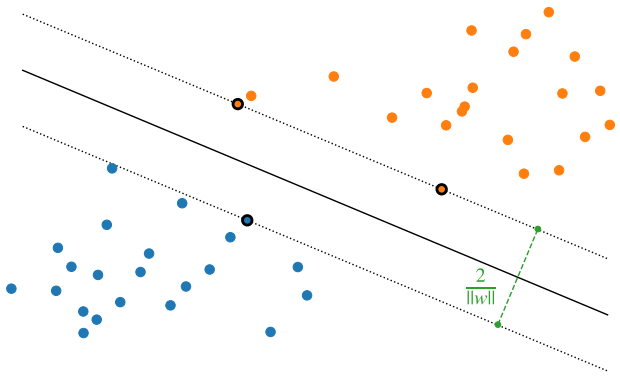


Figure 2.1: Result of training a linear hard margin SVM on linearly separable data, where orange and blue dots correspond to the two classes. The solid black line marks the decision boundary of the SVM. The margin is defined as the perpendicular distance between the dashed lines, which intersect the three support vectors marked with black circles.

size of the margin is equal to $2/\|\mathbf{w}\|$ and that the constraint $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1$ ensures that the data only lie outside of the margin. Putting this together, the primal SVM optimization problem which searches for the maximal margin can be rendered as the convex optimization

$$\begin{aligned} \underset{\mathbf{w} \in \mathbb{R}^s, b \in \mathbb{R}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \ \forall i, \end{aligned} \tag{2.10}$$

where $i \in \{1, 2, ..., M\}$ runs over the whole training set T.

Soft Margin

One way to treat data that are not linearly separable is by introducing slack variables $\xi_i \in \mathbb{R}^+$ that relax the constraints in (2.10) and simultaneously adding

a term that penalizes them. The resulting primal soft margin optimization problem remains convex and is then defined as

$$\begin{aligned} \underset{\mathbf{w} \in \mathbb{R}^s,\, b, \xi_i \in \mathbb{R}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_i \xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \;\forall i, \\ & \xi_i \geq 0 \qquad \forall i, \end{aligned} \tag{2.11}$$

where again $i \in \{1, 2, ..., M\}$ and a regularization parameter C is introduced that determines the trade-off between the size of the margin and points that make use of the slack because they are either misclassified or lie within the margin. See Fig. 2.2 for an example. For $C \to \infty$, the hard margin SVM is recovered.

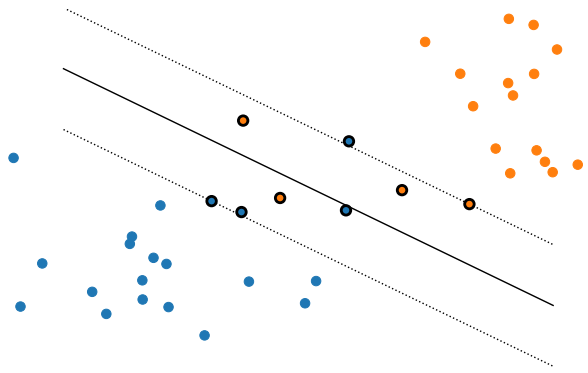


Figure 2.2: The soft margin SVM problem has a feasible solution even if the data are not linearly separable. The support vectors marked by black circles are all $\mathbf{x}_i \in T$ that are either at the edge of the margin (like in the hard margin case), inside the margin, or misclassified. Here, the regularization parameter is $C = 1$.

As a side note, (2.11) can be rewritten by observing that the two cases

$$\begin{aligned} y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 &\implies \xi_i = 0, \\ y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 1 &\implies \xi_i = 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b), \end{aligned} \tag{2.12}$$

together imply that

$$\xi_i = \max\Big[0, 1 - y_i(\mathbf{w}^\top \mathbf{x} + b)\Big] = \mathcal{L}_{\text{hinge}}(y_i, \mathbf{w}^\top \mathbf{x} + b). \tag{2.13}$$

This leads to the equivalent *hinge loss* formulation of the soft margin SVM

$$\underset{\mathbf{w} \in \mathbb{R}^s, b \in \mathbb{R}}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{M} \max\Big[0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\Big]. \tag{2.14}$$

2.2.2 Kernelized SVMs

The SVM formulations considered so far implement the decision function (2.9), which is linear in the input data $\mathbf{x}$. To adapt the SVM to be able to fit data without a linear structure, a nonlinear *feature map*

$$\phi : \mathbb{R}^s \to \mathcal{H}, \tag{2.15}$$

is introduced, which maps from the original data space to some, typically higher- and possibly infinite dimensional Hilbert space $\mathcal{H}$. This results in a modification to the classification function

$$\tilde{c}_{\text{SVM}}(\mathbf{x}) = \text{sign}\left[\langle \vec{w}, \phi(\mathbf{x})\rangle_{\mathcal{H}} + b\right], \tag{2.16}$$

where $\vec{w} \in \mathcal{H}$ and $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ is the inner product in $\mathcal{H}$. Here, the notation $\vec{w}$ for vectors in the Hilbert/feature space is established, in contrast to $\mathbf{w}$ denoting vectors in the original data space $\mathbb{R}^s$. While this classification function continues to be linear in feature space, it results in a nonlinear decision function in the original data space. In principle, the primal optimization problems in Section 2.2.1 can now be employed to find the optimal $\vec{w}$ in feature space.

However, this is not what is usually done. Instead of working in the feature space directly, which would entail the possibly inefficient calculation of explicit feature vectors $\phi(\mathbf{x})$, the dual problem of the optimization in (2.11) in feature space for $i, j \in \{1, 2, ..., M\}$

$$\begin{aligned} \underset{\alpha_i \in \mathbb{R}}{\text{maximize}} \quad & \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \left\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\right\rangle_{\mathcal{H}} \\ \text{subject to} \quad & 0 \le \alpha_i \le C \quad \forall i, \\ & 0 = \sum_i \alpha_i y_i \end{aligned} \tag{2.17}$$

is considered, which can be derived analogously to the dual of the ℓ^2-SVM discussed in a later section (see (2.50) to (2.58)). The optimization problem remains convex. The data $\mathbf{x}_i \in T$ where $\alpha_i \neq 0$ after training has completed are called the *support vectors*. The solution to the dual can be related to the solution to the primal optimization via

$$\vec{w} = \sum_i \alpha_i y_i \phi(\mathbf{x}_i), \tag{2.18}$$

which is the stationary condition of the Lagrangian of the primal with respect to $\vec{w}$. This leads to the alternative decision function in terms of the dual variables

$$\tilde{c}_{\text{SVM}}(\hat{\mathbf{x}}) = \text{sign}\left[\sum_{i=1}^{M} \alpha_i y_i \left\langle \phi(\mathbf{x}_i), \phi(\hat{\mathbf{x}})\right\rangle_{\mathcal{H}} + b\right]. \tag{2.19}$$

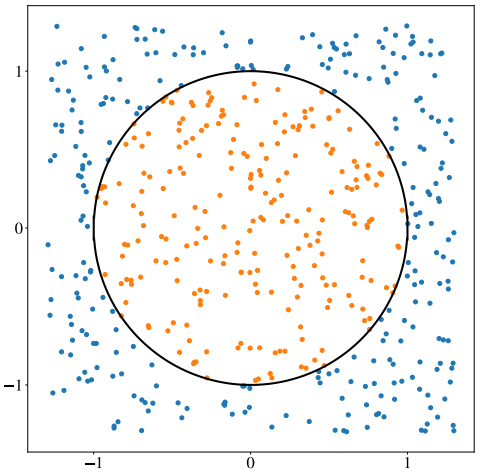
From (2.17) and (2.19) it is apparent that for the dual, both the optimization and the decision do not explicitly depend on any feature vectors $\phi(\mathbf{x})$, but only on inner products $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}}$ between them.

Therefore, the so called *kernel trick* can be applied: All occurrences of the inner product in $\mathcal{H}$ are replaced by evaluations of the *kernel function*

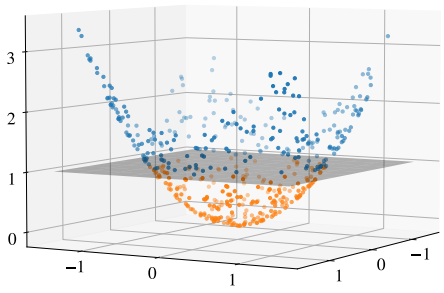
$$\kappa(\mathbf{x}, \mathbf{x}') \coloneqq \langle \phi(\mathbf{x}'), \phi(\mathbf{x}) \rangle_{\mathcal{H}}. \tag{2.20}$$

In this way, explicit calculation of feature vectors can be entirely avoided. This means that to efficiently employ an SVM in some feature space $\mathcal{H}$ accessed by ϕ, only the evaluation of the kernel function needs to be tractable.

Figure 2.3 gives an example for how data that are not linearly separable in the original space can become linearly separable in a feature space resulting from an appropriate nonlinear transformation.



(a) Original space: The nonlinear decision boundary is illustrated as the black line.



(b) Feature space: The linear decision boundary is illustrated as the black plane.

Figure 2.3: In this example, the ground truth determining the labels is $c(x) = \mathbf{1}_{\|\mathbf{x}\|<1}$ and the data lie in $\mathbb{R}^2$. In the three dimensional feature space accessed by the feature map $\phi(\mathbf{x}) = (x_1, x_2, x_1^2 + x_2^2)$, the two classes become linearly separable.

Implicit Bias Term There are cases when it is desirable to be able to drop the bias term b in the classification functions (2.16) or (2.19), while still implicitly including it. For a finite dimensional feature vector $\phi(\mathbf{x})$, this can be achieved by adding an additional component taking on a constant value like

$$\hat{\phi}(\mathbf{x}) \coloneqq (\phi(\mathbf{x}), c). \tag{2.21}$$

While this slightly changes the objective in (2.11) because the implicit bias shows up in the $\|\cdot\|^2$ term and the explicit bias does not, the contribution can be made arbitrarily small by increasing c. For the kernel, the implicit bias manifests itself as an additive term

$$\hat{\kappa}(\mathbf{x}, \mathbf{x}') \coloneqq \kappa(\mathbf{x}, \mathbf{x}') + c^2. \tag{2.22}$$

2.2.3 Quantum SVMs

The SVM variants considered so far are all entirely classical. In this section, a quantum machine learning model utilizing quantum resources to classify classical data is introduced, the *quantum support vector machine* (QSVM). Quantum computation enters the picture through the nonlinear feature map

$$\begin{aligned} \psi \colon \mathbb{R}^s &\to \mathcal{S}(2^q) \\ \mathbf{x} &\mapsto |\psi(\mathbf{x})\rangle\langle\psi(\mathbf{x})| \,, \end{aligned} \tag{2.23}$$

which accesses the exponentially large space of density matrices $\mathcal{S}(2^q)$ spanned by q qubits as the feature space. The kernel function is then defined with respect to the Hilbert-Schmidt inner product of matrices and given by

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= \operatorname{tr}\left[|\psi(\mathbf{x}')\rangle\langle\psi(\mathbf{x}')| \; |\psi(\mathbf{x})\rangle\langle\psi(\mathbf{x})|\right] \\ &= |\, \langle\psi(\mathbf{x}')|\psi(\mathbf{x})\rangle \,|^2, \end{aligned} \tag{2.24}$$

which can be simplified to the second line as the states are pure. It has been first observed in [1] that even when only considering pure states, $\mathcal{S}(2^q)$ is the correct feature space to work in, instead of the space of kets $\mathcal{H}(2^q)$, as the former choice ensures that physically meaningless global phases are canceled.

From the above definition, it follows that for a QSVM, only the evaluation of the quantum kernel in (2.24) has to be performed on a quantum computer and the training procedure in (2.17) can be performed classically. Since this limits the complexity of the required quantum circuits to the complexity of the feature map, QSVMs are especially suited to run on noisy devices.

Concretely, the feature map in (2.23) is implemented via a family of quantum circuits $\mathcal{E}(\mathbf{x}) \in \mathcal{U}(2^q)$ whose members are fixed by classical input vectors $\mathbf{x}$. Acting on the all zero state of the computational basis, the state

$$|\psi(\mathbf{x})\rangle = \mathcal{E}(\mathbf{x}) \, |0\rangle \tag{2.25}$$

is prepared. With this, the quantum kernel becomes

$$k(\mathbf{x}, \mathbf{x}') = |\, \langle 0| \, \mathcal{E}(\mathbf{x}')^\dagger \mathcal{E}(\mathbf{x}) \, |0\rangle \,|^2, \tag{2.26}$$

which can be approximated via *quantum kernel estimation*.[1] To this end, the circuit depicted in Fig. 2.4 is prepared and measured in the computational

[1] Alternatively, one can employ a SWAP test to measure the overlap in (2.26). This trades circuit depth for circuit width, as a SWAP test requires double the number of qubits compared to the adjoint method in the main text, but only applies $\mathcal{E}(\mathbf{x})$ and $\mathcal{E}(\mathbf{x}')$ separately instead of $\mathcal{E}(\mathbf{x}')^\dagger \mathcal{E}(\mathbf{x})$.

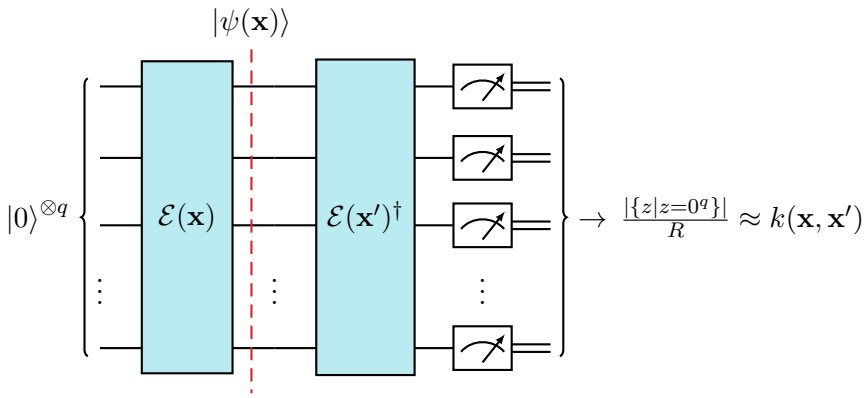


Figure 2.4: Quantum kernel estimation [1, 2, Algorithm 1 in Appendix C]: The quantum circuit is set up to prepare the state $\mathcal{E}(\mathbf{x}')^\dagger \mathcal{E}(\mathbf{x}) |0\rangle$. Then, measurement of all qubits in the computational basis is performed resulting in a bit string $z \in \{0,1\}^q$. This process is repeated for R measurement shots. The frequency of the all zero outcome then approximately determines the kernel value $k(\mathbf{x}, \mathbf{x}')$ in (2.26).

basis, resulting in a bit string $z \in \{0,1\}^q$. Then, the fraction of the all zero outcome is equal to (2.26) in the limit of an infinite number of measurement shots. The effect, which the finite accuracy of this estimation has on the overall QSVM, is explored in the later section on *finite sampling statistics*.

Example Feature Maps

The following examples for quantum feature maps, which are frequently used in the literature, and the classical kernels they correspond to, are adapted from [21, Section IV. A.].

For an *amplitude encoding*, the input dimension is fixed to $s = 2^q$ and the data are assumed to be normalized like $\|\mathbf{x}\|^2 = 1$. With this, the components of the input vector can directly define the amplitudes of the computational basis states such that

$$|\mathbf{x}\rangle = \sum_{i=0}^{2^q-1} x_i |i\rangle , \tag{2.27}$$

leading to the identity feature map

$$\psi \colon \mathbf{x} \mapsto |\mathbf{x}\rangle\langle\mathbf{x}| . \tag{2.28}$$

The quantum circuit needed to implement this feature map has to be capable of preparing arbitrary states, which generally calls for a number of gates that grows exponentially in the number of qubits [25]. The kernel function arising from this feature map is

$$k(\mathbf{x}, \mathbf{x}') = |\langle\mathbf{x}'|\mathbf{x}\rangle|^2 = |\mathbf{x}^\top \mathbf{x}'|^2, \tag{2.29}$$

which is the square of a classical linear kernel.

With *coherent state encoding*, quantum computers can realize the popular classical radial basis function (RBF) or Gaussian kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2), \tag{2.30}$$

by preparing coherent states, which is natural for optical quantum computers acting in the infinite dimensional Fock space, instead of a qubit space. For details, see [26, 21, Eqs. (22) to (25)].

In all of the numerical experiments performed in this thesis, some form of *rotation encoding*, where the classical input defines the angles in Pauli rotations, is employed. One basic example of this idea is applying Pauli-y rotations with angles $\omega_i = x_i$ for all components of $\mathbf{x}$ in parallel on $q = s$ qubits. The encoded state is then a product state

$$|\psi(\mathbf{x})\rangle = \bigotimes_{i=1}^{q} R_y(x_i) \, |0_i\rangle \,, \tag{2.31}$$

where $R_y(\omega)$ denotes a single qubit y-rotation and $|0_i\rangle$ is the single qubit zero state of i-th qubit (see Fig. A.1 on page 76 for a depiction of an equivalent circuit). The associated kernel function

$$k(\mathbf{x}, \mathbf{x}') = \prod_{k=1}^{s} \cos^2(x_k - x'_k), \tag{2.32}$$

is a squared classical cosine kernel. For all variants of rotation encoding, it is important to pre-process the data such that the rotation angles are all restricted to $[0, 2\pi]$. Otherwise, the feature map ceases to be injective, which can lead to avoidable misclassifications. Pre-processing of this form is applied to all of the two dimensional data appearing in example plots throughout this thesis, so whenever data are shown in a plane without axis labels, the data lie in $[0, 1]$ (because of π prefactors in the feature map definitions, see Appendix A.1).

The above kernels all possess simple closed form expressions which are tractable on classical computers. This potentially ceases to be the case when entanglement is added to the feature map and the circuit is repeated multiple times as in (2.102).

Quantum Advantage

It has been noted in [1] that classical intractability of the quantum kernel function in a QSVM is a necessary condition for a quantum advantage like for example an exponential speedup over classical SVMs. This implies that the

example kernels listed in the previous section are all uninteresting from this point of view, as they can be efficiently evaluated classically. A feature map introduced in [1] that is conjectured to be classically hard is drawn in Fig. A.2 on page 77. But as classical intractability does not pose a sufficient condition for quantum advantage, there is no reason to assume that this feature map is useful for a relevant problem.

Other theoretical research points to a concrete quantum advantage. Under standard complexity theoretic assumptions, there exists a provable exponential speedup for the combination of another feature map and a carefully engineered dataset [2]. How this result can be extended and which other sufficient conditions exist for a quantum speedup is an active research question [17].

On a separate note and as pointed out in [1], even though the feature space accessed by QSVMs is exponentially large in q, the sheer size can not offer a quantum advantage in itself, as even classically, infinite dimensional spaces (like for example the one corresponding to the Gaussian kernel) are tractable due to the kernel trick. In addition, there are results indicating that the exponential size of the Hilbert space leads to vanishing kernels [27, 28, Appendix B 1.] (related to the vanishing gradients found in variational circuit training), which require an exponential number of circuit evaluations to be distinguishable from zero. So in this case, the exponential size of the Hilbert space is a hindrance instead of an asset. How such vanishing kernels can be avoided is another open research question.

Finite Sampling Statistics

Even under the assumption that all of the quantum kernel evaluations are performed on a universal fault tolerant quantum computer, the Born rule tells that for a finite number of measurement shots, quantum mechanical expectation values can only be determined up to some statistical uncertainty. In this sense, running a quantum circuit on any actual hardware is fundamentally different from simulating state vectors on a classical computer, where the probabilities determining the measurement outcomes are directly accessible. This section explores how such uncertainty affects the QSVM training and prediction.

In a first step, define the kernel or Gram matrix $K \in \mathbb{R}^{M \times M}$ with entries

$$(K)_{ij} = k_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \quad \forall\, \mathbf{x}_i, \mathbf{x}_j \in T, \tag{2.33}$$

given by the kernel function evaluated for all combinations of data pairs in the training set. Like discussed in Section 2.2.3, the quantum kernel can be approximated with the adjoint method (see Fig. 2.4), where a quantum state $\mathcal{E}(\mathbf{x}')^\dagger \mathcal{E}(\mathbf{x}) \left|0\right\rangle$ is prepared and measured a total of R times.

A more formal treatment of the adjoint method introduces the Bernoulli distributed random variable $\hat{k}_{ij}$ taking on the values

$$\hat{k}_{ij} = \begin{cases} 1 & \text{if the zero state } |0\rangle \text{ is recovered in the measurement} \\ 0 & \text{otherwise} \end{cases} . \tag{2.34}$$

The kernel is then equal to the true mean

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{E}\left[\hat{k}_{ij}\right], \tag{2.35}$$

because the expectation value of the Bernoulli distribution is equal to the probability of $\hat{k}_{ij} = 1$. Fundamentally, we can only approximate this expectation value by the sample mean

$$\bar{k}_R(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{R}\sum_{l=1}^{R} \hat{k}_{ij}^{(l)}, \tag{2.36}$$

where the $\hat{k}_{ij}^{(l)}$ are the i.i.d. outcomes of R measurement shots, performed for every single entry of K. Denote by $\bar{K}_R$ the matrix with entries $\bar{k}_R(\mathbf{x}_i, \mathbf{x}_j)$. Closely following [18, Section V. C.], the goal is now to bound the operator distance between the ideal K and obtainable $\bar{K}_R$.

To do so, define the error

$$E_R \coloneqq \bar{K}_R - K, \tag{2.37}$$

and observe that

$$\mathbb{E}\left[(E_R)_{ij}\right] = \mathbb{E}\left[\bar{k}_R(\mathbf{x}_i, \mathbf{x}_j) - k(\mathbf{x}_i, \mathbf{x}_j)\right] \tag{2.38}$$

$$= \frac{1}{R}\sum_{l=1}^{R} \mathbb{E}\left[\hat{k}_{ij}^{(l)}\right] - \mathbb{E}\left[k(\mathbf{x}_i, \mathbf{x}_j)\right] = 0, \tag{2.39}$$

where the linearity of the expectation value and (2.35) have been used. The second moment of E_R can be calculated as

$$\mathbb{E}\left[|(E_R)_{ij}|^2\right] = \mathbb{E}\left[\left(\frac{1}{R}\sum_{\ell=1}^{R} \hat{k}_{ij}^{(\ell)} - k_{ij}\right)^2\right] \tag{2.40}$$

$$= \frac{1}{R^2}\,\mathbb{E}\left[\left(\sum_{l=1}^{R} \hat{k}_{ij}^{(l)}\right)^2\right] \underbrace{-\frac{2k_{ij}}{R}\sum_{l=1}^{R}\underbrace{\mathbb{E}\left[\hat{k}_{ij}^{(l)}\right]}_{=k_{ij}} + k_{ij}^2}_{=-k_{ij}^2} \tag{2.41}$$

$$= \frac{1}{R^2}\left(R\underbrace{\mathbb{E}\left[\left(\hat{k}_{ij}^{(l)}\right)^2\right]}_{(\star)} + 2\binom{R}{2}\underbrace{\mathbb{E}\left[\hat{k}_{ij}^{(l)}\hat{k}_{ij}^{(m)}\right]}_{(\dagger)}\right) - k_{ij}^2, \tag{2.42}$$

where in the step from the second to the third line, the sum inside the square is expanded into two terms collecting the two cases when both samples are identical and when they are different with $l \neq m$. The expression can be further simplified knowing

$$(\star) = \mathrm{Var}\left[\hat{k}_{ij}\right] - \left(\mathbb{E}\left[\hat{k}_{ij}\right]\right)^2 = k_{ij}(1 - k_{ij}) + k_{ij}^2 = k_{ij}, \tag{2.43}$$

because $\hat{k}_{ij}$ is Bernoulli distributed and

$$(\dagger) = \mathbb{E}\left[\hat{k}_{ij}^{(l)}\right] \mathbb{E}\left[\hat{k}_{ij}^{(m)}\right] = k_{ij}^2, \tag{2.44}$$

because the $\hat{k}_{ij}^{(\cdot)}$ are independently and identically distributed. Plugging in these results back into (2.42) and making use of

$$\binom{R}{2} = \frac{R!}{2!(R-2)!} = \frac{R(R-1)}{2} \leq \frac{R^2}{2} \tag{2.45}$$

finally yields

$$\mathbb{E}\left[|(E_R)_{ij}|^2\right] = \frac{1}{R^2}\left[R\,k_{ij} + 2\binom{R}{2}k_{ij}^2\right] - k_{ij}^2 \leq \frac{k_{ij}}{R} = \mathcal{O}\left(\frac{1}{R}\right). \tag{2.46}$$

Analogously, it can be shown that the fourth moment scales as

$$\mathbb{E}\left[|(E_R)_{ij}|^4\right] = \mathcal{O}\left(\frac{1}{R^2}\right). \tag{2.47}$$

Knowing this, the following result from random matrix theory can be harnessed:

Theorem 2.1 (Latala's theorem [29, Theorem 2, 30, Theorem 5.37]) *For any finite matrix X whose entries are independent random variables x_{ij} of zero mean, there exists a constant $c > 0$ such that*

$$\mathbb{E}\left[\|X\|_2\right] \leq c\left[\max_i\left(\sum_j \mathbb{E}\left[x_{ij}^2\right]\right)^{\frac{1}{2}} + \max_j\left(\sum_i \mathbb{E}\left[x_{ij}^2\right]\right)^{\frac{1}{2}} + \left(\sum_{ij}\mathbb{E}\left[x_{ij}^4\right]\right)^{\frac{1}{4}}\right], \tag{2.48}$$

where $\|\cdot\|_2 = \sigma_{\max}(\cdot)$ denotes the operator norm induced by the Euclidean vector norm.

The matrix E_R satisfies all of the assumptions in Theorem 2.1 and by (2.39), (2.46) and (2.47), yields

$$\mathbb{E}\left[\|E_R\|_2\right] = \mathbb{E}\left[\|\bar{K}_R - K\|_2\right] = \mathcal{O}\left(\frac{\sqrt{M}}{\sqrt{R}}\right). \tag{2.49}$$

With this, a bound on the accuracy of the kernel matrix is established. The next question is how the solution to the SVM optimization problem is affected by perturbations to the ideal K. The following treatment is based on [2, Appendix C].

Start by considering a linear soft margin SVM that solves

$$\begin{aligned} \underset{\mathbf{w} \in \mathbb{R}^s,\, b, \xi_i \in \mathbb{R}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{2}\sum_i \xi_i^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \; \forall i, \\ & \xi_i \geq 0 \qquad \forall i, \end{aligned} \tag{2.50}$$

which only differs from the optimization problem in (2.11) by a factor of one half introduced for convenience and the substitution of ξ_i^2 for ξ_i in the objective. The latter modification means that instead of regularizing the slack with respect to the ℓ^1-norm like usual, here it is regularized with the ℓ^2-norm. It can be shown that in this case, the constraint $\xi_i \geq 0$ can be lifted as its inclusion does not change the solution to the optimization problem. The Lagrangian corresponding to (2.50) is then equal to

$$L(\mathbf{w}, b, \xi_i; \alpha_i) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} + \frac{C}{2}\sum_{i=1}^{M} \xi_i^2 + \sum_{i=1}^{M} \alpha_i [1 - \xi_i - y_i(\mathbf{w}^\top \mathbf{x}_i + b)], \tag{2.51}$$

where

$$\alpha_i \geq 0 \; \forall i, \tag{2.52}$$

due to the inequality constraints. The dual of (2.50) can be derived by determining the Lagrangian's stationary conditions with respect to the original variables and inserting those results into the initial expression. Setting the derivative of the Lagrangian to zero, the condition for $\mathbf{w}$ reads

$$\nabla_{\mathbf{w}} L(\mathbf{w}, b, \xi_i; \alpha_i) = \mathbf{w} - \sum_{i=1}^{M} \alpha_i y_i \mathbf{x}_i \overset{!}{=} 0 \implies \mathbf{w}^\star = \sum_{i=1}^{M} \alpha_i y_i \mathbf{x}_i, \tag{2.53}$$

and analogously results in

$$\frac{\mathrm{d}}{\mathrm{d}b} L(\mathbf{w}, b, \xi_i; \alpha_i) \overset{!}{=} 0 \implies 0 = \sum_{i=1}^{M} \alpha_i y_i \tag{2.54}$$

for the bias b. Finally,

$$\frac{\mathrm{d}}{\mathrm{d}\xi_i} L(\mathbf{w}, b, \xi_i; \alpha_i) = C\xi_i - \alpha_i \overset{!}{=} 0 \implies \xi_i^\star = \frac{\alpha_i}{C} \tag{2.55}$$

directly relates the slack- to the dual variables. Plugging (2.53) to (2.55) into (2.51) yields

$$L(\mathbf{w}^\star, \xi_i^\star; \alpha_i) = \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \tag{2.56}$$

$$+ \frac{C}{2}\sum_i \left(\frac{\alpha}{C}\right)^2 + \sum_i \alpha_i \left(1 - \frac{\alpha_i}{C} - y_i \sum_j \alpha_j y_j \mathbf{x}_j^\top \mathbf{x}_i\right)$$

$$= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j - \frac{1}{2}\sum_i \frac{\alpha_i^2}{C}, \tag{2.57}$$

where the constraint $\alpha_i \geq 0 \ \forall i$ still applies and all sums range over $i, j \in \{1, 2, ..., M\}$. Maximizing (2.57) with respect to the α_i and re-introducing the quantum feature map in (2.23) finally results in the kernelized dual of the ℓ^2-SVM primal in (2.50)

$$\begin{aligned} \underset{\alpha_i \in \mathbb{R}}{\text{maximize}} \quad & \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \, k(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{2}\sum_i \frac{\alpha_i^2}{C} \\ \text{subject to} \quad & 0 \leq \alpha_i \ \forall i. \end{aligned} \tag{2.58}$$

The difference to the dual of the ℓ^1-SVM in (2.17) to note is the inclusion of an additional third sum, which manifests itself as a regularization in the following.

The dual can also be expressed in terms of matrices. Defining Q as the matrix with entries

$$(Q)_{ij} \coloneqq y_i y_j \, k(\mathbf{x}_i, \mathbf{x}_j), \tag{2.59}$$

Q is positive semi-definite if K is, since

$$Q = \text{diag}(\mathbf{y}) \, K \, \text{diag}(\mathbf{y}), \tag{2.60}$$

for the vector of training labels $\mathbf{y} \in \mathbb{R}^M$ and $\text{diag}(\mathbf{y})$ can be absorbed into the $\mathbf{v}$ in the definition of positive semi-definiteness of a matrix

$$\mathbf{v}^\top Q \, \mathbf{v} \geq 0 \quad \forall \mathbf{v} \in \mathbb{R}^M \setminus \{\mathbf{0}\}. \tag{2.61}$$

The kernel matrix K is always positive semi-definite [31, Section 2.2], implying that Q must be too. Now, the dual in (2.58) can be written like in [2, Eq. (D19)] as the quadratic program

$$\begin{aligned} \underset{\alpha \in \mathbb{R}^M}{\text{minimize}} \quad & \frac{1}{2}\alpha^\top \left(Q + \frac{1}{C}\mathbb{I}\right) \alpha - \mathbf{1}^\top \alpha \\ \text{subject to} \quad & \alpha \geq 0, \end{aligned} \tag{2.62}$$

where α denotes the vector with components α_i, $\mathbf{1} \in \mathbb{R}^M$ the all one vector and $\mathbb{I} \in \mathbb{R}^{M\times M}$ the identity matrix. From (2.62), the benefit, which the ℓ^2-SVM has over the ℓ^1-SVM, is apparent: a regularization term proportional to the identity is added to Q that ensures positive definiteness of the matrix inside the brackets and gives a lower bound for the size of its smallest eigenvalue, as proven in the next paragraph.

Because the identity commutes with all matrices, Q and $\frac{1}{C}\mathbb{I}$ can be simultaneously diagonalized. Since Q is positive semi-definite, its eigenvalues are all greater than or equal to zero. So when Q and $\frac{1}{C}\mathbb{I}$ are expressed in a basis where both matrices are diagonal, adding the two yields a diagonal matrix with entries greater than or equal to $\frac{1}{C}$, implying that

$$\lambda \geq \frac{1}{C} \tag{2.63}$$

for the the smallest eigenvalue λ of the matrix $\left(Q + \frac{1}{C}\mathbb{I}\right)$.

With this, the following theorem can be invoked to prove the robustness of the dual QSVM optimization when the kernel function is only evaluated up to some finite precision.

Theorem 2.2 (Daniel's Theorem [32, Theorem 2.1] [2, Lemma 16]**)** *Let x_0 be the solution to the quadratic program*

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}x^\top K x - c^\top x \\ \text{subject to} \quad & Gx \leq g, \\ & Dx = d, \end{aligned} \tag{2.64}$$

where K is positive definite with smallest eigenvalue $\lambda > 0$ and the dimensions of the vectors c, g, d and matrices G, D are such that all operations are well-defined. Let K' be a symmetric matrix such that $\|K' - K\|_2 \leq \varepsilon < \lambda$.[2] *Let x_0' be the solution to* (2.64) *with K replaced by K'. Then*

$$\|x_0' - x_0\| \leq \frac{\varepsilon}{\lambda - \varepsilon}\|x_0\|. \tag{2.65}$$

As a reminder, $\|\cdot\|$ denotes the Euclidean vector norm and $\|\cdot\|_2$ the operator norm induced by this norm.

From the ground up, classification with QSVMs can be divided into two subsequent steps, training and prediction. For training, the quantum kernel matrix K defined in (2.33) is initially evaluated on a quantum computer. Then, the QSVM is fit by running the dual program in (2.62) on a classical computer, yielding a solution vector $\alpha^\star$. For prediction, a new datum $\hat{\mathbf{x}} \in \mathbb{R}^s$ is assigned a

[2] Positive definiteness of K' is then implied.

class membership $\hat{y} \in \{-1, +1\}$ via the classification function $\tilde{c}_{\mathrm{SVM}}(\hat{\mathbf{x}})$ in (2.19). To this end, the quantum computer has to be employed again to determine the quantum kernel value $k(\hat{\mathbf{x}}, \mathbf{x}_i)$ for all $\mathbf{x}_i$ in the training set T.

From these two steps, it is clear that there are two separate instances in the QSVM algorithm where quantum kernels are evaluated and the statistical uncertainty inherent to quantum expectation values unavoidably enters. Because even on a fault tolerant quantum computer, fundamentally only an approximate kernel function $\bar{k}_R(\hat{\mathbf{x}}, \mathbf{x}_i)$ as defined in (2.36) is feasible. Consequently, for training, K has to be replaced by $\bar{K}_R$, in turn leading to a noisy $\bar{Q}_R$ analogous to (2.60). The solution to the dual when Q is replaced by $\bar{Q}_R$ is then denoted as $\bar{\alpha}^\star$. In the same way, the kernel evaluations in the decision function are replaced by evaluations of $\bar{k}_R(\hat{\mathbf{x}}, \mathbf{x}_i)$ in the prediction step. Putting all of this together, a statement on the robustness of the overall QSVM with respect to measurement uncertainty can be made on the level of

$$h(\hat{\mathbf{x}}) \coloneqq \sum_{i=1}^{M} \alpha_i^\star y_i \, k(\hat{\mathbf{x}}, \mathbf{x}_i) \quad \text{and} \quad \bar{h}_R(\hat{\mathbf{x}}) \coloneqq \sum_{i=1}^{M} \bar{\alpha}_i^\star y_i \, \bar{k}_R(\hat{\mathbf{x}}, \mathbf{x}_i), \tag{2.66}$$

the ideal and noisy version of the term inside the sign function in the classification function (2.19), where the components of the solution vectors $\alpha^\star$ and $\bar{\alpha}^\star$ are denoted as $\alpha_i^\star$ and $\bar{\alpha}_i^\star$. The goal is to show the robustness of QSVMs with respect to finite sampling noise by bounding the difference between these terms with high probability.

The following lemma is adapted from [2, Lemma 19].

Lemma 2.3 (Noise robustness of the ℓ^2-QSVM) *For some fixed $\varepsilon > 0$, suppose $R = \mathcal{O}\left(M^{8/3}/\varepsilon^2\right)$ measurement shots are taken for each quantum kernel estimation circuit. Furthermore, assume the setting of noisy halfspace learning defined in [2, Lemma 14]. Then, with fixed probability $p > \frac{1}{2}$, where the probability is stemming from the choice of random training samples and uncertainty coming from the finite sampling statistics, for every $\mathbf{x} \in \mathbb{R}^s$ we have*

$$|\bar{h}_R(\mathbf{x}) - h(\mathbf{x})| \leq \varepsilon. \tag{2.67}$$

Proof Start by considering the operator distance between the noisy matrix $\bar{Q}_R$ resulting from quantum kernel evaluations with a finite number of measurement shots R per entry and the ideal matrix Q. Making use of the definition of the operator norm $\|\cdot\|_2$ and the connection between Q and K in (2.60), the

expression $\|\bar{Q}_R - Q\|_2$ can be bounded like

$$\|\bar{Q}_R - Q\|_2 = \sup_{\mathbf{v}\neq 0} \frac{\|(\bar{Q}_R - Q)\mathbf{v}\|}{\|\mathbf{v}\|} \tag{2.68}$$

$$= \sup_{\mathbf{v}\neq 0} \frac{\|\mathrm{diag}(\mathbf{y})\,(\bar{K}_R - K)\,\mathrm{diag}(\mathbf{y})\mathbf{v}\|}{\|\mathbf{v}\|} \tag{2.69}$$

$$= \sup_{\mathbf{v}\neq 0} \frac{\sqrt{\sum_{i=1}^{M}\left|\sum_{j=1}^{M} y_i y_j(\bar{k}_R(\mathbf{x}_i,\mathbf{x}_j) - k(\mathbf{x}_i,\mathbf{x}_j))v_j\right|^2}}{\|\mathbf{v}\|} \tag{2.70}$$

$$\leq \sup_{\mathbf{v}\neq 0} \frac{\sqrt{\sum_{i=1}^{M}\left|\sum_{j=1}^{M} (\bar{k}_R(\mathbf{x}_i,\mathbf{x}_j) - k(\mathbf{x}_i,\mathbf{x}_j))v_j\right|^2}}{\|\mathbf{v}\|} \tag{2.71}$$

$$= \|\bar{K}_R - K\|_2, \tag{2.72}$$

where in the third to forth line $|y_i| = 1$ because $y_i \in \{-1, +1\}$ and the triangle inequality have been used. Then, (2.49) implies

$$\mathbb{E}\left[\|\bar{Q}_R - Q\|_2\right] = \mathcal{O}\left(\frac{\sqrt{M}}{\sqrt{R}}\right). \tag{2.73}$$

For the setting of *noisy halfspace learning* as defined in [2, Lemma 14, Definition 15], it can be shown [2, Remark 2] that

$$\mathbb{E}\left[\|\alpha^\star\|\right] = \mathcal{O}\left(M^{1/3}\right), \tag{2.74}$$

where $\alpha^\star$ is the solution to (2.62).

With Markov's inequality

$$\Pr[X \geq a\,\mathbb{E}[X]] \leq \frac{1}{a} \iff \Pr[X \leq a\,\mathbb{E}[X]] \geq 1 - \frac{1}{a} \stackrel{!}{=} p', \tag{2.75}$$

the statements (2.73) and (2.74) in expectation can be transformed into statements in probability. To this end, choose a such that $p' > 1 - \frac{1}{a} > \frac{1}{\sqrt[3]{2}}$ is fulfilled, yielding the bounds

$$\eta \coloneqq \|\bar{Q}_R - Q\|_2 = \mathcal{O}\left(\frac{\sqrt{M}}{\sqrt{R}}\right), \tag{2.76}$$

and

$$\|\alpha^\star\| = \mathcal{O}\left(M^{1/3}\right), \tag{2.77}$$

which hold with a probability greater than or equal to p' respectively.

Now, Theorem 2.2 can be leveraged as Q is positive-definite because there is a lower bound on its smallest eigenvalue λ in (2.63). Then, for $\delta_i \coloneqq \bar{\alpha}_i^\star - \alpha_i^\star$ and with probability at least p', (2.65) yields

$$\begin{aligned} \|\delta\| = \|\bar{\alpha}^\star - \alpha^\star\| \leq \frac{\eta}{\lambda - \eta}\|\alpha^\star\| = \left(\frac{\eta}{\lambda} + \mathcal{O}\left(\frac{\eta^2}{\lambda^2}\right)\right)\|\alpha^\star\| \\ = \mathcal{O}\left(\frac{\sqrt{M}}{\sqrt{R}}\right)\mathcal{O}\left(M^{1/3}\right) = \mathcal{O}\left(\frac{M^{5/6}}{\sqrt{R}}\right), \end{aligned} \tag{2.78}$$

where R has to be chosen such that $\eta < \lambda$, which is always feasible due to the lower bound on λ.

In a next step, denote by $\nu_i \coloneqq \bar{k}_R(\hat{\mathbf{x}}, \mathbf{x}_i) - k(\hat{\mathbf{x}}, \mathbf{x}_i)$ for $i = 1, 2, ..., M$ the difference between the obtainable and ideal kernel function evaluated for pairs of the datum to be classified $\hat{\mathbf{x}}$ and the elements in the training set $\mathbf{x}_i \in T$. As ν_i is a special case of the components in E_R in (2.37), $\mathbb{E}[\nu_i] = 0$ is implied by (2.39) and $\mathbb{E}[\nu_i^2] = \mathcal{O}\left(\frac{1}{R}\right)$ by (2.46). By making use of Chebyshev's inequality

$$\begin{aligned} &\Pr\left(|X - \mathbb{E}[X]| \geq a\sqrt{\mathrm{Var}[X]}\right) \leq \frac{1}{a^2} \\ \iff &\Pr\left(|X - \mathbb{E}[X]| \leq a\sqrt{\mathrm{Var}[X]}\right) \geq 1 - \frac{1}{a^2} \stackrel{!}{=} p', \end{aligned} \tag{2.79}$$

and

$$\mathrm{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2, \tag{2.80}$$

the above expectation values can be converted into the statement that

$$\|\nu\| = \mathcal{O}\left(\frac{\sqrt{M}}{\sqrt{R}}\right) \tag{2.81}$$

holds with probability at least $p' > \frac{1}{\sqrt[3]{2}}$, where the $\sqrt{M}$ scaling comes from the euclidean norm $\|\cdot\|$ and the fact that the vector ν has M components ν_i.

Returning to the terms inside the ideal and noisy classification functions, all

of the bounds (2.76), (2.77) and (2.81) can be combined in

$$\begin{aligned}
\left|\bar{h}_R(\hat{\mathbf{x}}) - h(\hat{\mathbf{x}})\right| &= \left|\sum_{i=1}^{M} \bar{\alpha}_i^\star y_i \, \bar{k}_R(\hat{\mathbf{x}}, \mathbf{x}_i) - \sum_{i=1}^{M} \alpha_i^\star y_i \, k(\hat{\mathbf{x}}, \mathbf{x}_i)\right| && (2.82)\\
&= \left|\sum_{i=1}^{M} (\alpha_i^\star + \delta_i) y_i (k(\hat{\mathbf{x}}, \mathbf{x}_i) + \nu_i) - \sum_{i=1}^{M} \alpha_i^\star y_i \, k(\hat{\mathbf{x}}, \mathbf{x}_i)\right| && (2.83)\\
&= \left|\sum_{i=1}^{M} \alpha_i^\star \nu_i + \delta_i k(\hat{\mathbf{x}}, \mathbf{x}_i) + \delta_i \nu_i\right| && (2.84)\\
&\leq \sum_{i=1}^{M} |\alpha_i^\star \nu_i| + |\delta_i k(\hat{\mathbf{x}}, \mathbf{x}_i)| + |\delta_i \nu_i| && (2.85)\\
&\leq \|\alpha^\star\| \cdot \|\nu\| + \sqrt{M}\|\delta\| + \|\delta\| \cdot \|\nu\| && (2.86)\\
&= \mathcal{O}\left(\frac{M^{4/3}}{\sqrt{R}}\right), && (2.87)
\end{aligned}$$

which is true with probability at least $p \coloneqq (p')^3 > \frac{1}{2}$ for any $\hat{\mathbf{x}} \in \mathbb{R}^s$. In the second to last step, the Cauchy-Schwarz inequality and property $k(\hat{\mathbf{x}}, \mathbf{x}_i) \in [0, 1]$ have been used.

With this, the desired accuracy

$$\left|\bar{h}_R(\hat{\mathbf{x}}) - h(\hat{\mathbf{x}})\right| \leq \varepsilon \tag{2.88}$$

for the QSVM classification function is obtained with probability greater than or equal to p for

$$R = \mathcal{O}\left(\frac{M^{8/3}}{\varepsilon^2}\right) \tag{2.89}$$

measurement shots per quantum kernel evaluation. □

Training From Lemma 2.3 and the fact that the symmetric quantum kernel matrix K has $M(M+1)/2 = \mathcal{O}\left(M^2\right)$ unique entries, it follows that overall

$$R_{\text{train}} = \mathcal{O}\left(\frac{M^{14/3}}{\varepsilon^2}\right) = \mathcal{O}\left(\frac{M^{4.67}}{\varepsilon^2}\right) \tag{2.90}$$

measurement shots are needed to train a QSVM so that it fulfills the accuracy requirement defined there. This marks an improvement over the result

$$R = \mathcal{O}\left(\frac{M^4}{\varepsilon^2}\right) \quad \text{and} \quad R_{\text{train}} = \mathcal{O}\left(\frac{M^6}{\varepsilon^2}\right) \tag{2.91}$$

derived in [2] under the same assumptions. Note that the finding

$$R = \mathcal{O}\left(\frac{M}{\tilde{\varepsilon}^2}\right) \quad \text{and} \quad R_{\text{train}} = \mathcal{O}\left(\frac{M^3}{\tilde{\varepsilon}^2}\right) \tag{2.92}$$

from [18, Section V. C.] is not comparable to (2.90) and (2.91), since it is derived directly from the quantum kernel matrix K and the result does not take into account the quadratic program that is used to train the QSVM. Therefore, ε and $\tilde{\varepsilon}$ denote different quantities.

Prediction Once training has completed and the $\alpha^\star$ vector is fixed, the QSVM decision function can be employed to assign an arbitrary datum $\mathbf{x}$ a class membership. The goal is to achieve an accuracy of ε like in (2.67), but for error free $\alpha^\star$. To this end, consider the sum in (2.19), which runs over all S support vectors $\mathbf{x}_i$ distinguished by $\alpha_i \neq 0$. To determine the sum up to a standard deviation equal to ε, each term needs to be estimated with accuracy $\varepsilon/\sqrt{S}$, because the terms in the sum are independently distributed and the variance is additive. Consequently, S/ε^2 quantum circuit evaluations are required per term by the standard error of the mean, resulting in a total of

$$R_{\text{pred}} = \mathcal{O}\left(\frac{S^2}{\varepsilon^2}\right), \tag{2.93}$$

quantum circuit evaluations for the sum. To convert this statement concerning the standard deviation of the term inside the sign function in (2.19) to a probably correct ($p > \frac{1}{2}$) inequality of the form of (2.67), Chebyshev's inequality in (2.79) can be invoked to get the same scaling for R_{pred}.

Note that for kernelized SVMs, typically $S \ll M$ since $S \sim M$ leads to overfitting, as for example formalized in the generalization bound in [33, Eq. (93)]. For context, the number of support vectors usually increases with increasing regularization parameter C in the case of a kernelized SVM. Furthermore, S is strongly dependent on the training set and the kernel function itself. For data that are linearly separable in feature space, S does not scale with M.

Quantum Amplitude Estimation As an aside on the training and prediction complexity on fault tolerant quantum computers, the kind of overlap evaluations in (2.26) lend themselves to a quadratic speedup by employing the quantum amplitude estimation algorithm [19, 20].

2.2.4 Pegasos Algorithm

So far, the SVM optimization problem has been solved via the kernelized dual in (2.17). The Pegasos ("Primal Estimated sub-GrAdient SOlver for SVM") algorithm developed in [3] offers an alternative as it finds an approximate

solution to the primal SVM optimization in (2.11) and its formulation in feature space.

In the classical case where kernels are known exactly and there is no analogue to the statistical noise inherent to quantum kernels, the main appeal of Pegasos is a training runtime that scales as $\mathcal{O}(M/\delta)$ for

$$\delta \coloneqq \left| \left(\frac{1}{2}\|\mathbf{w}^\star\|^2 + \frac{C}{2}\sum_i (\xi_i^\star)^2 \right) - \left(\frac{1}{2}\|\mathbf{w}^\mathrm{P}\|^2 + \frac{C}{2}\sum_i \left(\xi_i^\mathrm{P}\right)^2 \right) \right|, \tag{2.94}$$

where $w^\star, \xi^\star$ denote the exact solution to the primal optimization in (2.11) and $w^\mathrm{P}, \xi^\mathrm{P}$ the solution found by Pegasos. This is in contrast to the $\mathcal{O}\left(M^2\right)$ scaling achievable when training via the dual, where the quadratic complexity comes from the necessity to construct the full $M \times M$ kernel matrix for the training set T.

As seen in the previous section on QSVMs utilizing quantum kernel evaluations that are inherently subjected to finite sampling statistics, the computational complexity of the dual is even more unfavorable in that setting with a total of $R_\mathrm{train} = \mathcal{O}\left(M^{4.67}/\varepsilon^2\right)$ quantum circuit evaluations needed to get an ε-accurate classifier in the sense of (2.67). Since with such a large polynomial scaling, the computational cost can quickly become practically prohibitive for larger datasets, the question whether Pegasos offers better training complexity also in the quantum case is of great interest. The later subsection on *finite sampling statistics* analyzes Pegasos from this point of view.

First, the algorithm itself is briefly introduced. The kernelized version of Pegasos is included in Algorithm 1. For the linear variant, see [3, Fig. 1]. Even though Algorithm 1 minimizes the primal and not the dual, it can be formulated in a way that only necessitates the evaluation of inner products and not of explicit feature vectors [3, Section 4], so the kernel trick can be applied just like in Section 2.2.2. In Algorithm 1, the notation $\vec{\alpha}[j]$ denotes the j-th component of the M dimensional vector $\vec{\alpha}$ and the subscript $\vec{\alpha}_t$ refers to the intermediate result at the t-th step.

For the Pegasos algorithm, the bias has to be included in the kernel function by adding a constant factor like in (2.22). For more sophisticated approaches, see [34, Section 6]. The α_i in (2.17) and final $\vec{\alpha}_t$ in Pegasos take on analogous roles

$$\alpha_i \mathrel{\widehat{=}} \frac{C}{t}\,\vec{\alpha}_t[i], \tag{2.95}$$

so the decision function of an SVM that has been trained with the Pegasos algorithm is identical to (2.19). Explicitly written to be consistent with the notation in Algorithm 1, it is

$$\tilde{c}_\mathrm{Pegasos}(\hat{\mathbf{x}}) = \operatorname{sign}\left[\sum_{i=1}^{M} \vec{\alpha}[i] y_i\, k(\mathbf{x}_i, \hat{\mathbf{x}}) \right], \tag{2.96}$$

Algorithm 1 Kernelized Pegasos [3, Fig. 3]

1: **Inputs:**
2: training data $T = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_M\}$
3: labels $L = \{y_1, y_2, ..., y_M\}$
4: regularization parameter $C \in \mathbb{R}^+$
5: number of steps $\tau \in \mathbb{N}$
6:
7: **Initialize:** $\vec{\alpha}_1 \leftarrow \vec{0} \in \mathbb{N}^M$
8:
9: **for** $t = 1, 2, ..., \tau$ **do**
10: Choose $i_t \in \{0, ..., M\}$ uniformly at random.
11: **for all** $j \neq i_t$ **do**
12: $\vec{\alpha}_{t+1}[j] \leftarrow \vec{\alpha}_t[j]$
13: **end for**
14: **if** $y_{i_t} \frac{C}{t} \sum_j \vec{\alpha}_t[j] y_{i_t} k(\mathbf{x}_{i_t}, \mathbf{x}_j) < 1$ **then**
15: $\vec{\alpha}_{t+1}[i_t] \leftarrow \vec{\alpha}_t[i_t] + 1$
16: **else**
17: $\vec{\alpha}_{t+1}[i_t] \leftarrow \vec{\alpha}_t[i_t]$
18: **end if**
19: **end for**
20:
21: **Output:** $\vec{\alpha}_{\tau+1}$

where the constant prefactor can be dropped and the bias b has to be implicitly included in the kernel. As in the case of the regular QSVM, training data $\mathbf{x}_i \in T$ where $\alpha_i \neq 0$ after training has completed are called *support vectors*.

When training QSVMs with Pegasos for a fixed hyperparameter τ and training set, the total number of kernel evaluations necessary to run the algorithm strongly depends on the regularization parameter C, which is not the case for QSVMs trained via the dual. An empirical relationship is included in Fig. 2.5.

Finite Sampling Statistics

Just as the regular QSVM, the Pegasos trained QSVM has quantum kernel evaluations like (2.24) at its core. Since the expectation value in the kernel function can fundamentally only be known up to an additive error due to finite sampling statistics even on a fault tolerant quantum computer, the effect this has on the algorithm has to be analyzed. In the following, the focus is on the number of kernel evaluations necessary to run the algorithm.

Training Line 14 in Algorithm 1 implies that during the t-th step, at most t kernel evaluations need to be performed, since initially $\vec{\alpha_1} = \vec{0}$ and not more

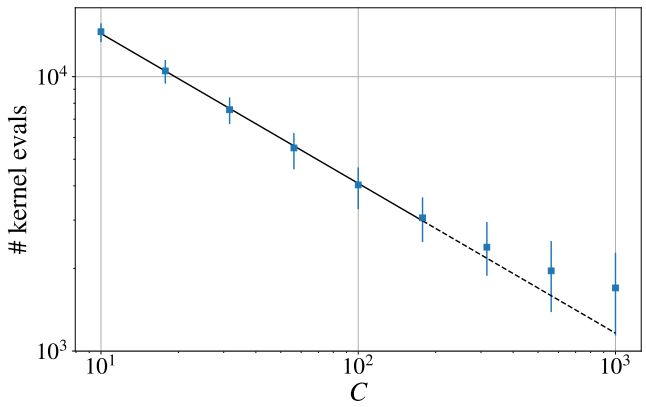


Figure 2.5: The total number of kernel evaluations (taking the expectation values as exact and neglecting the R measurement shots that actually have to be taken) occurring in a run of Pegasos strongly depends on the regularization parameter C. For this example, the number of kernel evaluations is found to be approximately proportional to $C^{-0.55}$ for the six smallest values of C considered (the solid black line marks the fit). For larger C, the points deviate from the fit (dashed black line). The training set T under consideration is generated like described at the beginning of Section 3.3 with $M = 1000$ and $\mu = 0.2$, such that the classes are linearly separable in feature space. The squares and error bars correspond to the mean and the interval between the 5th and 95th percentile over 100 runs of the algorithm. For reference, constructing the full kernel matrix as necessary for the dual trained SVM results in around $5 \cdot 10^5$ evaluations of the kernel function, independent of C.

than one component of $\vec{\alpha}$ becomes greater than zero in each step (see line 15). If the sum in line 14 now ought to be δ-accurate in terms of the standard deviation, the individual i.i.d. kernel evaluations have to be $(\delta/\sqrt{t})$-accurate, which implies t/δ^2 samples per term. Since there are t terms, overall t^2/δ^2 kernel evaluations are necessary for step t. There is a total of τ such iterations. Summing up $\sum_{t=1}^{\tau} t^2/\delta^2$ then leads to $\mathcal{O}\left(\tau^3/\delta^2\right)$ samples. For the exact kernel evaluations treated in [3], it is proven that $\tau = \mathcal{O}\left(1/\delta\right)$ iterations are enough to find a solution to the SVM optimization problem that achieves accuracy δ in the sense of (2.94). Under the assumption that this also holds if the sum in line 14 is merely δ-accurate instead of exact, the results can be combined to a total of

$$R_{\text{train}} = \mathcal{O}\left(\frac{1}{\delta^5}\right) \tag{2.97}$$

necessary quantum circuit evaluations to train a QSVM with Pegasos. The conjecture directly above (2.97) is supported empirically by the experiments in Section 3.3.1, as they suggest that introducing statistical uncertainty to line 14 of the algorithm only has a small impact on its performance (see for example Figs. 3.14, 3.15 and B.6).

Observe that the complexity in (2.97) is independent of the size of the dataset M, which is fundamentally different to the scaling for a traditionally trained QSVM found in Section 2.2.3. See Fig. 3.16 on page 65 in Section 3.3.1 for an

empirical example of the M independence in the setting where the data are linearly separable in feature space.

Alternatively, the overall scaling law

$$R_{\text{train}} = \mathcal{O}\left(\frac{M^2}{\delta^3}\right) \tag{2.98}$$

for Pegasos training can be derived by bounding the number of evaluations in the t-th iteration with respect to M instead of t. To get to the result in (2.98), the same assumption as above (2.97) is made. Altogether, the minimum of (2.97) and (2.98) holds.

Finally, like for the standard QSVM training, the kernel evaluations in the Pegasos algorithm can be sped up quadratically by utilizing the quantum amplitude estimation algorithm on fault tolerant hardware. Then, only $\mathcal{O}\left(1/\delta^{7/2}\right)$ or $\mathcal{O}\left(M^{3/2}/\delta^2\right)$ quantum circuit evaluations are required for training.

Prediction Under the assumption that the uncertainty in the solution $\vec{\alpha}$ found by the Pegasos algorithm can be neglected, predicting the class membership for a new datum $\mathbf{x}$ after the QSVM has been trained, entails the following computational complexity: The goal is an ε-accurate (in terms of the standard deviation) value inside the sign function in (2.96). There, a total of $S < M$ terms with nonzero $\vec{\alpha}[i]$ are summed (the associated $\mathbf{x}_i$ are called support vectors). Because the variance is additive for independent random variables, this implies that the individual terms in the sum have to be known with accuracy $\varepsilon/\sqrt{S}$, which requires S/ε^2 measurement shots. So overall, the number of quantum circuit evaluations

$$R_{\text{pred}} = \mathcal{O}\left(\frac{S^2}{\varepsilon^2}\right) \tag{2.99}$$

is necessary for ε-accurate prediction, in full analogy to the result derived for the dual trained QSVM in Section 2.2.3. Like there, this statement in standard deviation can be converted to an inequality that holds with probability $p > \frac{1}{2}$ by application of Chebyshev's inequality in (2.79).

Implications of Approximating the Optimization

When considerations on training complexity can be left aside, another question to explore is how the quality of the solution compares between a QSVM that has been trained via the dual and one that has been trained with Pegasos.

The Pegasos algorithm only finds an δ-accurate solution to the primal optimization problem in (2.11), not the exact solution. However, the ultimate goal of an SVM is not to perfectly solve the optimization problem, which its training poses, but to achieve good generalization performance on the

previously unseen test set E. There is research that indicates that within limits, approximating the training does not have significant implications for the generalization performance [34]. This is possibly similar to how classical neural networks also only approximately minimize their training objective with stochastic gradient descent and still generalize well.

As a side note, there are also other approximations to the classical SVM apart from Pegasos that might be able to be adapted to the QSVM setting. An example is the *random Fourier features* model first introduced in [35].

2.3 Quantum Neural Networks

A *quantum neural network* (QNN) is a parameterized quantum circuit [14] that performs a machine learning task, such as regression or classification. In this thesis, QNNs are set up to perform binary classification and consist of two successive sub-circuits: a *feature map* acting on the all zero state and a *variational form*. See Fig. 2.6 for an illustration. This setup is identical to the *quantum variational classifier* in [1], on which the following definitions are largely based.

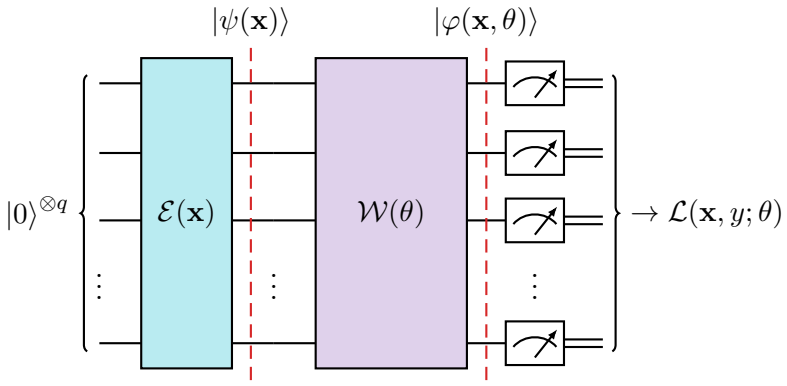


Figure 2.6: General quantum circuit structure of a QNN, which is defined to consist of a feature map circuit $\mathcal{E}(\mathbf{x})$ followed by a variational form circuit $\mathcal{W}(\theta)$. This means that there is a strict separation between circuit parameters fixed by the classical input vector $\mathbf{x}$ and trainable parameters θ.

Feature Maps

In the first step, a classical input has to be encoded into the Hilbert space the quantum computer acts in, in particular, the space of density matrices. To this end, a quantum circuit $\mathcal{E}(\mathbf{x}) \in \mathcal{U}(2^q)$ fixed by a classical input datum $\mathbf{x}$ is applied to the zero state of the computational basis resulting in the state

$$|\psi(\mathbf{x})\rangle := \mathcal{E}(\mathbf{x})\,|0\rangle\,, \tag{2.100}$$

which in turn defines a feature map

$$\begin{aligned} \psi \colon \mathbb{R}^s &\to \mathcal{S}(2^q) \\ \mathbf{x} &\mapsto |\psi(\mathbf{x})\rangle\langle\psi(\mathbf{x})| \,, \end{aligned} \tag{2.101}$$

where the space of quantum states on q qubits is the feature space. Notice that this structure is identical to the feature maps for QSVMs considered in Section 2.2.3. For concrete examples, see the earlier discussion in Section 2.2.3 on page 15 and the circuits depicted in Appendix A.1 on page 76.

Typically, the feature map circuit $\mathcal{E}$ can be divided into identical layers $\mathcal{E}^{(\cdot)}$ that are repeated, resulting in a quantum circuit structured like so

$$\mathcal{E}(\mathbf{x}) = \mathcal{E}^{(l)}(\mathbf{x}) \ \cdots \ \mathcal{E}^{(2)}(\mathbf{x})\, \mathcal{E}^{(1)}(\mathbf{x}), \tag{2.102}$$

for l layers. As already mentioned in Section 2.2.3, the data have to be pre-processed in accordance with the feature map. For the rotation encodings in Appendix A.1, this means $\mathbf{x} \in [0,1]^s \subset \mathbb{R}^s$ for example.

Note that in general, the dependence of $|\psi(\mathbf{x})\rangle$ on $\mathbf{x}$ is nonlinear.

Variational Forms

Taking the state prepared by the feature map as its input, the *variational form* is another quantum circuit $\mathcal{W}(\theta) \in \mathcal{U}(2^q)$ and prepares the state

$$|\varphi(\mathbf{x}, \theta)\rangle \coloneqq \mathcal{W}(\theta)\, \mathcal{E}(\mathbf{x})\, |0\rangle \,. \tag{2.103}$$

It is the only part of the QNN that contains free parameters θ that can be adjusted during a training procedure. For a variational form containing d parameters, θ is d-dimensional. Throughout this thesis, the θ always correspond to angles in Pauli rotations and are therefore restricted to $\theta \in [0, 2\pi]^d$ to avoid periodicity.

The variational form quantum circuit can be typically grouped into repeating layers $1, 2, ..., l$, where each layer consists of a rotation block containing single qubit gates and an entanglement block made up of two qubit gates. This results in a composition

$$\mathcal{W}(\theta) = \underbrace{\mathcal{W}_{\text{ent}}^{(l)}\left(\vartheta_l'\right) \mathcal{W}_{\text{loc}}^{(l)}\left(\vartheta_l\right)}_{l\text{th layer}} \ \cdots \ \underbrace{\mathcal{W}_{\text{ent}}^{(1)}\left(\vartheta_1'\right) \mathcal{W}_{\text{loc}}^{(1)}\left(\vartheta_1\right)}_{\text{1st layer}}, \tag{2.104}$$

where θ encompasses all of the ϑ and ϑ'. For an example quantum circuit layer, see Appendix A.2 on page 77.

Note that since $\mathcal{W}(\theta)$ is a unitary operation, it acts linearly on its quantum mechanical input state.

Decision Function

With the previous definitions, the QNN is complete and binary classification can now be performed: Given a datum $\mathbf{x}$, the quantum state $|\varphi(\mathbf{x},\theta)\rangle = \mathcal{W}(\theta)\,|\psi(\mathbf{x})\rangle$ is prepared and measured in the computational basis of q qubits. A single resulting bitstring $z \in \{0,1\}^q$ is associated with a class membership via the boolean function

$$\begin{aligned} f\colon \{0,1\}^q &\to \{-1,+1\} \\ z &\mapsto \tilde{y} \end{aligned} \tag{2.105}$$

fixed in advance.

Because of the nature of quantum mechanics, the measurement outcome z is probabilistic and a single realization can not be used to assign $\mathbf{x}$ a class membership $y \in \{-1,+1\}$. Instead, the randomness in z is suppressed by averaging in the expectation value

$$\mathrm{QNN}_\theta(\mathbf{x}) \coloneqq \langle\psi(\mathbf{x})|\,\mathcal{W}(\theta)^\dagger\,\mathcal{F}\,\mathcal{W}(\theta)\,|\psi(\mathbf{x})\rangle \in [-1,+1], \tag{2.106}$$

which is defined with respect to the diagonal operator

$$\mathcal{F} \coloneqq \sum_{z\in\{0,1\}^q} f(z)\,|z\rangle\langle z| \tag{2.107}$$

in the computational basis. The class label is finally assigned via the *classification function*

$$\tilde{c}_{\mathrm{QNN}_\theta}(\mathbf{x}) \coloneqq \mathrm{sign}\left[\mathrm{QNN}_\theta(\mathbf{x}) + b\right], \tag{2.108}$$

where a bias term $b \in \mathbb{R}$ has been introduced.

This decision function can also be justified by the following considerations: Expressing (2.106) explicitly yields

$$\langle\psi(\mathbf{x})|\,\mathcal{W}(\theta)^\dagger\,\mathcal{F}\,\mathcal{W}(\theta)\,|\psi(\mathbf{x})\rangle = 1\cdot p_{f(\mathbf{x})=+1} + (-1)\cdot p_{f(\mathbf{x})=-1}\,. \tag{2.109}$$

Since $p_{f(\mathbf{x})=1} = p_{+1}$ is the probability that $\mathbf{x}$ belongs to class $+1$ and likewise for -1, the probabilities of class membership predicted by the QNN can be expressed as

$$p_y = \frac{1}{2}\left(1 + y\;\mathrm{QNN}_\theta(\mathbf{x})\right), \tag{2.110}$$

and the classification function (2.108) is equivalent to basing the decision off the question whether

$$p_{+1} > p_{-1} - b \tag{2.111}$$

is true or false.

Throughout the thesis, f is chosen to be the *parity function*

$$f(z) = \begin{cases} -1, & \text{if } z \text{ has odd parity} \\ +1, & \text{if } z \text{ has even parity} \end{cases} \implies \mathcal{F} = Z^{\otimes q}, \tag{2.112}$$

leading to a global observable. Observables depending on all qubits like this might suffer from vanishing gradients as a result (see the last paragraph in Section 4.1). Considering different local $\mathcal{F}$ is a promising direction for future research.

Optimization

Before the QNN decision function in (2.108) can be meaningfully applied, it has to be fitted to the machine learning task at hand.

Ultimately, the goal is to minimize the *expected risk* of a previously fixed loss function $\mathcal{L}$

$$\mathcal{R}(c_{\mathrm{QNN}_\theta}) = \mathbb{E}\left[\mathcal{L}(Y, c_{\mathrm{QNN}_\theta}(X))\right] \tag{2.113}$$

defined as an expectation value over X and Y, where X is the random variable of which $\mathbf{x}$ is a realization and Y the random variable of which y is a realization. Since the distribution $P(X, Y)$ is unknown, (2.113) can only be approximated via the *empirical risk*

$$\hat{\mathcal{R}}(c_{\mathrm{QNN}_\theta}, T) = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}(y_i, c_{\mathrm{QNN}_\theta}(\mathbf{x}_i)), \tag{2.114}$$

which is defined with respect to the training set T and labels L like in Section 2.1. With this, the intention of training is to find a classifier $\tilde{c}_{\mathrm{QNN}_\theta}$ that minimizes the empirical risk:

$$\tilde{c}_{\mathrm{QNN}_\theta} \in \underset{c_{\mathrm{QNN}_\theta}}{\arg\min} \quad \hat{\mathcal{R}}(c_{\mathrm{QNN}_\theta}, T). \tag{2.115}$$

However, this optimization problem is generally intractable; so instead, a local minimum of $\hat{\mathcal{R}}(c_{\mathrm{QNN}_\theta}, T)$ is taken as the solution and denoted $\tilde{c}_{\mathrm{QNN}_\theta}$. Typically, a gradient descent algorithm is employed to find such a local minimum.

Two choices for the loss function are employed in this thesis. Firstly, the cross entropy or *logarithmic loss* depicted in Fig. 2.7 and defined as

$$\mathcal{L}_{\log}(\mathbf{x}, y; \theta) = -\frac{1}{M} \sum_{i=1}^{M} \left(\frac{1 + y_i}{2} \log\left[p_{+1}(\mathbf{x}_i, \theta)\right] + \frac{1 - y_i}{2} \log\left[1 - p_{+1}(\mathbf{x}_i, \theta)\right] \right), \tag{2.116}$$

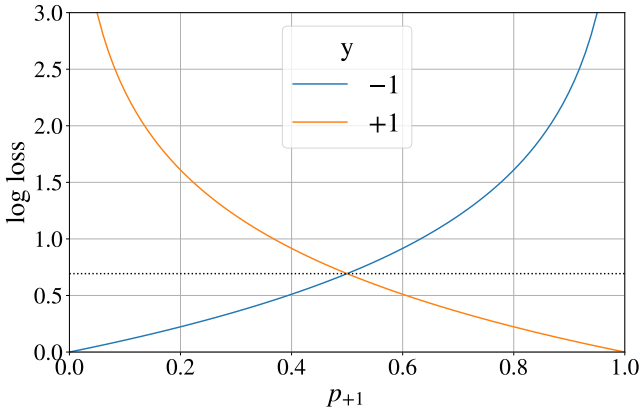


Figure 2.7: Cross entropy or logarithmic loss for a single ground truth label y and probability prediction p_{+1} for membership in class $+1$. Random guessing incurs a loss of $\log(2) \approx 0.69$.

where the probability p_{+1} is calculated like in (2.110), making this loss function a natural fit to the QNN. Secondly, a regularized *hinge loss* motivated in Section 2.3.1 is considered. It is given by

$$\mathcal{L}_{\text{hinge}}(\mathbf{x}, y; \theta, a, b) = \frac{1}{a} + \frac{C}{M} \sum_{i=1}^{M} \max\left[0, 1 - y_i \, a \left(\text{QNN}_\theta(\mathbf{x}_i) + b\right)\right], \quad (2.117)$$

where C is a fixed hyperparameter determining the trade-off between the size of the margin and misclassifications.

The gradients of these loss functions can be exactly computed by employing a parameter shift rule [36, 13, Eq. (3)] to calculate the derivative of the expectation value in (2.106) like

$$\begin{aligned} \frac{\partial \langle \varphi(\mathbf{x}, \vartheta) | \, \mathcal{F} \, | \varphi(\mathbf{x}, \vartheta) \rangle}{\partial \vartheta} = \big[& \langle \varphi(\mathbf{x}, \vartheta + \pi/2) | \, \mathcal{F} \, | \varphi(\mathbf{x}, \vartheta + \pi/2) \rangle \\ & - \langle \varphi(\mathbf{x}, \vartheta - \pi/2) | \, \mathcal{F} \, | \varphi(\mathbf{x}, \vartheta - \pi/2) \rangle \big] / 2 \end{aligned} \quad (2.118)$$

for all of the parameters ϑ making up θ and then plugging this into the analytically derived expression

$$\nabla_\theta \, \mathcal{L}_{\log}(\mathbf{x}, y; \theta) = -\frac{1}{M} \sum_{i=i}^{M} \left(\frac{1 + y_i}{2 \, p_{+1}(\mathbf{x}, \theta)} - \frac{1 - y_i}{2 \, (1 - p_{+1}(\mathbf{x}, \theta))} \right) \nabla_\theta \, p_{+1}(\mathbf{x}, \theta), \quad (2.119)$$

for the log loss as an example, where

$$\frac{\partial}{\partial \vartheta} \, p_{+1}(\mathbf{x}, \theta) = \frac{1}{2} \frac{\partial}{\partial \vartheta} \, \text{QNN}_\theta(\mathbf{x}). \quad (2.120)$$

Finite Sampling Statistics

Fundamentally, the expectation value $\text{QNN}_\theta(\mathbf{x})$ in (2.106) can not be directly accessed and only approximated as the sample mean of some finite number

of measurement outcomes. This means that even on a fully error corrected quantum computer, $\mathrm{QNN}_\theta(\mathbf{x})$ is only known up to an additive error determined by the number of measurement shots.

Denote by Q the random variable of which $\mathrm{QNN}_\theta(\mathbf{x})$ is the expectation value. Then, every measurement corresponds to one realization of the random variable and for R measurement shots, $Q_1, Q_2, ..., Q_R$ are i.i.d. according to the probability mass function

$$\mathrm{pmf}(Q; p_{+1}) = \begin{cases} p_{+1} & \text{if } Q = +1 \\ 1 - p_{+1} (= p_{-1}) & \text{if } Q = -1, \end{cases} \tag{2.121}$$

where $p_y \in [0, 1]$ is the probability of class membership $y \in \{-1, +1\}$. With this, we know that $\frac{1}{2}(Q + 1) \sim \mathrm{Bernoulli}(p_{+1})$. For the *sample mean* defined as

$$\bar{Q}_R \coloneqq \frac{1}{R} \sum_{i=1}^{R} Q_i, \tag{2.122}$$

the standard deviation is given by the *standard error of the mean*

$$\sigma_{\bar{Q}_R} = \frac{\sigma_Q}{\sqrt{R}}, \tag{2.123}$$

where σ_Q denotes the standard deviation of the distribution from which the Q are sampled. For the Bernoulli distribution, we know

$$\sigma^2_{\mathrm{Bernoulli}} = p_{+1}(1 - p_{+1}) \leq 0.25 \tag{2.124}$$

and therefore

$$\sigma^2_Q = 4\, \sigma^2_{\mathrm{Bernoulli}} \leq 1. \tag{2.125}$$

Putting this together, the standard error of the mean is upper bounded by

$$\sigma_{\bar{Q}_R} \leq \frac{1}{\sqrt{R}}. \tag{2.126}$$

This means that due to the sampling statistics,

$$R = \mathcal{O}\left(\frac{1}{\varepsilon^2}\right) \tag{2.127}$$

measurement shots are needed to determine $\mathrm{QNN}_\theta(\mathbf{x})$ up to a standard deviation equal to ε. To bring this result in line with (2.67) for comparability, Chebyshev's inequality in (2.79) can be invoked to find that for some constant a determined by the desired p, due to (2.126) the inequality

$$|\bar{Q}_R - \mathrm{QNN}_\theta(\mathbf{x})| \leq \frac{a}{\sqrt{R}} \tag{2.128}$$

holds with probability $p > \frac{1}{2}$. So to achieve an accuracy of ε in the sense of fulfilling

$$|\bar{Q}_R - \mathrm{QNN}_\theta(\mathbf{x})| \leq \varepsilon \tag{2.129}$$

with probability p, a total of

$$R = \mathcal{O}\left(\frac{1}{\varepsilon^2}\right) \tag{2.130}$$

measurement shots are needed per evaluation of $\mathrm{QNN}_\theta(\mathbf{x})$.

Analogous considerations apply to the gradient calculations needed to train the QNN, where the measurement outcomes are also restricted to $\{-1, +1\}$, as is evident from (2.118). The finite sampling statistics have the following implications for training and prediction.

Training Since the QNN approach is heuristic, it is difficult to make purely theoretical statements and empirical evidence has to be invoked too. In the following, the number of quantum circuit evaluations required for training is analyzed.

First of all, there is a linear dependence $\mathcal{O}(d)$ on the number of parameters in the variational form d for gradient descent algorithms that employ the full gradient, as every direction in ∇_θ necessitates two evaluations of quantum expectation values. Note that the SPSA optimizer would be an exception to this as it only evaluates the gradient in a random direction per step.

For QNNs trained with stochastic gradient descent (see Section 3.3.2 on page 67 for a definition) and on data that are linearly separable in feature space, there are numerical experiments in Figs. 3.22 and B.10 indicating that the speed of convergence during training is independent of the size of the dataset, as a function of the number of expectation values calculated. At first glance, this is unexpected since even for stochastic gradient descent, every datum has to be seen at least once in expectation (which would contribute a factor $\mathcal{O}(M)$ to the complexity) to make use of the full information contained in the training set. A justification for why this is not the case for QNNs comes from their linear structure in feature space derived in the following Section 2.3.1. The result obtained there implies that the trained part of the QNN, the variational form, implements a hyperplane in feature space and has no further flexibility. Then, the number of data lying on the two different sides of that linear boundary does not play a role in determining a suitable orientation. In this way, it is possible that the size of the training set M does indeed not have an impact on training.

Lastly, the goal is to achieve an ε-accurate term QNN_θ inside the decision function $\tilde{c}_{\mathrm{QNN}_\theta}$ in (2.108), under finite sampling statistics, analogous to the result for QSVMs in (2.67). When stochastic gradient descent is employed and

the number of terms the average is taken over in the loss function consequently constant, having an accuracy of ε for the sample mean necessitates $\mathcal{O}\left(1/\varepsilon^2\right)$ measurement shots. Furthermore, assume that $\mathcal{O}\left(1/\varepsilon\right)$ stochastic gradient descent steps are enough to converge to parameters θ that yield an ε-accurate classifier in that setting. Taken together, these hypotheses yield a conjectured scaling for the number of quantum circuit evaluations necessary for training

$$R_{\text{train}} = \mathcal{O}\left(\frac{d}{\varepsilon^3}\right), \tag{2.131}$$

which has to be empirically supported by numerical experiments as the above assumptions are theoretically questionable.

Prediction To get an ε-accurate term inside the sign function in $\tilde{c}_{\text{QNN}_\theta}(\mathbf{x})$ in (2.108) after training has completed, a total of

$$R_{\text{pred}} = \mathcal{O}\left(\frac{1}{\varepsilon^2}\right) \tag{2.132}$$

measurement shots are necessary due to (2.127) for a statement in standard deviation and (2.130) for an inequality that holds with probability $p > \frac{1}{2}$.

2.3.1 Relation to QSVMs

Linearity of the QNN Decision Boundary

The following section is adapted from [1, Supplementary information]. The expectation value in (2.106) can be equivalently written in terms of density matrices and the Hilbert-Schmidt inner product as

$$\text{QNN}_\theta(\mathbf{x}) = \text{tr}\left[\mathcal{W}(\theta)^\dagger \mathcal{F} \mathcal{W}(\theta)\ |\psi(\mathbf{x})\rangle\langle\psi(\mathbf{x})|\right]. \tag{2.133}$$

This motivates the following decomposition with respect to an orthogonal basis of the vector space of Hermitian matrices. As a concrete example, consider the Pauli group

$$\mathcal{P}_q = \langle X_i, Y_i, Z_i \rangle_{i=1,\ldots,q} \tag{2.134}$$

and restrict it to elements $P_\alpha \in \mathcal{P}_q$ for $\alpha \in \{1, ..., 4^q\}$ which have phase $+1$. These P_α fulfill the orthogonality condition

$$\text{tr}\left[P_\alpha P_\beta\right] = 2^q\ \delta_{\alpha,\beta}. \tag{2.135}$$

Knowing this, the two matrices in (2.133) can be decomposed like

$$\mathcal{W}(\theta)^\dagger \mathcal{F} \mathcal{W}(\theta) = \frac{1}{2^q} \sum_\alpha w_\alpha(\theta) P_\alpha \tag{2.136}$$

and

$$|\psi(\mathbf{x})\rangle\langle\psi(\mathbf{x})| = \frac{1}{2^q}\sum_\alpha \psi_\alpha(\mathbf{x})P_\alpha, \tag{2.137}$$

where the $w_\alpha(\theta)$ and $\psi_\alpha(\mathbf{x})$ are real coefficients defined as

$$w_\alpha(\theta) \coloneqq \operatorname{tr}\left[\mathcal{W}(\theta)^\dagger \mathcal{F}\, \mathcal{W}(\theta)\; P_\alpha\right] \tag{2.138}$$

and

$$\psi_\alpha(\mathbf{x}) \coloneqq \operatorname{tr}\left[|\psi(\mathbf{x})\rangle\langle\psi(\mathbf{x})|\;\; P_\alpha\right]. \tag{2.139}$$

Inserting these expansions into (2.133) and simplifying using (2.135) then yields

$$\begin{aligned}\mathrm{QNN}_\theta(\mathbf{x}) &= \operatorname{tr}\left[\frac{1}{2^q}\sum_\alpha w_\alpha(\alpha)P_\alpha\;\frac{1}{2^q}\sum_\beta \psi_\beta(\mathbf{x})P_\beta\right] \\ &= \frac{1}{2^q}\sum_\alpha w_\alpha(\theta)\psi_\alpha(\mathbf{x}).\end{aligned} \tag{2.140}$$

The significance of this result can be seen by considering the decision rule in (2.108) again. It can now be cast as

$$\tilde{c}_{\mathrm{QNN}_\theta}(\mathbf{x}) = \operatorname{sign}\left[\frac{1}{2^q}\sum_\alpha w_\alpha(\theta)\psi_\alpha(\mathbf{x}) + b\right]. \tag{2.141}$$

Comparing this to the generic decision function of an SVM classifier in feature space already encountered in (2.16) on page 12, it is apparent that the functions are of the same form. Therefore, the QNN implements a linear decision boundary in feature space.

The main difference between the QNN and QSVM now lies in the parameterized coefficients $w_\alpha(\theta)$ that make up the vector $\vec{w}(\theta) \in \mathcal{S}(2^q)$ that defines the direction of the hyperplane in feature space according to which class membership is assigned by the QNN. Because these coefficients are determined through a variational circuit $\mathcal{W}(\theta)$, they can not take on arbitrary values in general. In particular, there is no guarantee that the solution to the dual QSVM optimization problem in (2.17) on page 12 denoted by $\vec{w}^\star$ is accessible to the QNN. But $\vec{w}^\star$ is guaranteed to be found by the QSVM due to convexity and guaranteed to maximize the (regularized by slack) margin in feature space by strong duality. With this, the QNN can be interpreted as an approximation to the QSVM that finds a solution that is at best equally good, depending on the expressivity of the variational form. If $\mathcal{W}(\theta)$ can express any unitary in $\mathcal{U}(2^q)$, the QNN has access to the optimal solution $\vec{w}^\star$ and can in principle find the same hyperplane in feature space as the QSVM as a result.

Similar arguments on the relation between QNNs and QSVMs can also be found in Appendix C of [37] (from the perspective of unitaries) and Section VI of [21] (from the perspective of the reproducing kernel Hilbert space (RKHS), a concept from classical kernel theory).

See Fig. 2.8 for an example of how more expressive variational forms can find better minima.

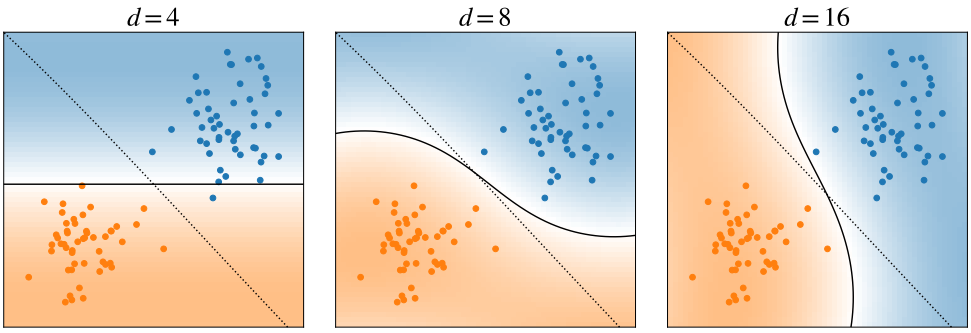


Figure 2.8: Example of *log loss* (see (2.116)) trained QNN prediction (background and solid black line) for a generic linearly separable two dimensional dataset (sklearn blobs) consisting of two classes (orange and blue points). With an increasing number of parameters d in the variational form, the QNN can find minima of decreasing loss; here $d = 4, 8, 16$ result in $\mathcal{L}_{\text{log}} \approx 0.21, 0.10, 0.09$. For reference, the optimal decision boundary for the selected feature map obtained by the QSVM is plotted as the dashed black line. As the log loss objective differs from the QSVM objective, the QNN is not expected to recover the same decision boundary even for an arbitrarily expressive variational form. An analogous plot for a different dataset and feature map is included in Fig. B.1.

The QNN & QSVM Feature Spaces Are Identical

From (2.141) it is clear that the QNN implements a linear classifier in feature space. What remains to be shown to make the comparison between QNNs and QSVMs complete is that for identical feature maps $\mathcal{E}(\mathbf{x})$, both act in the same feature space.

To this end, consider the classification function $\tilde{c}_{\text{SVM}}$ of a kernelized SVM in terms of the dual variables already encountered in (2.19) and the quantum kernel k defined for the Hilbert-Schmidt inner product as in (2.24).

The two density matrices inside the trace in the kernel function can then be expressed with respect to the P_α basis and decomposed like (2.137). Inserting

the result into (2.19) and making use of (2.135) finally yields

$$\tilde{c}_{\mathrm{QSVM}}(\hat{\mathbf{x}}) = \operatorname{sign}\left[\sum_{i=1}^{M} \alpha_i y_i \frac{1}{2^q} \sum_{\alpha} \psi_\alpha(\mathbf{x}_i)\psi_\alpha(\hat{\mathbf{x}}) + b\right] \tag{2.142}$$

$$= \operatorname{sign}\left[\frac{1}{2^q} \sum_{\alpha} \underbrace{\sum_{i=1}^{M} y_i \alpha_i \psi_\alpha(\mathbf{x}_i)}_{=:w_\alpha} \psi_\alpha(\hat{\mathbf{x}}) + b\right], \tag{2.143}$$

from which it is apparent that the vector $\vec{w}$ with entries w_α defined like above acts in the same feature space as the QNN. So the different decision rules for QNNs (2.141) and QSVMs (2.19) in fact operate in identical feature spaces when the feature map $\mathcal{E}(\mathbf{x})$ is used for both models.

As a side note, the w_α are only considered on paper and never explicitly calculated as that would entail the evaluation of explicit feature vectors, which the kernel trick avoids.

Approximate QSVMs Explicitly

The previous two subsections show the mathematical equivalence between the classification functions implemented by QNNs and QSVMs. However, this does not imply that for the same data, the two trained quantum machine learning models end up with the same decision boundary. For that to be the case, the $w_\alpha(\theta)$ in (2.141) have to be equal to the implicit w_α in (2.143). This section aims to find a suitable objective for QNN training such that the resulting QNN classifier can be interpreted as an approximation to the QSVM.

For simplicity, start with the standard linear hard margin SVM optimization problem for linearly separable data already encountered in (2.10) on page 10. In the quantum feature space from (2.23) and for a bias b fixed to zero, this is equivalent to

$$\begin{aligned} &\underset{\vec{w} \in \mathcal{S}(2^q)}{\text{minimize}} && \vec{w} \\ &\text{subject to} && y_i \left\langle \vec{w}, \psi(\mathbf{x}_i) \right\rangle_{\mathcal{S}} \geq 1 \; \forall i, \end{aligned} \tag{2.144}$$

where the coefficient $\frac{1}{2}$ and square have been dropped. The above constraints can be rewritten as

$$y_i \frac{\left\langle \vec{w}, \psi(\mathbf{x}_i) \right\rangle_{\mathcal{S}}}{\|\vec{w}\|} \geq \frac{1}{\|\vec{w}\|}. \tag{2.145}$$

Because the labels $y_i \in \{-1, +1\}$ and the quantum feature vectors $\psi(\mathbf{x}_i)$ are automatically normalized as $\|\psi(\mathbf{x}_i)\| = 1$, the left hand side is guaranteed to lie in $[-1, 1]$. Since the data are linearly separable and no misclassifications are allowed to occur for viable solutions, the l.h.s. is even in $[0, 1]$. So $c \coloneqq 1/\|\vec{w}\|$

is also restricted to $c \in [0, 1]$ and the optimization problem can be restated as

$$\begin{aligned} \underset{\vec{w} \in \mathcal{S}(2^q),\ c \in [0,1]}{\text{minimize}} \quad & \frac{1}{c} \\ \text{subject to} \quad & y_i \frac{\langle \vec{w}, \psi(\mathbf{x}_i) \rangle_{\mathcal{S}}}{\|\vec{w}\|} \geq c \ \forall i. \end{aligned} \tag{2.146}$$

Written this way, the optimization over $\vec{w}$ only determines the orientation of the separating hyperplane and is decoupled from the margin, which has size $2c$ and is symmetric around the decision boundary at zero.

We can now re-introduce the bias by considering an asymmetric margin determined by c and g fulfilling $-1 \leq c < g \leq 1$ and get

$$\begin{aligned} \underset{\vec{w} \in \mathcal{S}(2^q),\ c, g \in [-1,1]}{\text{minimize}} \quad & \frac{1}{g - c} \\ \text{subject to} \quad & \frac{\langle \vec{w}, \psi(\mathbf{x}_i) \rangle_{\mathcal{S}}}{\|\vec{w}\|} \leq c \text{ if } y_i = -1, \\ & \frac{\langle \vec{w}, \psi(\mathbf{x}_i) \rangle_{\mathcal{S}}}{\|\vec{w}\|} \geq g \text{ if } y_i = +1, \end{aligned} \tag{2.147}$$

where the y_i dependence is explicitly written in two constraints. The decision boundary of the resulting classifier is still the middle of the margin and therefore now lies at $(g + c)/2$. In summary, the minimization problem in (2.147) is an equivalent reformulation of the standard hard margin SVM in the case where the inner product is limited to values in $[-1, 1]$.

By defining $a \coloneqq 2/(g - c) > 0$ and $b \coloneqq (g + c)/(g - c)$ and combining the two constraints into one again, we get back to the more familiar form

$$\begin{aligned} \underset{\vec{w} \in \mathcal{S}(2^q),\ a \in \mathbb{R}^+, b \in \mathbb{R}}{\text{minimize}} \quad & \frac{1}{a} \\ \text{subject to} \quad & y_i \left(a \, \frac{\langle \vec{w}, \psi(\mathbf{x}_i) \rangle_{\mathcal{S}}}{\|\vec{w}\|} + b \right) \geq 1 \ \forall i. \end{aligned} \tag{2.148}$$

Finally, replacing $\frac{\langle \vec{w}, \psi(\mathbf{x}_i) \rangle_{\mathcal{S}}}{\|\vec{w}\|} \in [-1, 1]$ with $\mathrm{QNN}_\theta(\mathbf{x}) \in [-1, 1]$ from (2.106) and relaxing the constraints to the soft margin SVM formulation in (2.14) yields

$$\underset{\theta \in \mathbb{R}^d, a \in \mathbb{R}^+, b \in \mathbb{R}}{\text{minimize}} \quad \frac{1}{a} + \frac{C}{M} \sum_{i=1}^{M} \max\left[0, 1 - y_i(a \, \mathrm{QNN}_\theta(\mathbf{x}_i) + b)\right], \tag{2.149}$$

where it can be seen that the explicit minimization with respect to $\vec{w} \in \mathcal{S}(2^q)$ is replaced by the parameters in the variational form $\theta \in \mathbb{R}^d$. The resulting expression is equivalent to minimizing the empirical risk of a scaled QNN with respect to hinge loss formulated in Section 2.3.

From the derivation it is apparent that this QNN objective is identical to the QSVM objective. The caveats discussed in the previous section apply and (2.149) must be interpreted as a mere approximation to the QSVM since the values which $\mathrm{QNN}_\theta(\mathbf{x})$ can take on are always limited by the variational form. Furthermore, when going from the optimization problem in (2.148) to the one in (2.149), convexity and with it the guarantee to find the global optimum are lost.

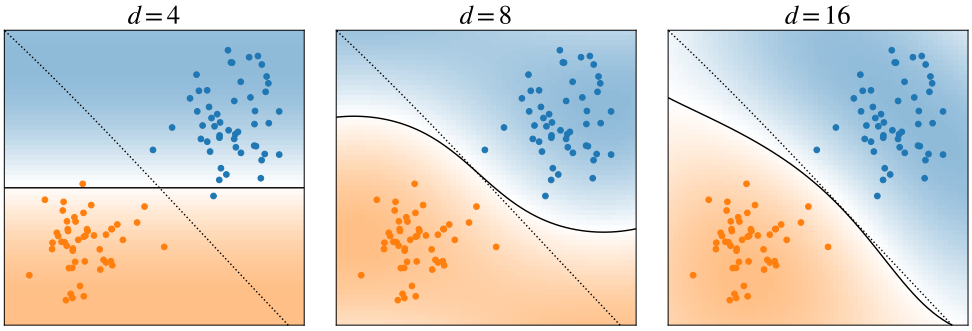


Figure 2.9: Example of *hinge loss* (see (2.117) and (2.149)) trained QNN prediction (background and solid black line) for the same data as in Fig. 2.8. The decision boundary maximizing the margin for the selected feature map, as is obtained by the QSVM, is plotted as the dotted black line. The subplots show different numbers of parameters d in the variational form. Since the objectives are aligned, the QNN decision boundary becomes more similar to the QSVM result for increasing d, unlike in Fig. 2.8. For another example considering a different dataset, see Fig. B.2.

See Fig. 2.9 for results obtained when training QNNs with respect to the objective in (2.149), compared to a QSVM employing the same feature map.

Other Considerations

With the knowledge that quantum neural networks of the form depicted in Fig. 2.6 can be interpreted as approximate QSVMs, the QNNs inherit the justifications of QSVMs. So when a quantum speedup for QSVMs like in [2] is found and gives a reason to employ a QSVM instead of a classical machine learning model, it is implied that QNNs might achieve a similar advantage. This answers the question why to deploy a quantum model in the first place.

When on the other hand the question is under which circumstances to choose a QNN over a QSVM, even though the former is only an approximation to the latter, a possible answer is that the QNN enjoys better training complexity, see the theoretical considerations in Section 2.3 and empirical results in Section 3.3.2. However, this advantage might not remain intact when QSVMs trained with the Pegasos algorithm are considered, see Section 2.2.4 for a theoretical analysis and Section 3.3.1 for numerical experiments.

As pointed out in [1], the close connection between QNNs and QSVMs also

implies that for the QNN to possibly offer a quantum advantage, the feature map circuit has to be classically intractable. This is the case since a QNN with classically tractable feature map circuit can always be replaced by a tractable classical SVM, even if the total QNN circuit including the variational form can not be efficiently simulated classically.

On current noisy hardware, any practical comparison between the models also has to take the quantum resources required into account. For an identical feature map, a QNN and QSVM call for the same number of qubits if the kernel evaluations for the QSVM are performed with the adjoint method like in Fig. 2.4. Then, the difference can still lie in the circuit depth. While the QSVM runs the possibly deep feature map circuit twice, the QNN consists of the same feature map circuit applied only once, followed by a variational form. So which of the two quantum circuits is deeper overall hinges on whether the feature map or variational form circuit has greater depth. It is known that the space of unitaries in $\mathcal{U}(2^q)$ has dimension $4^q - 1$. For the QNN, the hope is that in spite of this, the number of parameters d in the variational form needed does not grow exponentially in q. Because then, the depth of the variational form would always overtake a feature map circuit of subexponential depth. Furthermore, any polynomial speedup with respect to M in training complexity the QNN might have compared to the QSVM, would be irrelevant in that case since, training complexity of the QNN also scales with d, as discussed in Section 2.3.

2.3.2 Relation to Classical Neural Networks

In this section, classical and quantum neural networks are briefly contrasted since the naming scheme suggests a comparison. While for *mechanics* or *computation* it is known that "quantum" is more general than "classical" in the sense that "quantum" encompasses "classical", this is not the case for neural networks.

A classical fully connected feed forward neural network (NN) is a function of the form

$$\mathrm{NN}(\mathbf{x}) = \alpha^{(l)} \circ L^{(l)} \circ \ldots \circ \alpha^{(2)} \circ L^{(2)} \circ \alpha^{(1)} \circ L^{(1)}(\mathbf{x}), \tag{2.150}$$

where the L are linear functions

$$\begin{aligned} L\colon \mathbb{R}^n &\to \mathbb{R}^m \\ \mathbf{x} &\mapsto W\mathbf{x} + b \end{aligned} \tag{2.151}$$

with $W \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ and the α are nonlinear activation functions applied component wise like

$$\begin{aligned} \alpha\colon \mathbb{R}^n &\to \mathbb{R}^n \\ \mathbf{x} &\mapsto (\hat{\alpha}(x_1), \hat{\alpha}(x_2), ..., \hat{\alpha}(x_n)). \end{aligned} \tag{2.152}$$

Typical choices include the *ReLU-function* $\hat{\alpha}(z) = \max[0, z]$ or the *sigmoid-function* $\hat{\alpha}(z) = 1/(1+\exp(-z))$ [38]. For binary classification, the final output dimension of (2.150) is typically one-dimensional. When a sigmoid function is then applied, the result lies in the interval $[0, 1]$ and can be interpreted as the probability for membership in one of the two classes. The network can then be trained analogously to the QNN with respect to the log loss (see (2.116)) and the decision function takes the form in (2.111) for $b = 0$. As the optimization problem posed by NNs does not come with any guarantees, they are heuristic methods just like QNNs.

From these equations it is apparent that any additional layer in a NN introduces more nonlinearity with respect to the input $\mathbf{x}$. This is the key difference with respect to QNNs, where only the feature map is nonlinear in the data and increasing the depth of the variational form does not change the fact that the variational form ultimately implements a linear decision in feature space (see (2.141)). From a practical perspective, this means that adding more layers to a NN can enable it to fit previously not fittable data while for a QNN, which data can be fit and which cannot is always constrained by the initial feature map (see also Sections 3.1 and 3.2).

Because of the apparent mathematical dissimilarity between QNNs and NNs, the comparison between QNNs and QSVMs based on Section 2.3.1 is much more natural.

2.4 Advanced Heuristic Models

This section gives a brief overview of additional quantum machine learning models that are related to the QSVMs and QNNs as they are defined in the previous sections, but do not strictly fit that framework. Just like in the case of QNNs, the training procedures under consideration here lack the guarantees that come with QSVMs and are thus heuristic in nature.

2.4.1 Kernel Alignment

The quantum feature maps $\psi(\mathbf{x})$ defined by circuits $\mathcal{E}(\mathbf{x})$ encountered so far are fully determined by the classical input $\mathbf{x}$ and do not contain additional parameters that can be adjusted during training. When this strict definition is relaxed and the feature map circuit is allowed to contain free parameters θ like $\mathcal{E}_{\text{var}}(\mathbf{x}, \theta)$, the kernel function becomes

$$k_\theta(\mathbf{x}, \mathbf{x}') \coloneqq |\langle 0| \mathcal{E}_{\text{var}}(\mathbf{x}', \theta)^\dagger \mathcal{E}_{\text{var}}(\mathbf{x}, \theta) |0\rangle|^2, \tag{2.153}$$

where the θ can also be understood as hyperparameters in machine learning parlance. When a training procedure is employed to find values for the θ that are good with respect to some objective, this is called *kernel alignment*. Two eponymous proposals from the literature follow.

The first approach as for example championed in [17, Eq. (12) and Algorithm 1] adds an outer optimization to the kernelized SVM known from (2.17) in Section 2.2.2, resulting in the twofold optimization problem

$$\begin{aligned} \underset{\theta \in \mathbb{R}^d}{\text{minimize}} \quad & \underset{\alpha_i \in \mathbb{R}}{\text{maximize}} \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \, k_\theta(\mathbf{x}_i, \mathbf{x}_j) \\ & \text{subject to} \quad 0 \leq \alpha_i \leq C \quad \forall i, \\ & \qquad\qquad\quad 0 = \sum_i \alpha_i y_i, \end{aligned} \tag{2.154}$$

where $i, j \in \{1, 2, ..., M\}$ runs over the training set. Note that the inner maximization is convex, but the whole problem including the outer minimization is not. So kernel alignment defines a heuristic approach overall, even though the inner SVM optimization comes with theoretical guarantees. While (2.154) can not be efficiently solved in general, a local minimum can be found by a hybrid approach where the θ values are adjusted via gradient descent and the α are determined through the usual quadratic programming. The downside to this procedure is its computational expense as it means running the whole standard SVM optimization multiple times (the exact number depends on the type of optimizer and gradient type used) for each step in the outer gradient descent. Alternatively, a θ can be found by running a grid search as is often done in the context of hyperparameter tuning. Note that this approach is not suited for large d as for a constant distance between parameters under consideration on the grid, the total number of points grows exponentially in d.

A second approach has been brought forward to the quantum machine learning literature in [18, Eq. (27)]. There, the so called *kernel target alignment* defined as

$$\mathrm{KTA}(k_\theta) \coloneqq \frac{\sum_{i,j} y_i y_j \, k_\theta(\mathbf{x}_i, \mathbf{x}_j)}{\sqrt{\left(\sum_{i,j} k_\theta(\mathbf{x}_i, \mathbf{x}_j)^2\right) \left(\sum_{i,j} y_i^2 y_j^2\right)}}, \tag{2.155}$$

is maximized directly to find suitable θ, where again $i, j \in \{1, 2, ..., M\}$. Like in (2.154), the maximization of $\mathrm{KTA}(k_\theta)$ does not come with any guarantees, such that gradient descent or a similar algorithm has to be employed in the search for some local maximum. Note that this procedure also necessitates the repeated calculation of the whole $M \times M$ quantum kernel matrix at every gradient ascent step, so the quantum computational resources needed to find local optima for (2.154) and (2.155) are similar. The form of (2.155) is motivated by geometrical considerations relating the actual kernel to an unknown ideal kernel with entries

$$k^\star(\mathbf{x}, \mathbf{x}') = \begin{cases} +1 & \text{if } y = y' \\ -1 & \text{otherwise} \end{cases}. \tag{2.156}$$

For the derivation, see [18, Section IV.].

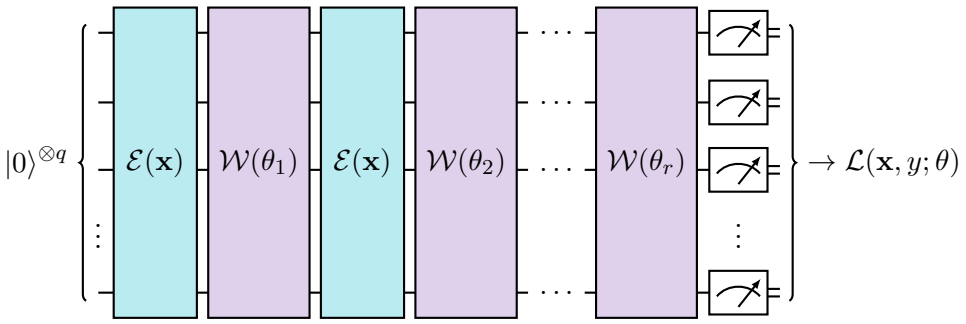


Figure 2.10: Structure of the quantum circuit making up a QAN. Feature map circuits $\mathcal{E}(\mathbf{x})$ are alternated with variational circuits $\mathcal{W}(\theta_i)$ a total of r times, such that the data encoding is repeated r times. The parameters that are fixed by the classical input and variational ones appear throughout the whole quantum circuit, unlike for the QNN where they are strictly separated. For simplicity, all of the $\mathcal{E}$ and $\mathcal{W}$ in the different layers are identical here, but this does not have to be the case in general. At the output stage of the circuit, measurement in the computational basis is performed in order to evaluate a predetermined loss function in an analogous fashion to the QNN case.

2.4.2 Quantum Alternating Networks

In this thesis, QNNs are defined like in Fig. 2.6 in Section 2.3 as the succession of a fixed feature map circuit $\mathcal{E}(\mathbf{x})$ and a trainable variational circuit $\mathcal{W}(\theta)$. Proposals to consider the more general case where feature map- and variational layers are instead alternated throughout the quantum circuit have been made in [22, 23, 24]. Applied to the zero state, such an approach results in the quantum state

$$|\Psi(\mathbf{x}, \theta)\rangle \coloneqq \mathcal{W}(\theta_r)\mathcal{E}(\mathbf{x}) \cdots \mathcal{W}(\theta_2)\mathcal{E}(\mathbf{x})\mathcal{W}(\theta_1)\mathcal{E}(\mathbf{x}) |0\rangle , \tag{2.157}$$

for r repetitions. In full analogy to the derivation in Section 2.3, the expectation value of this state with respect to the observable $\mathcal{F}$ defined in (2.107) is then given by

$$\mathrm{QAN}_\theta(\mathbf{x}) \coloneqq \langle \Psi(\mathbf{x}, \theta)| \mathcal{F} |\Psi(\mathbf{x}, \theta)\rangle , \tag{2.158}$$

and can be inserted into the QNN decision function in (2.108) to yield binary classification. In the previous, the parameter vector θ encompasses all $\theta_1, \theta_2, ..., \theta_r$ for ease of notation.

The name *quantum alternating network* (QAN) is proposed here for this kind of model as it consists of alternating layers of the sub-circuits making up QNNs. For an illustration of a generic QAN quantum circuit, see Fig. 2.10.

Differences to QNNs

Seen from the outside as black boxes, QANs are indistinguishable from QNNs: In the quantum circuits of both, there are some parameters that are fixed by

classical input data and some which are trained. Furthermore, both employ the same classification function defined in terms of analogous expectation values.

The difference becomes apparent when going back to the QSVM picture. For the QNN, the single purpose of the variational part is to determine a linear decision boundary in feature space (see Section 2.3.1). For the QAN, only the final variational layer $\mathcal{W}(\theta_r)$ right before the measurement acts in this way. The rest of the circuit can be interpreted as one large variational feature map

$$\mathcal{E}_{\mathrm{QAN}}(\mathbf{x}, \theta) \coloneqq \mathcal{E}(\mathbf{x})\mathcal{W}(\theta_{r-1})\mathcal{E}(\mathbf{x}) \cdots \mathcal{W}(\theta_1)\mathcal{E}(\mathbf{x}), \tag{2.159}$$

which can be used to define a parameterized kernel like k_θ in (2.153) in the previous section. Such a kernel can in principle be trained via the kernel alignment procedures in (2.154) and (2.155). In contrast, the kernel corresponding to a QNN with feature map $\mathcal{E}(\mathbf{x})$ is defined like in (2.26) and does not contain any trainable parameters.

Note that these considerations are mainly of theoretical interest as training a QAN like a QNN (see Section 2.3) is less computationally expensive than kernel alignment for a sufficiently large number of trained parameters and size of the training set. Since both approaches are heuristic in nature, in addition, there is no formal argument to be made in favor of kernel alignment.

The difference between QANs and QNNs is also apparent from the point of view of (non-)linearity: Because apart from measurement, the quantum operations in ideal quantum circuits are always linear on the level of the quantum states, nonlinearity with respect to the data $\mathbf{x}$ only enters through the nonlinearity present in the encoding $\mathcal{E}(\mathbf{x})$. In a QNN, all of the trained parameters occur after the encoding, so the variational part always acts linearly on the states encoding the data, see Section 2.3.1. The situation is different for a QAN, as the parameters θ_i are mixed with encoding layers $\mathcal{E}(\mathbf{x})$ throughout the quantum circuit. Therefore, the same distinction can not be made and the trained part of the circuit is allowed to act nonlinearly with respect to the data. In this sense, QANs are more similar to classical neural networks than QNNs are (see Section 2.3.2 for the relation between QNNs and classical neural networks and Figs. 3.5 and 3.8 for a numerical example of the consequences).

Additional Comments

Schuld et. al in [39] relate a special case of the circuit in Fig. 2.10 to partial Fourier series. Then, they formally show that such variational quantum circuits where the data encoding is repeated can be universal function approximators. This gives another formal motivation to consider QANs instead of QNNs, as this is not the case for QNNs.

The setting where not the variational form, but the feature map is trained, is also analyzed in [22] under the name of *quantum metric learning*. This is

closely related to the quantum kernel target alignment in (2.155) as pointed out in [18, Appendix A].

The separation between feature map and variational form layers like in the QAN definition presented here can also be lifted for an even more general approach where parameters that are fixed by the data and ones that are trained are arbitrarily mixed throughout the quantum circuit.

Chapter 3

Numerical Experiments

The goal of this chapter is to make the previous theoretical results more tangible by presenting a selection of numerical examples. Furthermore, some of the questions left unanswered by theory are investigated empirically.

As a general note, the numerical experiments presented here focus on training and not how well the models generalize on unseen data, which is the actual goal of the binary classification task defined in Section 2.1. This is done intentionally because the overall aim is not to find a machine learning model that is particularly suited for a specific task and to make statements on model selection, but to better understand the models themselves. And to that end, looking at the performance on the training set is often sufficient. Furthermore, the simple two dimensional toy data for which decision boundaries can be plotted, lend themselves to intuitive visual characterization and quantifying every result obtained there in terms of generalization performance would only complicate the discussion without adding anything directly meaningful.

For all square plots containing data and decision boundaries, the two dimensional $\mathbf{x} \in \mathbb{R}^2$ are depicted as circular points and the two colors, orange and blue, correspond to the class labels $y \in \{-1, +1\}$. When there are no axis ticks and labels, the data always lie in the interval $[0, 1]^2$. In the background, either the binary class membership in the case of QSVMs, or the continuous probability of class membership in the case of QNNs, is illustrated, as noted in the captions.

Implementation-wise, all of the experiments are realized with the Python library Qiskit [40] and make use of statevector simulations, run on classical hardware. Since the classical resources needed to perform the simulations scale exponentially in the number of qubits, the experiments performed are confined to small q, often equal to two. If nothing else is stated, the size of the training set is $M = 100$ by default. For QSVMs, the implementation in Qiskit is used in conjunction with feature map classes implementing the circuits in

Appendix A.1. For QNNs, a custom implementation is employed throughout. Unless stated otherwise, the optimization problem posed by training QNNs and QANs is always handled with the ADAM algorithm. Throughout, some additional functions like data generators from the Python library sklearn [41] are used. All of the code used to run the experiments and create the plots can be sent on request.

3.1 Significance of the Feature Map

For a binary classification task, both QSVM and QNN machine learning models can be viewed as successively performing two steps: Initially, classical data are encoded nonlinearly in the Hilbert space of quantum states, thereby acting as the feature space. Subsequently, a hyperplane in feature space that defines the decision boundary according to which the data are linearly classified is constructed. See Sections 2.2.2 and 2.3.1 for derivations. As a consequence, a QSVM or QNN can only perfectly classify a training set that is linearly separable in the feature space it operates in. In other words, the choice of the feature map always limits the capability of these quantum machine learning models. This section gives some examples for the key role, which the feature map circuit plays.

The feature maps under consideration implement different forms of *rotation encoding* (see Section 2.2.3 for an example) and are included in Appendix A.1 on page 76. They are denoted by $\mathcal{E}_A$, $\mathcal{E}_B$ (see Fig. A.1) and $\mathcal{E}_C$ (see Fig. A.2).

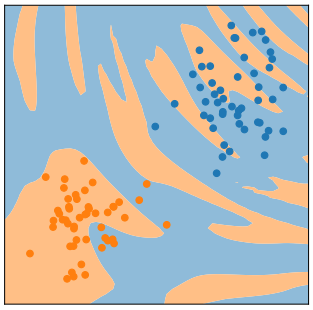


Figure 3.1: Example for a bad choice of a feature map, where the QSVM decision boundary filling the background exhibits structure that is blatantly absent from the data (sklearn blobs). Different choices for the regularization hyperparameter C, here equal to 1, also do not lead to visually better classification results, suggesting that the mismatch is caused by the feature map itself and not by overfitting. Here, the feature map $\mathcal{E}_C(\mathbf{x})$ from Fig. A.2 is employed for $r = 2$, but with factors of $9\pi^2$ instead of π in the Z rotations sandwiched between the CNOT-gates.

Figure 3.1 depicts a cautionary example for a bad choice of feature map, which results in classification boundaries that show a pattern visibly absent from the data. This is not the case because of failed training, as the convex QSVM problem is guaranteed to find the best solution obtainable with the feature map. Rather, there is a deeper mismatch between the data and the feature map.

In Fig. 3.2, an example is given where the different feature map circuits in Appendix A.1 are applied to the same data. No substantial difference between

them is visible, irrespective of whether they include entanglement or not. Figure 3.3 includes an example of how the decision boundary changes when the encoding is repeated a different number of times l, as defined in (2.102). Visibly, increasing l results in a more tightly curved decision boundary.

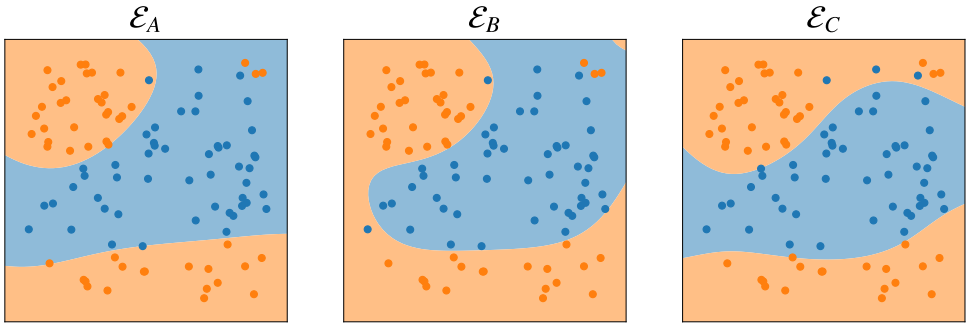


Figure 3.2: For two dimensional toy data shown here (the data are artificially generated as described at the beginning of Section 3.3 from a fourth feature map, such that they are not necessarily linearly separable in feature space for the feature maps depicted) and QSVMs with constant hyperparameters $C = 1$ and $r = 2$, the different feature maps in Appendix A.1 perform very similarly. In this setting, there is significant difference between circuits that implement entangled states ($\mathcal{E}_B$ and $\mathcal{E}_C$) and those that do not ($\mathcal{E}_A$).

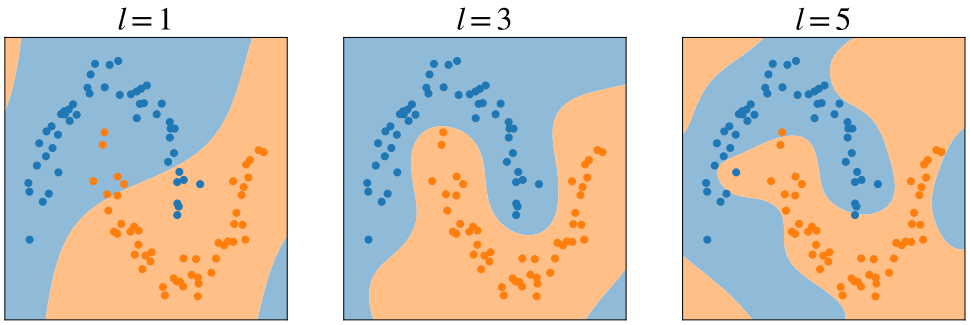


Figure 3.3: The feature maps under consideration are composed of l repetitions of a single sub-circuit like in (2.102). Not modifying the QSVM ($C = 10$) apart from varying l, the decision boundary changes. A larger l is not necessarily more capable of fitting data and there is no generally applicable rule to find a suitable choice. In this example, the $\mathcal{E}_B$ feature map from Fig. A.1 is employed and the toy data depicted come from the moons set in sklearn.

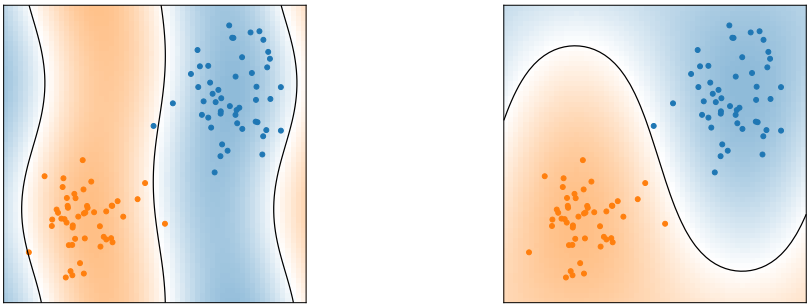
3.1.1 Affine Transformation

It has been previously noted in Section 2.2.3 that for a rotation encoding, it is important to pre-process the data such that the angles encoding the classical components lie in the interval $[0, 2\pi]$. Instead of fixing this in advance, a more general pre-processing can also be learned for a feature map that is

parameterized like in Section 2.4. As a simple example, consider an affine transformation on the input like

$$\mathcal{E}_{\text{affine}}(\mathbf{x}; A, c) \coloneqq \mathcal{E}(A \cdot \mathbf{x} + c), \tag{3.1}$$

where $A \in \mathbb{R}^{s \times s}$ is diagonal for simplicity and $c \in \mathbb{R}^s$. The feature map can then be either optimized in an outer loop for the QSVM case (see (2.154)), or A and c are trained together with the θ in a QNN. An example for the latter setting is illustrated in Fig. 3.4, where the decision boundary found by the QNN is compared between a rigid and a learned classical pre-processing of the data.



(a) There are no free parameters in the feature map $\mathcal{E}(\mathbf{x})$ and the data are pre-processed such that all rotation angles are in $[0, 2\pi]$. The final loss on the training set is $\mathcal{L}_{\text{log}} \approx 0.13$.

(b) A diagonal affine transformation $A \cdot \mathbf{x} + c$ is learned in $\mathcal{E}_{\text{affine}}(\mathbf{x}; A, c)$, which enables the QNN to reduce the minimal value of the training objective to $\mathcal{L}_{\text{log}} \approx 0.07$.

Figure 3.4: The probability prediction of the trained QNNs is plotted in the background and the decision boundary included as the black line(s). In both sub-figures, the feature map is $\mathcal{E}_A$ from Fig. A.1 for $r = 1$. The variational form quantum circuit is $\mathcal{W}_A$ from Fig. A.3 for $d = 16$, a rotation block consisting of R_x and R_y gates, and CNOT-gates in the entanglement layer. Training is done with respect to the logarithmic loss in (2.116) and gradient descent is performed for a total of 100 steps in ADAM, which is sufficient for convergence. The data are the same as in Fig. 3.1 and generated with sklearn. In Fig. B.3, an otherwise identical example, but for the same feature map as employed in Fig. 3.1, is given.

3.1.2 Quantum Alternating Network

As discussed in Section 2.4.2, a QAN where the encoding is repeated r times can be interpreted as a deep parameterized feature map $\mathcal{E}_{\text{QAN}}(\mathbf{x}; \theta_1, ..., \theta_{r-1})$, followed by a single variational layer $\mathcal{W}(\theta_r)$ implementing a linear decision boundary in feature space in the same way a QNN does. Then, the main difference to the QNN is that the mapping to feature space is trainable since it includes variational parameters. With this, a natural question to ask is how malleable such a variational feature map is with respect to data in practice.

Because any added layer in a QAN introduces additional nonlinearity in the data, increasing the depth potentially enables the model to fit data it previously could not. Figures 3.5 to 3.7, which all stem from the same numerical experiment, are different illustrations of how the fitting capability of QANs increases with their depth. In contrast, this is not the case when adding trained parameters to the variational form of a QNN that is limited by the non-parameterized feature map it employs (see Figs. 3.8 and 3.9 in the next section).

The QAN architecture employed throughout the section alternates feature map and variational form quantum circuits like in Fig. 2.10 and the sub-circuits depicted there are single layers of $\mathcal{E}_A$ (no entanglement, $r = 1$) and $\mathcal{W}_A$ (R_x and R_y in the rotation block and CNOT in the entanglement block) as found in Appendix A. In the experiments, training is repeated for different random initializations of the variational parameters. In Fig. 3.5, an average and confidence interval over these initializations is shown, while Figs. 3.6 and 3.7 depict results obtained from the single initialization achieving the lowest value of the training objective. In all cases, the QANs are trained with the ADAM algorithm with a learning rate equal to 0.1 and for a total of 100 steps, which is chosen to be sufficient for the training loss to converge to a stable value. The results included in the plots are obtained after the training procedure has concluded. The dataset considered in the experiments is the moons set from sklearn.

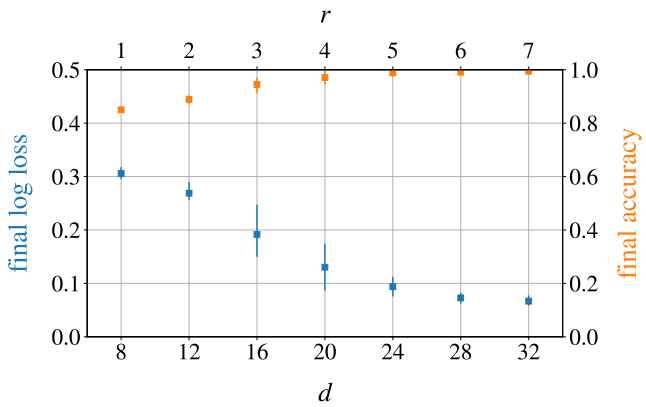


Figure 3.5: Increasing the number of times r the feature map is repeated, yields QANs that achieve lower logarithmic loss and better classification accuracy on the training set after training has completed. For the architecture chosen, the number of trainable parameters is $d = 4(r + 1)$. The squares mark the mean value and the error bars correspond to the 5 and 95 percentile over a total of 12 different random initializations of the trained parameters.

The experiments suggest that QANs are in danger of overfitting the training set when they consists of too many layers. Consequently, the depth r and with

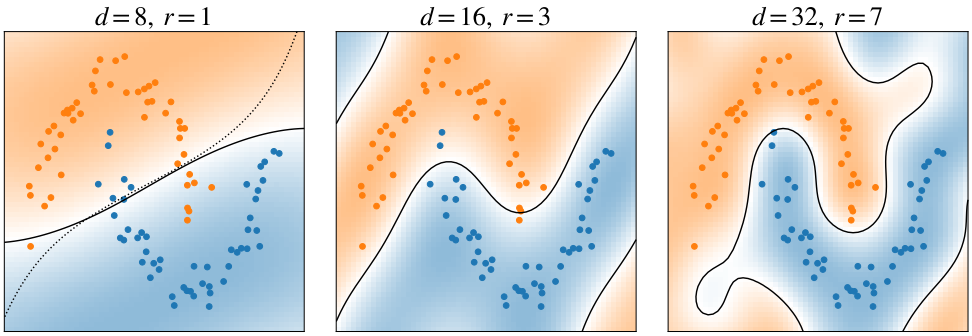


Figure 3.6: In the background, the QAN class membership prediction after training has completed is plotted, and the associated decision boundary is included as the solid black line(s). For the case where $r = 1$, which is equivalent to having a non-parameterized feature map, the corresponding QSVM (with $C = 1$) decision boundary is drawn as the dotted line. Visually, larger r leads to a better fit of the data, which is in line with the quantitative results in Figs. 3.5 and 3.7, which are based on the same set of experiments. Analogous plots only differing in the dataset under consideration can be found in Figs. B.4 and B.5.

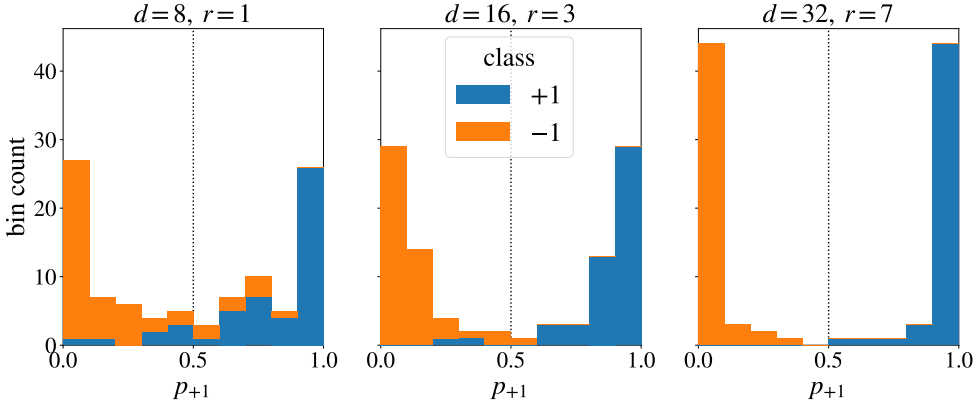


Figure 3.7: The underlying experiments are the same as in Figs. 3.5 and 3.6. The histogram bins contain the probability p_{+1} for class membership $+1$ according to which the class is assigned conditioned on whether the inequality $p_{+1} > 0.5$ (dotted line) is true or not, like in (2.111). For $r = 7$, no misclassifications occur on the training set and the extreme bins are dominant.

it, number of trained parameters d, are important hyperparameters to tune when performing model selection. Framed more positively, QANs are powerful tools to fit data, provided these hyperparameters are adequately chosen.

3.2 Significance of the Variational Form

A QNN consists of a feature map followed by a variational form in which all of the trained parameters are located. As noted in the previous section, such a QNN can in principle only perfectly classify a training set if the data are linearly separable in the feature space it operates in (see Section 2.3.1 for the

derivation) and whether this is the case or not is solely determined by the feature map.

However, the variational form can also limit the performance of a QNN: For perfect classification of the training set, apart from linear separability of the data in feature space, another necessary condition is that the variational form can in fact implement a decision boundary that separates the feature vectors. More concretely, the variational form has to be capable of parameterizing the coefficients $w_\alpha(\theta)$ in (2.141) such that they define a separating hyperplane in feature space. In this setting, the expressivity of the variational form relates to the flexibility it has in specifying $w_\alpha(\theta)$. In the extreme case when the variational form can prepare any unitary, the $w_\alpha(\theta)$ can take on any real value (see (2.138)).

One significant factor for the expressivity of a variational form, is the number of trainable parameters d it contains, which is coupled to the depth of the circuit. Under the assumption that the extra static entanglement layers consisting of CNOTs can be neglected in this discussion, for the variational form $\mathcal{W}_A$ (see Fig. A.3) considered in the experiments, a larger d increases the expressiveness of the circuit. Since in that case, a deeper variational form can express a shallower one by setting the angles in the additional rotation gates to zero. In Figs. 2.8 and 2.9 in Section 2.3.1 on page 39 and on page 42, illustrations of this are included. There, increasing d improves the final training loss and classification accuracy. Such a result indicates that the variational form and not the feature map immediately constrains the performance of the QNN in those instances.

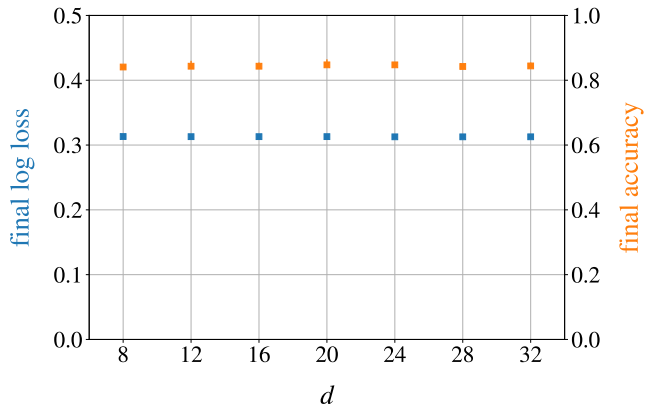


Figure 3.8: For this example (see the main text for the experimental setup), increasing the number of variational parameters d in the QNN does not improve the achievable training loss and accuracy, implying that the QNN is limited by the feature map. This is in contrast to what is obtained in Fig. 3.5 for the QAN. The error bars marking the 5 and 95 percentile over a total of 12 random initializations of the trained parameters are obscured by the squares, which mark the mean.

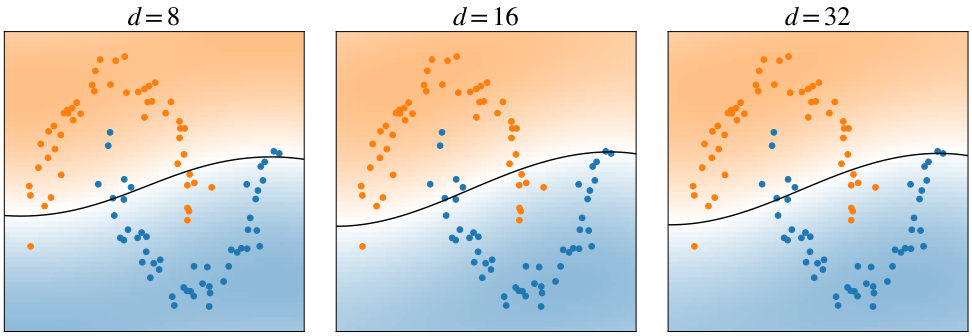


Figure 3.9: For the same numerical experiment as depicted in Fig. 3.8, the QNN prediction on two dimensional toy data (moons from sklearn) shows that not just the loss (as depicted in Fig. 3.8), but also the decision boundary (solid black line) is virtually identical for the different values of d. This implies that in this example (see the main text for the experimental setup), the QNN is limited by the feature map and not the variational form. Note the difference to Fig. 3.6, where increasing the number of parameters results in an increasingly complex decision boundary.

As is implied by the previous discussion of the role of the feature map in a QNN, larger values of d do not necessarily lead to better training performance. Figures 3.8 and 3.9 show the case when the loss and accuracy achieved on the training set do not change for increasingly expressive variational forms, which indicates that the feature map is restricting. As increasing d does not improve the result, the limiting factor is not the hyperplane in feature space, but the arrangement of the feature vectors in that space. Note the difference to Figs. 3.5 and 3.6, which depict analogously obtained results for QANs instead of QNNs.

For the numerical experiments underlying Figs. 3.8 and 3.9, the QNN circuit is held constant apart from the number of parameters d in the variational form. The feature map employed is $\mathcal{E}_A$ (see Fig. A.1) for $r = 1$ and no entanglement and the variational form is $\mathcal{W}_A$ (see Fig. A.3) with rotation blocks consisting of R_x, R_y and CNOT-gates in the entanglement layer. The training procedure employed here is identical to the one in Section 3.1.2, so 100 steps with a learning rate equal to 0.1 are performed by ADAM and care is taken that this number is sufficient for training to have converged. Likewise, the moons dataset is considered.

Apart from the depth of the variational form coupled to d, the specific architecture of the quantum circuit is key in determining the achieved expressivity. Figure 3.10 shows an example for how variational forms that include the same number of trainable parameters, but differ in the types of rotation gates they consist of, cause the training procedure to converge to different final loss values. Illustrating this fact in another way, Fig. 3.11 includes the decision boundary of a QNN with such a restrictive variational form and compares it to the performance of a QSVM that implements the same feature map.

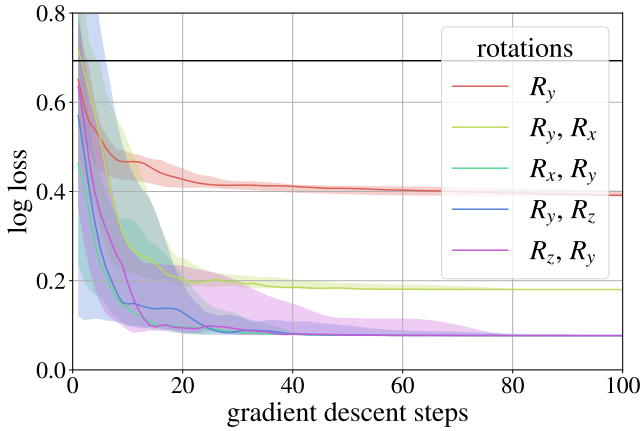
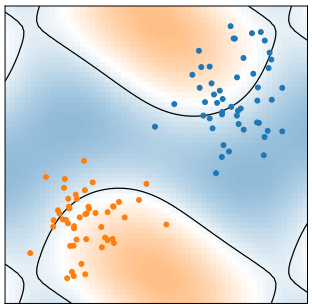
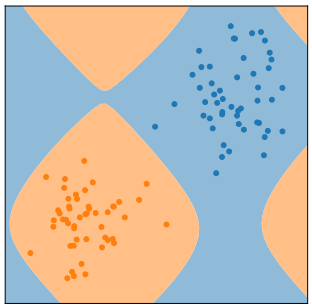


Figure 3.10: For variational forms of the same architecture $\mathcal{W}_A$ defined in Fig. A.3, the types of rotation gates included in the rotation block greatly influence the quality of the final decision boundary. Throughout, the entanglement layers are identical and exclusively consists of CNOT-gates. The solid lines trace the median loss over 12 random initializations of the trained parameters and the shaded areas correspond to the values between the 15.9 and 84.1 percentile. These values are chosen to approximate the probability contained within one standard deviation around the mean of a Gaussian distribution. In black, the log loss incurred by random guessing $\mathcal{L}_{\text{log}} = \log(2)$ is included. From the plot, it is apparent that the number of training steps performed is sufficient for convergence. So the difference in the final loss obtained cannot be attributed to a lack of training steps in ADAM (learning rate 0.1), but is the result of the difference in expressivity, which the variational forms have.



(a) QNN prediction after training (100 steps in ADAM with learning rate $= 0.1$) has completed for the $\mathcal{W}_A$ variational form configured to only contain R_y- and CNOT-gates.



(b) QSVM decision boundary for the same feature map and $C = 1$. Here, all classes are correctly predicted.

Figure 3.11: These plots are complementary to Fig. 3.10. Like for the red graph converging to a higher loss there, the QNN in (a) fails to find a good decision boundary. The QSVM prediction in (b) shows that this is the case because of a lacking variational form and not due to the feature map, which as the QSVM achieves zero misclassifications, evidently maps to linearly separable clusters in feature space. In both sub-figures, the same feature map ($\mathcal{E}_A$ from Fig. A.1 for $r = 1$) and identical data (sklearn blobs) are considered.

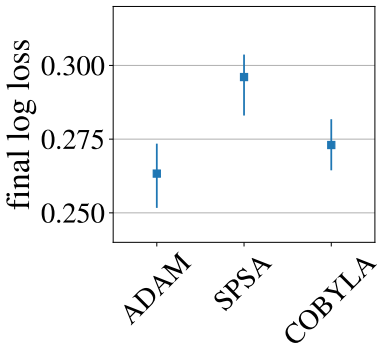


Figure 3.12: Typical example of the log loss obtained after training a QNN with three different optimizers, namely ADAM (learning rate 0.1, 100 steps), SPSA (default parameters in Qiskit, 1000 steps), and COBYLA (default parameters in Qiskit, 1000 steps). The number of steps (not depicted) is chosen to achieve comparable runtimes.

Here and in the rest of the chapter, the QNN training is exclusively realized with the ADAM optimizer for the following reasons. The COBYLA and SPSA optimizers tend to perform worse for a similar training duration in these experiments, see a typical example in Fig. 3.12. Using ADAM is also beneficial for interpretability as it is deterministic when the loss is calculated over the whole dataset, which corresponds to gradient descent (this does not hold true for the *stochastic* gradient descent introduced in the later Section 3.3.2). Furthermore, the fact that ADAM is gradient based adds transparency, as for example the convergence to a local minimum can always be verified by checking whether the gradients are close to zero.

In this section, the uncertainty arising from finite sampling statistics is neglected and all expectation values as exactly calculated by the classical statevector simulation are directly employed.

3.3 Impact of Finite Sampling Statistics

As discussed at length in Section 2.2.3 for QSVMs trained via the dual, Section 2.2.4 for QSVMs trained with Pegasos and Section 2.3 for QNNs, the exact quantum mechanical expectation values that underlie these models are fundamentally inaccessible to any quantum computer by Born's rule. Instead of obtaining them directly, they can only be evaluated approximately, by estimating the expectation values as the sample mean of a large number of measurement shots. When analyzing the performance these models achieve, the statistical uncertainty that is invariably injected into the algorithms in this fashion has to be taken into account, unless it can be shown to be of negligible effect. All of the results here are obtained by running classical statevector simulations in Qiskit, such that the exact expectation values are in fact available. Therefore, the statistical uncertainty has to be modeled separately. It is implemented as an additive error on the expectation values, as noted in detail for the different models in the corresponding sections.

The following presents numerical experiments that investigate some of the consequences the finite sampling statistics have, especially on the training procedure and the training loss achieved once it has completed. The focus is on QSVMs that are fit with the PEGASOS algorithm and QNNs. For QNNs, another kind of statistical uncertainty is introduced on top of the finite sampling, in the form of stochastic gradient descent.

Artificial Data

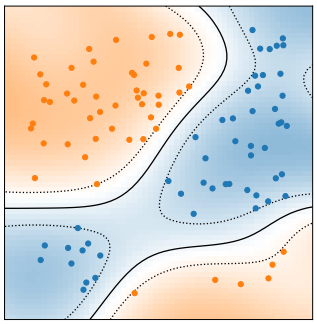


Figure 3.13: Example for an artificially generated dataset of size $M = 100$ and margin $\mu = 0.2$. Class membership is determined by a QNN, the probability prediction of which is depicted as the color of the background. The solid lines mark the QNN decision boundary and the dotted lines the margin. The precise QNN circuit employed here is like in Figs. 3.8 and 3.9 in Section 3.2 for $d = 8$.

The experiments in this section are designed with the intention of suppressing factors that distract from the training procedure. As elaborated in Section 3.1, the feature map circuit is key to the capability of both QSVMs and QNNs to fit a training set. To take it out of the equation, all of the toy datasets under consideration here are artificially generated from QNNs in the following way: For a fixed QNN architecture with a set feature map and randomly chosen fixed variational parameters θ, a total of M data $\mathbf{x}_i$ are randomly sampled from the uniform distribution on the s-dimensional interval $[0, 1]^s$ and assigned a class membership y_i according to $\tilde{c}_{\mathrm{QNN}_\theta}(\mathbf{x}_i)$. In this way, a training set T and set of training labels L are obtained. From Section 2.3.1 on the linearity of the QNN decision boundary, it is then clear that data generated in this way are always linearly separable in feature space. Therefore, both QNNs of the same architecture initialized with different random $\tilde{\theta}$ and QSVMs employing the same feature map are guaranteed to be capable of obtaining zero misclassifications on the training set. Furthermore, a margin can be introduced by excluding data for which the probability of class membership lies inside a set interval around the decision boundary. A similar approach has been chosen in [1, Fig. 3 B] and [2, 17] include further examples from the literature where the data are artificially generated with the feature map in mind. In Fig. 3.13, a two dimensional example dataset artificially generated in this way is plotted and the precise algorithm is formulated in Algorithm 2 on page 83.

3.3.1 Training Noisy QSVMs with Pegasos

At the heart of all QSVMs, there is the expectation value in the quantum kernel function, see (2.24). Since this expectation value has to be estimated from finite sampling, invariably there is uncertainty associated with it and this statistical noise can be modeled by employing a random variable

$$\eta \sim \mathcal{N}(0, \sigma^2), \tag{3.2}$$

and adding random samples of it to the exact quantum kernel

$$\bar{k}_R(\mathbf{x}, \mathbf{x}') = \bar{k}_{\sigma^{-2}}(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') + \eta, \tag{3.3}$$

where $\bar{k}_R(\mathbf{x}, \mathbf{x}')$ corresponds to $R = 1/\sigma^2$ measurement shots like in (2.36) in Section 2.2.3 on page 17. The relationship between R and θ is justified by the same considerations on the standard error of the mean as in Section 2.3. By the central limit theorem, the noise model in (3.3) is valid for sufficiently large R because then, the distribution of the sample mean is approximated well by a Gaussian. Note that in the following, calculating the term in (3.3) is denoted one kernel evaluation and means something different than the number of quantum circuit evaluations R necessary to determine (3.3) up to the standard deviation of η.

Experimental Setup

In the numerical experiments underlying the results depicted in Figs. 3.14 to 3.17, the feature map quantum circuit $\mathcal{E}_C$ from Fig. A.2 for $r = 2$ determines the quantum kernel function $k(\mathbf{x}, \mathbf{x}')$ by the definition in (2.26). Furthermore, the size of the artificially generated dataset is fixed to $M = 1000$ with a margin of $\mu = 0.1$ throughout (unless in Fig. 3.17 where $\mu = -0.4$) and the regularization parameter is set to $C = 1000$. The latter two choices are connected: The artificial data are guaranteed to be linearly separable in feature space, such that a large C resulting in an optimization problem that approaches the hard margin case (which is equivalent to $C \to \infty$) is suitable. An added benefit is that a relatively large C also speeds up training drastically, as is evident from the relationship between C and the number of kernel evaluations included in Fig. 2.5 in Section 2.2.4 on page 27. The number of steps t the algorithm is run for is varied throughout the section and given as in the axis labels. In experiments where the noise free case ($\sigma = 0$) is compared to the noisy case, the standard deviation $\sigma = 0.1$ is considered for the noisy runs. This value is chosen since for the $R = 100$ measurement shots $\sigma = 0.1$ corresponds to, the noise model in (3.2) is well justified. Note that the error in (3.3) is exclusively applied to the kernel evaluations in line 14 of Algorithm 1 and not in the quantities that are only calculated to be plotted like the hinge losses and predictions underlying the accuracy values in Figs. 3.14 to 3.17.

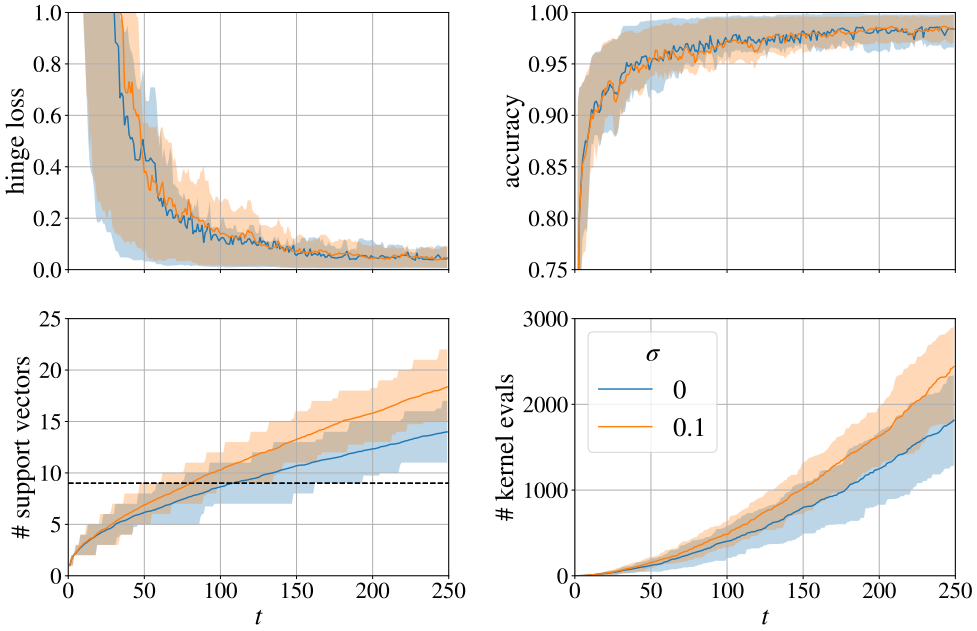


Figure 3.14: The experimental setup is as given in Section 3.3.1. A comparison between noiseless ($\sigma = 0$, blue) and noisy ($\sigma = 0.1$, orange) training of a QSVM with the PEGASOS algorithm is depicted. Throughout all subplots, the solid lines mark the mean value and the shaded areas the interval between the 15.9 and 84.1 percentile over a total of 100 random runs of the algorithm. By construction, a loss of zero and accuracy of 1 are feasible. The hinge loss and accuracy (upper subplots) incurred on the training set do not differ for the two cases in any statistically significant way. However, the number of support vectors and number of kernel evaluations (lower subplots) do, as both quantities grow more quickly in the noisy case. Note that these quantities are coupled by line 14 in Algorithm 1. For the number of support vectors, the value obtained by an identical QSVM trained with the dual optimization is included as the dashed black line. Both the noisy and noiseless instances of PEGASOS find a larger number of support vectors for large enough t. For this dataset of size $M = 1000$, constructing the full kernel matrix K takes 500,500 evaluations of the kernel function. In Fig. B.6, the same plot for t up to 1000 is included, which does not change the qualitative behavior.

Experimental Results

In Fig. 3.14, the training progress of the PEGASOS algorithm is compared between samples of a noiseless and an example noisy classifier. For the two cases, the evolving hinge loss and classification accuracy on the training set are virtually indistinguishable. However, the number of support vectors grows more quickly (and with it the number of kernel evaluations as well), when statistical uncertainty is incorporated. The reason for this is that the condition in line 14 in Algorithm 1 is fulfilled more often then.

The observation that the hinge loss converges to the same value for the noiseless and noisy case has the following important implication. Note that by (2.14), the hinge loss is equal to the objective in the primal SVM optimization. Therefore, it is directly linked to δ in (2.94). So the fact that the speed of convergence of

the hinge loss the algorithm attains seems to not be adversely affected by the statistical uncertainty, gives a plausible justification for the assumption made above (2.97) in Section 2.2.4 on page 29.

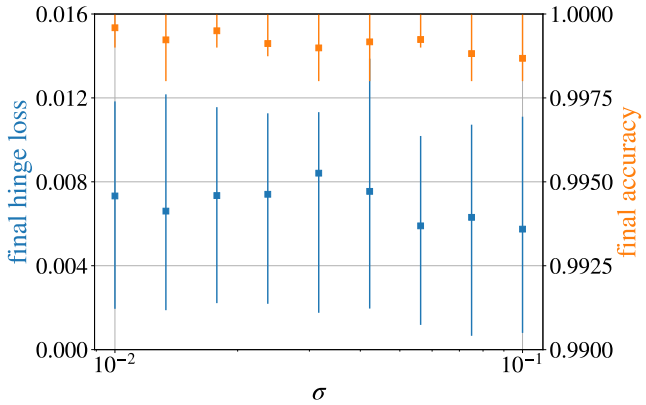


Figure 3.15: For the linearly separable data described in Section 3.3.1 with $M = 1000$ and $\mu = 0.1$, Pegasos is insensitive to the additive error η of standard deviation σ as defined in (3.3). In this experiment, the number of iterations is fixed to $\tau = 1000$, corresponding to less than 25,000 kernel evaluations in all instances. The squares mark the mean value and the error bars the 15.9 and 84.1 percentiles after training has completed, over 100 random runs of the algorithm. Note the scaling of the two y-axes, from which it is clear that both the loss and accuracy are close to the ideal value in all cases. Also important to observe is that there is no statistically significant trend along the x-axis.

A more comprehensive analysis of the impact which the additive error η in (3.3) has on the loss value the algorithm converges to, is included in Fig. 3.15. There, it is evident that Pegasos is very resilient with respect to the noise model in (3.2), as the final hinge loss and final accuracy do not show a statistically significant trend for the different standard deviations under consideration.

Figure 3.16 shows that in the experimental setting under consideration, the size of the training set M does not play a role for the convergence of the algorithm. This is in line with the absence of M in the $R_{\text{train}} = \mathcal{O}\left(1/\delta^5\right)$ scaling derived in (2.97).

Taken together, the experiments suggest that even for $M = 1000$, which is a moderately sized dataset, there is promise for the Pegasos algorithm as the number of kernel evaluations needed to run it is consistently orders of magnitude smaller than what would be needed to train the QSVMs via the dual, while the quality of the final results is similar. Because the quantum kernel evaluations are the only part of QSVMs that is run on a quantum computer, they pose the likely bottleneck of the models, so this fact potentially has far reaching implications on the feasibility of QSVM models. The Pegasos algorithm holds the promise that it enables the training of QSVMs even when the classical approach via the dual becomes practically intractable. Furthermore, from

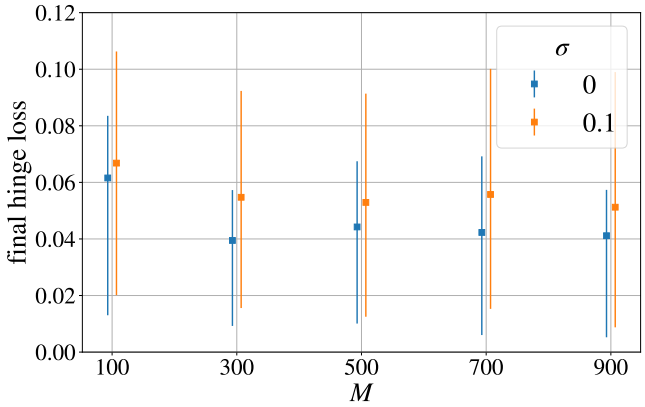


Figure 3.16: For the experimental setup defined at the beginning of Section 3.3.1, there is no statistically significant relationship between the size of the dataset M and the hinge loss obtained after training has completed for a total number of steps $\tau = 250$. In the figure, the squares mark the mean value and the error bars the 15.9 and 84.1 percentile for 100 runs of the PEGASOS algorithm. The pairs of blue and orange values next to each other in fact are of the same value M indicated by the ticks, but are slightly offset along the x-axis for legibility.

the theoretical results concerning the number of quantum circuit evaluations necessary for training, it is clear that the advantage PEGASOS has over the dual only grows more pronounced when larger datasets are considered: to get an ε-accurate classifier, training via the dual necessitates $\mathcal{O}\left(M^{4.67}/\varepsilon^2\right)$ quantum circuit evaluations (see Section 2.2.3), while the PEGASOS algorithm entails $\mathcal{O}\left(M^2/\delta^3\right)$ or $\mathcal{O}\left(1/\delta^5\right)$ circuit evaluations (see Section 2.2.4). However, the practical relevance of these theoretical worst case results remains an open question. While the numerical experiments considered in this section do not quantitatively confirm the scaling for PEGASOS, they imply that at least in the experimental setting under consideration, the algorithm is in fact very resilient to statistical uncertainty, indicating that the theoretical results are pessimistic compared to the real world performance.

Single Measurement Shot

Motivated by the resilience to statistical uncertainty observed in the previous section, the noise model in (3.3) is replaced by

$$\tilde{k}(\mathbf{x}, \mathbf{x}') \sim \text{Bernoulli}\left(k(\mathbf{x}, \mathbf{x}')\right), \tag{3.4}$$

which corresponds to the extreme case where only a single measurement shot is taken ($R = 1$) for every evaluation of the quantum kernel function. In this case, the sampling noise in PEGASOS is maximal.

For the numerical experiment presented in Fig. 3.17, the setup is like described in Section 3.3.1, apart from the margin in the data generation process, which is set to $\mu = -0.4$. Then, the data are not linearly separable in feature

space and the QNN underlying the data generation process itself misclassifies approximately 20% of the training set. This corresponds to the more realistic case where the situation in feature space is intuitively comparable to Fig. 2.2: while the two classes still exhibit a structure that can be exploited by a linear decision boundary, they are not linearly separable.

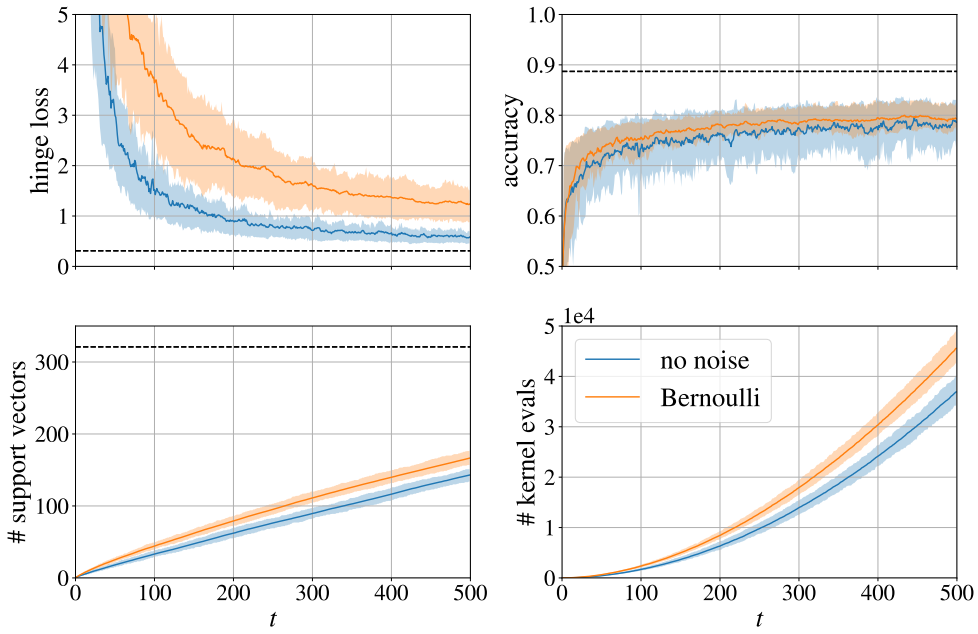


Figure 3.17: In this figure, the same quantities as in Fig. 3.14 are included. However, here the two extreme cases $R \to \infty$ (no noise) and $R = 1$ (Bernoulli) are compared. The solid lines mark the average and the shaded areas the confidence interval between the 15.9 and 84.1 percentile over 100 random runs of the Pegasos algorithm. For the hinge loss, accuracy and number of support vectors, the dashed black lines mark the values which a QSVM trained via the dual obtains for the same feature map and identical hyperparameter $C = 1000$. In that case, the full kernel matrix K has to be constructed, requiring 500,500 kernel evaluations. Note that even for the edge case depicted here, the qualitative behavior in the lower two subplots is like in Fig. 3.14. But unlike there, here the separation between the noisy and noiseless case is statistically significant for the loss and accuracy in the upper two subplots. For an extension of the experiment to larger t, see Fig. B.7.

From Fig. 3.17 it is clear that even under these adversarial conditions, Pegasos still manages to find a respectable solution to the SVM optimization problem. A plausible explanation as to why the algorithm performs this well even when only a single measurement shot is conducted, is that the sum in line 14 in Algorithm 1 still gives meaningful statistics as it goes over a large number of quantum kernel evaluations, with one of the entries of $k(\cdot, \cdot)$ being fixed. Also note that while the hinge loss incurred on the training set is consistently larger, surprisingly, the single shot *Bernoulli* case on average achieves a higher accuracy than the instances of the noiseless classifier. Why this is the case is

an open question, but the larger number of support vectors, which the noisy classifiers include, is likely part of the answer.

An interesting question for further experiments is how the generalization performance on the test set E is affected in all of these cases where only the training set is considered.

3.3.2 Training Noisy QNNs with (Stochastic) Gradient Descent

In line with the discussion in Section 2.3, for QNNs the statistical noise arising from finite sampling can be modeled by adding independent realizations of the random variable η distributed like

$$\eta \sim \mathcal{N}(0, \sigma^2), \tag{3.5}$$

onto all occurrences of the expectation value $\mathrm{QNN}_\theta(\mathbf{x})$ in (2.106) and all instances of expectation values making up the gradient evaluations in (2.118). The standard deviation σ can be related to the number of measurement shots by (2.126), such that fixing it to a constant value is equivalent to always assuming the worst case standard deviation for $R = 1/\sigma^2$ measurement shots. Note that this noise model is only valid for sufficiently large R, when by the central limit theorem the distribution of the sample mean approaches a Gaussian.

Stochastic Gradient Descent vs. Gradient Descent

In addition to the statistical noise modeled in (3.5), another source of uncertainty is introduced when QNNs are trained with *stochastic gradient descent* (SGD) as opposed to *gradient descent* (GD). For gradient descent considered exclusively up to here, the loss function $\mathcal{L}$ to be minimized is averaged over the entire training set T of size M, like for example in (2.116) and (2.117) on page 35 and on page 36. In contrast, for stochastic gradient descent, the average is not taken over all $i \in \{1, 2, ..., M\}$, but only a subset of indices sampled at random without replacement. The number of samples is constant and is denoted as the *batch size*. When the loss is not averaged over the whole training set, but only a part of it, it is called the *batch loss*. In Fig. 3.18, the total loss is compared to the batch loss for identical data and QNNs. Visually, the batch loss is more noisy, but has the same expectation value as the total loss calculated over the whole training set. This does not come as a surprise since the data included in a batch are independently sampled in subsequent steps, so the two losses are equal in expectation.

Throughout the section, the number of quantum expectation values occurring during training is given by

$$\#\text{ expectation values} = (2d + 1) \cdot \text{batch size} \cdot \#\text{ (S)GD steps}, \tag{3.6}$$

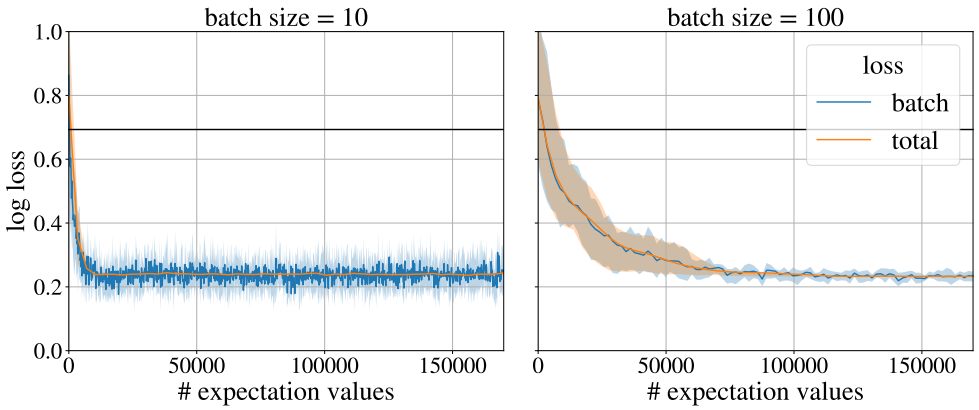


Figure 3.18: In the two subplots corresponding to different batch sizes and both depicting QNN training, the loss averaged over the whole training set of size $M = 500$ (orange) is compared to the batch loss (blue), which is an average over 10 and 100 data respectively. From the plot it is apparent that on average, there is close agreement between these two quantities, as is expected since the data considered in the batch loss are randomly sampled. The solid lines and shaded regions correspond to the mean and interval between the 15.9 and 84.1 percentile over 12 initializations.

where in the first factor the term $2d$ comes from the gradient calculations for all directions in θ and $+1$ from the loss evaluation. Note that this quantity is related to the number of physical quantum circuits that have to be evaluated by R, the number of measurement shots performed per expectation value. Furthermore, it is identical to the number of virtual quantum circuits that are evaluated with the statevector simulation in Qiskit, which has direct access to the exact expectation values. The motivation to consider the training convergence with respect to the number of expectation values is that when this quantity is equal, training takes approximately the same time under the assumption that R is the same in the compared cases. In this way, the number enables a fair comparison between different training runs, even for experiments where the size of the dataset M or batch sizes differ.

Experimental Setup

Throughout all of the experiments in this section, from which results are presented in Figs. 3.18 to 3.22, the same QNN architecture is used: The feature map quantum circuit employed is $\mathcal{E}_C$ (see Fig. A.2) for $r = 2$ and the variational form is fixed to $\mathcal{W}_A$ (see Fig. A.3) for $d = 8$, rotation blocks consisting of R_x and R_y gates, and entanglement blocks consisting of all to all connected CNOT-gates. Furthermore, the ADAM optimizer with a learning rate equal to 0.1 is used for all of the experiments. The number of gradient descent steps can always be related to the number of quantum expectation values by (3.6) and is noted in the captions. In many of the experiments, the case where $\sigma = 0$ (corresponding to the limit $R \to \infty$) is contrasted with $\sigma = 0.1$ (corresponding

to $R = 100$). The latter choice is fixed such that the statistical uncertainty is large enough to have a discernible impact on the results, while the noise model in (3.5) is simultaneously still valid.

Experimental Results

The three subplots in Fig. 3.19 show for two instances of SGD (batch sizes 10 and 100) and one of GD (batch size equal to $300 = M$) how the loss obtained evolves during training. In all three cases, it is apparent that while the loss for $\sigma = 0.1$ defined like in (3.5) is slightly higher than when there is no additive error on the expectation values, the QNN training still shows the same trend and converges to a comparable stable value.

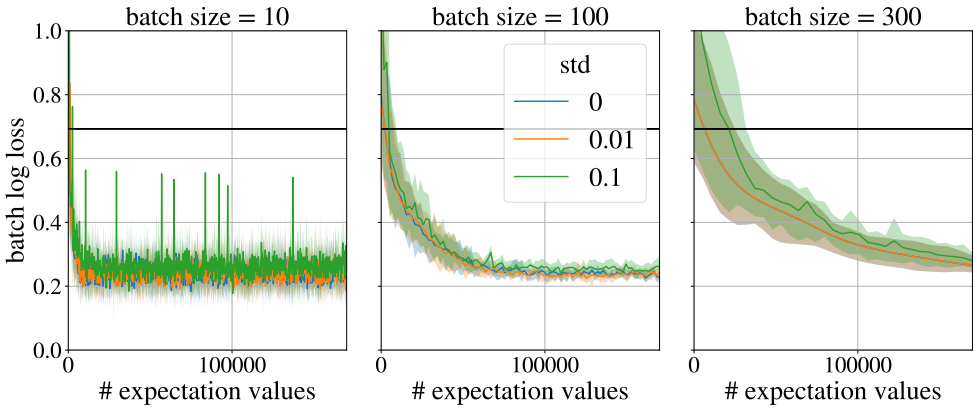


Figure 3.19: For an artificially generated toy dataset consisting of $M = 300$ data, the logarithmic loss evaluated on random batches (first and second subplot) and the full dataset (third subplot) is plotted during training. In this case, the number of expectation values considered for training is constant throughout the subplots and corresponds to 1000, 100 and 33 gradient descent steps respectively. The experiment is repeated for 12 random initializations of θ and the shaded area is bounded by the 15.9 and 84.1 percentiles. The mean over those runs is marked by the solid lines. Note that the blue and orange lines largely overlap. In Fig. B.8, the same experiment for $M = 100$ and batch sizes 1, 10 and 100 is depicted.

In Fig. 3.20, the logarithmic loss averaged over the full training set is plotted for QNNs that have been trained with SGD and GD, as a function of the standard deviation σ of the additive error η in (3.5). For the range under consideration, SGD seems to be insensitive to the noise and there is no significant trend visible. For GD on the other hand, the final loss and size of the error bars clearly increase with σ. This suggests that the SGD algorithm is more robust to the noise introduced to the algorithm by η than GD is. However, apart from this observation, the result in Fig. 3.20 more broadly indicates that the performance of classifiers that have been trained with SGD achieve a final performance that is comparable and not worse than when they are trained with GD.

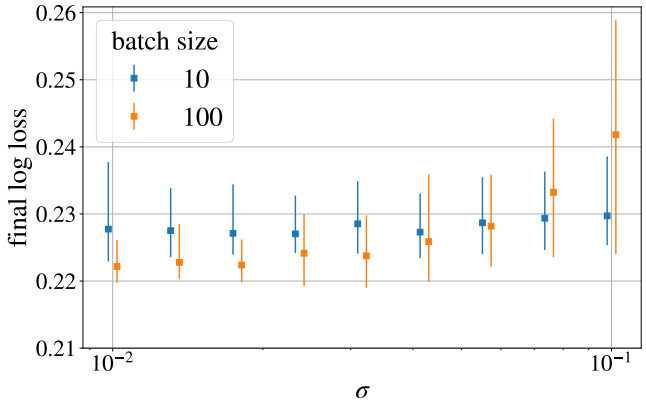


Figure 3.20: The final loss (averaged over the entire training set of size $M = 100$) achieved after the ADAM optimizer has converged to a constant value is plotted dependent on the standard deviation σ of the additive noise defined in (3.5). Blue (SGD) and orange (GD) points appearing next to each other share identical values of σ, but the positions along the x-axis have been slightly offset for legibility. The squares correspond to the mean values and the error bars to the 15.9 and 84.1 percentiles over a total of 100 random initializations of the trained parameters θ. The number of steps performed for training is 700 for SGD and 70 for GD for a total of 119,000 expectation values in both cases. Overall, the results obtained for the two different batch sizes are roughly comparable. Note however the upwards trend for the orange markers, which is largely absent for the blue markers.

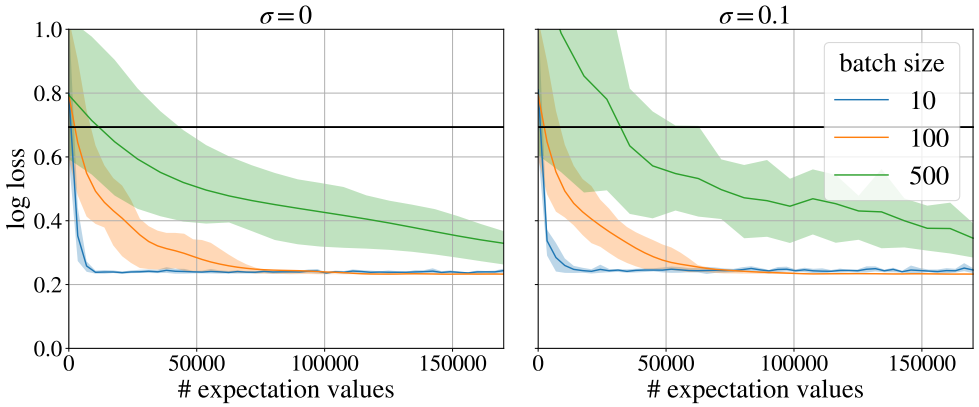


Figure 3.21: The two subplots correspond to noiseless and noisy training and they qualitatively agree. In both cases, the smaller the batch size is, the faster the convergence to a low loss happens as a function of the number of expectation values. In these plots, the loss is averaged over the whole training set of size $M = 500$. The number of expectation values respectively corresponds to 1000, 100 and 20 steps in ADAM (learning rate 0.1) for the batch sizes 10, 100 and 500. The solid lines and shaded regions mark the means and intervals between the 15.9 and 84.1 percentile over 12 random initializations of the trainable parameters. For a plot in terms of the batch loss, and where enough expectation values are considered for all graphs to converge, see Fig. B.9.

Figure 3.21 includes a comparison of the training convergence for different batch sizes. As the number of expectation values is equal in all cases, a fair

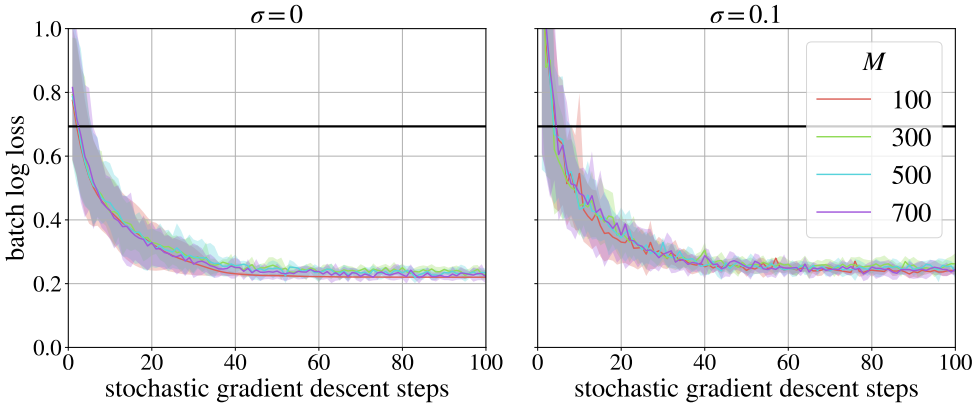


Figure 3.22: The two subplots correspond to the noise free ($\sigma = 0$) and an example noisy ($\sigma = 0.1$) case. In both instances, the speed of convergence for the QNN training is independent of the size of the training set M. Training is accomplished with a stochastic gradient descent algorithm with batch size equal to 100 (for a batch size equal to 10, see Fig. B.10). When the same experiment is conducted for non-stochastic gradient descent (corresponding to a batch size of M), the graphs do not overlap like here because then, the speed of convergence relative to the number of expectation values depends on the different batch sizes equal to M like in Fig. 3.21. This experiment is included in Fig. B.11. In all cases, the training has been repeated for 12 different random initializations of θ. The solid lines mark the means over these, while the shaded areas match the confidence intervals between the 15.9 and 84.1 percentile.

comparison can be made. In the experiment, SGD is observed to converge much faster to a lower training loss than GD, both when finite sampling noise is present, and when it is not. Furthermore, for SGD, the smallest batch size converges the fastest. Together with the fact that the quality of the solution is comparable in all cases (in other experiments like for example Figs. 3.20 and B.9 it is found that GD, which does not converge to a stable value in Fig. 3.21 for a lack of stochastic gradient descent steps/expectation values, obtains the same loss value as SGD after training has reached a stable value), this result implies that there is no reason to employ GD when training QNNs, and SGD only offers advantages.

Included in Fig. 3.22 is the result that at least for the setting under consideration here, where the data are artificially generated and guaranteed to be linearly separable in feature space, the speed of convergence for the QNN training does not depend on the size of the dataset M when SGD is employed. A plausible justification of why this could be the case is included in Section 2.3 on page 38.

Chapter 4

Conclusion

To conclude, it has been shown that the comparison between *quantum neural networks* (QNNs) and *quantum support vector machines* (QSVMs) comes naturally, since for supervised binary classification, both models implement decision functions of the same mathematical form

$$\tilde{c}(\mathbf{x}) = \operatorname{sign}\big[\, \langle \vec{w},\, \psi(\mathbf{x}) \rangle_{\mathcal{S}} + b \,\big], \tag{4.1}$$

where $\mathbf{x} \in \mathbb{R}^s$ is the datum to be classified, ψ is a quantum feature map, $\mathcal{S}$ denotes the space of density matrices and $\vec{w}$ defines the direction of the hyperplane in feature space according to which class membership is assigned [1]. The difference between these two approaches lies entirely in their different implementation of $\vec{w}$. Because QSVMs come with theoretical guarantees, which the heuristic QNNs lack, the latter can then be seen as approximations to the former. An important implication of taking this perspective is that QNN models at least partially inherit the formal motivations brought forward for QSVMs, which include provable quantum speedups [2, 17].

With this, the question remains, what can QNNs offer over QSVMs to warrant their study? One answer is that for QNN models, the number of quantum circuit evaluations necessary to accurately train them, scales favorably in the size of the training set when compared to QSVMs that are fit by solving the dual optimization problem. However, the introduction of an alternative way to fit QSVMs called the Pegasos algorithm [3] might be competitive with QNNs in this regard and further (numerical) studies are needed to settle this matter. The present experiments on the noise robustness of Pegasos suggest that a practically relevant average case scales significantly better than the theoretically treated worst case. Throughout the thesis, the asymptotic computational complexities for training and prediction are analyzed in the realistic setting where it is taken into account that quantum expectation values can only be estimated up to a statistical uncertainty, even on ideal quantum computers. An overview of the findings is highlighted in Table 4.1.

	QSVM (dual)	QSVM (Pegasos)	QNN
training	$\mathcal{O}\left(M^{4.67}/\varepsilon^2\right)$	$\mathcal{O}\left(\min\left[M^2/\delta^3,\, 1/\delta^5\right]\right)$	$\mathcal{O}\left(d/\varepsilon^n\right)$
prediction	$\mathcal{O}\left(S^2/\varepsilon^2\right)$	$\mathcal{O}\left(S^2/\varepsilon^2\right)$	$\mathcal{O}\left(1/\varepsilon^2\right)$

Table 4.1: Asymptotic complexity of the number of ideal quantum circuit evaluations necessary for training and prediction. M is the size of the training set, S the number of support vectors, d the number of trainable parameters, $n = 3$ conjectured, ε a bound on the difference between the ideal and noisy classification function that holds with probability $p > \frac{1}{2}$ and δ the accuracy with which the global optimum is approximated. For additional details, see Table 1.1.

Overall, QNN and QSVM models complement each other and can be productively studied in parallel. Then, a theoretical focus for QSVMs and an empirical focus for QNNs appears natural. In a wider perspective, heuristic extensions of these models, where trainable parameters are included into the feature map, blur the separation-line between them. Judging from the experiments on toy data conducted in this thesis, such models show most promise for applications, due to their additional versatility.

4.1 Outlook

In this final section, concrete research questions left unanswered in this thesis and some more broad directions for future investigation are collected.

Chapter 2 on theory includes a key result in Lemma 2.3, where an upper bound on the number of quantum circuit evaluations necessary to train a QSVM via the dual optimization is derived (see also Table 4.1). The finding marks an improvement over the bound previously reported in the literature [2, Lemma 19]. An interesting open question in this context is whether the assumption of *noisy halfspace learning* [2, Lemma 14], which is included in Lemma 2.3 and utilized in (2.74), can be replaced with another, more general one.

Another open theoretical question is how the quantity δ (see (2.94) in Section 2.2.4) can be expressed in terms of ε (see (2.67) in Section 2.2.3). These different measures of accuracy also occur under the same name in Table 4.1 and throughout Sections 2.2 and 2.3 in the statements on R_{train} and R_{pred}.

Concerning Chapter 3 as a whole, in view of practical application, an important direction for future extension of the results is repeating some of the experiments and invoking the test sets to measure the generalization performance of the models. As in this thesis, the analysis is strictly restricted to how the models perform on the training sets.

Similarly pertaining to the entire chapter, repeating some of the experiments on real quantum hardware would offer additional practical insights, as the numerical results have been exclusively obtained from simulations running on

classical computers. Furthermore, in this way the number of qubits utilized in the QML algorithms could be increased, as the available classical computational resources have limited this. Considering a larger number of qubits would allow the consideration of more complex datasets and in itself entails interesting questions like whether barren plateaus or vanishing kernels (see a below paragraph) emerge.

For QNNs concretely, the scaling of training with respect to ε conjectured in Section 2.3, has not been quantitatively probed by numerical experiments yet. Linked to this is the question of how accurately the gradients have to be estimated for the training procedure to still converge.

In the same vein, there is a theoretical derivation of the training complexity of the Pegasos algorithm presented in Section 2.2.4. While there are numerical experiments (see Fig. 3.14) that support the assumptions (see directly above (2.97) in Section 2.2.4) made in the derivation, the concrete scaling result with respect to δ has not been quantitatively investigated yet.

More generally, from Section 3.1 it is clear that for both QSVMs and QNNs, the central question all subsequent considerations hinge on is what makes a *good feature map*. Because if the feature map is not suitable for the data, all following steps in the machine learning model are hampered by this fact. Always coupled to this question is which data to consider. Is it the goal to fit a toy dataset classical algorithms have no trouble with, or to handle something that cannot be done classically? While the latter approach is obviously the way to go for justifying the deployment of quantum machine learning models, simultaneously finding such data and a quantum feature map that allows the quantum model to solve the task, is a difficult endeavor. So far, this has only been shown for academic examples [2, 17]. The experiments performed in this thesis indicate that when classically tractable data are considered instead, the heuristic models learning the feature map (kernel alignment and QANs) are promising and it seems worthwhile to (empirically) pursue this further. A concrete question to investigate is whether for a QAN, an M dependence surfaces in the training complexity, unlike for QNNs (see Section 2.3 and Figs. 3.22, B.10 and B.11).

For QNNs, the question of what makes a *good variational form* also remains rather unexplored. With the knowledge from Section 2.3.1 that the variational form parameterizes the coefficients defining the direction of a hyperplane in feature space, the question can be rephrased as asking which variational quantum circuit architectures offer a large expressivity in terms of these coefficients. A follow up work on [42] with respect to this metric seems a promising direction. A closely connected, and for the applicability of QNNs, ultimately crucial question is how the number of qubits q in the model and the number of trainable parameters d included in the variational form relate. As the dimension of the feature space, which the qubits access is exponentially increasing in their number, QNNs are only viable models when it can be shown

that the number of trainable parameters that is needed to implement a useful model does not scale the same way. Otherwise, any polynomial speedup QNNs enjoy over QSVMs becomes meaningless for increasing numbers of qubits. These considerations also apply to QANs.

Another topic increasingly coming into the focus of QML research not touched upon in this thesis concerns barren plateaus [14, 43, 44, 45, 46] for QNNs and vanishing kernel entries [27, 28] for QSVMs. The main fact in both cases is that a global observable involving all qubits leads to quantum expectation values that exponentially decrease as the size of the quantum space increases exponentially in the number of qubits. While the quantum kernel estimation approach presented for QSVMs (see Fig. 2.4 in Section 2.2.3) does not circumvent this, employing a local observable (see (2.107) and note that the parity function in (2.112) employed throughout this thesis is global) in QNNs is possible and the added flexibility might offer an advantage over QSVMs when considering larger numbers of qubits.

4.2 Acknowledgments

Firstly, I want to express my deep thanks to David Sutter for the countless discussions, mathematical clarifications, and without exception fantastic day-to-day supervision. My project was thoroughly shaped both by his concrete guidance and by his encouragement to pursue the questions I myself find the most interesting. Furthermore, I want to thank Amira Abbas for always taking the time to talk and offer essential advice, the vital role she had in guiding the project, and for the careful feedback she has given me on this document. Equally big thanks go to Stefan Woerner for offering me the opportunity to work on this exciting project in the first place, the helpful discussions we had, and his insightful supervision, regarding both the big picture and details. Special thanks go to Julien Gacon for his ample technical help with Qiskit and feedback on Chapters 1 and 4, as well as to the whole group at IBM for creating a great atmosphere to conduct research in. In addition, I want to thank Maria Schuld for her illuminating inputs in numerous calls and Knud Thomsen for proofreading this document. Finally, I want to express my gratitude to Prof. Dr. Renato Renner for generously supervising the thesis at ETH, enabling me to conduct it.

Appendix A

Quantum Circuits

A.1 Feature Maps

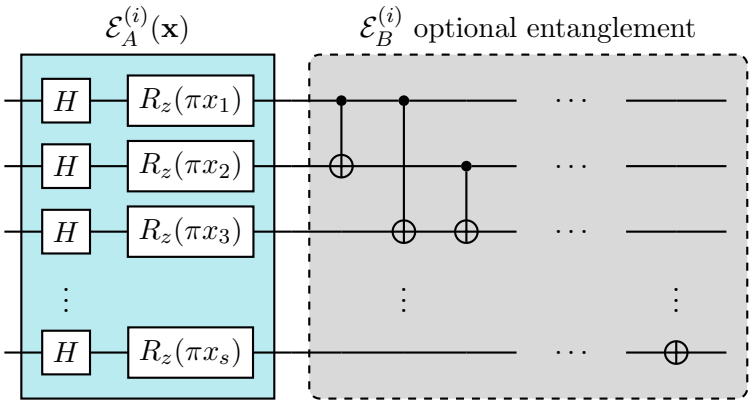


Figure A.1: A single layer (here, the i-th) of the $\mathcal{E}_A$ feature map circuit consists of the content of the blue box. Optionally, the all to all entangling CNOT-gates in the gray box can be added to the circuit as part of the same single layer. In that case, the feature map is denoted $\mathcal{E}_B$. In both cases, the total feature map circuit is structured like in (2.102) and is composed of l layers. For $\mathcal{E}_A$, where no entanglement is present, the resulting state is a product state and the quantum kernel function induced by the feature map can be efficiently calculated classically, regardless of l. The circuit is similar to ZFeatureMap in Qiskit.

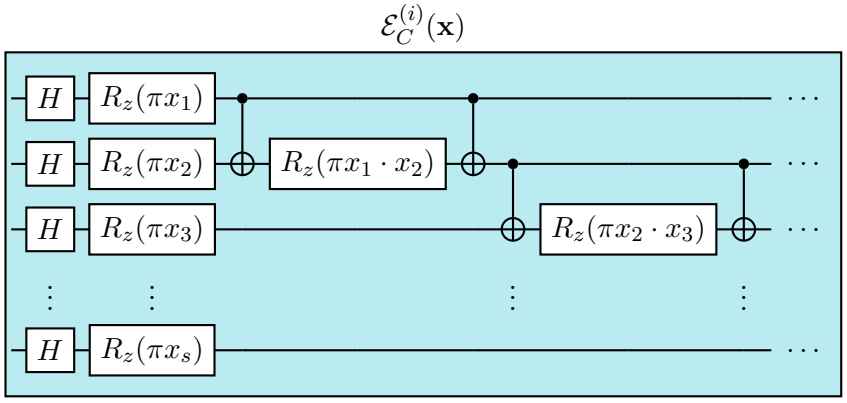


Figure A.2: Depicted is one layer of the $\mathcal{E}_C$ feature map circuit as proposed in [1] and also used in [14]. For l layers, the total feature map is structured like in (2.102). For $l \geq 2$, the quantum kernel function induced by the feature map is conjectured to be classically intractable [1]. This circuit is similar to ZZFeatureMap in Qiskit.

A.2 Variational Forms

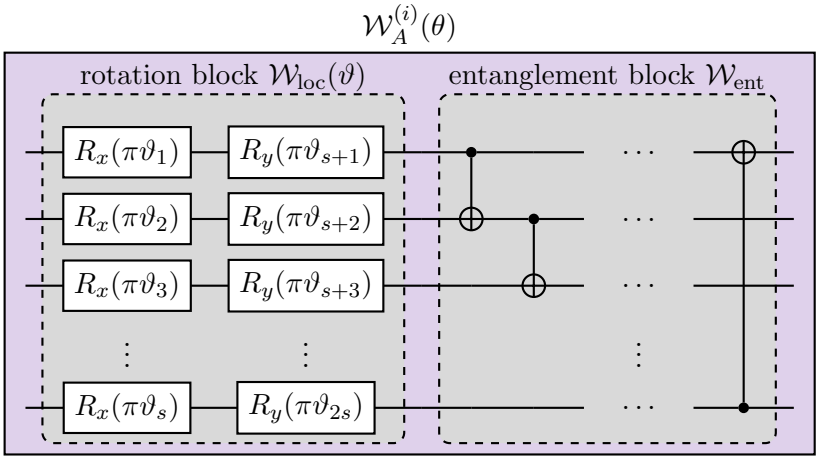


Figure A.3: Depicted is one layer of the $\mathcal{W}_A$ variational form. It consists of a *rotation block* and *entanglement block*. The rotation block consists of any choice of R_x, R_y or R_z single qubit gates applied to all qubits in parallel. The entanglement block consists of either CNOT or controlled R_x entangling two qubit gates, which are placed with either nearest neighbor, all-to-all or circular (like nearest neighbor but with one additional final connection between furthest neighbors) connectivity. The figure depicts a choice of R_x and R_y gates followed by CNOT with circular entanglement as an example. This circuit is similar to RealAmplitudes in Qiskit.

Appendix B

Additional Plots

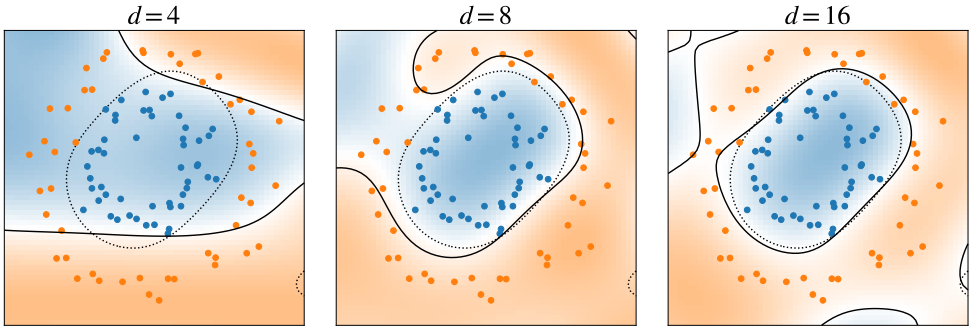


Figure B.1: Like Fig. 2.8, but for the sklearn circles dataset and the $\mathcal{E}_C$ feature map instead of $\mathcal{E}_A$.

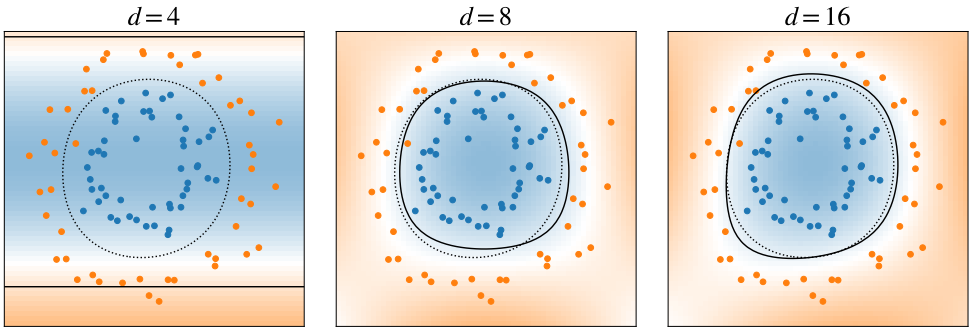
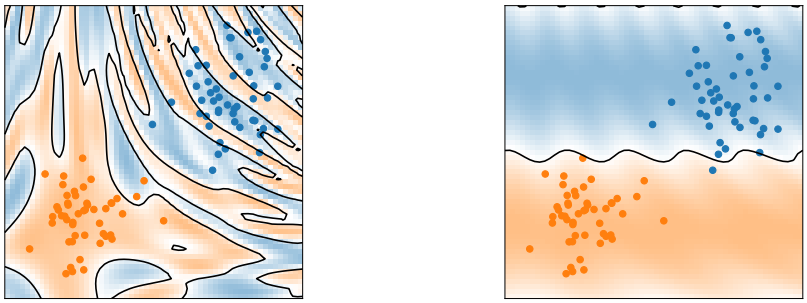


Figure B.2: Like Fig. 2.9, but for the sklearn circles dataset.



(a) The pre-processing is fixed such that all rotation angles are in $[0, 2\pi]$.

(b) A diagonal affine transformation $A \cdot \mathbf{x} + c$ is learned.

Figure B.3: Like Fig. 3.4, but for the cautionary example feature map employed in Fig. 3.1. In this extreme case, the benefit of learning an affine transformation is more apparent.

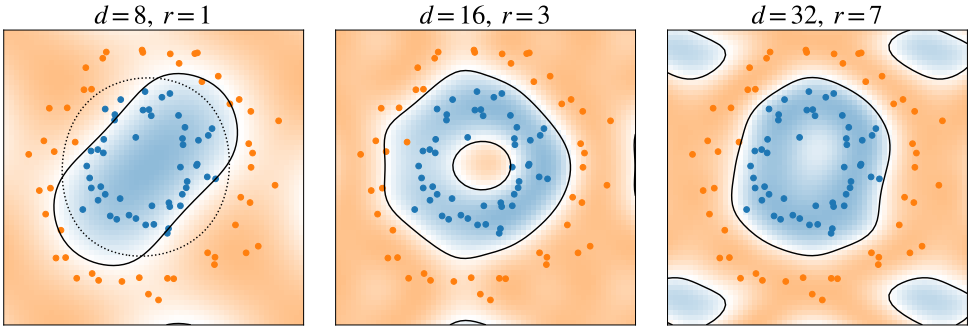


Figure B.4: Like Fig. 3.6, but for the sklearn circles dataset.

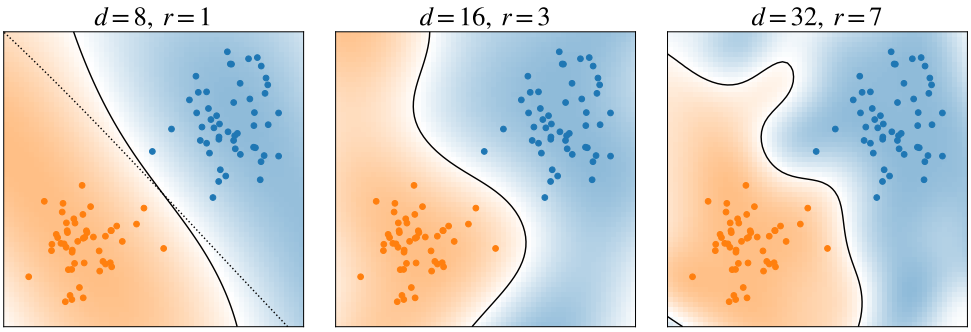


Figure B.5: Like Fig. 3.6, but for the sklearn blobs dataset. This experiment clearly illustrates the danger of overfitting; more complex models are not necessarily better.

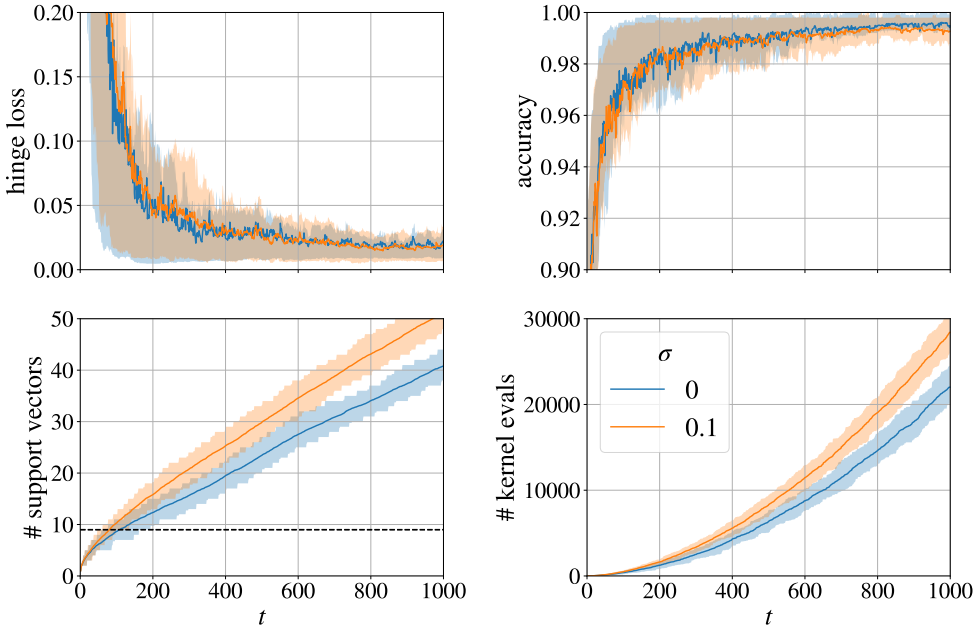


Figure B.6: Like Fig. 3.14, but extended from $\tau = 250$ to $\tau = 1000$ Pegasos steps.

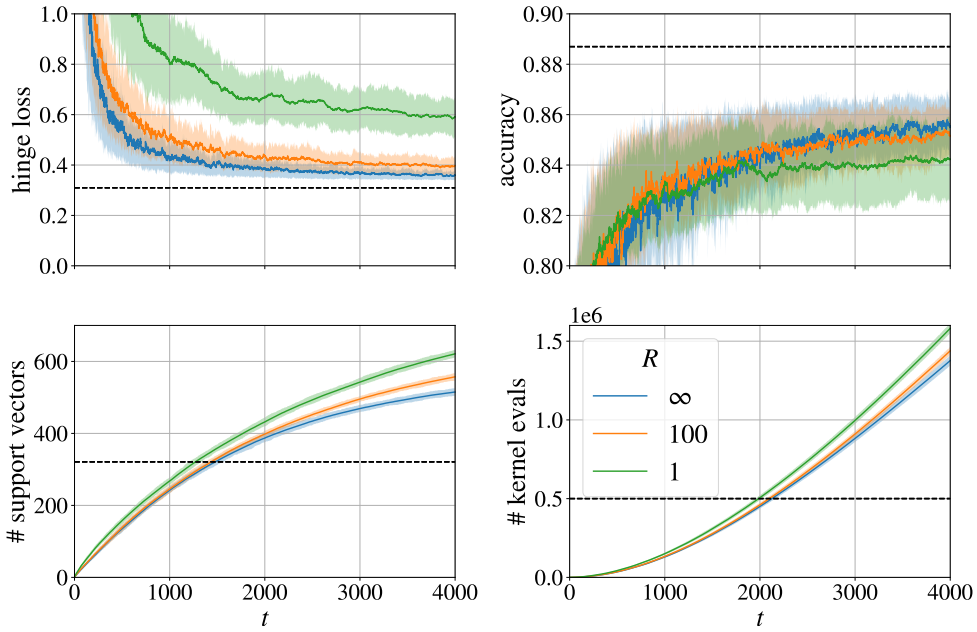


Figure B.7: Like Fig. 3.17, but for a larger number of Pegasos steps. The Gaussian noise model is also included analogous to Fig. 3.14. The values $R = \infty, 100, 1$ for the (modeled) number of measurements shots per kernel evaluation corresponds to "no noise", "$\sigma = 0.1$" and "Bernoulli" in the nomenclature in Section 3.3.1. Note that the number of kernel evaluations is not equal to the number of quantum circuit evaluations. The comparison in the plot is on approximately equal footing with respect to the former quantity (see lower right plot), but not regarding the latter.

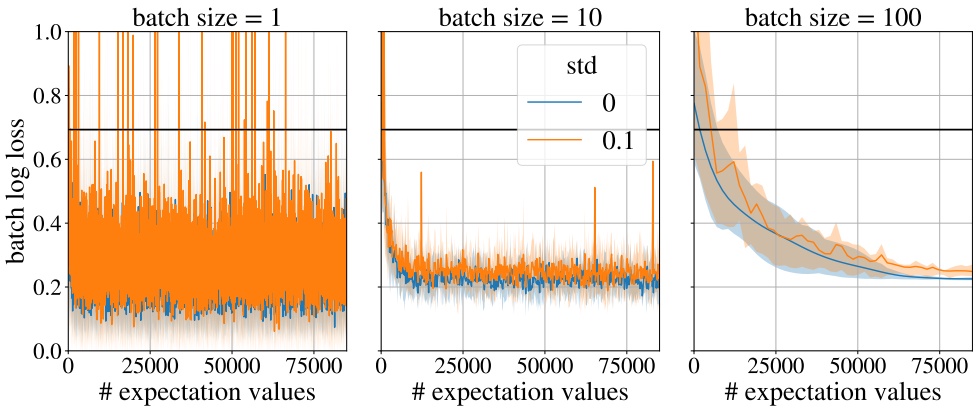


Figure B.8: Like Fig. 3.19, but for a dataset of size $M = 100$ and the batch sizes 1, 10 and 100. Here, these correspond to 5000, 500 and 50 ADAM steps respectively.

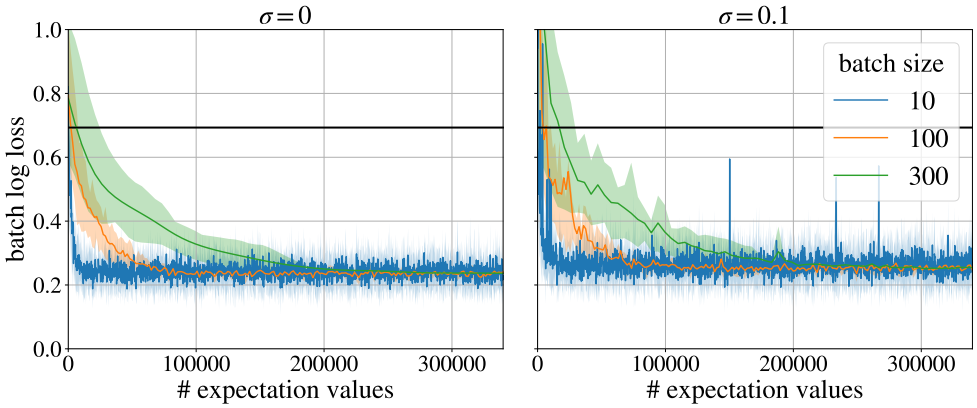


Figure B.9: Like Fig. 3.21, but for the batch loss instead of total loss, $M = 300$ and a larger number of (stochastic) gradient descent steps corresponding to 2000, 200 and 66 for the batch sizes 10, 100 and 300. It is apparent that while all train procedures converge to approximately the same final value, the smaller the batch size, the faster the convergence takes place.

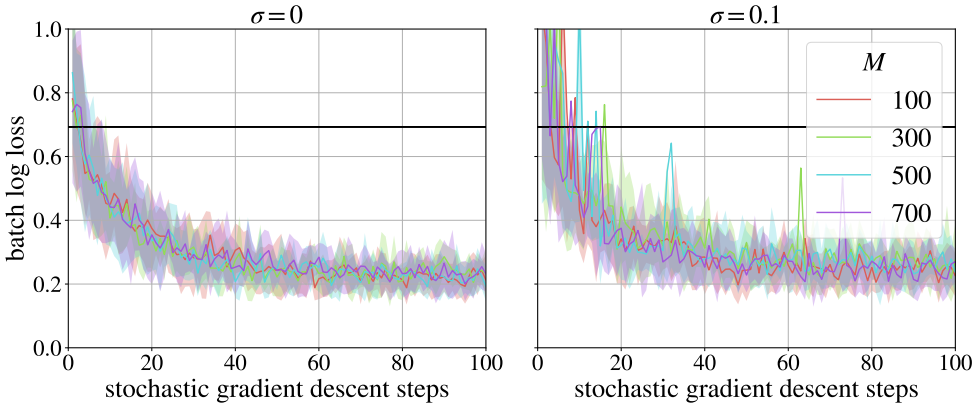


Figure B.10: Like Fig. 3.22, but for a batch size of 10.

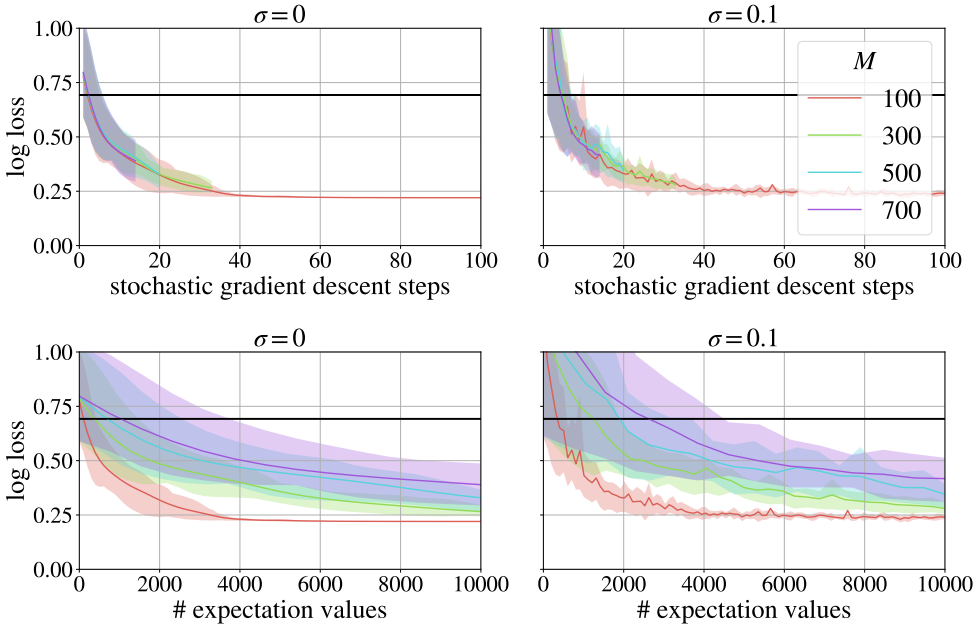


Figure B.11: Like Fig. 3.22, but for a batch size equal to the size of the dataset M, such that non-stochastic gradient descent is performed here. The graphs in the upper and lower sub-plots are identical apart from rescaling of the x-axis. Note that the train convergence is identical between different M as a function of the gradient descent steps in ADAM. However, for larger M the convergence is slower when training is seen as a function of the number of circuit evaluations.

Appendix C

Additional Algorithms

Algorithm 2 Generating Artificial Data

Inputs:
$\text{QNN}_\theta(\mathbf{x})$ with fixed θ,
size of the dataset $M \in \mathbb{N}$,
size of the margin $\mu \in \mathbb{R}_+$

$i = 0$
while $|y = -1| \leq \frac{M}{2} \wedge |y = 1| \leq \frac{M}{2}$ **do**
  Sample $\mathbf{x} \sim \mathcal{U}_{[0,1]} \times \mathcal{U}_{[0,1]}$ once
  $\tilde{y} = \text{QNN}_\theta(\mathbf{x})$
  if $\tilde{y} \leq -\frac{\mu}{2}$ **then**
    $\mathbf{x}_i \leftarrow \mathbf{x}$
    $y_i \leftarrow -1$
    $i \leftarrow i + 1$
  end if
  if $\tilde{y} \geq +\frac{\mu}{2}$ **then**
    $\mathbf{x}_i \leftarrow \mathbf{x}$
    $y_i \leftarrow +1$
    $i \leftarrow i + 1$
  end if
end while

Output: Balanced train set $T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_M, y_M)\}$

Bibliography

[1] Vojtěch Havlíček et al. "Supervised learning with quantum-enhanced feature spaces." In: *Nature* 567.7747 (2019), pp. 209–212. ISSN: 14764687. DOI: 10.1038/s41586-019-0980-2. arXiv: 1804.11326. URL: http://dx.doi.org/10.1038/s41586-019-0980-2.

[2] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. "A rigorous and robust quantum speed-up in supervised machine learning." In: *Nature Physics* (2021). ISSN: 1745-2481. DOI: 10.1038/s41567-021-01287-z. URL: https://doi.org/10.1038/s41567-021-01287-z.

[3] Shai Shalev-Shwartz et al. "Pegasos: Primal estimated sub-gradient solver for SVM." In: *Mathematical Programming* 127.1 (2011), pp. 3–30. ISSN: 00255610. DOI: 10.1007/s10107-010-0420-4.

[4] David Silver et al. "Mastering the game of Go with deep neural networks and tree search." In: *Nature* 529.7587 (2016), pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: https://doi.org/10.1038/nature16961.

[5] Andrew W Senior et al. "Improved protein structure prediction using potentials from deep learning." In: *Nature* 577.7792 (2020), pp. 706–710. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1923-7. URL: https://doi.org/10.1038/s41586-019-1923-7.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Ed. by B. Schölkopf M. Jordan, J. Kleinberg. Springer Science+Business Media, LLC, 2006. ISBN: 9780387310732.

[7] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Harlow: Pearson Education, Limited, 2016. ISBN: 9781292153964.

[8] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN: 026201825X.

[9] Peter W Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172. URL: https://doi.org/10.1137/S0097539795293172.

[10] Lov K Grover. "A Fast Quantum Mechanical Algorithm for Database Search." In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866.

[11] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum Algorithm for Linear Systems of Equations." In: *Phys. Rev. Lett.* 103.15 (2009), p. 150502. DOI: 10.1103/PhysRevLett.103.150502. URL: https://link.aps.org/doi/10.1103/PhysRevLett.103.150502.

[12] Maria Schuld and Francesco Petruccione. *Supervised Learning with Quantum Computers*. 2018. ISBN: 9783319964232.

[13] Maria Schuld et al. "Quantum Machine Learning in Feature Hilbert Spaces." In: *Physical Review Letters* 122.4 (2019), p. 40504. ISSN: 1079-7114. DOI: 10.1103/PhysRevLett.122.040504. URL: https://doi.org/10.1103/PhysRevLett.122.040504.

[14] Amira Abbas et al. "The power of quantum neural networks." In: *Nature Computational Science* 1.June (2020). ISSN: 2662-8457. DOI: 10.1038/s43588-021-00084-1. arXiv: 2011.00027. URL: http://arxiv.org/abs/2011.00027{\%}0Ahttp://dx.doi.org/10.1038/s43588-021-00084-1.

[15] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. "A Training Algorithm for Optimal Margin Classifiers." In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. New York, NY, USA: Association for Computing Machinery, 1992, pp. 144–152. ISBN: 089791497X. DOI: 10.1145/130385.130401. URL: https://doi.org/10.1145/130385.130401.

[16] Vladimir N Vapnik. *The Nature of Statistical Learning Theory*. Springer Science+Business Media, LLC, 2000. ISBN: 9781441931603.

[17] Jennifer R. Glick et al. "Covariant quantum kernels for data with group structure." In: (2021), pp. 1–9. arXiv: 2105.03406. URL: http://arxiv.org/abs/2105.03406.

[18] Thomas Hubregtsen et al. "Training Quantum Embedding Kernels on Near-Term Quantum Computers." In: (2021), pp. 1–20. arXiv: 2105.02276. URL: http://arxiv.org/abs/2105.02276.

[19] Gilles Brassard et al. "Quantum amplitude amplification and estimation." In: *Quantum computation and information (Washington, DC, 2000)*. Vol. 305. Contemp. Math. Amer. Math. Soc., Providence, RI, 2002, pp. 53–74. DOI: 10.1090/conm/305/05215. URL: https://doi.org/10.1090/conm/305/05215.

[20] Dmitry Grinko et al. "Iterative quantum amplitude estimation." In: *npj Quantum Information* 7.1 (2021), p. 52. ISSN: 2056-6387. DOI: 10.1038/s41534-021-00379-1. URL: https://doi.org/10.1038/s41534-021-00379-1.

[21] Maria Schuld. "Supervised quantum machine learning models are kernel methods." In: (2021), pp. 1–25. arXiv: 2101.11020. URL: http://arxiv.org/abs/2101.11020.

[22] Seth Lloyd et al. "Quantum embeddings for machine learning." In: *arXiv* (2020), pp. 1–11. ISSN: 23318422. arXiv: 2001.03622.

[23] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. "Effect of data encoding on the expressive power of variational quantum-machine-learning models." In: *Physical Review A* 103.3 (2021), pp. 1–16. ISSN: 24699934. DOI: 10.1103/PhysRevA.103.032430. arXiv: 2008.08605.

[24] Francisco Javier Gil Vidal and Dirk Oliver Theis. "Input Redundancy for Parameterized Quantum Circuits." In: *Frontiers in Physics* 8 (2020), p. 297. ISSN: 2296-424X. DOI: 10.3389/fphy.2020.00297. URL: https://www.frontiersin.org/article/10.3389/fphy.2020.00297.

[25] Raban Iten et al. "Quantum circuits for isometries." In: *Phys. Rev. A* 93.3 (2016), p. 32318. DOI: 10.1103/PhysRevA.93.032318. URL: https://link.aps.org/doi/10.1103/PhysRevA.93.032318.

[26] Rupak Chatterjee and Ting Yu. "Generalized coherent states, reproducing kernels, and quantum support vector machines." In: *Quantum Information and Computation* 17.15 (2017), pp. 1292–1306.

[27] Jonas M. Kübler, Simon Buchholz, and Bernhard Schölkopf. "The Inductive Bias of Quantum Kernels." In: (2021), pp. 1–27. arXiv: 2106.03747. URL: http://arxiv.org/abs/2106.03747.

[28] Evan Peters et al. "Machine learning of high dimensional data on a noisy quantum processor." In: (2021), pp. 1–20. arXiv: 2101.09581. URL: http://arxiv.org/abs/2101.09581.

[29] Rafał Latała. "Some estimates of norms of random matrices." In: *Proceedings of the American Mathematical Society* 133.5 (2005), pp. 1273–1282. ISSN: 0002-9939. DOI: 10.1090/s0002-9939-04-07800-1.

[30] Roman Vershynin. "Introduction to the non-asymptotic analysis of random matrices." In: *Compressed Sensing: Theory and Applications* (2009), pp. 210–268. DOI: 10.1017/CBO9780511794308.006. arXiv: 1011.3027.

[31] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. "Kernel methods in machine learning." In: *Annals of Statistics* 36.3 (2008), pp. 1171–1220. ISSN: 00905364. DOI: 10.1214/009053607000000677. arXiv: 0701907 [math].

[32] James W. Daniel. "Stability of the solution of definite quadratic programs." In: *Mathematical Programming* 5.1 (1973), pp. 41–53. ISSN: 00255610. DOI: 10.1007/BF01580110.

[33] C J C Burges and Chris J C Burges. "A Tutorial on Support Vector Machines for Pattern Recognition." In: *Data Mining and Knowledge Discovery* 2 (1998), pp. 121–167. URL: https://www.microsoft.com/en-us/research/publication/a-tutorial-on-support-vector-machines-for-pattern-recognition/.

[34] Shai Shalev-Shwartz and Nathan Srebro. "SVM optimization: Inverse dependence on training set size." In: *Proceedings of the 25th International Conference on Machine Learning* (2008), pp. 928–935.

[35] Ali Rahimi and Benjamin Recht. "Random Features for Large-Scale Kernel Machines." In: *Proceedings of the 20th International Conference on Neural Information Processing Systems*. NIPS'07. Red Hook, NY, USA: Curran Associates Inc., 2007, pp. 1177–1184. ISBN: 9781605603520.

[36] K Mitarai et al. "Quantum circuit learning." In: *Phys. Rev. A* 98.3 (2018), p. 32309. DOI: 10.1103/PhysRevA.98.032309. URL: https://link.aps.org/doi/10.1103/PhysRevA.98.032309.

[37] Hsin-Yuan Huang et al. "Power of data in quantum machine learning." In: *Nature Communications* 12.1 (2021), p. 2631. ISSN: 2041-1723. DOI: 10.1038/s41467-021-22539-9. URL: https://doi.org/10.1038/s41467-021-22539-9.

[38] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[39] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. "Effect of data encoding on the expressive power of variational quantum-machine-learning models." In: *Physical Review A* 103.3 (2021), p. 32430. ISSN: 0031-899X. DOI: 10.1103/PhysRevA.103.032430. URL: https://doi.org/10.1103/PhysRevA.103.032430.

[40] M D SAJID ANIS et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2021. DOI: 10.5281/zenodo.2573505.

[41] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[42] Sukin Sim, Peter D Johnson, and Alán Aspuru-Guzik. "Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms." In: *Advanced Quantum Technologies* 2.12 (2019), p. 1900070. DOI: https://doi.org/10.1002/qute.201900070. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/qute.201900070.

[43] Jarrod R. McClean et al. "Barren plateaus in quantum neural network training landscapes." In: *Nature Communications* 9.1 (2018), pp. 1–6. ISSN: 20411723. DOI: 10.1038/s41467-018-07090-4. arXiv: 1803.11173. URL: http://dx.doi.org/10.1038/s41467-018-07090-4.

[44] Patrick Huembeli and Alexandre Dauphin. "Characterizing the loss landscape of variational quantum circuits." In: *Quantum Science and Technology* 6.2 (2021), pp. 1–10. ISSN: 20589565. DOI: 10.1088/2058-9565/abdbc9. arXiv: 2008.02785.

[45] Kaining Zhang et al. "Toward trainability of quantum neural networks." In: *arXiv* (2020), pp. 1–37. ISSN: 23318422. arXiv: 2011.06258.

[46] Zoë Holmes et al. "Connecting ansatz expressibility to gradient magnitudes and barren plateaus." In: 1 (2021), pp. 1–20. arXiv: 2101.02138. URL: http://arxiv.org/abs/2101.02138.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Comparing Quantum Neural Networks and Quantum Support Vector Machines

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Thomsen	Arne

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Rüschlikon, 6.9.2021	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.