12-2020

Exploring Information for Quantum Machine Learning Models

Michael Telahun
University of Louisville

Exploring Information for Quantum Machine Learning Models

By Michael Telahun

Master of Engineering Thesis

Director: Daniel Sierra-Sosa

Co-Director: Adel S. Elmaghraby

Department of Computer Science and Engineering

J.B. Speed Scientific School of Engineering

University of Louisville

Louisville KY, USA, 40292

December 2020

Exploring Information for Quantum Machine Learning Models

Abstract

Quantum computing performs calculations by using physical phenomena and quantum mechanics principles to solve problems. This form of computation theoretically has been shown to provide speed ups to some problems of modern-day processing. With much anticipation the utilization of quantum phenomena in the field of Machine Learning has become apparent. The work here develops models from two software frameworks: TensorFlow Quantum (TFQ) and PennyLane for machine learning purposes. Both developed models utilize an information encoding technique *amplitude encoding* for preparation of states in a quantum learning model. This thesis explores both the capacity for amplitude encoding to provide enriched state preparation in learning methods and a deep analysis of data properties that provide insights into training data using a Variational Quantum Classifier (VQC). The advent of these new methods begs the question of how to best use these tools, we aim to give some overview explanation for the applicable state of quantum machine learning given actual device constraints. The results show there is a clear advantage for using amplitude encoding over other methods as we show using a hybrid quantum-classical neural network in TFQ. Additionally, there are several steps of preprocessing that can lead to more feature rich data when utilizing a VQC, in essence the no free lunch theorem holds true for quantum learning methods as it does in classical techniques. Information albeit encoded in a quantum form does not change the steps of preparing data but involves new ways to comprehend and appreciate these novel methods.

Acknowledgements

I would first like to thank my mother and father for supporting me through my journey in life so far. I would like to thank my advisor, Dr. Adel Elmaghraby for allowing me the opportunity to work in this exciting and innovative field of Quantum Machine Learning.

Additionally, I would like to thank my advisor, Dr. Daniel Sierra-Sosa who without his guidance and mentorship I would likely never have working knowledge of this topic let alone have anything tangle to show for my effort. Through the ups and downs I appreciate what you all have done for me.

Table of Contents

1. Introduction

Quantum Machine Learning (QML) is an interdisciplinary field that merges Quantum Computing (QC) and Machine Learning (ML). There are many algorithms present in the field of QC such as Grover's and Bernstein-Vazirani which have presented speedups to a number of problems such as integer factorization and database search. These algorithms are beginning to see the light of day now that devices coined Noisy Intermediate Scale Quantum (NISQ) have become a genuine implementation of Quantum Processing Units (QPU)s [1,2]. NISQ era devices have opened up the door for researchers to begin the process of developing these algorithms in earnest. These devices have also raised the interest around the topic, significantly spanning to domains outside of the typical computational thesis such as into finance, chemistry, biology, among others [3, 4, 5, 6, 7]. New developments in quantum technologies are beginning to spawn, and implementations of learning models for these new devices is growing. NISQ era devices provide the unique opportunity to test and develop QML techniques which were not physically possible before [2].

At the cornerstone of QC research and physical implementations, NISQ era devices have brought about the development of several programming suites for simulated quantum devices. Together with modern processing power these allow for a much richer experience when preparing to perform experiments on a real quantum device as well as trying to understand realistic expectations of current systems. In this realm we have two major components in terms of Application Programable Interfaces (APIs): the supporting code for development of circuits and the compilation code that runs on the physical device. Major organizations such as IBM, Google, and Xanadu allow development of models on both simulated and real quantum devices [8, 9, 10]. These software suites allow for the fundamental device-agnostic gate implementations that can be used to build quantum algorithms. With the developmental code there is the ability to create circuits and oracles out of gates. Oracles are considered "black boxes" in a quantum circuit which apply some set of gates to perform a change to the computational basis of a quantum state(s) [11, 12]. Some of these languages include Qiskit Aer, Cirq, and PennyLane. These are created and supported by IBM, Google, and Xanadu respectively. There is also so to speak the "backend" which needs to handle both software and physical problems. The backend handles aspe cts such as, transpiling of code, mapping of qubits to device topology, and noise during exe cution of an experiment. Each of these code sets handles these backend steps in their own ways.

There are several other companies that are also joining and excelling the race in the space of QC. In some cases these companies are developing their own QCs while others are developing software suites such as Amazon with BraKet [13], Microsoft with Q# [14], and Honeywell who uses the open Quantum Assembler (QASM) API [15]. Industry is beginning to bow to this quantum race as projections suggest a quantum supremacy just on the horizon. We are speeding towards a state where quantum computers will perform intensive computational tasks faster than classical devices [16]. The recent announcement of the experiment on Google's Sycamore QC showed achieving quantum dominance over a classical device could happen any day now.

Some of these key players in QML have developed open source frameworks such as IBM's Qiskit & Q Experience, Google's TensorFlow Quantum, and Xandu's PennyLane. These three are used in the work presented here primarily because they have the most extensive package s, support, and involved communities for QML. Each of these frameworks has capabilities to extend their software suites by utilizing tertiary APIs as well as utilize their frameworks on devices outside of their respective companies. For example, TensorFlow (classic) and PyTorch can both be used to train neural networks utilizing PennyLane. Additionally, circuits written with PennyLane, QASM, or Cirq can be run on IBM's quantum devices [10]. These abilities make developing QML experiments and models as natural as their classical counterparts. It should be noted that we are still not at the point where we can say that QML on NISQ devices surpasses classical methods; however, with these new tools, increased funding of research, and interest in the topic growing rapidly, we are getting closer to an inevitable outperformance of current leading technologies.

Much of the work presented here relates to the processing both before and after a learning methodology is applied, perhaps one of the most fundamental requirements for utilizing QML, or ML for that matter, is a firm foundation in general data mining procedures. Apart from developing models for quantum systems there is the body of work that will need to answer the questions: How do we improve results?, When do we use a method of embedding over a method of encoding?, Why did a model perform the way it did?, etc. Several methods we will develop are classical and help to answer these questions. One quantum method used here has been come to be known as amplitude encoding. This method as we will show is far superior to other methods of encoding raw classical data into a quantum state [17].

This thesis is organized with the following sections. We further develop a simple introduction of classical data processing methods, machine learning methods, and quantum computing in the Theoretical Background. A literature review, description of the technologies used, and the individual frameworks for quantum machine learning are given in the section Technical Background. We continue with the datasets explored both generated and toy datasets in our exploration of utilizing quantum machine learning in the section Datasets. The section Experiments covers the work in applications of data processing, development of quantum learning models, and what we have come to understand as some advantages/disadvantages. Finally, a conclusion on presented material and discussions for future work are given in the Conclusions section.

2. Theoretical Background

There are many components that make up this highly specialized field of QML. The goal of this section is to give enough background to the reader by briefly developing several these topics before moving on to a more technical overview.

2.1 Classical Overview of Data Processing

Preprocessing and post analysis of data are essential for gaining a firm awareness of information that will be explored or learned from. We will describe several of these methods that can be used before applying a learning technique, and after, methods that are used to understand the results. Briefly, the data storage system in this thesis was Comma Separated Values (CSV) and JavaScript Object Notation (JSON). We exclude a through explanation into data collection and data storage methods as they are not the focus.

2.1.1 Raw Data as a Whole

Raw data from a system or software is often captured without any pretext other than its immediate intended use. There are a number of different forms data can take such as continuous real variables, categorical discrete variables, binary variables, non-structured text, multimedia (audio, images, and video), among others. In their raw form, data are often not ready to be learned from or utilized in an analytical way. Even just grasping the bare meaning of the data can often be daunting without some level clarification or set of steps that simplify and facilitate an understanding of the information [18].

Three issues often associated with data processing involve missing records, imbalanced classes, and outliers. Missing records pose a very difficult problem as they would otherwise contain information that would be valuable to a learning technique. There are several methods for dealing with missing data such as replacing them with the mean of the feature, but this also causes potentially useful contents to be lost. Imbalanced classes are also a major issue when handling data especially when applied to a learning heuristic. Class imbalance occurs when a set of classes or a single class has more samples than other classes. Simply the population of a single class is greater than another. In the extreme case there is the possibility a technique will simply judge all the data as the majority class over the minority [18]. To alleviate this issue data can be balanced simply by dropping records from the majority class to match the minority class. Additionally, methods for oversampling the minority class have been developed such as Synthetic Minority Oversampling Technique (SMOTE) and Tomek Links [19, 20]. The third issue of outliers is often more complex in its analysis and handling. An outlier exists when there are data on the edge of the distribution of the samples. Simply put the data that does not conform to the same general behavior of the data [18, 21]. In the simplest case this can be handled by excluding samples greater than some number of standard deviations. In this thesis we handle all three issues in different circumstances. As we will discuss later the Scikit-Learn datasets which we generated fortunately do not have many of these issues but others such as the Wine dataset face some of these issues.

2.1.2 Feature Dependent Processing

Direct methods that transform and manipulate data in the preprocessing phase are regularly needed to clarify the information in some features. A few of these methods include discretization, normalization, and smoothing. As we describe further in the body of the experimentation each of these results in dramatic changes to a model's learning behavior when utilizing QML techniques.

2.1.3 Discretization

Feature discretization is a method for transforming raw continuous variables to a discrete and less complex feature space. In general, this can be considered as value reduction where there are two questions that we want to answer: where to stop or start a discrete set or interval? and how to determine what represents a discrete set or interval? The simplest method for discretization is to simply sort the data and then split it based on bins of size m where m is the number of elements in a bin. The bin value then becomes the mean of each bin and then the values are converted to those bin values. This method struggles with finding what size m needs to be to achieve the best results and can require several iterations of trial and error [18].

A second method known as the ChiMerge Technique has three steps for discretization: first sort the data in ascending order, define an initial interval such that only one value is in each interval (using the mean of every pair of values), for every adjacent interval compute X^2 of each interval and determine if the X^2 value is below the threshold, finally, if X^2 is below a certain threshold merge the two intervals, if not, the intervals cannot be merged. The lower bound of the first interval and upper bound of the second interval will replace the bounds of a new merged interval. To implement the ChiMerge a contingency table must be constructed that uses the number of classes in the dataset to determine the values used in the calculation of X^2. This makes the method typically useful for only classification methods. The ChiMerge Technique provides a statistical method for determining the intervals and results in significantly different results than the method for binning above.

2.1.4 Normalization

Normalization is one of the most common, yet important transformations applied to data. Normalization scales data between some predetermined range such as [0, 1]. The values are arrange based on some method which considers all of the samples for one feature. Therefore, normalization occurs column wise across the whole dataset where applicable [18]. This is because each feature's values are independent of every other feature. The concept of normalization is to simplify the impact any singular value can have on biasing a learning techniques behavior when observing the sample. If a technique sees that in certain cases a value is very large/small, it may over/under weigh the importance of a feature's value and misguide the learning process.

Before normalizing, outliers of data must be removed as it can cause the normalization to reduce most of the data to a small interval of values. Although normalization reduces the interval of values to some range, if outliers are included a learning technique may have a harder time understanding the other features of the data. If they are included, they also weigh in on the scale for normalization. Without removing outliers utilizing normalization can lead to errors that become harder to comprehend further in the pipeline of analysis. Normalization takes several forms such as standard deviation normalization, decimal normalization, and minimum-maximum (min-max) normalization. These equations are shown in Table 1.

Table 1. Normalization methods for scaling data.

Normalization Method	**Standard Deviation**	**Decimal**	**Min-Max**
Equation	$\frac{v_i - \text{mean}(v)}{}$	$\frac{v_i}{10^k}$	$\frac{v_i - \min(v_i)}{\max(v_i) - \min(v_i)}$

Where the column being normalized is v and the column value is v_i. The value k in decimal normalization is the power needed to make the largest value in the column less than or equal to one and std is standard deviation. Each of these methods can be modified to fit the data appropriately. For example, when using standard deviation normalization you may want to decrease the weight of values and apply a coefficient in the denominator, or in minimum-maximum normalization you may choose to normalize between [0, 1] or [-1, 1] depending on the classification task [18]. In either case normalization and its application is data dependent and should be evaluated before and after the learning process. We develop the normalizations in more detail as we build upon the process of preparing different procedures in the Experiments.

2.2 Post Analysis

After training a model, post analysis of results is a critical component as it explains the learning outcomes of an applied technique. In almost every case the best way to perform analysis of a learning method in either the classical or quantum realm is on a holdout or test set of samples which have not been used anywhere in the learning process. This requires the dataset be split into at least two groups such as training and testing or in some cases three groups where we have training, validation, and testing [18]. The validation set as we will is used in some methods to validate and provide feedback to a learning method while it is actively learning. Testing on the validation set should not be done as the technique will be privy to this set. The holdout or testing set again must never be used by the model. The holdout set must also be prepared using the same methods as the training data. This means the same preprocessing steps such as normalization and discretization must also be applied. From a practical implementation point of view, it is best to split the testing set from the data immediately prior to beginning any training, this way one can be sure all the necessary steps have been applied correctly.

The most common metric used here is Accuracy of the learning model on the new testing set. We also consider Precision, Recall, F1-Score, Confusion matrices (True Positive, True Negative, False Positive, and False Negative), and Receiver Operator Curves (ROC) when evaluating a model's results. We will show how these metrics imply the learning outcome of an amplitude encoded dataset using TFQ significantly outperforms other state preparation methods [17]. We will also show how different transformations to data in quantum models can significantly change the metric scores gathered. The main reason for using accuracy is that we primarily work with synthetic data here and as we have control over the data. For example, with an imbalanced dataset it would make sense to maximize and concern ourselves with F1-score [18]. Due to the lack of abnormalities in the data such as imbalances and outliers, accuracy is less questionable. That is not to say we did not evaluate almost all the data using the aforementioned methods, but due to the behavior of datasets we worked with, accuracy was an appropriate choice. In most cases we still evaluate results using all the metrics.

Post analysis of data can be tricky as it is tries to explain the output of the learning technique. It does not try to explain the learning process of an applied technique but the outcome. In this thesis the general goal of post analysis is to show which forms of encoding and transformations result in better performance in terms of a model's ability to learn the data in a quantum space/representation. For this reason, we fully define and develop the post analysis tools later in the section Experiments. We will also show that graphical results in some cases are able to capture the behavior of a technique which can make these numerical metrics misleading at a glance.

2.3 Machine Learning

Learning methods in modern times have grown very complex with new hardware paving the way for Deep Learning and advancements in Artificial Intelligence. Given these advancements, many of the underlying methods for learning have stayed the same. We cannot cover the entirety of machine learning in our brief overview but cover some of the basic components. We also discuss some simple methods, albeit old still perform exceedingly well.

In a broad sense machine leaning techniques can be categorized into three types: prediction, classification, and clustering. These three types although complex in their various implementations can be simplified in their explanation. In general, prediction is a task which aims to determine with some level of exactness or accuracy a value given a set of inputs. Both prediction and classification are concerned with accuracy in a similar way, but prediction accuracy is measured against the immediate result of a prediction. Classification aims to determine which class (whether there be two or two hundred) a sample identifies or corresponds to. Accuracy in terms of a classification model is determined based on the set of correctly classified samples. Clustering methods are typically based on some kind of distance metric which considers a "spatial" component of the data such that those closer or spatially nearer to each other in n-dimensional space are clustered together [22]. The goal of clustering is the most different from the three. It aims to produce some measurable explanation within a dataset by organizing or grouping subsets together, often it is used as a descriptive method before prediction or classification [18, 22].

An additional split in machine learning definitions comes with the approach for supervised and unsupervised learning. Supervised learning is a process by which samples of data are fed to a method with the class label or expected output. For any method of supervised learning the goal is to let the method run, applying whatever methods are available to it, and with its current state of knowledge try to make a guess about the expected output. The result or set of results from these guesses is then measured against the true output, internal functions or often model parameters are minutely updated, and the process begins again. This is essentially how most how machine learning and recently deep learning methods attain their highest results. Unsupervised learning such as clustering approaches the problem differently as its applications are generally not the same as supervised learning. They do not have labels to measure against after an iteration or part of learning. That is not to say clustering is only unsupervised, when applied correctly clustering techniques are among the top performers for classification in many cases [23]. Unsupervised methods typically have the goal of making some formal descriptions about data. An example of this is the Restricted Boltzmann Machines (RBM) which learn probability distributions of a dataset and the Apriori algorithm for market-basket analysis. For completeness, there is also semi-supervised learning which takes elements from both supervised and unsupervised learning [24].

2.3.1 Learning Methods

One machine learning technique which has its roots in statistical learning theory is a Support Vector Machines (SVM) [25]. SVMs are a method of supervised learning which in its basic form is a linear classifier that separates two classes from one another via a hyperplane. A hyperplane is defined based on the dot product of input vectors. This is one of the reasons why the method Quantum Support Vector Machine (QSVM) has become popular in the field of QML [26]. Several hyperplanes exist between classes so an SVM also seeks to maximize the distance between classes. This maximally spaced hyperplane exists when it is furthest away from the closest sample from both (all) classes. A margin is also important component to an SVM as it provides the ability to compromise when data is not perfectly separable as in most cases. The margin is the boundary space, containing the hyperplane, between the classes but with additional support for allowing overlap between classes. Optimization of these hyperplanes is performed using a Lagrangian transformation in most cases. Support vector machines have been expanded to include nonlinear classifiers based on what are called *kernel* tricks/methods/functions. Perhaps the most popular of these kernel methods is the radial basis function or RBF. These kernel methods replace the dot products into more robust nonlinear generalizations of SVMs [18]. In Figure 1. we plot three of these kernel methods: LINEAR, POLYNOMIAL, and RBF. SVMs are largely dependent on these functions and choosing the best one is data driven. The example contains ten samples, five for each class. In the figure the solid white lines are the separating hyperplanes, and the dotted white lines are the margins.

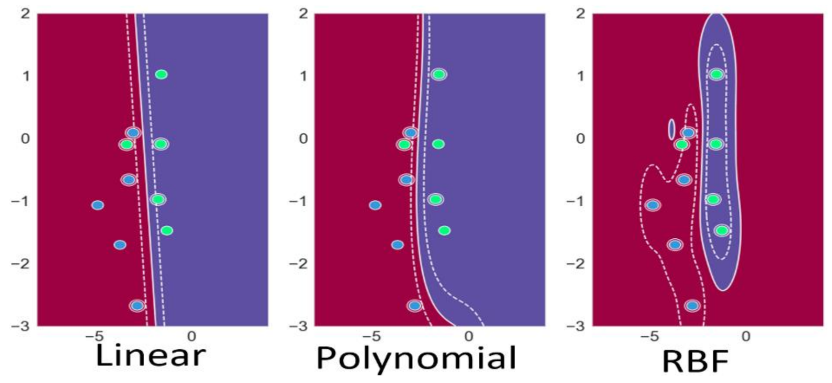


Figure 1. Application of SVM kernel methods on ten samples showing the different boundaries, margins, and hyperplanes for the same dataset. RBF is the only kernel which is almost able to classify all 10 samples correctly.

For clustering there are several techniques that fall into categories such as hierarchical methods, partitional methods, and density-based methods. One of the most well-known clustering methods is a partitional method called the KMEANs algorithm [27]. This method is rather straightforward as it tries to cluster data into k groups of equal variances by reducing the *inertia* or within-cluster sum-of-squares. The algorithm requires that a user pick the value of k which is often found either by trial and error, expert opinion, class labels, or a combination of these. Inertia is calculated by applying the Equation 1. where x_i is the sample and u_j is the cluster mean the sample is in. The naïve approach is performed by assigning clusters based on k and then updating the centroids or cluster centers by calculating the least squared Euclidean distance of the samples.

$$\sum_{i=0}^{n} \min\left(\left|\left|x_i - u_j\right|\right|^2\right) \tag{1}$$

Visualization of clustering methods also makes them attractive when attempting to explain the method or gain intuition. Figure 2. shows the results of the KMEANs algorithm on a very simple two-dimensional sample with four clusters. These results are visually well grouped and easily distinguishable in comparison to other datasets.

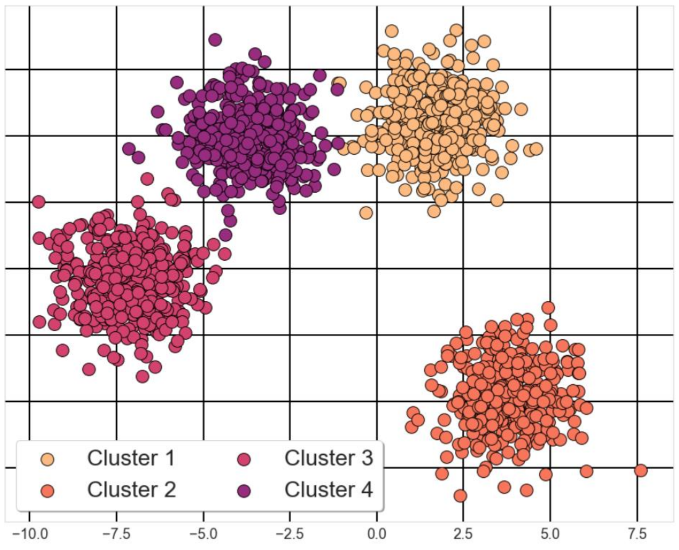


Figure 2. Four classes clustered by the KMEANs algorithm. The simple dataset here shows the capability of descriptive mining methods on convex data.

Deep learning's path has been primarily paved by way of the perceptron or multilayer perceptron (MLP). The perceptron was developed in 1958 making it over 60 years old [28, 29]. Later in this work we will apply an MLP to a quantum technique making it quantum-classical implementation. At its core a perceptron is a simple function which takes as an input a vector and takes its dot product with a real-valued weight. In the case of binary classification, the output value will be *one* when the dot product is greater than *zero* and *zero* otherwise. A perceptron is very simple and because of this it is not able to solve nonlinear problems [28]. The graphic in Figure 3. shows that there is not one single hyperplane that can separate the red triangles from the orange dots. This example shows that even a simple XOR logic gate is not linearly separable. To solve this problem we can add multiple perceptrons stacked together in the form of a “layer”, several layers (typically three or more) create an MLP. An MLP allows for nonlinear approximations to be learned. An MLP connects each of its nodes (neurons) with all other nodes and is dubbed a fully connected layer or dense layer [18, 28, 29].

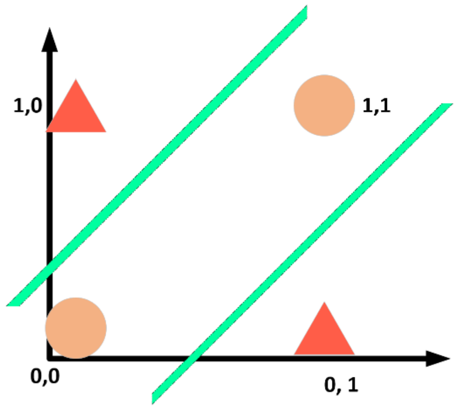


Figure 3. Classical problem on XOR gate that shows a simple perceptron cannot solve nonlinear problems. An MLP solves this by selecting the samples inside the boundary (dots) as one group and samples outside the boundary (triangles) as the second group.

Multiple “layers” of MLPs are the very basics of so-called deep learning as we are at a depth of typically several (sometimes hundreds/thousands) of layers. With this representation we also have the notion of a hidden layer which is any layer between the input and output layers of the network. The MLP used in [17] uses two dense layers of 64 and 32 neurons and one neuron in the output layer, a visualization of this can be seen in Figure 4. We have scaled it to ¼ the size due to the space limit of a page. The output layer of a perceptron can be extensively modified with software packages. It can not only perform binary classification but multiclass classification or continuous value regression. This behavior is controlled by an *activation function* which we will discuss later in this section [30].

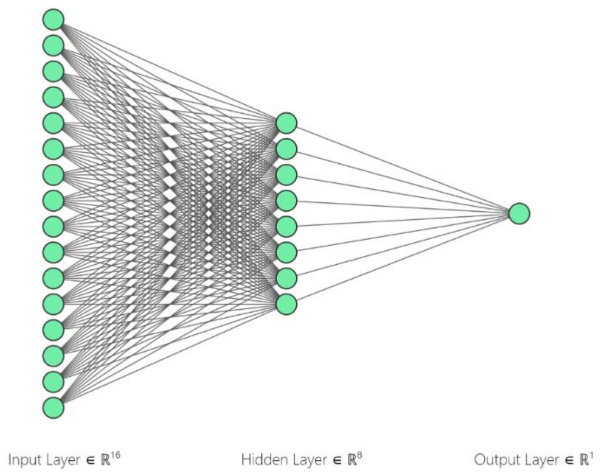


Figure 4. Artificial Neural Network (ANN) with 16, 8, 1 perceptrons for the input, hidden, and output layers respectively. The model used in the TFQ experimentations uses a similar network with four times the nodes in the first two layers.

2.3.2 Learning – Optimization and Loss

Although the recipes of different learning techniques can range from subtle to poles apart to even contradicting the underlying ingredient is optimization. Optimization is what breathes the concept of learning into any of these techniques. In terms of learning an optimization means solving for either a non-linear or linear equation of some feature space such as y = ax + b in the linear case. We cannot begin to cover the vast number of optimization functions that exist. But in general optimization has the goal of finding or 'fitting' to the equation that predicts, classifies, or clusters a dataset in the best possible way. To do this we introduce two concepts the learner or heuristic and the loss function.

If optimization is the method for learning than we can define the learning method as a heuristic. The heuristic is given control of a special type of parameters called hyperparameters. In the SVM this might be the tolerance for the margin or in the case of deep learning the number of neurons. Initially the one who is designing or implementing a technique will setup these hyperparameters which cannot be changed during training. These hyperparameters are used to develop the learner as they determine how to calculate and optimize the applied model [31]. These are often decided using additional heuristics, trial and error, or a combination of both. The learner uses these parameters within the model to determine or calculate its own model parameters which are unseen to the user. Model parameters are not imaginary or conceptual. In a simple algorithm we can show and precisely define the values these parameters will produce; however, as models scale to larger datasets and more complex feature spaces the size and number of these parameters tends toward combinatorial explosion [32].

One frequently applied optimization which in deep learning is the Stochastic Gradient Descent (SGD). SGD is an iterative method that tries to converge at some minima (local or minimal) within a function that fits to the data [32]. It has two main parameters to compute the gradient of a function: w or weight and η which is the learning rate or step size. SGD is a differentiable function which takes the form of a summation of gradients. It is computed using Equation 2. where $Q(w)$ is the function being minimized $Q_i(w)$ is therefore the ith example of $loss$ of the sample.

$$w \coloneqq w - \eta \nabla \mathrm{Q(w)} = \mathrm{w} - \frac{\eta}{n}\sum_{i=1}^{n} \nabla \mathrm{Q_i}(w) \tag{2}$$

In its process the SGD outputs a new weight for after every iteration. An illustrative example of this process is shown in Figure 5 (a). and in this case the optimizer will find a local minimum in place of the true global minimum. Although we would like our optimizer to reach the lowest point this is contingent on several factors. In terms of data we would like our data to exhibit some convex behavior in order to more frequently reach the global minimum. As this is data dependent, we are stuck tunning parameters, in the case of Figure 5. we show two examples of how two different learning rates would can impact SGD. The graphic in Figure 5 (a). shows the application of a large learning rate and below we can see a small learning rate in (b). Determining which is better is often data dependent. As the graphic depicts, a bigger learning rate is susceptible to jumping behavior and will likely skip over/not find a "good" local minimum. On the other hand, small learning rates are much smoother but might get trapped in any minimum such as the next minimum (#2 in the graphic). The key here is balance, the SGD needs to be able to be large such that it can move out of "poor" minimums and small enough to not bounce out of "good" minimums [28].

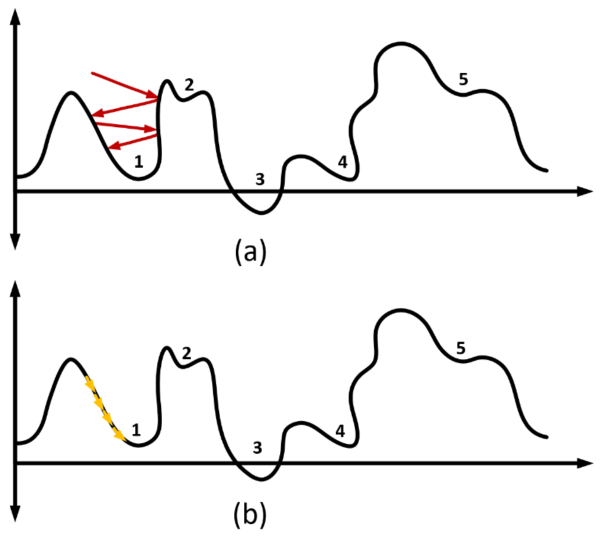
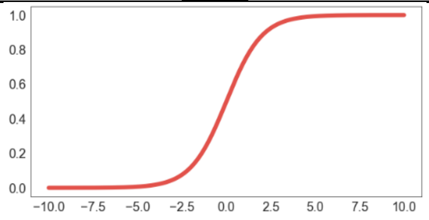
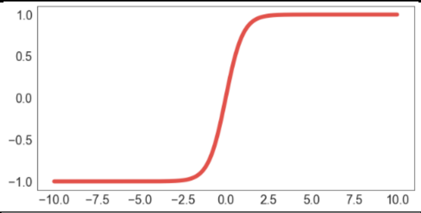
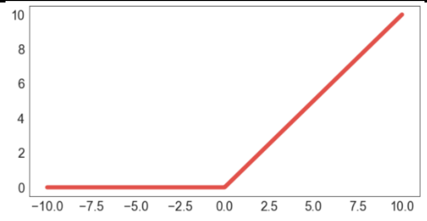


Figure 5. Example of gradient based methods. Learning rate of the top figure (a) is set to high which may result in jumping behavior. Bottom figure (b) has small learning rate which may lead to getting trapped in poor local minima.

We have labeled the minima in this diagram to show that there are four local minimums three of which are "good" minimums and #3 is the global minima. With a big learning rate, it would not be surprising to see the end result end up in the second or fifth minima. With a small learning rate the function may never exit the first minima. In our example this would by chance be an acceptable outcome. Later in the Experiments section we will discuss Adaptive Moment Estimation (Adam) Optimizer. Other optimizers apply additional parameters to control the learning behavior such as momentum to solve issues that SGD struggles with, like saddle points [33]. As always there is a tradeoff between choosing one method over another. In general, stochastic optimizers are the bread winner of optimizers.

Conceptually a "guide" is implemented in the form of a *loss function* to steer the optimizer towards these local minima [34]. They do not take the optimizer out of a minimum so to speak but attempt to make them tend toward the minima. Loss can be applied in the middle of an iteration, on a batch, and/or the end of an epoch. In general, loss determines how the heuristic has done thus far. By implementing loss, we can determine how learning is increasing or decreasing the overall results. It also helps explain what is known as *overfitting* if the learner has gone too far or in the wrong direction. The loss function of a neural network or other learning methods are fed values that have been weighted by the model, these weighted values are produced from a neuron or node by an *activation function* that shapes the output to some desired form. There are several activation functions that can be applied to a neuron. Three very popular choices are shown in Table 2., both equations and graphics are shown. Of these three, Rectified Linear Unit or ReLU has made waves in the deep learning community for showing that it is capable of converging faster than the other two [34]. Once each neuron has had an activation applied to it then the loss function can be applied to the layer or entire network. At the end of any training instance or rather a pair of training instances a decrease in loss signifies an increase in performance, typically accuracy, this implies that the heuristic has learned some component of the data [35]. As we develop in the Experiments section the loss function can also imply training has taken a turn for the worse and begun to, rightfully named, overfit.

Table 2. Common Activation functions for artificial neural networks.

Activation	**Function**	**Plot**
Sigmoid	$\frac{1}{1+e^{-x}}$	
Hyperbolic Tangent	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	
ReLU	$\mathrm{f(x)} = \max(0, \mathrm{x})$	

Overfitting is the state when a learning function has over optimized or in a sense memorized the training data it is privy to. This problem does not arise because the heuristic is attempting to memorize the data but rather that fitting to a function draws this behavior from an optimizer. Overfitting is an issue and without a holdout or validation set of data it is impossible to realize a model has overfit. When overfitting the optimizer has stopped fitting to feature information and started to optimize for decreasing the loss function on the training samples. Simply adding a validation dataset to use within the model will not mend the situation as the problem lies in the method or application of the method itself. But it can help spotting the problem much easier as we show in the Figure 6. using the public University of California Irvine Wine dataset. On the left the figure shows that while the training loss is continuing to decrease the validation loss is not matching this behavior, and similarly the accuracy (right) appears to drop back to its initial value for validation. This model has been intentionally overfit to show this issue. There are several methods for managing a model that overfits. The main issue of overfitting in this thesis occurs when data has been fit as best as it can, or simply the model has trained for too long. The issue has several solutions such as decreasing the complexity of the learning method, decreasing the training time, and/or forgetting some of the information learned in an epoch. Better yet, as we show from our paper using TensorFlow quantum, using the wrong quantum state preparation techniques will lead to *underfitting* [18].

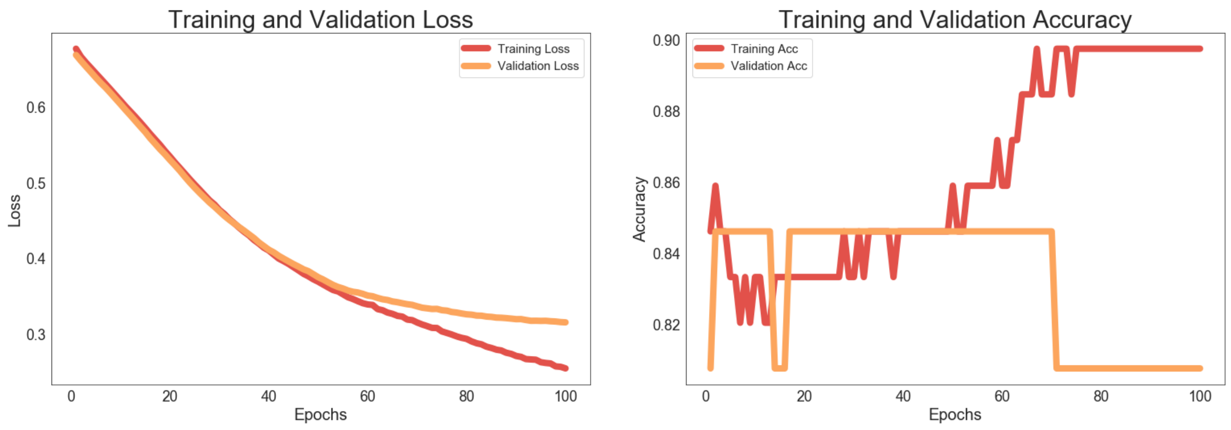


Figure 6. Example of a model overfitting on the Wine dataset. Noting the red are training metrics and orange are validation metrics. This is an over simplified model that needs tuning or better preprocessing of the data.

2.4 Quantum Computing

Quantum computation has had a longer history than many may be led to believe; however, it was in the early 1980's when suggestions regarding analog quantum computers by Richard Feynman, Paul Benioff, and Yuri Manin began to appear [36, 37]. In the following years, contributions led to the development of the first algorithms by Deutsch and Jozsa [38]. It was not until Peter Shor's algorithm for integer factorization and discrete logarithms in 1994 [39] that interest of Quantum Computing stirred within the scientific community. What is known now as Shor's Factoring algorithm showed that cryptanalysis techniques could exponentially be sped up, jeopardizing methods used to protect stored data and communication in both civil and governmental applications.

Among the many technical challenges associated with the physical implementation of quantum computers [40], one of these challenges is that quantum gates should be faster than the loss of information to the environment; this is known as decoherence. This phenomenon, imposes a constraint on computing, making some algorithms impractical. The development of superconducting metals allows for the creation of resonant circuits capable of providing coherent lifetimes of *milliseconds*, making quantum processors a reality. Although a large-scale noise free quantum computer seems beyond the horizon, there are several quantum algorithms that can be executed with current technologies [41].

The research done on Noisy Intermediate Scale Quantum devices, known as NISQ, produces algorithms and simulations for the current technology and hardware development state. With classical computers, tasks such as tracking and describing qubits in quantum simulations would be impossible; quantum computers are able to do this with ease. Without NISQ, comprehending qubits would require an exponentially large set of classical numbers, something quantum computers are readily able to produce. At a fundamental level this is why classical computers fail to perform many of the tasks that researchers and theorists believe quantum computing can [36].

Quantum Mechanics postulates are of algebraic nature, meaning there exists an intrinsic relation between quantum computation and algebraic operations [ref]. Multiple advances in the field of quantum information processing have provided promising prospects relying on that advantage. Therefore, it has been proven that Quantum Computing could lead to exponential speed-up in different data processing and machine learning methods, including Principal Component Analysis (PCA) [42, 43], K-means Clustering [44] and regularized Support Vector Machines (SVM) [45]. Advances in NISQ devices imply the development of more diverse and meaningful applications, increasing the relevance of conducting research in this area.

In the 1990s physicists began to analyze and consider what aspects would be needed to develop a quantum computer and with that how to program one. The result of this early thinking has led to the concept of qubits or the quantum dual of the binary bit and quantum circuits. Qubits or quantum bits are represented as the state $|\psi\rangle$ (read state psi or ket psi). A qubit is defined by a set of probability amplitudes α_n where n is the number of basis states. A generalization for a qubit is given in Equation 3. and the constraints for n are given in Equation 4., where the probability amplitudes must sum to one.

$$|\psi\rangle = \alpha_0|0 \dots 00\rangle + \alpha_1|0 \dots 01\rangle + \cdots \alpha_n|1 \dots 1\rangle \tag{3}$$

$$\sum_{i=0}^{n} |x_i|^2 = 1 \tag{4}$$

The need for quantum computing comes from the desire to model the real world with much greater detail. Computationally the classical computer has grown into quite a powerful tool which has been able to solve a myriad of tasks [36, 46]. In the most basic case, a few qubits are computationally intractable for classical devices to simulate and track. That is not to say classical devices will be obsolete when/if quantum devices reach supremacy over them. In fact, as we will show, classical devices and quantum devices both have their part to play. Quantum devices can be viewed as a secondary computational unit outside of the typical Computing Processing Unit (CPU) which handles calculations with much greater complexity like a Graphics Processing Unit (GPU).

One way quantum devices are able to perform intractable calculations that classical devices cannot is by taking advantage of what is known as the Hilbert space. A Hilbert space is a generalization of vector space with the structure of an inner product, it is a complete space [36, 46]. A Hilbert space is a real or complex inner product space that is also a complete metric space with respect to the distance function induced by the inner product. The following properties satisfy a Hilbert space:

1. The inner product of a pair of elements is equal to the complex conjugate of the inner product of the swapped elements. $\langle x, y\rangle = \overline{\langle y, x\rangle}$
2. The inner product is linear in its first argument and for all complex numbers a and b. $\langle ax_1 + bx_2, y\rangle = a\langle x_1, y\rangle + b\langle x_2, y\rangle$

3. The inner product of an element with itself is positive definite.

$$\begin{cases} \langle x,x\rangle > 0 & x \neq 0 \\ \langle x,x\rangle = 0 & x = 0 \end{cases}$$

The Hilbert space allows for generalizations of change of basis and linear operations which are requirements to achieve quantum computation [36, 46]. To oversimplify, quantum devices are not constrained to singular values as their minimum computational unit like classical devices. Their fundamental unit is a vector space. Quantum devices are not constructed to logics gates that utilize Boolean algebra instead they leverage quantum gates which resemble matrix operations and utilize linear algebra.

Quantum computers today take the form of Noisy Intermediate Scale Quantum (NISQ) devices. These devices by no means are the final state of quantum computer, but a steppingstone to prepare, develop, and test algorithms. Scaling these devices poses several challenges, although it is not the purpose of this thesis it must be mentioned. Physical systems are prone to error due to issues such as stochastic noise. Research in this area is referred to as quantum error correction and it works to increase the fidelity of quantum systems under these unideal circumstances [40, 47].

Quantum computing devices when created using a lithographic process have a physical connection between qubits called a Josephson junction [48]. The Josephson junction is a tunnel junction composed of two superconducting metals separated by an insulation barrier. The phenomenon is a product of quantum tunneling [36]. Quantum devices with this characteristic include IBM devices such as those in Figure 7. These devices have a different number of qubits but that does not directly relate to their compute capabilities. IBM has dubbed the term quantum volume [49] to indicate the capacity of a quantum computer taking into consideration several factors such as number of qubits, circuit depth before the level of error is to large, topological connectivity, crosstalk, U gate error, CNOT error, among others [50].

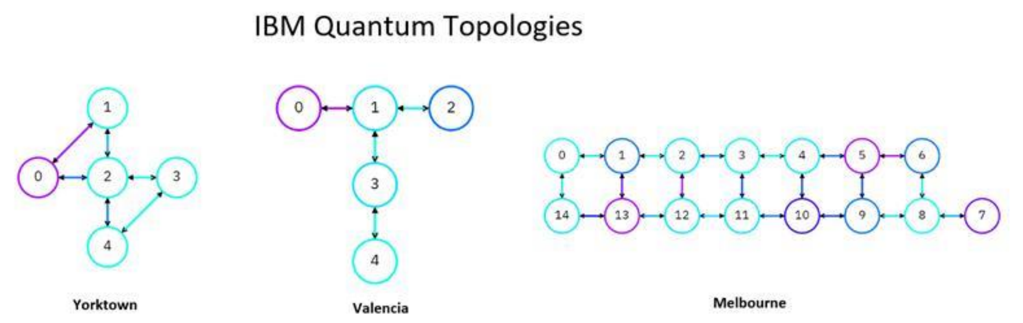


Figure 7. Three IBM quantum device topologies, taken from IBM Q Experience. The coloring of connections and qub its is indicative of their error rates. Darker colors indicate higher error rates. The devices are regularly reset which changes the error rates for better or worse. Error rates are also displayed for fundamental gates in the IBM Q Experience application.

Before discussing quantum machine learning, we wrap up the discussion of quantum computing and quantum information with a basic data transformation. Data can be transformed to represent quantum states using any arbitrary change to the computational basis of a qubit. A simple dataset such as $\{x, y, z\}$ can apply what is known as a unitary gate or U gate. A unitary is operated over a set of inputs producing some set of outputs to obtain a new set of transformed states. This concept or schema for applying a unitary to data is given in Figure 8.

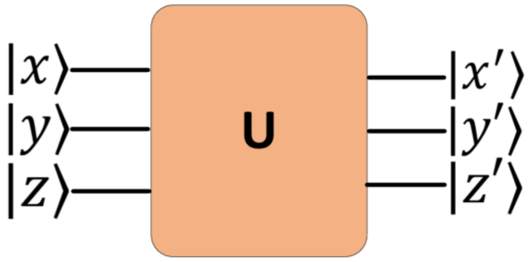


Figure 8. Unitary gate applied to create state changes.

2.5 Quantum Machine Learning

The term quantum machine learning (QML) can have more than one meaning depending on its use. What we will mean by it in this thesis is the use of machine learning on quantum devices (simulators) or quantum-assisted machine learning. The goal of this marriage is to discover whether the addition of quantum components can be leveraged to increase the learning of classical methods. It seeks to answer whether there are patterns that quantum information is better suited for, if with less information can similar/better results be achieved, or if quantum computers speedup the learning in certain classical optimization problems. The QML models we will apply in this work are all performed on simulated quantum devices so we will mainly focus our attention in the second meaning, quantum-assisted machine learning. This type of learning can be split into a number of different arrangements but perhaps the clearest example is given in [46] as a four-quadrant map. The map in Figure 9. represents the four types of quantum-assisted machine learning approaches that can be taken. The components are a combination of two letters where the first is the data archetype and the second is the computing device. The first letter is either quantum "Q" or classical "C" data and the second letter is either quantum "Q" or classical "C" devices. Therefore, the combination of QC only in this section is an abbreviation meaning quantum data on a classical device.

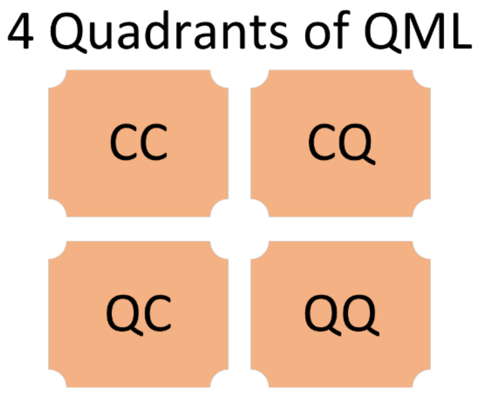


Figure 9. Quadrant map of data and system relationship "Q" is quantum and C is classical. First character represents the data source and the second represents the device. CQ is read classical data on a quantum device.

Computing in QML has mostly evolved to a state where there are several working models that have been theoretically developed and maintained in a quantum "Zoo" online [41]. These methods share crossover from standard classical models and in some cases are simply adaptations of quantum data in a classical model. Two of these models are implemented here they are the

Quantum Convolutional Neural Network (QCNN) [46] and a Variation Quantum Classifier (VQC) [46]. In general, we discuss supervised learning algorithms in this research. As we will develop the QCNN in the section TFQ Experimental Setup we mainly discuss the VQC model here. Both of these models are characterized as parametric quantum optimizations. Parametric meaning that some value, in our case a sample's values from a dataset, can be fed as parameters to a quantum circuit and the behavior of the evolved state can be estimated and then optimized.

One method we will discuss later applies the notion of "shots" in their learning methodology [51]. The concept of shots or repeated experiment runs is a method for handling the noise in NISQ era devices. Shots are performed such that a distribution of the probabilities for the outcomes can be ascertained. The value in the distribution of outcomes with the largest value is considered the true value output of an experiment. For example, in Figure 10. we have the distribution of outputs in Grover's algorithm for four qubits. These outputs suggest that the circuit is outputting the key value 1111 or 15.

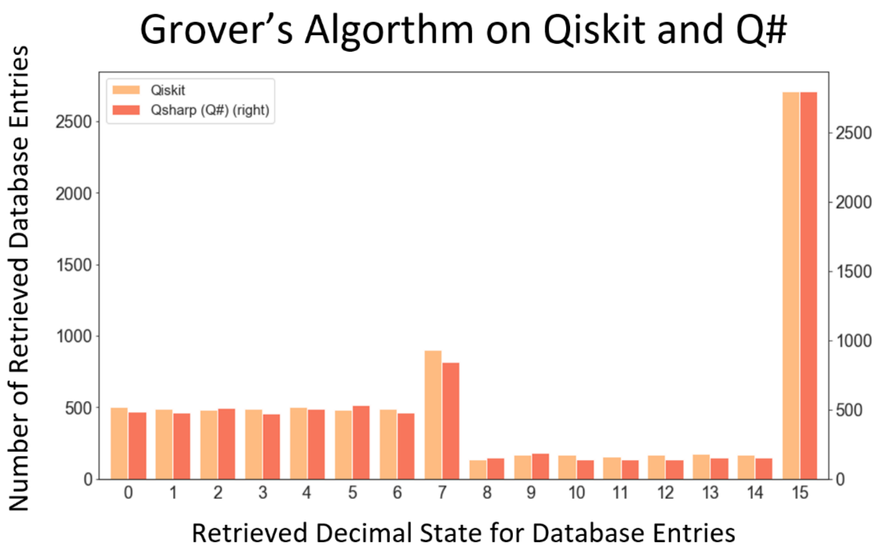


Figure 10. Grover's algorithm for four qubits using over 600 gates an example of a circuit that needs a high number of shots in order to be sure about the final value. An output layer for binary outputs needs a sufficient number of shots to ensure loss in the correct direction.

The application of shots in terms of a QML model is applied per iteration of a model. In this way if we perform twenty iterations of optimization with a batch size of five and ten shots, we will perform a total of 1,000 experiments on a quantum device. This is indicative of the state of QML on NISQ era devices and it is a rather large number of experiments. The number of shots needed is dependent on the complexity of the circuit and how infrequently we expect to get a noisy output from the system. We implemented the experiment in Figure 10. showing a circuit that searches for the binary value of 1111. The circuit is a four-qubit variation of Grover's algorithm that required 632 gates. We show the example on both Qiskit and Q# as the problem with shots is device/language agnostic. The number of experiments performed in both cases is 8,192. As you can see 15 is the result of the system roughly 1/3 of the time. This implies 15 was the value we were searching for.

2.5.1 Parametric Quantum Classifiers

We now introduce the topic of parametric quantum classifiers using the variational quantum classifier (VQC). We have shown that in order to calculate the value in Grover's algorithm several shots must be performed due to issues with error correction. Variational circuits and more specifically the VQC algorithm do not have this issue directly [46]. Instead VQC applies a hybrid application to the learning process by performing the optimization update within a classical device. This significantly decreases the complexity of a fully quantum circuit

There are three components to this methodology, following Havlíček et al [51]: a feature map applied to the data, a variational circuit, and the optimization. Like the SGD algorithm, VQC is an iterative method. Coincidentally the optimization of the model in a classical device produces a new source of error mitigation even when inputs to the classical devices are noisy measurements. As we will show in our experiments, we follow the steps from [46, 51] for state preparation using amplitude encoding, a proven method for VQC applications. In Figure 11. we show the architecture used in a typical VQC implementation.

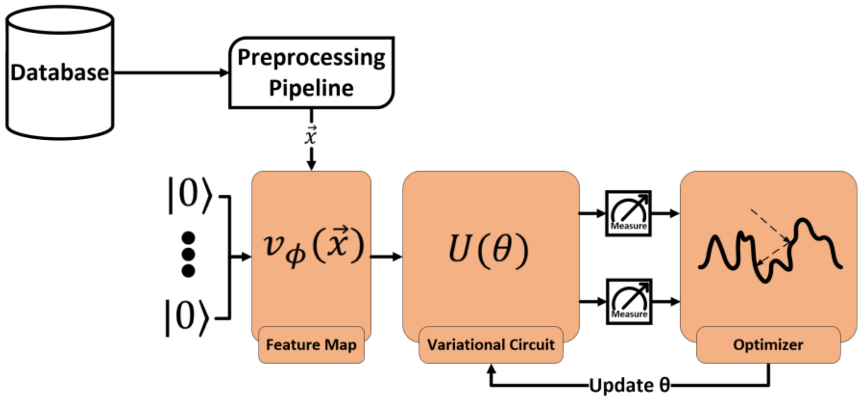


Figure 11. Pipeline for a VQC model. Once data has been preprocessed the feature map $v_\phi(x)$ is applied to the data for robust learning in Hilbert spaces. The variational quantum circuit is then applied. Results from the circuit are fed to an optimizer on a classical device which will update the parameter θ.

In Figure 11. we can see the feature map is applied before performing the optimization on the variational circuit. The feature map maps the classical data input into a higher-dimensional Hilbert space for the quantum system. The feature map in Havlíček's work is a "black-box" encoding of classical data to a quantum state $|\psi(x_i)\rangle$ that is performed using transformations to the ground state $|0\rangle^n$. This implementation of the feature map is given in Equation 5. where H is the applied Hadamard gate and Equation 6. is the diagonal gate in the Pauli-Z basis.

$$v_\phi(x) = U_\phi H^{\otimes n} U_{\phi(x)} H^{\otimes n} \tag{5}$$

$$U_{\phi(x)} = \exp\left(i \sum_{s \subseteq [n]} \phi s(x) \prod_{i \in S} Z_i \right)$$

3. Background

This section is a combination of a literature review and a background into the frameworks we utilized to program our experiments. The literature review is primarily concerned with discussing recent works in the realm of quantum information and quantum assisted machine learning. We present several recent research efforts in this domain. We conclude this section with the background knowledge of the programing "stacks" used to prepare quantum experiments in this thesis. This spans both quantum related frameworks and non-quantum specific packages. The code for this thesis can be found in Michael Telahun's GitHub repository. This is listed in the Appendix.

3.1 Literature Review

Schuld and Lloyd, who are veterans in this upcoming research field, present a quantum embedding method for increasing the performance of learning in high-dimensional Hilbert spaces in [52]. This is done by a paradigm shift in the way we consider optimizing a model where instead of fitting to the objective function the goal is to maximally separate two classes in the Hilbert space. The first component creates a quantum feature map to encode classical samples into quantum states, as we will discuss later amplitude encoding is one of these methods. The second component is a quantum measurement that gets returned from the model. Optimization of the quantum feature map is done by a parametric circuit, in this case a variational quantum classifier (VQC) that separates data based on the measurement performed. They describe a fidelity measurement and a Helstrøm measurement. The fidelity measure is a set of SWAP gates performing an inversion to the state of the samples. This method of measurement will maximize the Hilbert-Schmidt distance, or loss function in this case, ensuring the minimization of empirical risk or fidelity. The second component requires knowledge of which objective function is needed to minimize the classification loss [52]. Not all datasets can utilize the Hilbert-Schmit distance, in the specific case of this work the data appears to only contain a few features and the task is binary classification. In general, this work presents a practical enhancement and a solution for parametric classifiers on NISQ era devices. The authors also show their method is able to combine in a quantum-classical model that utilizes ResNet, a backbone deep learning set of model weights, for a Quantum Approximate Optimization Algorithm (QAOA) model [52].

Havlíček et al propose two binary classifiers to process data that is provided classically that uses the quantum state spaces as feature spaces [51]. The first approach is a variational quantum circuit which applies a binary measurement. The second approach follows from the classical SVM utilizing the construction of hyperplanes to estimate a kernel method. These experiments are performed on a five-qubit quantum processor from IBM. Their circuit and methodology have been embedded into the Qiskit API and Qiskit documentation as a fundamental example for quantum machine learning on NISQ era devices [51]. Perhaps the most striking component of this work is the highly nonlinear kernel method that must be constructed in order for a high accuracy to be achieved. The circuit is able achieve 100% accuracy on the generated dataset using the circuit shown in Figure 12. and it is one of several attempts to solve a similar problem [51]. They define the variational classifier in four steps: first map data to a quantum state by a feature map, second apply a short quantum circuit to the feature state, next apply a measurement in the Z-basis or via a Z-gate, and finally apply a decision rule by performing several "shots" or runs to obtain an empirical distribution of outcomes and assign the label for the largest probability [51]. Their method draws upon the notion of shots due to the noise and current capabilities of NISQ era devices.

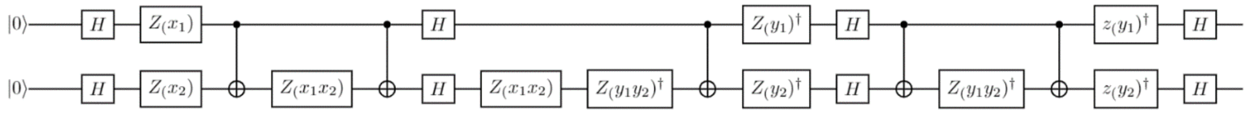


Figure 12. Havlíček quantum circuit for classifying a small highly nonlinear dataset.

Rebentrost et al derive the quantum equivalent of the gradient descent, an iterative optimization which tries to minimize a function, by considering the curvature information [53]. They apply Newton's method for the gradient descent which often improves convergence and can be useful in high dimensional problems that require a small number of iterations. The authors work with a class of polynomials which are constrained by sparsity conditions, meaning the optimizations can be used for certain smaller order functions. The authors also mention that the input dimensions of the vectorized data should conform to a binary space, or 2^N. This paper mentions annealing to show that their quantum gradient descent is agnostic of the device. They point out that the by applying Newton's method they are able to circumvent orthogonal movement in relation to the contour lines of a gradient and instead can also evaluate curvature. This method takes advantage of projecting the descent into spherical constraints. They implement three quantum oracles as different variations of their quantum gradient descent [53]. Under spherical constraints they have also extended their method to optimize a class of polynomials constrained by sparsity conditions of Hamiltonian simulation methods. Because the method for optimization exploits Newton's method they theorize that a highly accurate solution for any convex problem can be found within 5-25 iterations. They also mention similar issues to classical optimization such as "saddle points" in high dimensional space for Newton's method. They alleviate the saddle point issue by replacing the eigenvalues of the Hessian with absolute values. They conclude by stating theoretically their method will lead to exponential improvements to classical gradient descent-based methods [53].

Kiloran et al discuss methods for continuous variable (CV) quantum neural network architectures using parametric quantum circuits following the principal of a fully connected layer. They design a fully connected quantum scheme for a neural network with the availability for classical neural network support i.e. quantum-classical networks. They show that a CV architecture is capable of handling a fully connected (FC) network as well as several other network types such as Recurrent Networks (RNN), Convolutional Networks (CNN), and Residual Networks (ResNet). They train four models two of which use hybrid quantum/classical architectures with the remaining two strictly quantum architectures. In the development of these four models one major component of their work is the generation of cost functions. Aside from the curve fitting model [54] they develop a cost function for each model. The curve fitting model use mean square error (MSE) as the loss function. This is a common loss function we utilize later in the TFQ work.

The first hybrid model was trained using supervised learning to detect fraudulent transactions in credit card purchases. They use an exponential linear unit or ELU as their activation function and define their cost function in this model as Equation 7.,

$$C = \sum_{i \epsilon data} (1 - p_i)^2 \tag{7}$$

where p_i is the probability of the single photon being detected as "on" or "off" correctly [54]. They show that given the constraints of the quantum simulator used, simplicity of the network, and restriction to both size and depth of the quantum circuit their results are a proof of principle. This is clearly shown by their results in false negatives. The second hybrid architecture is an autoencoder which they state has a resemblance to a variational autoencoder. It consists of 25 layers and tries to generate the Fock states $|0\rangle, |1\rangle, |2\rangle)$ based on the one-hot vector representation $(0,0,1), (0,1,0), (1,0,0)$ that is input to the network. Here their cost function is identical to the cost function they use for the fully quantum neural network for the Tetris game shapes. This cost function is in Equation 8. where $\gamma = 100$ and $|A\rangle$ are the input states of the three Fock states, and P is the trace penalty. The results of this network were 99.5% when tested only on the quantum decoder.

$$C = \sum_{i=0}^{2} (|\langle i|\psi_i\rangle|^2 - 1)^2 + \gamma P(\{|\psi_i\rangle\}) \tag{8}$$

The second fully quantum method they explore tries to generate "LOTISJZ" tetromino shapes in the form of images for the game Tetris. They use the cost function from Equation 9. where $\gamma = 100$ and $|A\rangle$ are the seven input image states for each tetromino, and P is the trace penalty. Visually these results appear just as the tetromino shapes do in the game Tetris. They use 11 photons in the simulation [54]. They use Strawberry Fields from Xanadu to implement all their experiments.

$$C = \sum_{i=1}^{7} |\langle \psi_i|A_i\rangle|^2 + \gamma P(\{|\psi_i\rangle\}) \tag{9}$$

3.2 Programming & Frameworks

Quantum computing packages are fundamental to developing QML techniques. There are several packages and libraries which give us the capability of doing so. The ones we discuss here are widely used and provide through groundwork in not only machine learning but also quantum computing's various components. Utilizing these packages for QML requires working knowledge of both quantum computing and machine learning; however, with libraries such as TFQ and PennyLane many of the computational components are ready to use out of the box. Similar to data analysis packages for commercial and research purposes, quantum computing libraries contain many of the underlying components or functions to create basic gates, use a simulator, measure results, create oracles, among other things. QML libraries on the other hand provide a different set of tools which can be used on top of QC packages. Two packages stand more in the eye of this work than others, these are TensorFlow Quantum and PennyLane. These libraries as we show use elements from machine learning on top of quantum computing libraries. The leveraging of either library was suited to the experiments we conduct in the section Experiments.

3.2.1 TensorFlow Quantum

TensorFlow Quantum (TFQ) was announced by Google at the beginning of 2020 as a new library for quantum machine learning. Its implementation and current support are for the Python Language only. TFQ is made publicly accessible, guidelines for how to develop, test, and design simple models are provided in their documentation. The TFQ library builds upon its base TensorFlow (TF) which has become a notorious leader for deep learning development. Deep learning libraries such as TF and PyTorch have also been included as plugins to the stack for PennyLane and can be integrated with Qiskit but the TFQ library is an entire computing platform much like the original TensorFlow. TFQ intends to provide rapid prototyping of hybrid quantum-classical machine learning models [9]. The TFQ library works in conjunction with two other libraries for symbolic mathematics [55] and quantum logic circuit design [56], Sympy and Cirq respectively. These libraries are fundamental in order to create learning models in TFQ. Because TFQ requires these other packages to perform QML they must also be developed as part of the software stack for our experiments later.

Cirq as we have mentioned is a circuit design package which provides several of the same capabilities as QASM, Qiskit Aer, and PennyLane. Cirq is a couple years older, released in 2018, than TFQ. It was developed by the Google AI Quantum Team. At its core it is the component of the software stack that allows for quantum computing. Cirq was intended to be usable on local simulators of users' machines [56]. As it performs universal quantum operations, if transpiled correctly, it can be device agnostic when used on actual quantum devices. In our work we used Cirq for the very thing, in our TFQ model it was used to implement amplitude encoding and the circuits for quantum convolutions.

Sympy is a library for symbolic mathematics and is a full-featured Computer Algebra System (CAS) with a longer history than recent quantum computing libraries. Sympy is meant to be leveraged by those in need of true mathematical computation. The uses include Calculus, Discrete mathematics, Geometry, Physics, Combinatorics, among others [55]. Sympy was initially released in 2006 and is not a result of Google's venture into the quantum space. The usage of Sympy in our work was to control parameters within the quantum model. They are different than typical parameters as the TensorFlow library builds upon the two components namely placeholders and the TensorFlow graph for computation of deep learning models. Sympy is used within the graph as a the parametric quantum variable placeholders for intermediate values provided by the quantum calculation within the model.

3.2.2 PennyLane

PennyLane is a cross-platform library for differentiable programming of quantum computers. The language is supported by the company Xanadu in Toronto, CA. PennyLane is essentially designed for machine learning techniques in quantum computers. It allows for most other quantum and non-quantum machine learning libraries to interact with it making it perhaps the most robust framework currently available [10]. It is able to interface with IBM devices, Google devices, Rigetti devices, and is prepared to hand Microsoft devices. PennyLane is all encompassing and uses the same code to create low level instructions like gates and circuits unlike TFQ and Cirq. PennyLane provides automatic differentiation of quantum circuits to create both hybrid quantum-classical and fully quantum models. Many of the functions and type interfacing are done by way of NumPy which is a linear algebra library we use extensively in the development of our experiments [10].

PennyLane also offers prebuilt algorithms for many quantum learning algorithms. These include: Variational Quantum Classifiers (VQC), Quantum Approximate Optimization Algorithm (QAOA), Quadratic Unconstrained Binary Optimization (QUBO), Variational Quantum Eigen solvers (VQE), Ensemble Classification, Quantum Generative Adversarial Neural Networks (QGANN), Quantum Convolutional Neural Networks (QCNN), Variational Quantum Linear Solvers (VQLS), among several others. Each of these models is given a robust introduction in their documentation [10, 57].

Xanadu produces another framework called Strawberry Fields which is targeted at more low-level logical functions. The framework is intended to work with error mitigation and hardware optimization. Both PennyLane and Strawberry fields are designed to run on the Xanadu photonic quantum computers which are different from other super conducting devices. Companies such as IBM and Google use super conducting devices. Briefly, the advantage to these devices over super conductors is that they can run at room temperature making them a more versatile implementation. It contains an interface which is similar to the IBM Quantum Experience user interface which allows you to drag, drop, and run circuits in a web application [10].

3.2.3 Development & Non-Quantum Packages

Coding, experimentation, and development of this thesis was done in Python3 for the portion of work in TFQ we used Python 3.6.10 and we used Python 3.7.8 for the portion in PennyLane. Anaconda is a program which allows for simple package management and environment control, it was used to create separate environments for both TFQ and PennyLane. Most packages were installed using either the main ‘anaconda’ channel or ‘conda-forge’, when these two channels did not have a specific package the Package Installer for Python (PIP) was used. We used Visual Studio Code and Jupyter Notebook as the Integrated Development Environments when developing code.

Numpy is a library for linear algebra and vector/matrix operations, it was extensively used both in applying several of the preprocessing steps and post analysis. Scikit-Learn was used for creating the datasets, splitting data, and performing many of the post analysis steps. We will discuss the creation of these data sets in the section Scikit-Learn Generator Datasets later as well as the toy datasets such as Iris. The Scikit-Learn library is a large library with sub modules for imbalanced datasets, image processing, and many data mining tasks. It is a library generally revolving around generalized learning methods, preparation of data for learning methods, and predictive analytics. It is built using Matplotlib, Numpy, and Scipy [58, 59]. Scikit-Learn was additionally used in the post analysis steps for generating ROC/AUC curves and gathering the metrics of the learning outcomes. Several of the preprocessing steps defined in the Experiments section were done using the Scikit-Learn library. We used the Pandas library to manipulate and view data by way of DataFrames which make handling and transforming data simpler. DataFrames also allow for functions to be applied column wise making many complex steps easy.

Visualization was a key component for facilitating understanding of many of the transformations. It is also the major primary medium for expressing the preprocessing steps, and the results of this thesis. Matplotlib was used for most of the plotting and is largely tied to Numpy both in practical application and internal development [58]. It provides functions for plots such as scatter, line, histogram, density, pie charts, among others. Seaborn was also used for plotting, it extends Matplotlib by adding styling and some additional plotting functions when using Pandas DataFrames. Poincare plotting was done using Plotly, we use these plots in our analysis of Stokes parameters. Plotly is a multilanguage visualization library with extensive plotting capabilities like Matplotlib. For storing results and data we used both comma separated value (CSV) and JavaScript Object Notation (JSON) files. CSVs make viewing data very simple when being shared and evaluated individually. Both CSV and JSON are simple to use in Python and have built in libraries for handling both. The writing was done using Microsoft Word and the online Overleaf editor for LaTeX. We used Microsoft Visio to create graphics unique from plots of data.

4. Datasets

In exploring datasets with quantum methods, we wanted to test an appropriate number of different distributions and shapes. The datasets we worked with are in the majority of synthetic and some popular toy datasets. The reason we use synthetic datasets is to show how different the quantum learning methods behave on them. We also wanted to control the shape of the data when developing the datasets to answer our hypothesis. We generate several datasets using the Scikit-Learn Generator methods. Additionally, we use 'toy' datasets which are often a utility before testing methods on real datasets. The 'toy' datasets we use are the Iris dataset which we will discuss in much detail as it is critical to our analysis, and the Wine dataset. One key component of the datasets, mostly the generator datasets, is that we use very few features. The primary reason for this is the faultiness of quantum devices. To work with a dataset that is large in dimensionality on a simulator is possible. But with the limited capabilities of physical NISQ devices these datasets would perform poorly. With the interest of testing and working with real systems we avoid large datasets here.

4.1 Scikit-Learn Generator Datasets

The following four datasets in this section were created using Scikit-Learn. We include two datasets from the scikit-learn Toy datasets and modify them all from them out of the box design to fit into the analysis here. These are later elaborated in the section Toy Datasets. The Datasets in Sickit-Learn's Generator class have several parameters which can be set to match whatever objective is trying to be met. Generator functions in Scikit-Learn have controllable parameters for number of features, number of samples, random state (for reproducibility), number of repeated values, and a parameter typically unique to the type of data that can be generated by that function. This parameter typically controls the separation between classes in a dataset.

4.1.1 Make Blobs Dataset

In the Generator MakeBlobs the CenterBox parameter determines how spread out each sample is within a class. When the number of classes is just two the CenterBox parameter becomes the centroids of each class along the positive and negative y-axis. In two dimensions, the CenterBox parameter when equal to (-4.5, 4.5) will result in class one centered around -4.5 and class two centered around 4.5 both along the y-axis. When the number of classes is larger than two the CenterBox is no longer a centroid but the bounding box for each cluster center. In both cases a larger range in the CenterBox parameter implies more compactly distributed samples per class with generally less overlap, while a smaller range implies more overlap between classes and less compactness. The value for CenterBox can range from (-10, 10). A two-dimensional sample of the MakeBlobs data is shown in Figure 13. the CenterBox is (-3, 3). The MakeBlobs Generator was used for the KMEANs algorithm in the Introduction.

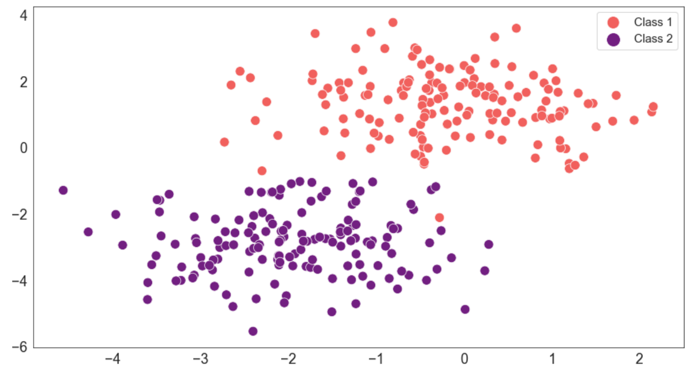


Figure 13. Scatter plot of MakeBlobs dataset.

4.1.2 Make Circles Dataset

In the Generator MakeCircles the factor parameter determines the space factor between two concentric circles. The dataset only has two features so when used for three- or four-dimensional data third and fourth features can either be generated from a normal distribution, via padding with a constant, or both. With a factor of 0.9 the inner circle will be very close to the outer circle almost overlapping it. If the factor is small such as 0.2 the inner circle will be much smaller and have much more distance between it and the outer circle. The MakeCircles generator only generates two output classes. Noise can also be added to both circles in the form of a standard deviation for the Gaussian distribution applied when generating the circles. An example of a generated MakeCircles sample is show in Figure 14., it contains very little noise and is has a very small factor.

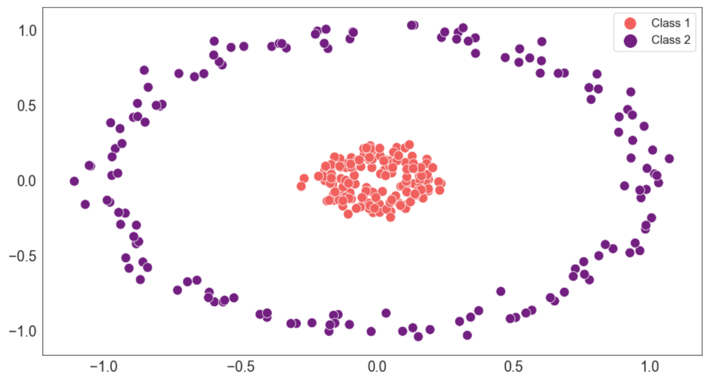


Figure 14. Scatter plot of MakeCircles dataset.

4.1.3 Make Moons Dataset

In the Generator MakeMoons there is no additional parameter to control the shape or location of the two classes. This generator makes two half-moons or arcs where one end of each classes' "moon" is at the crest of the other. The dataset only has two features so when used for three- or four-dimensional data third and fourth features can either be generated from a normal distribution, via padding with a constant, or both. Noise can be added to both moons in the form of a standard deviation for the Gaussian distribution applied when generating the moons. A visual of MakeMoons is shown in Figure 15., it contains very little noise.

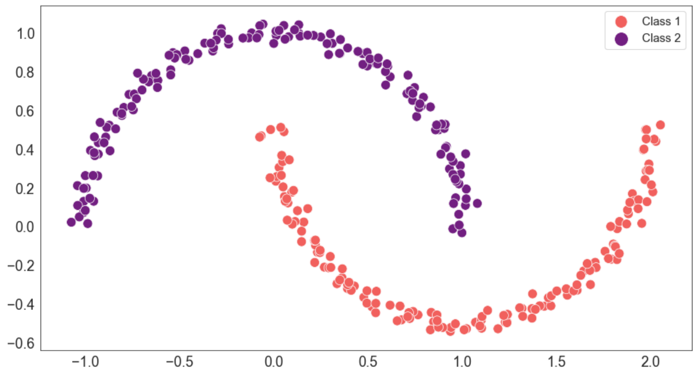


Figure 15. Scatter plot of MakeMoons dataset.

4.1.4 Make Swiss Role Dataset

In the Generator MakeSwissRole there are a number of controllable parameters to shape the output dataset. The single most important detail is that there is no class label that defines the components of the dataset. The SwissRole dataset needs to be classified with a different method to determine the classes per sample of the data. The method applied follows directly from the documentation as a method which can be most reproducible, but it must be stated that it most likely is not the single best method. This adds a layer of complexity as we are applying a clustering method to generate the class labels for a dataset and then expecting the QML model to recognize the content from the data when splitting the classes. The entire dataset can be seen in Figure 16. after it has been clustered using the Agglomerative Clustering method to produce the class labels. The Agglomerative method produces a total of six class when clustered on the dataset. As we do not work with multiclass datasets in the QML experiments we reduce these to just two classes when using the dataset. The data is also reduced from three dimensions to the first two.

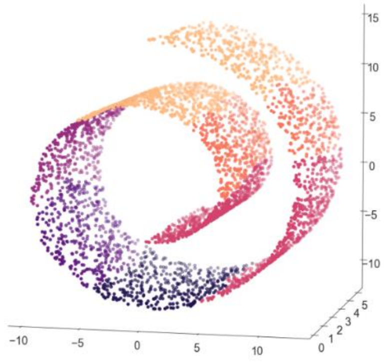


Figure 16. Scatterplot of 3D Make Swiss Role dataset.

4.2 Toy Datasets

Toy datasets are not synthetic data but have some well-behaved trends within them. There is often little to no missing data or very specific components that lead to near perfect results. They are typically utilized in the facilitation of discussions and in testing/preparation before applying techniques to real datasets. These datasets are used only in the work done with PennyLane. They are publicly available and accessible through Scikit-Learn or from the UCI ML website.

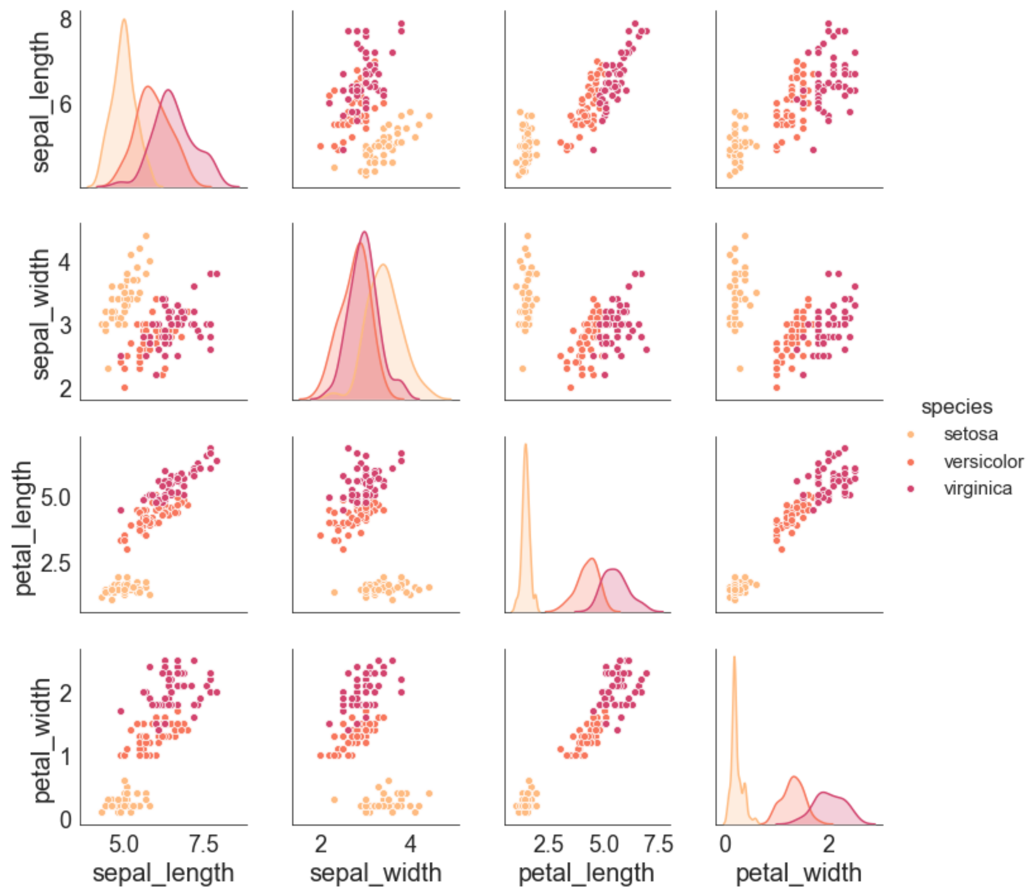


Figure 17. Matrix of scatterplots for each feature pair in the Iris dataset. The diagonals of the matrix are the distributions of each class for the feature pairs.

4.2.1 Iris Dataset

The Iris Dataset is a public testing or toy dataset which can be found in most software packages that apply data analysis, data mining, or analytics to some degree. The dataset's origin can be found from the public repository of databases on the UCI website for Machine Learning. The Iris dataset is considered in many frameworks as the go to for a basic application of tools on a "real" set of steps to apply a model on. This is mainly because it is easy to achieve very high results for a classification model with this data. It also only contains four features and three output classes that correspond to the types of Iris flower. The four features Petal Length, Petal Width, Sepal Length, and Sepal Width correspond to the flower's physical properties. The three classes are Setosa, Versicolour, and Virginica. The goal when using the Iris Dataset is to determine which class of flower the features represent. This dataset is utilized extensively to develop several of the conclusions later in this work. As we will see there are several preprocessing steps that can be applied in order to enhance the performance of a QML model and others that appear to have little effect. The Iris dataset is plotted in Figure 17. in two forms. The first is the distribution of values for each feature (along the diagonal) and a scatter plot of each feature pair is also shown. Please note that the lower and upper triangles of the matrix contain the same scatter plots, both are included for viewing preference.

4.2.2 Wine Dataset

The Wine dataset is another public toy dataset which can be found inside of Scikit-Learn or the UCI website for ML. The Wine data contains 14 features, and the goal is to use these features to classify one of three types of wine. These classes are given as (0,1,2). The correlation matrix is given per class in Figure 18. Showing the relationship between every two features in the data. The correlation matrix shows mainly that for class 0 the features are mostly negatively correlated while the features in class 2 are mostly positively correlated. We can use this information to create a classification model for these two classes. We can also see there is somewhat of a good mix of strongly negative and positively correlated features in the class 1. The dataset contains only continuous positive values. Features include alcohol, malic_acid, ash, flavonoids, color_intensity, and hue to name a few. This dataset was initially intended for use in the TFQ model. We did not retrieve enough conclusive results to apply the dataset there and instead include it in the work we planned to do with PennyLane.

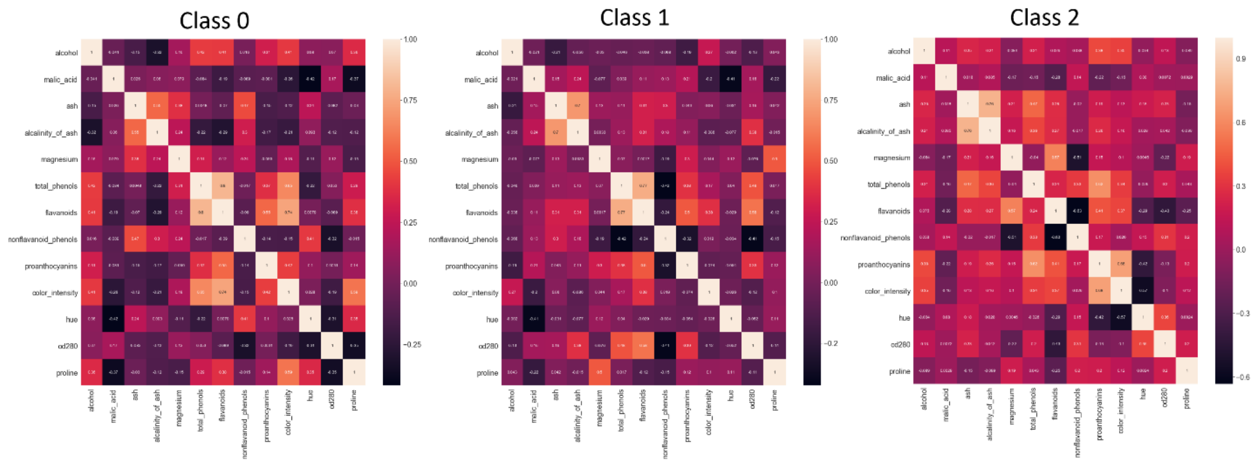


Figure 18. Wine dataset correlation matrices for each wine class.

5. Experiments

Experiments are split in general into two separate projects. The first of these was using TensorFlow Quantum (TFQ) from the work in [17] and the second was with PennyLane. The work done in PennyLane is mostly disjoint from the TFQ experiments. The experiments from TFQ aim to build a hybrid quantum-classical model that is able to surpass the results of the TFQ documentation model for the MakeBlobs dataset. We do this by making some changes to the encoding of the data, namely applying Amplitude Encoding. This method was developed by Schuld et al and as results will show are exceedingly better at encoding information [17].

After developing and building the argument for Amplitude Encoding, we then move to more concrete data analysis and steps of preprocessing, some of these steps are repeated from the TFQ section. The premise for preprocessing and transformation made while using PennyLane is much deeper and covers a large variety of steps and procedures. The work in TFQ is relatively confined to an analysis of Amplitude Encoding which we take for granted in the section Transformations for Learning In Quantum Models because it is fully developed through the TFQ experiments.

5.1 Quantum State Preparation

As we show quantum state preparation is a crucial factor for Quantum Machine Learning (QML) techniques to be successful. These steps are applied in the preprocessing phase as a means for encoding information prior to performing any learning. In this section we first develop three methods of state preparation: basis encoding, angle encoding, and amplitude encoding which are methods for preparing quantum states from classical data. We follow with the experimental setup and development of the TFQ models we tested. Finally, we outline the expectations for what we sought to solidify by using amplitude encoding in place of other methods.

5.1.1 Basis Encoding

Perhaps the most straightforward method of encoding techniques is basis encoding. Basis encoding is a method for preparing the computational basis of a qubit using an n-bit-string such as 0101. If a given feature vectors value is 5, we would want something resembling $|5\rangle$ but with basis encoding we prepare the binary bit-string making the state resemble $|0101\rangle$. It is important to note here that it is only the representation from decimal to binary that has changed, the feature vector value is still 5. The written form is $|0101\rangle$ but we are describing a matrix of binary values [46].

To perform this conversion and then preparation little is needed in terms of computation power. A simple binary parser will be able to convert a decimal number to binary, for the sake of brevity this can be done using the Equation 10. where the k^{th} value in binary will produce the binary bit string x that is encoded based on the desired precision of the binary string τ [46].

$$x = \sum_{k=0}^{\tau} b_k \frac{1}{2^k} \tag{10}$$

Then the super position of the basis states can be prepared to relate the binary input using Equation 9. where the binary string $x^m = (b_1^m, \ldots, b_N^m)$ and $b_i^m \in \{0,1\}$ for $i = 1, \ldots, N$. Resulting in the superposition of states $|D\rangle$. In Equation 11. this is performed by considering the binary data for two feature vectors of dimension two, in their binary state, they are $x^1 = (00,11)$ and $x^2 = (10,11)$.

$$|D\rangle = \frac{1}{\sqrt{2}}|0011\rangle + \frac{1}{\sqrt{2}}|1011\rangle \tag{11}$$

An amplitude vector therefore will have $\frac{1}{\sqrt{m}}$ for entries of the basis states for a given binary feature vector and zero in the rest as shown in Equation 12.

$$\alpha = \left(0,0,0,\frac{1}{\sqrt{2}},0,0,0,0,0,0,0,\frac{1}{\sqrt{2}},0,0,0,0\right) \tag{12}$$

In general, this is an exceptional way of encoding and preparing data for a quantum device. However, as the dimensionality grows the method requires more and more qubits. Even for a simple dataset, if values are continuous the number of qubits required is just to large even with discretization. For simple categorical variables with few dimensions this method could perhaps be used. The total number of amplitudes for a feature vector will be $2^{N\tau}$ which certainly makes this method unsuitable on NISQ era devices with today's capabilities. The following methods are both viable options for NISQ era devices granted the same parameter issues on a much smaller scale. In conclusion basis encoding may never see the light of day because of the large number of qubits required to effectively prepare a dataset. That said it is also unclear if it performs better than the following methods as it is not possible to test at this time.

5.1.2 Angle Encoding

Before beginning down the path of amplitude encoding let us introduce the encoding method it is compared to. In many online packages, TFQ included, another form of information encoding is applied, simply angle encoding. This name is not necessarily standard but literally describes the method. Angle encoding is a simple and effective method for information encoding, but it is not robust, and it does not map information from a classical state in well-defined fashion. Angle encoding is essentially the most basic form of encoding classical data into a quantum state. It has good results for problems such as parity checking or working with specific finite ranges of values that are largely unapplicable to real datasets.

Our explanation of angle encoding is not meant to be discouraging, but rather point out some of the inefficiencies of this method. Angle encoding is performed by applying a gate rotation about the x-axis $R_x(v_i)$ or y-axis $R_y(v_i)$ where v_i is the value to encode. In a Hilbert space a rotation about the y-axis applies an angle rotation, usually based on some π, hence the name angle encoding. Consider a classical dataset with three features where one record is represented by the vector $v = ([0.1],\ [0.2],\ [0.3])$. In an angle rotation the number of rotations applied will be same number of features in a dataset, i.e. we apply the R_x on v three times once for each dimension. The resulting sample in its gate form is shown in Figure 19. where $|q1\rangle$, $|q2\rangle$, $|q3\rangle$ are the qubits which will take the new states $|\psi_1\rangle$, $|\psi_2\rangle$, $|\psi_3\rangle$ representing the encoded vector values v_1, v_2, v_3.

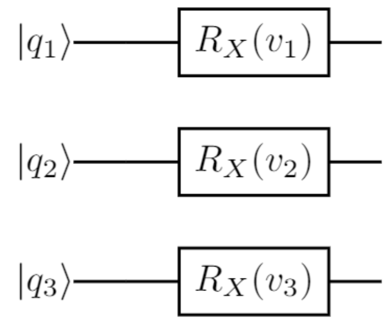


Figure 19. Angle preparation of classical data into the state $|\psi_1\rangle$, for a 3-Dimensional sample. This method is utilized in the TFQ experiments for the base model following the documentation.

We can now see that an n-dimensional sample would take n number of qubits to generate the set of quantum states. This method creates a simple representation of the data with roughly the same complexity as it did in its classical representation. This makes angle encoding attractive for simple datasets which may have few discrepancies between samples. As we have mentioned NISQ era devices have a limited number of qubits and keeping more than a few coherent for an experiment is difficult.

5.1.3 Amplitude Encoding

Amplitude encoding maps classical data into the amplitude of a qubit. Conceptually it can be thought of as any other encoding that must represent but also losslessly be able to be encoded and decoded. As an example we use one-hot encoding in classical data preprocessing to take a dense vector such as ([1], [2], [3]) to a sparse vector like ([0, 0, 1], [0,1,0], [1,0,0]) where each integer value in the dense vector is represented by a one at the index of the value in the sparse vector. The difference with amplitude encoding is that it changes the *computational basis* for allowing supposition of states differently than basis and angle encoding. One-hot encoding changes a sample's *form* from dense to sparse [46]. In either the classical or quantum case this preprocessing step can largely impact the performance of a learning method, in some cases it even determines whether a method will learn at all [46].

The process of applying amplitude encoding begins by converting a dataset to their angle representations with multi-controlled rotations. This process is performed using the Equation 13., where the angle θ is created via a vector, v^i represents i^{th} classical sample, and β is the angle based on the $arcsin$ of the number of dimensions in the sample space [46].

$$|\psi\rangle = R\left(v^i, \beta\right)|q_1 \ldots q_{s-1}\rangle|q_s\rangle \tag{13}$$

In the circuit implementation a state $|\psi\rangle$ is *prepared* by a "cascade" of n R_y rotations where n represents the power in binary for encoding a feature vector v^i. For example, if a dataset has ten features, n needs to be four, because n equal to three at most encodes samples with eight features. The complexity or depth of a circuit can be seen in the circuit of Figure 20. Just by comparing the number of gates between Figure 19. and Figure 20. we can see a dramatic difference. With just three qubits we can also see the limitations of applying amplitude encoding for large datasets in NISQ era devices. Although the problem does not generally become the number of qubits representing the state $|\psi\rangle$, we face the issue of several gate operations, so many that coherence is again a problem. While in the TFQ work we only use three qubits (amplitude encoding) in place of four (angle encoding) the number of quantum gates applied is roughly ten times the angle encoding method [46].

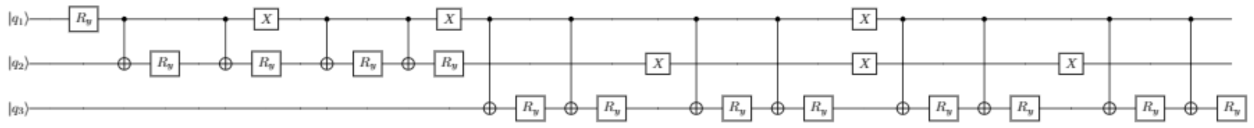


Figure 20. Amplitude encoding of the state $|\psi_1\rangle$, for a 3-Dimensional sample. Noting that the complexity of this method is clearly larger than a simple angle preparation method.

In summarization to encode a sample using amplitude encoding there are two steps: (1) compute the angle using Equation 13. then (2) apply the cascade of R_y on the computed angles such as in Figure. For large datasets, this method has been generalized recently by Araujo et al. and takes into account both steps [60]. They exploit the classical divide and conquer algorithm to encode n-dimensional samples.

With these three encoding methods we have developed a basis for converting classical data into quantum states. As basis encoding suggests the method is essentially to exhaustive for NISQ era devices albeit theoretically a powerful solution. Following this we showed that angle encoding is a rather mundane solution as it simply encodes a feature vector to a set of qubits by applying the a rotation to the values. This method we consider as the simplest approach for preparing a quantum state and other gates could be used in place of R_y such as R_x. In terms of performance we show this method is noisy and hard to interpret [17]. On the other hand, with amplitude encoding we apply a much more complex solution which only grows more complex as the dimensionality of a dataset increases. With a more complex circuit or increased circuit depth amplitude encoding begins to be concerned with issues such as decoherence. With time and developments in QC amplitude encoding will likely become more broadly adopted. This method is vastly superior to angle encoding and we show these findings in our experiments.

5.2 TFQ Experimental Setup

Although the experiments in this portion are performed in a classical device which simulate a quantum computer the methodology and setup are the same. Again, this was done mainly because there is currently no quantum computer publicly made available from Google for TFQ. We perform two sets of experiments based on the number of training epochs the first being eight-epochs (experiment one) and the second is fifty-epochs (experiment two). In both cases the task is to classify the two classes -1 and +1. Our hypothesis was to evaluate the training behavior in order to show (i) amplitude encoding leads to a model which will converge at a higher accuracy sooner and (ii) that amplitude encoding leads less erratically and therefore more effective training over time.

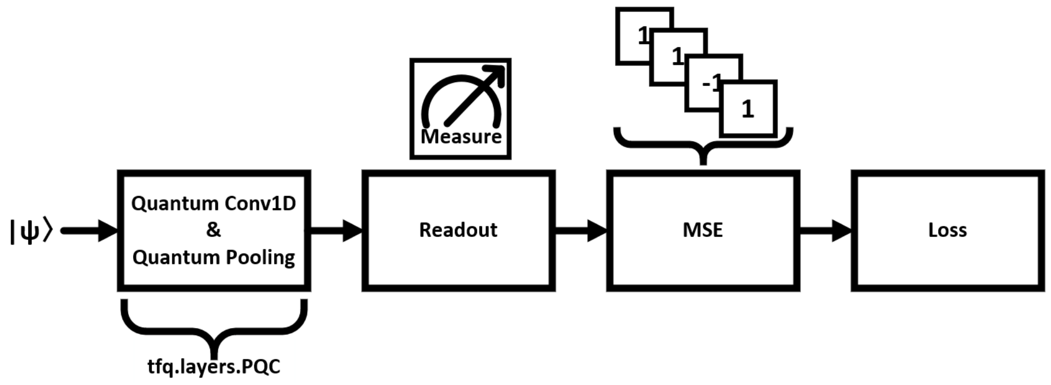


Figure 21. Simplified architecture implemented for each TFQ model. This is the entire architecture for the QCNN model we develop here uses. The additional two models add an MLP between the 'Readout' and 'MSE' blocks.

The models we mainly compare in this section are of the type quantum-classical meaning that a portion of the network is developed using quantum methods and another portion is classically developed. As we have said the classical component is an MLP. The model's quantum component is a Quantum Convolution Neural Network (QCNN). The QCNN model architecture we used is shown in Figure 21. This is the core of our model which can be found in the TFQ documentation and it is also used in the analysis but left unchanged as a baseline. The model consists of some number of one dimensional quantum convolutions (QConv1D) as the, in our case there are two, with a quantum pooling (QPool) layer immediately following each QConv1D. Readout of the quantum state performed after the circuit is done by applying a Z gate on the qubits after the final layer of pooling. The circuit for QConv1D is shown at the top of Figure 22. as well as the QPool circuit which is below. As you may notice the QPool layer is non-parametric, it simply applies the Pauli X, Y, and Z gates to the circuit.

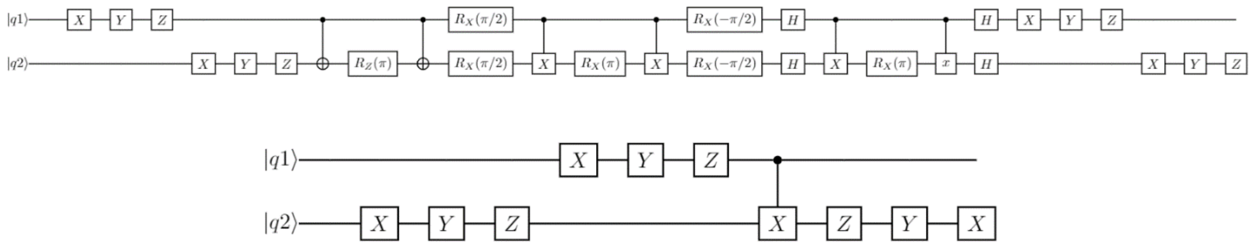


Figure 22. Quantum circuits used in the QCNN for parametric 1D quantum convolution (Top) and quantum pooling (bottom).

Our TFQ experiments here use three models, two of these models follow from TFQ's documentation for Quantum Convolutional Neural Networks (QCNN). The third is of our own design. We call TFQ's models (a) QCNN as the base model which contains the fundamental quantum layers in all three models, (b) Angle-Hybrid which applies angle encoding for state preparation, and (c) Amplitude-Hybrid which applies amplitude encoding for state preparation. The latter two are hybrid models so they will contain the QCNN and MLP in a sequential order. Our methodology is applied to the MakeBlobs datasets shown in Figure 23. In total, for both eight and fifty-epochs, we use eight datasets. Each dataset is given in a two-dimensional plot to show the range of difficulties based on centroid. The data begins with a centroid of 0.6 where the two classes are mostly overlapping and ends with a centroid of 2.0 where the two classes are much more separable. We do not include any noise for these datasets. The datasets each contain four features and we split the data to have 2,048 samples for training, 512 for validation within the model, and 512 for testing or evaluation after training.

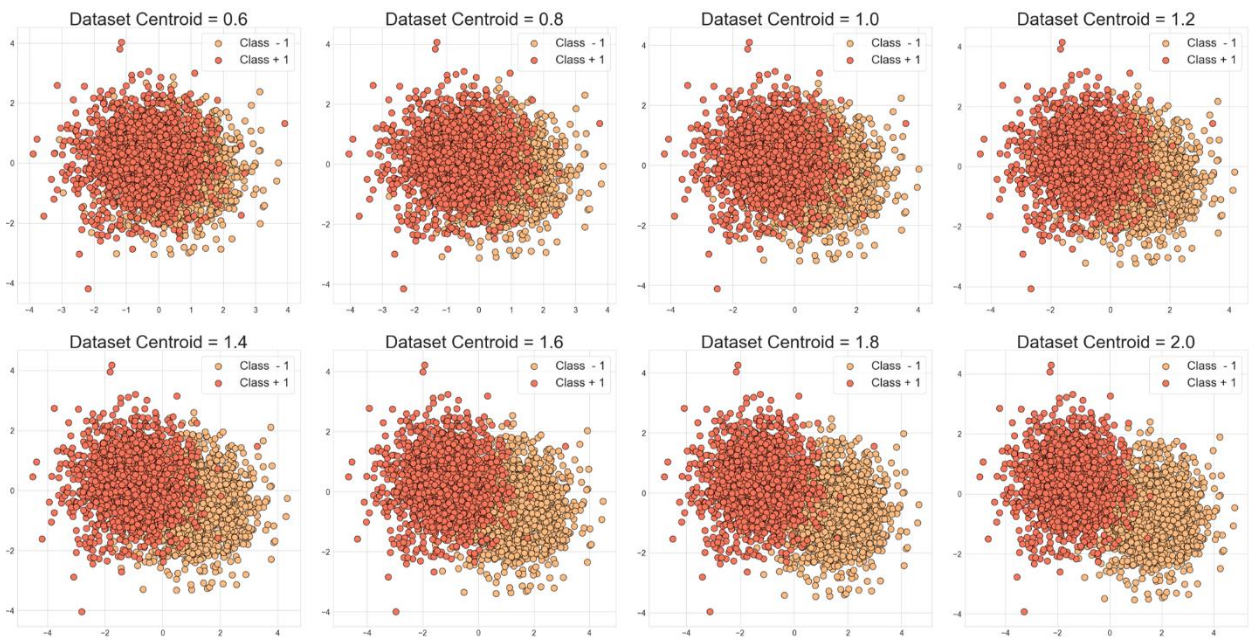


Figure 23. The eight MakeBlobs datasets used in throughout the TFQ experiment process. Data Centroid or CenterBox was moved progressively by 0.2 for each dataset to have a range of complexities.

We next show the models' function parameters and the metric calculations we apply in the analysis. We calculate and show the results for loss, accuracy, precision, recall, and F1-score. Many of the same model functions are applied from the TFQ documentation. The accuracy is calculated as the sum of correct predictions in Equation 14. and loss is calculated using mean square error (MSE) in Equation 15.

$$accuracy = \frac{1}{n}\sum_{i=0}^{n}(y_i = \widetilde{y}_i) \tag{14}$$

$$loss = MSE = \frac{1}{n}R\sum_{i=0}^{n}\left(y_i - \widetilde{y}_i\right)^2 \tag{15}$$

In both equations for accuracy and loss y_i is the observed or real value and $\widetilde{y}_i$ is the predicted value. The optimizer used in the MLP of the model is Adaptive Moment Estimation (Adam) optimizer. We set the learning rate η equal to 0.02, following the same approach as the TFQ documentation. In Equation 16. θ_{t+1} is the current gradient of the stochastic gradient descent (SGD) based on the previous gradient θ_t,

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v_t}}}\widehat{m_t}, \tag{16}$$

where the weight $\widehat{v_t}$ in Equation 17., and momentum $\widehat{m_t}$ in Equation 18. are defined as:

$$\widehat{v_t} = \frac{\beta_2 v_{t-1} + (1 - B_2)g_t^2}{1 - \beta_2^t} \tag{17}$$

$$\widehat{m_t} = \frac{\beta_1 v_{t-1} + (1 - B_1)g_t}{1 - \beta_1^t}. \tag{18}$$

Therefore, $\widehat{v_t}$ and $\widehat{m_t}$ are estimates of the gradients' mean and variance respectively, and β_1 and β_2 are the forgetting factors. Momentum and forgetting are the two key factors which make Adam widely adopted optimizer over the standard SGD. The final dense layer of each model (QCNN, Angle Hybrid, and Amplitude-Hybrid) apply a $tanh$ as the activation function since the two classes we are trying to classify are (-1, +1). The tanh function or hyperbolic tangent is defined using the exponential in Equation 19. where x is the current sample weight

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{19}$$

Precision in Equation 20. is calculated as a ratio of true positive (TP) and false positive (FP)

$$Precision = \frac{TP}{TP + FP}, \tag{20}$$

whereas recall in Equation 21. is calculated as a ratio of TP and false negative (FN)

$$Recall = \frac{TP}{TP + FN}. \tag{21}$$

F1-score is calculated as the relationship or harmonic mean between precision and recall in Equation 22.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{22}$$

It can also be shown that F1-score, Precision, and Recall derive accuracy. These last three Equations 20, 21, and 22. are used to gather additional statistics about each model's behavior not only in this section but also later in the section with PennyLane.

5.3 Analysis of TFQ Experiments – Hybrid Models

In our first set of tests we discuss and run the eight datasets for eight-epochs. In general, we hypothesized that running the model for a short number of epochs with amplitude encoding would lead to better learning results. This would also imply that fewer training epochs are needed to achieve better results. The values (-1) and (+1) in the two tables of this subsection header refer to the class label's individual metric. The second set of tests we let the model run for fifty-epochs to get a representative sample of the learning history. The major analysis comes from the combination of both the training plots and table. We note that the QCNN model is mostly included for reference as we go through the analysis. This section is mainly concerned with the two hybrid models' ability to perform and the results the two encoding methods provided.

5.3.1 Experiment: Eight-epochs

The training validation results are given in Table 3. These results indicate Amplitude-Hybrid was the top performer in every evaluation metric. Let us first take a look at the cluster's centroid distance as our metric for classification difficulty. For the first two centroids, 0.6 and 0.8, the most difficult datasets of these experiments, the Amplitude-Hybrid model achieved roughly 2% better results for accuracy than the Angle-Hybrid model. Working down through the table we see that at 1.4 centroid distance the Amplitude-Hybrid model achieves an accuracy of 90%. In contrast Angle-Hybrid's achieves an accuracy of 90% only at the 2.0 centroid distance.

For each model we see that it improves as the centroid distances spreads further apart or as we move down through the table. For Angle-Hybrid by the time we are at 1.4 centroid distance we see that almost every metric, excluding loss, for each class is above 90%. What we show in the plots of Figure 24. are rather interesting when evaluating Angle-Hybrid with Amplitude Hybrid. These plots show the training validation accuracy and loss of each dataset per each model (QCNN, Angle-Hybrid, and Amplitude-Hybrid). Plotting here shows that with Angle-Hybrid the models learning behavior is flat which does not associate steady optimization and learning. The opposite can be said about the Amplitude-Hybrid models. Aside from the 1.8 and 2.0 centroid distances (which are the easiest) the models appeared to learn fast and consistently over the eight epochs. These results with Table 3's results paint a compelling picture that in just a few epochs a model using amplitude encoding is far superior.

Averaging the results of Table 3. shows some additional results, these are also in favor of Amplitude-Hybrid. We determine the best model here again by looking at accuracy and the combined class results per metric. With the QCNN and Angle-Hybrid results show they are similar in many cases. These results are not surprising as they utilize the same angle encoding method and because of this the results are tightly correlated for every difficulty. Therefore, with or without the addition of the classical/hybrid component there is little accuracy improvement over the eight epochs. In the best case, Angle-Hybrid was only two percent better in accuracy over the QCNN model but in the majority of cases the accuracy was within 0.5% for both these models. It follows that the models utilizing angle encoding limit the ability of overall learning and from Table 3. the additional components of the Angle-Hybrid model do not improve the performance. In the next experiment there are similarities that are reminiscent of these results.

Table 3. Eight-Epoch Post Analysis Model Metrics.

Model	Centriod	Loss	Accuracy	Precision (**-1**)	Recall (**-1**)	F1-Score (**-1**)	Precision (**1**)	Recall (**1**)	F1-Score (**1**)
QCNN	0.6	0.829	0.683	0.755	0.564	0.646	0.639	0.808	0.713
Angle-Hybrid	0.6	0.965	0.687	0.729	0.618	0.669	0.655	0.760	0.703
Amplitude-Hybrid	0.6	0.889	0.708	0.767	0.618	0.684	0.667	0.804	0.729
QCNN	0.8	0.739	0.726	0.796	0.625	0.700	0.679	0.832	0.748
Angle-Hybrid	0.8	0.808	0.740	0.779	0.687	0.730	0.708	0.796	0.749
Amplitude-Hybrid	0.8	0.702	0.763	0.819	0.690	0.749	0.721	0.840	0.776
QCNN	1.0	0.660	0.767	0.832	0.683	0.750	0.720	0.856	0.782
Angle-Hybrid	1.0	0.672	0.781	0.807	0.751	0.778	0.757	0.812	0.783
Amplitude-Hybrid	1.0	0.533	0.826	0.864	0.782	0.821	0.792	0.872	0.830
QCNN	1.2	0.595	0.792	0.848	0.725	0.781	0.750	0.864	0.802
Angle-Hybrid	1.2	0.563	0.800	0.812	0.793	0.803	0.789	0.808	0.798
Amplitude-Hybrid	1.2	0.396	0.865	0.897	0.832	0.863	0.836	0.900	0.867
QCNN	1.4	0.544	0.826	0.864	0.782	0.821	0.792	0.872	0.830
Angle-Hybrid	1.4	0.475	0.833	0.836	0.839	0.838	0.831	0.828	0.829
Amplitude-Hybrid	1.4	0.281	0.902	0.930	0.874	0.901	0.875	0.932	0.903
QCNN	1.6	0.507	0.859	0.874	0.847	0.860	0.844	0.872	0.858
Angle-Hybrid	1.6	0.406	0.853	0.845	0.874	0.859	0.863	0.832	0.847
Amplitude-Hybrid	1.6	0.183	0.939	0.949	0.931	0.940	0.929	0.948	0.938
QCNN	1.8	0.484	0.871	0.862	0.889	0.875	0.880	0.852	0.865
Angle-Hybrid	1.8	0.357	0.876	0.861	0.904	0.882	0.894	0.848	0.870
Amplitude-Hybrid	1.8	0.113	0.966	0.965	0.969	0.967	0.967	0.964	0.965
QCNN	2.0	0.473	0.892	0.881	0.912	0.896	0.904	0.872	0.887
Angle-Hybrid	2.0	0.286	0.902	0.878	0.938	0.907	0.931	0.864	0.896
Amplitude-Hybrid	2.0	0.101	0.974	0.966	0.984	0.975	0.983	0.964	0.973

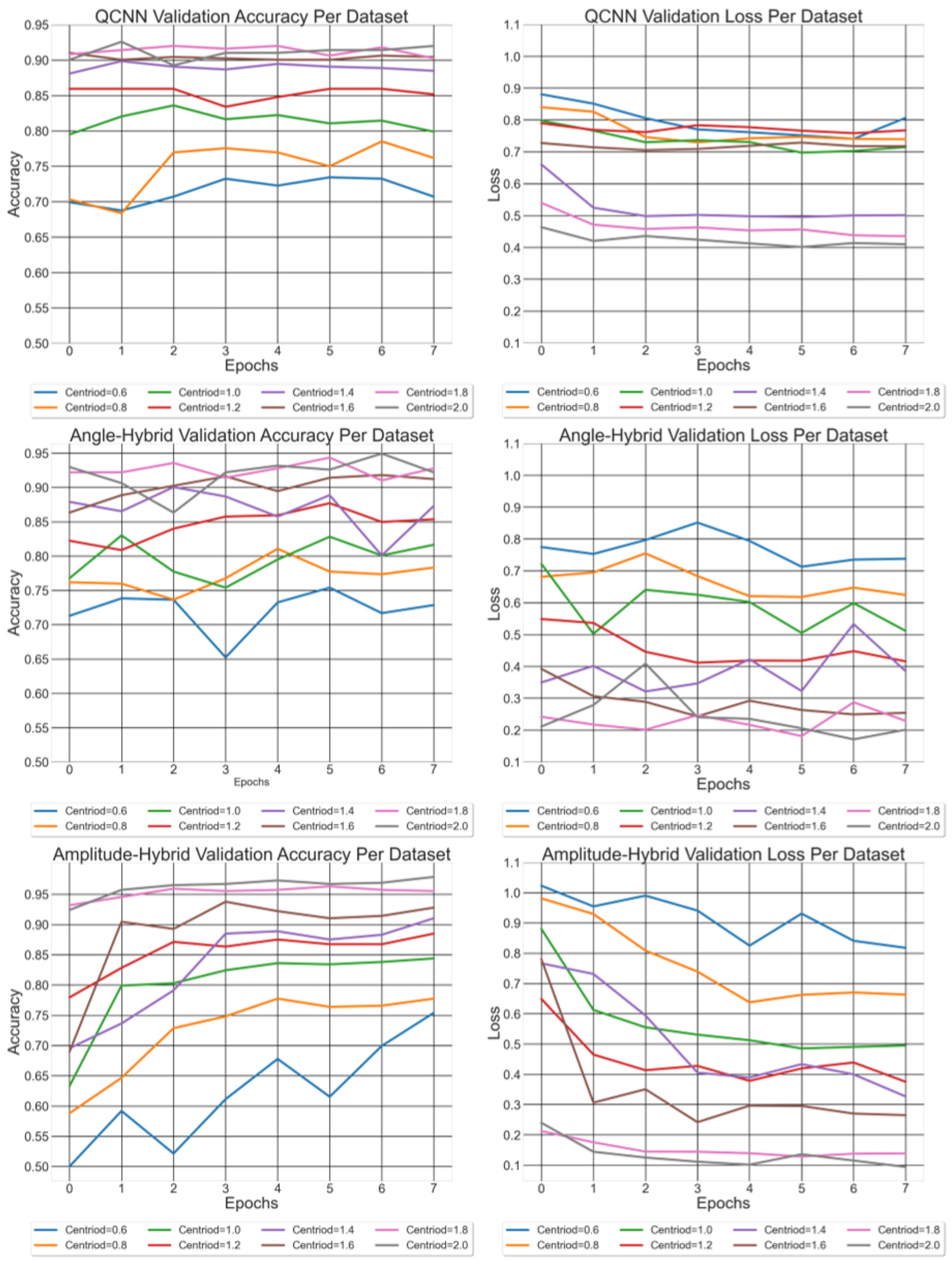


Figure 24. Training histories of each model over eight epochs. Top to bottom are the models QCNN, Angle-Hybrid, and Amplitude-Hybrid. Left to right we show each model's Accuracy and Loss history.

5.3.2 Experiment: Fifty-epochs

We share the same form of output in the plot of Figure 25. and Table 4. for the results here but this time allow the model to run for fifty epochs. Here we are looking to show consistency over the learning period and any scenarios that stick out as red flags. First, we recap that over the eight epochs of training, where things generally looked to be increasing in the right places for each model. This trend for the most part continues again but it is apparent that a few models performed worse than they did after fifty epochs. Glancing at Table 3. and comparing it with Table 4. will show that in some cases Angle-Hybrid overall now appears more appealing than before. We must mention that arbitrarily training for a larger number of epochs is not always an effective means of achieving increased accuracy or any other metric for that matter.

Much of the evaluation is given graphically for the fifty-epoch experiment. This is in part because we want to look at the history and see what behavior the model exhibits. Why does the table of metrics not answer these questions? Table 4. results occur after fifty-epochs and shows the testing of data on the final epoch of the model. These singular values do little to shed light on the history.

Taking a look at the Amplitude-Hybrid plots at the bottom of Figure 25. the range of 10-20 epochs is a region that answers our hypothesis. It is in this range that every model appears to have fit to the data as best as it can, by the peaks in accuracy, drops in loss, and what appears to be gradual overfitting after. In a sense we wanted to show the model overfitting after some number of epochs. It is by chance that several of these were in a range of roughly ten epochs. The appearance of overfitting as we discussed implies the model will begin to retain to much influence from the training data. Overfitting here signals the model has done all it can to learn the information present in the data. This behavior to some degree is desirable because it signals our model is learning from the data in a consistent expected behavior whereas we will soon discuss angle encoding appears to not. Overall, this implies the best learning behavior that can be achieved is by using Amplitude encoding and it is consistent.

With the QCNN and Angle-Hybrid models we can see the difference in historical outcomes from epoch to epoch in Figure 25. Results for both indicate these models never had a best fit to the data. In a few cases the models made good improvements to their overall metrics. However, in several cases some of the outcomes train well over the first two dozen epochs and then again appear to improve after another two dozen later. Key to this analysis is this appearance things are improving. Looking again at the 10-20 epoch region it is clear previous conclusions for Amplitude-Hybrid cannot be draw here. The Angle-Hybrid model in this region jumps from a “high” accuracy and then in a few epochs drops 5% to 15%. This occurs not only here but during the entire training period. This behavior implies that learning is failing as the heuristic tries to “guess” a better solution than it did in a previous iteration or epoch guessing is a much less consistent behavior we would want to see in our models.

Now that we have discussed the visual observation of the histories let us go back to Table 4. and make a few more remarks. Reviewing the two hybrid models for centroid distances 1.0, 1.2, and 1.4 we see that the difference in accuracy, in favor of Amplitude-Hybrid, is 0.0%, 0.976%, and 3.125% respectively. Not only accuracy but also the other metrics such as F1-score show there is little difference between these models. The Recall and Precision of the Angle-Hybrid model is actually better here than Amplitude-Hybrid. Although this is the case, the information in Table 4. is misleading and we show it here to disprove any counter arguments for what we have discussed so far. Over the training period for these datasets we can see each Angle-Hybrid model is bouncing 5-10% epoch to epoch while the Amplitude-Hybrid model is very slightly changing for these datasets. Again, looking at 1.2 more specifically in terms of Figure 25. we can see that the Amplitude Model roughly peaks at 14 epochs stays there for a few epochs and slowly declines for the rest of training. When we look for similar behavior in Amplitude-Hybrid we can see the same thing is happening for every other dataset. However, this is not visible in the Angle-Hybrid model. When considered with Table 4. we conclude that the table results are not as strong because of overfitting. The same is not true for the Angle-Hybrid model. We believe the training in Angle-Hybrid was so erratic that the model performance cannot be accurately assessed other than to say it improved over fifty-epochs, though we could go so far as to say this improvement is no more than coincidence.

Table 4. Fifty-Epoch Post Analysis Model Metrics.

Model	Centriod	Loss	Accuracy	Precision (-1)	Recall (-1)	F1-Score (-1)	Precision (1)	Recall (1)	F1-Score (1)
QCNN	0.6	0.798	0.687	0.729	0.618	0.669	0.655	0.760	0.703
Angle-Hybrid	0.6	1.013	0.683	0.771	0.541	0.636	0.634	0.832	0.719
Amplitude-Hybrid	0.6	1.171	0.685	0.798	0.515	0.626	0.629	0.864	0.728
QCNN	0.8	0.708	0.753	0.785	0.713	0.748	0.726	0.796	0.759
Angle-Hybrid	0.8	0.818	0.742	0.821	0.633	0.715	0.690	0.856	0.764
Amplitude-Hybrid	0.8	0.953	0.718	0.835	0.561	0.671	0.657	0.884	0.754
QCNN	1.0	0.631	0.787	0.814	0.755	0.784	0.762	0.820	0.789
Angle-Hybrid	1.0	0.641	0.796	0.859	0.721	0.784	0.750	0.876	0.808
Amplitude-Hybrid	1.0	0.727	0.796	0.898	0.679	0.773	0.732	0.920	0.815
QCNN	1.2	0.568	0.830	0.832	0.835	0.834	0.827	0.824	0.825
Angle-Hybrid	1.2	0.497	0.839	0.891	0.782	0.833	0.797	0.900	0.845
Amplitude-Hybrid	1.2	0.522	0.849	0.930	0.763	0.838	0.791	0.940	0.859
QCNN	1.4	0.520	0.849	0.843	0.866	0.854	0.855	0.832	0.843
Angle-Hybrid	1.4	0.384	0.865	0.903	0.824	0.862	0.831	0.908	0.868
Amplitude-Hybrid	1.4	0.367	0.896	0.948	0.843	0.892	0.853	0.952	0.899
QCNN	1.6	0.486	0.876	0.864	0.900	0.882	0.891	0.852	0.871
Angle-Hybrid	1.6	0.300	0.896	0.916	0.877	0.896	0.877	0.916	0.896
Amplitude-Hybrid	1.6	0.260	0.925	0.970	0.881	0.924	0.886	0.972	0.927
QCNN	1.8	0.467	0.888	0.875	0.912	0.893	0.903	0.864	0.883
Angle-Hybrid	1.8	0.242	0.917	0.926	0.912	0.919	0.909	0.924	0.916
Amplitude-Hybrid	1.8	0.160	0.949	0.975	0.923	0.949	0.924	0.976	0.949
QCNN	2.0	0.347	0.902	0.889	0.923	0.906	0.916	0.880	0.897
Angle-Hybrid	2.0	0.300	0.894	0.906	0.885	0.895	0.882	0.904	0.893
Amplitude-Hybrid	2.0	0.115	0.970	0.976	0.965	0.971	0.964	0.976	0.970

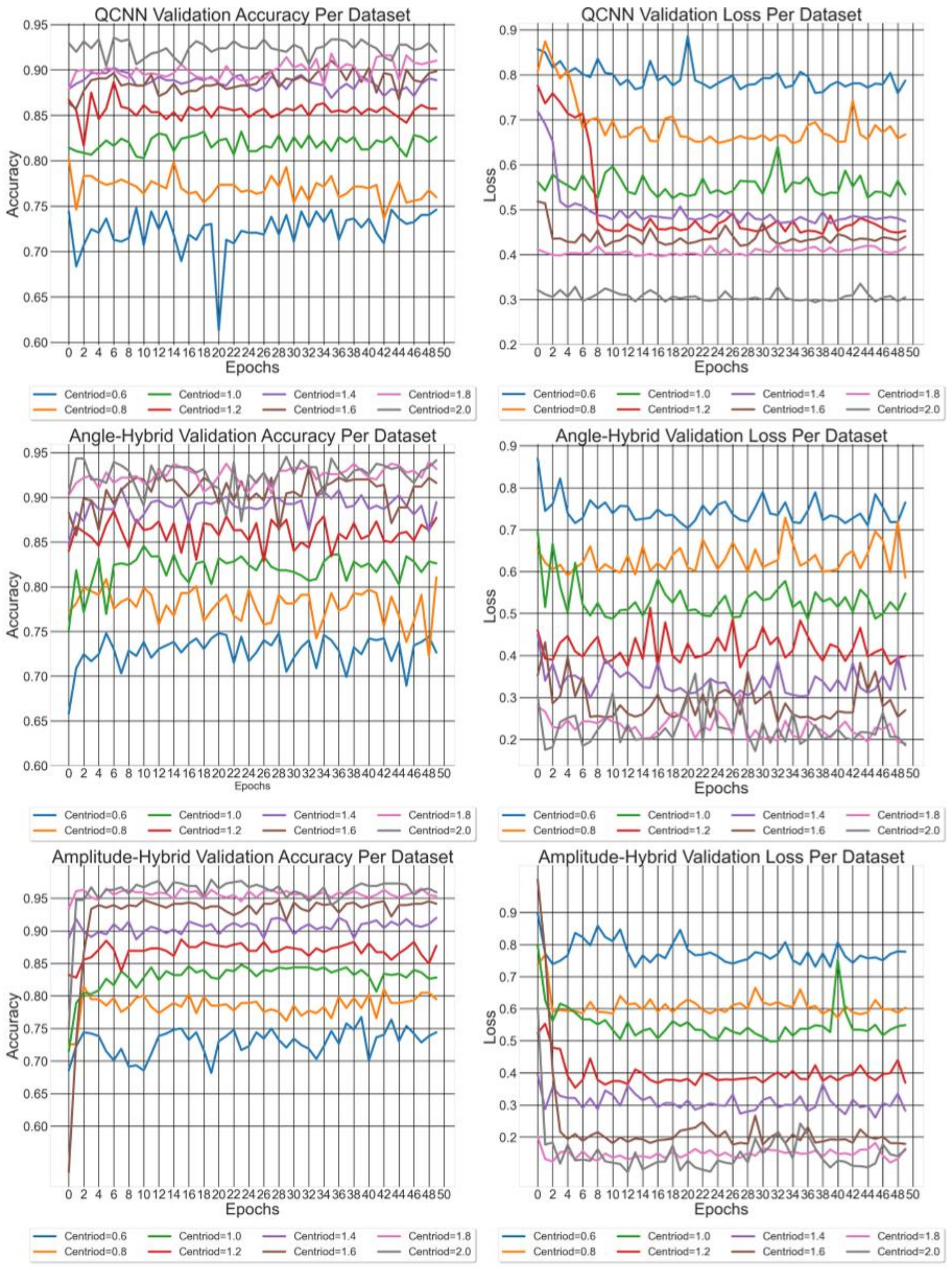


Figure 25. Training histories of each model over fifty epochs. Top to bottom are the models QCNN, Angle-Hybrid, and Amplitude-Hybrid. Left to right we show each model's Accuracy and Loss history.

5.4 Transformation for Learning in Quantum Models

The experimentation process of this section is done using the PennyLane Variational Quantum Classifier (VQC) to evaluate how certain preprocessing steps can impact the training of fully quantum algorithms. We cannot cover every method that can be applied to the dataset but test several and evaluate them on different data sources. Most of the methods here stray from any hyperparameter tunning as that is not the same as preprocessing data. We show one example of this tuning and note that increased performance would undoubtedly be achieved via hyperparameter tunning. We want to focus more on generalizations of the methods for preprocessing and whether there is some commonality among them. Many of the transformations of the data are easily performed by software packages, here we develop most methods using either simply Numpy array manipulations or packaged Scikit-Learn functions. Each dataset was created with a sample size of 400 points and a random state of 11.

5.4.1 Analysis of Make Blobs Dataset

The MakeBlobs generator we have mentioned above has the ability to control several parameters when generating a dataset. This ability is utilized in the generation of three datasets we call Blobs-4F, Blobs-3F, and Blobs-2F. The three datasets are created with the intention of performing binary classification with only two features. The following explanation of the three blob datasets is given with a note that the reader could implement a similar solution using different methods. In Blobs-4F we define the number of features to be four and the number of target output classes (centers in the function parameter list) as three, and set the CenterBox equal to (-3.5, 3.5). These parameters generate different distributions for the features than if we used two classes and two features. This different distribution is what we aimed to capture using these features. The plot of the points below in Figure 26. contains these three datasets. It shows how the points are spread out differently than intuition would initially guide the user of the generator function to believe. The third parameter of CenterBox is chosen as reasonably simple so the classifiers would not need to formulate highly non-linear solutions in determining the decision boundary of the two classes.

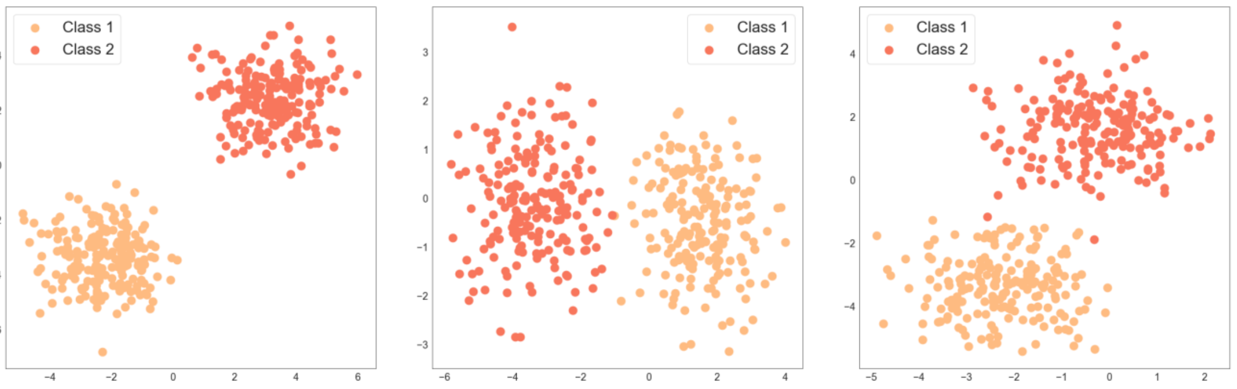


Figure 26. Three blobs datasets before any preprocessing or transformations. Left to right are Blobs-4F, Blobs-3F, and Blobs-2F.

We then select the first two features of the dataset. As the Blobs-4F dataset has four features this is the zeroth and first columns of the data. We choose these essentially because in Blobs-3F we must choose two columns again and in Blobs-2F we only have two columns to choose from.

In general, we are purely interested in gathering three slightly different distributions of points such that the datasets can vary in complexity and representation. Following the selection of columns, we begin to formally apply methods of preprocessing and then aim to answer a few questions that will become apparent and will be elaborated on shortly. So, the reader is comfortable in our explanation of dataset creation we consider the extreme case of a philosophical dataset we could have created with fifty-four features and twelve output classes. In this case, we perform the same selection process where we would choose the zeroth and first columns and just two of the output classes. The systematic process of this decision must be kept the same for Blobs-3F and Blobs-2F as different results can be gathered from different features e.g. features twenty seven and thirty four might be perfectly separable for the two classes with the naked eye while features two and three may might share large overlap for the two classes.

We now develop the preprocessing that was gathered from the initial investigation into the Iris dataset and apply additional steps to garner more comprehension of the VQC capacity with two dimensional datasets for binary classification. The preprocessing steps were applied to the datasets in different combinations to individually test which combinations would perform better collectively or individually. Preprocessing is often performed in an iterative manner and we develop the applications of each method here in an iterative manner. When reading this section, we build on points made and briefly refer back to them again as they steps are repeated or slightly changed.

We begin the preprocessing phase by utilizing the MinMaxScalar function on each of the datasets Blobs-4F, Blobs-3F, and Blobs-2F. MinMaxScalar is applied using the same generalized minimum maximum normalization in Equation 23. with the additional piece for scaling given in Equation 24 where X is the feature vector or column and x is the column value, $x_{normlized}$ is the intermediate min-max normalization performed. Each dataset is therefore normalized roughly between [0,1]

$$x_{normlized} = \frac{x - \min(X)}{\max(X) - \min(X)} \tag{23}$$

$$x_{scaled} = x_{normlized} * \frac{1}{2}(\max(X) - \min(X)) + \min(X) \tag{24}$$

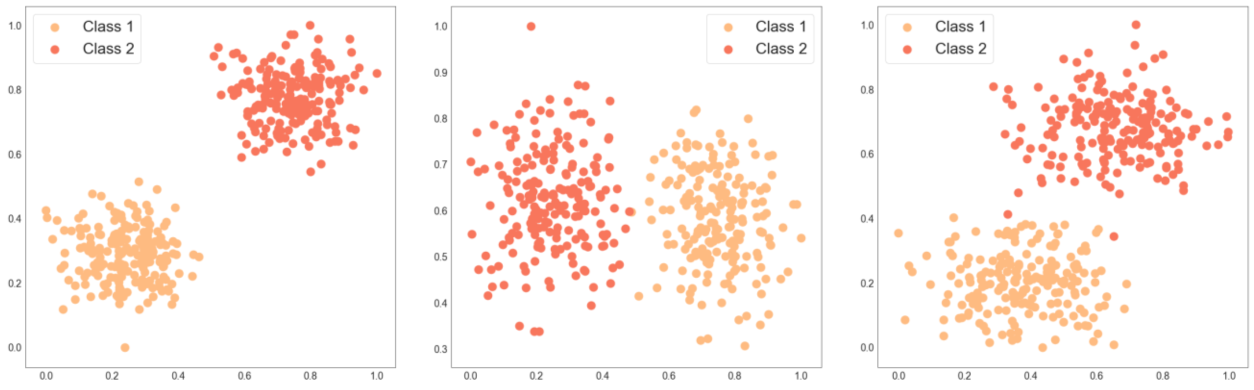


Figure 27. Blobs after applying PennyLane normalization. Left to right are Blobs-4F, Blobs-3F, and Blobs-2F.

The application of the transformation is performed column wise such that each column is normalized within itself, values from other columns do not have influence outside of their column. The utilization of MinMaxScalar follows from the Iris dataset. The Iris dataset in the example online was scaled first using this technique. Additionally, we found that in the preprocessing of Iris each feature is multiplied by a coefficient; however, the reason for this additional constant coefficient bewilders the authors comprehension. We test results of this scaling coefficient on several datasets and, in some cases, it led to better results in several cases. The coefficient is calculated column wise similar to Equation 22. where the final addition of the minimum of a column is replaced with division by two. We produced this conclusion after several tests. At best we understand that this might have been to separate the features from one another making the data have more space between classes [52], but this essentially rescales the data to a larger feature space than normalization left it. We include this step of preprocessing as we investigate the methods further, each dataset set is tested with and without this constant coefficient.

Once scaled via MinMaxScalar and/or multiplied via the coefficients the data is padded to four dimensions with two columns of all 0.3 and 0.0 for features three and four respectively. This was done in the Iris example in the PennyLane documentation and padding by the specific value does not have a large impact on the learning behavior. The padding of the features must be done here because the VQC function utilizes two qubits. The two qubits encode four features in the amplitude encoding method by the angle method described above in Quantum State Preparation. Once in their angle form state preparation is applied and the classifier can be applied to the data to calculate loss and accuracy as any other learning model. The data by this point looks oldly dissimilar from the initial blobs of data generated. This is in part because an additional normalization is applied to the data aside from MinMaxScalar. This follows from PennyLane as well and is shown in Equation 25. where x_i is the value of a column.

$$normalization = \sqrt{\sum_{i=0}^{n} x_i^2} \tag{25}$$

This is applied column wise to each feature in the data. The plot of the data is shown in Figure 28. where the second row of scatter plots is the zeroth and third features of the padded then normalized data after applying the angle method. The angles method essentially creates three angles with the input vector and outputs five values as a feature vector. The values are most heavily represented in the zeroth and third features of the feature vector.

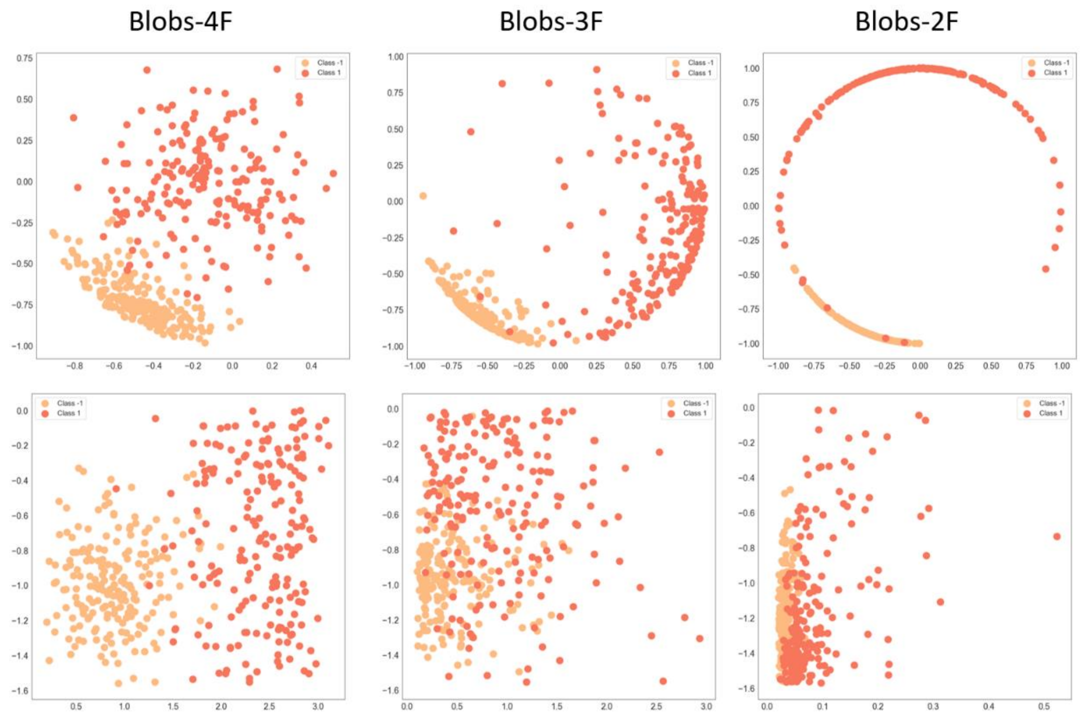


Figure 28. Top: blobs datasets after applying MinMaxScalar and padding to four features. Bottom: feature vectors after applying angle translation method to data above. Note these are only two features of the feature vectors.

We continue here with the model parameters defined in Table 5. below. We use the same random seed as in the documentation to keep consistent with the starting point of the random batch values. The same is true for number of layers and starting point of optimizer and the optimizer's step size. We do not use any momentum. The training of every model discussed in the following analysis utilizes this configuration and later an additional two layers are used and discussed.

Table 5. Parameters utilized in basic VQC from PennyLane Documentation

Qubits	Layers	Optimizer	Step Size	Optimizer Shape	Training Set	Test Set	Batch Size
2	6	Nesterov	0.01	(6,2,3)	75%	25%	5

As a method of confirmation, we develop and perform the same preprocessing explained above as the PennyLane documentation utilizing the Iris dataset within Scikit-Learn and achieve essentially the same results. This sanity check is a vital component to the rest of the analysis. The preprocessing of the Iris dataset is not made publicly available, the steps discussed so far were captured after several iterations of trial and error. Training the Iris dataset with the values as is or with a different normalization such as L1 normalization or L2 normalization can yield exceptional post training results for accuracy, precision, recall, and F1-Score. However, the results and the behavior of training are different from the PennyLane documentation. We admit at this time we found no reasonable conclusion for scaling the data the way which is given in the Iris dataset and also consider the possibility the data was scaled using a set of different steps but produces the same values of the features. We include the training results for the Iris dataset below in Figure 29. for the Scikit Learn dataset we processed to match PennyLane's in the form of the VQC kernel and hyperplane plotted over both datasets after learning.

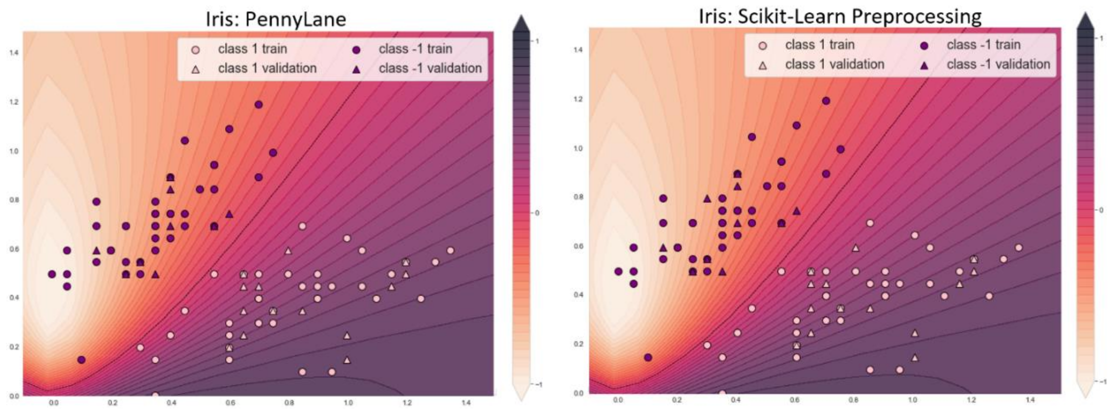


Figure 29. VQC decision boundaries for the Iris datasets. Left: PennyLane sample of data from already preprocessed. Right: Scikit-Learn Iris dataset after discovering and applying the preprocessing methods as the PennyLane dataset.

Before proceeding we describe two final characteristics of the two datasets. The first being the order of values in the dataset we processed is not the same as PennyLanes. Order of samples can have some influence on the training behavior and results of the model. We choose to ignore this as the Iris dataset is not truly focal point of this work and once finding the preprocessing steps we move on to its application to other datasets. The second is that the output classes (originally the flower name in the genuine dataset), in the Scikit Learn dataset were converted from (0, 1) to (-1, 1). This involved removing the third class as we mentioned above the problem at hand is binary classification, therefore only two classes can exist. The values (-1, 1) are specific to quantum machine learning the polarity of the values is better suited for the Hilbert space than (0, 1) which is traditionally used in classical techniques.

One change we made is to rotate the datasets by -90 degrees about the origin. This change was done with the intention of making the dataset separable with a line having slope one. This intentionally is done to gauge whether the classifier can fit to the simplified decision boundary better. We assumed this would greatly simplify the fitting function and lead to faster converging of the classifier. The question we ask here is why is it better at splitting the data this way versus the same amount of space between the two classes rotated differently? The results are as we expected, the classifier achieves a significant improvement for each case, even for Blobs-2F. Although none of the datasets are classified perfectly, we achieve roughly the same results for both Blobs-4F and Blobs-3F when using the additional two layers. The resulting decision boundaries for both datasets are also much more linear as we expected.

Interestingly, this is the single best change we make for Blobs-2F dataset. The change in coordinate space does improves the learning ability. Consider the decision boundaries for the baseline, additional two layers, and here with a rotation about the origin by -90 degrees. We can see that even with additional layers the classifier tries to produce a vertical line even though it classifies roughly fifty percent of the samples incorrectly. The result when rotating is a again a vertical line but is much more accurate after the rotation. In the case of the former two, similar results should have been achieved by some horizontal line. This rotation begs the question of other rotations, which improve, and which further hinder the classifier? Before answering that question, we add here that including the coefficients to the features for rotation appeared to show no additional outstanding improvement with rotations. Therefore, we exclude it in the analysis of rotations as we continue. The equation for the rotation is show in Equation 26 where x_1 and y_1 are the two feature values, θ is the angle to rotate by, and x_0 and y_0 are the coordinates of the origin (0,0).

$$r_{cartesianx} = \cos\theta * (x_1 - x_0) - \sin\theta * (\mathrm{y}_1 - \mathrm{y}_0)$$
$$(26)$$
$$r_{cartesiany} = \sin\theta * (x_1 - x_0) + \cos\theta * (y_1 - y_0)$$

We hypothesized that some rotations would lead to very poor performance while others increased it. With this in mind we applied the rotations from -90 to +90 by three in total training 61 times. This was done to evaluation if there is a periodic pattern related to the rotations performed on the datasets. In Figure 27. we have plotted the AUC score of each training outcome. The figure below shows that roughly every fifty degrees we end up with repeated behavior. These results are for Blobs-4F and best results appear to occur every 60 degrees. Although we apply the rotation to just two features a rotation matrix can be created for n-dimensional datasets applied in a similar way.

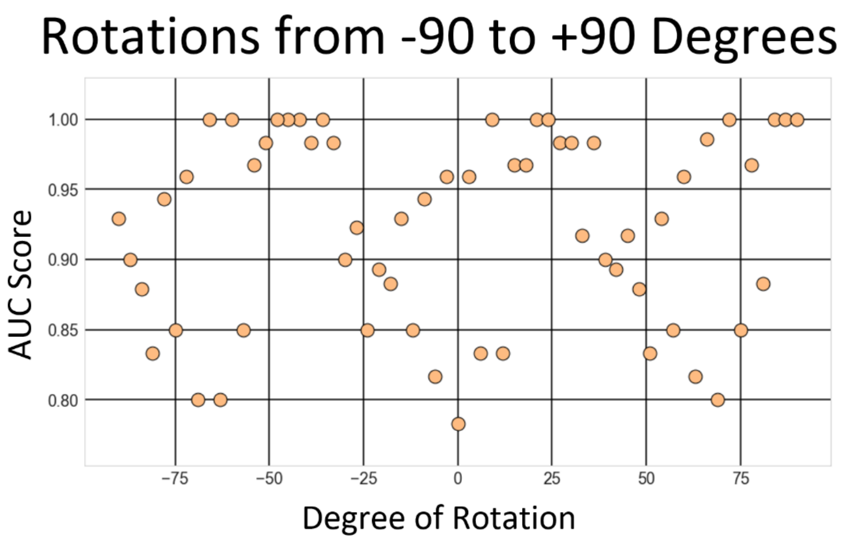


Figure 30. AUC scores of 61 models trained based on rotations of the dataset. Rotations are done every three degrees. Obvious periodic behavior can be seen from the results roughly every fifty degrees.

5.4.2 Additional Analysis Leading to Future Work

We have covered all of the work developed in this master thesis. Over the experimentation process we have been left with several partially developed insights. We include this section as it is not removed from the work we present here, albeit less analytical and more descriptive. We have developed several datasets but only discussed MakeBlobs and the transformations applied to Iris. We continue here with shallow exploration into the other datasets. We will begin with the Wine dataset and continue with the other generator datasets.

The Wine dataset possess several interesting challenges in order train it using the quantum methods we have described. The primary problem being which features are best for a quantum device or more generally which features are separable enough to learn from. In terms of binary classification this problem is not difficult, and it is trivial to show that the first and third classes are dissimilar in many of the features. This can be seen in Figure 18. We have trained on several of these features and applied the basic steps for state preparation and from the Iris dataset. Several of the training samples performed well but for the features selected these results were expected. We then selected “less” separable features and the training of these data performed poorly. We expect these can be improved either by applying rotations or different scaling/normalization methods. Further analysis here would need to be performed on each of the subsets utilized for with/without rotations of varying degrees.

We also tested different factors applied to the data after the initial normalization. These factors were applied to two datasets, Wine and MakeSwissRolls. The factors as we have shown separate the data more than normalization initially intends. Although there is little evidence for this application on other datasets, we considered it in our analysis. We note that applying this factor performed well in many cases, including MakeBlobs, when the factor is a multiple of the computed factor. Taking that into account, random factors appear to cause poor learning. Simply multiplying one feature by ten and the other by five cause decreased performance for Iris, Wine, MakeBlobs, and MakeSwissRolls. Additional analysis should be applied to determine if factors correspond to a distribution of values and testing of interval values should be evaluated.

We also applied Stokes parameters after padding and then removing the third feature from the analysis. Stokes parameters have been leveraged in QML methodologies and have shown applicable results for preprocessing. In terms of effectiveness we applied it to MakeBlobs, MakeCircles, and MakeMoons. For MakeBlobs the application of Stokes parameters was a top performer for Blobs-4F but performed worse than the baseline in all other cases. For MakeCircles Stokes parameters worked only when the data was highly separated which provided little insight. For MakeCircles the results were poor but marginally worse than the baseline model.

In the majority we have developed and tested our methods on two dimensional samples, but we have additionally expanded to three, four, and five dimensional datasets. The conclusion of the experimentation with larger higher dimensional samples is that the simplicity of the VQC model impedes anything other than a shallow level of optimization across all of the tested datasets. We tried to apply the VQC to for more than two dimensions of data but basic hyperparameter tuning and several different preprocessing steps, some of which we have not discussed, appeared to have no success. This conclusion although useless in terms of quantification shows that the VQC model is not truly suited for complex feature spaces. More generally, there are several VQC models in existence and other parametric quantum classifiers such as QSVM and QAOA that are able to hand a spectrum of problems. We additionally note that the model as presented in the documentation of PennyLane is misleading. To an extent, the VQC applied to Iris is deceiving and has little computational applicability outside of the discussion we have developed. The root of this stems from the preprocessing applied to the Iris dataset. There is little doubt to the authors that the data was prepared in the way it was for the applied method.

The last transformation we discuss was the applying a transformation typically performed after Stokes parameters but in this case we did so intentionally without doing so. The plots of this data are shown in Figure 31. for two of the datasets, although it was applied to each Generator dataset. This method of preprocessing we will call "Poincare transformation" because it plots the data to the Poincare sphere without applying Stokes parameters first. The two datasets in Figure 31. originally have two features only. For each generator with only two features we padded a third feature in two different ways: by a constant or set of constants and by some distribution of values. We varied the distributions and constants to find interesting samples that may or may not perform well in a quantum model. The datasets in Figure 31are among the most promising of these generated datasets. Among these promising datasets are also MakeGaussianQuantiles, MakeSCurve, and MakeSwissRolls (for certain class pairs). Two of these, MakeGaussianQuantiles and MakeSCurve we have not developed as they were only tested in this Poincare transformation unlike the other datasets.

We also applied different normalization methods to some of the Poincare data. The images in Figure 31 use L-1, L-2, and L-Maximum normalization. We tested these different normalizations on Iris, MakeBlobs, MakeMoons, and MakeSwissRoll. Of these, Iris was the only dataset that classified with an accuracy above 90%. The rest of the datasets performed better when applying MinMaxScalar to the data.

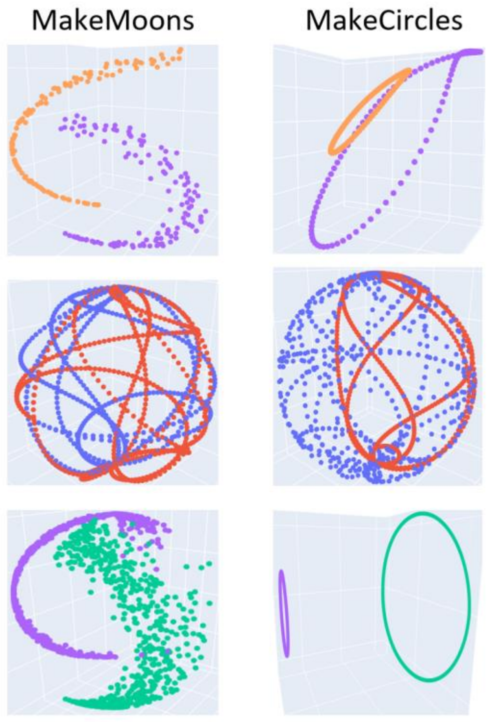


Figure 31. Examples of manipulated datasets in 3D. Left are three transformations applied to the MakeMoons dataset and right are three similar transformations to the MakeCircles dataset.

6. Conclusion

We have worked through and implemented components of preprocessing and QML models. In the first half of our experimentation we have applied and tested the method of state preparation known as amplitude encoding. We achieved benchmark like results for two applications of state preparation and in doing so have essentially removed most doubt about other state preparation methods. These experiments were performed using the TensorFlow Quantum framework. The second half of this work has tried to explain several steps that make the training of quantum models more robust. These methods in part were performed as an analysis to understand what needs to be considered in order to prepare methods in the future for QML. We have applied our methods to several synthetic datasets in this portion of the work using PennyLane's framework. We have developed essentially only binary methods in this work as much of the literature in this field boasts results of the same type. Future work in this area will require expansions into multiclass classification techniques which are developed enough to be applicable in works such as this one.

6.1 Future Discussion

Although NISQ era devices are faulty and have a long way to go they are appropriate for testing datasets with small number features. The utilization of a NISQ device on the transformations and analysis applied in this thesis would be the immediate next step. There are many issues with using modern NISQ era devices from noise and decoherence to allotted time and space constraints. These things among others are what discouraged their use in place of the more widely available simulated devices. Quantum simulators have come a long way and have provided viable and we believe acceptable results to move forward using a noisy implementation of a quantum computer. Due to the capabilities of quantum devices today, we expect the results to be generally worse. The hope is that similar results or patterns with the learning outcomes appear. We can say for certain that the methods applied here are not guaranteed to behave the same but with anticipation expect large similarities.

Appendix X

Link to code for reproducing many of the components in this thesis:
https://github.com/m0tela01/

The code is open source and has no copyright, so you are free to use it in any capacity. While cleaning many of the irrelevant notebooks and scripts some of the examples or components may have been lost. I have tried to comment code as much as it makes sense. Many of the notebooks use the submodule created in the folder /MEngCode.

References

[1] Britt, Keith A., and Travis S. Humble. “High-performance computing with quantum processing units.” *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13, no. 3 (2017): 1-13.

[2] Preskill, John. "Quantum Computing in the NISQ era and beyond." Quantum 2 (2018): 79.

[3] Orus, Roman, Samuel Mugel, and Enrique Lizaso. "Quantum computing for finance: overview and prospects." Reviews in Physics 4 (2019): 100028.

[4] Cao, Yudong, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan et al. "Quantum chemistry in the age of quantum computing." Chemical reviews 119, no. 19 (2019): 10856-10915.

[5] Cory, David G., Amr F. Fahmy, and Timothy F. Havel. "Ensemble quantum computing by NMR spectroscopy." Proceedings of the National Academy of Sciences 94, no. 5 (1997): 1634-1639.

[6] Zeng, William, and Bob Coecke. "Quantum algorithms for compositional natural language processing." arXiv preprint arXiv:1608.01406 (2016).

[7] Patra, Bishnu, Rosario M. Incandela, Jeroen PG Van Dijk, Harald AR Homulle, Lin Song, Mina Shahmohammadi, Robert Bogdan Staszewski et al. "Cryo-CMOS circuits and systems for quantum computing applications." IEEE Journal of Solid-State Circuits 53, no. 1 (2017): 309-321.

[8] Cross, Andrew. "The IBM Q experience and QISKit open-source quantum computing software." APS 2018 (2018): L58-003.

[9] Broughton, Michael, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, Jae Hyeon Yoo, Sergei V. Isakov, Philip Massey et al. "Tensorflow quantum: A software framework for quantum machine learning." arXiv preprint arXiv:2003.02989 (2020).

[10] Bergholm, Ville, Josh Izaac, Maria Schuld, Christian Gogolin, M. Sohaib Alam, Shahnawaz Ahmed, Juan Miguel Arrazola et al. "Pennylane: Automatic differentiation of hybrid quantum-classical computations." arXiv preprint arXiv:1811.04968 (2018).

[11] Berthiaume, André, and Gilles Brassard. "Oracle quantum computing." Journal of modern optics 41, no. 12 (1994): 2521-2535.

[12] Boneh, Dan, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. "Random oracles in a quantum world." In International Conference on the Theory and Application of Cryptology and Information Security, pp. 41-69. Springer, Berlin, Heidelberg, 2011.

[13] Amazon. 2019. Quantum computing | Amazon Bracket (AWS). 2019 https://aws.amazon.com/de/braket/

[14] Quantum Development Kit|Microsoft. Microsoft quantum—US (English). https://www.microsoft.com/en-us/quantum/development-kit. Accessed 22 Nov 2020

[15] Gaebler, John, Bryce Bjork, Dan Stack, Matthew Swallows, Maya Fabrikant, Adam Reed, Ben Spaun, Juan Pino, Joan Dreiling, and Caroline Figgatt. "Progress toward scalable quantum computing at Honeywell Quantum Solutions." APS 2019 (2019): S01-103.

[16] Arute, Frank, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas et al. "Quantum supremacy using a programmable superconducting processor." Nature 574, no. 7779 (2019): 505-510.

[17] Sierra-Sosa, Daniel, Telahun, Michael, and Elmaghraby, Adel S. "TensorFlow Quantum: Impacts of Quantum State Preparation on Quantum Machine Learning Performance" IEEE Access, doi: 10.1109/ACCESS.2020.3040798.

[18] Kantardzic, Mehmed. Data mining: concepts, models, methods, and algorithms. John Wiley & Sons, 2011.

[19] Chawla, Nitesh V., Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research 16 (2002): 321-357.

[20] Kubat, Miroslav, and Stan Matwin. "Addressing the curse of imbalanced training sets: one-sided selection." In Icml, vol. 97, pp. 179-186. 1997.

[21] Asghari, Mohsen, Daniel Sierra-Sosa, Michael Telahun, Anup Kumar, and Adel S. Elmaghraby. "Aggregate density-based concept drift identification for dynamic sensor data models." Neural Computing and Applications (2020): 1-13.

[22] Xu, Rui, and Donald Wunsch. "Survey of clustering algorithms." IEEE Transactions on neural networks 16, no. 3 (2005): 645-678.

[23] Caruana, Rich, and Alexandru Niculescu-Mizil. "An empirical comparison of supervised learning algorithms." In Proceedings of the 23rd international conference on Machine learning, pp. 161-168. 2006.

[24] Zhu, Xiaojin Jerry. Semi-supervised learning literature survey. University of Wisconsin-Madison Department of Computer Sciences, 2005.

[25] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20, no. 3 (1995): 273-297.

[26] Rebentrost, Patrick, Masoud Mohseni, and Seth Lloyd. "Quantum support vector machine for big data classification." Physical review letters 113, no. 13 (2014): 130503.

[27] MacQueen, James. "Some methods for classification and analysis of multivariate observations." In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol. 1, no. 14, pp. 281-297. 1967.

[28] Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. Deep learning. Vol. 1, no. 2. Cambridge: MIT press, 2016.

[29] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521, no. 7553 (2015): 436-444.

[30] Cybenko, George. "Approximation by superpositions of a sigmoidal function." Mathematics of control, signals and systems 2, no. 4 (1989): 303-314.

[31] MacKay, David JC. "Hyperparameters: optimize, or integrate out?." In Maximum entropy and bayesian methods, pp. 43-59. Springer, Dordrecht, 1996.

[32] Robbins, Herbert, and Sutton Monro. "A stochastic approximation method." The annals of mathematical statistics (1951): 400-407.

[33] Ge, Rong, Furong Huang, Chi Jin, and Yang Yuan. "Escaping from saddle points—online stochastic gradient for tensor decomposition." In Conference on Learning Theory, pp. 797-842. 2015.

[34] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." In ICML. 2010.

[35] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15, no. 1 (2014): 1929-1958.

[36] Nielsen, Michael A., and Isaac Chuang. "Quantum computation and quantum information." (2002): 558-559.

[37] Benioff, Paul. "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines." Journal of statistical physics 22, no. 5 (1980): 563-591.

[38] Deutsch, David, and Richard Jozsa. "Rapid solution of problems by quantum computation." Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences 439, no. 1907 (1992): 553-558.

[39] Shor, Peter W. "Algorithms for quantum computation: discrete logarithms and factoring." In Proceedings 35th annual symposium on foundations of computer science, pp. 124-134. Ieee, 1994.

[40] Preskill, John. "Reliable quantum computers." Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences 454, no. 1969 (1998): 385-410.

[41] Montanaro, Ashley. "Quantum algorithms: an overview." npj Quantum Information 2, no. 1 (2016): 1-8.

[42] Wold, Svante, Kim Esbensen, and Paul Geladi. "Principal component analysis." Chemometrics and intelligent laboratory systems 2, no. 1-3 (1987): 37-52.

[43] Lloyd, Seth, Masoud Mohseni, and Patrick Rebentrost. "Quantum principal component analysis." Nature Physics 10, no. 9 (2014): 631-633.

[44] Xiao, Jing, YuPing Yan, Jun Zhang, and Yong Tang. "A quantum-inspired genetic algorithm for k-means clustering." Expert Systems with Applications 37, no. 7 (2010): 4966-4973.

[45] Rebentrost, Patrick, Masoud Mohseni, and Seth Lloyd. "Quantum support vector machine for big data classification." Physical review letters 113, no. 13 (2014): 130503.

[46] Schuld, Maria. Supervised learning with quantum computers. Springer, 2018.

[47] Knill, Emanuel, Raymond Laflamme, Rudy Martinez, and Camille Negrevergne. "Benchmarking quantum computers: the five-qubit error correcting code." Physical Review Letters 86, no. 25 (2001): 5811.

[48] Makhlin, Yuriy, Gerd Scöhn, and Alexander Shnirman. "Josephson-junction qubits with controlled couplings." nature 398, no. 6725 (1999): 305-307.

[49] Cross, Andrew W., Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta. "Validating quantum computers using randomized model circuits." Physical Review A 100, no. 3 (2019): 032328.

[50] IBM Quantum Experience|IBM. Quantum on the Cloud—US (English). https://www.ibm.com/quantum-computing/experience/. Accessed 29 Nov 2020

[51] Havlíček, Vojtěch, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. "Supervised learning with quantum-enhanced feature spaces." Nature 567, no. 7747 (2019): 209-212.

[52] Lloyd, Seth, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. "Quantum embeddings for machine learning." arXiv preprint arXiv:2001.03622 (2020).

[53] Rebentrost, Patrick, Maria Schuld, Leonard Wossnig, Francesco Petruccione, and Seth Lloyd. "Quantum gradient descent and Newton's method for constrained polynomial optimization." New Journal of Physics 21, no. 7 (2019): 073023.

[54] Killoran, Nathan, Thomas R. Bromley, Juan Miguel Arrazola, Maria Schuld, Nicolás Quesada, and Seth Lloyd. "Continuous-variable quantum neural networks." Physical Review Research 1, no. 3 (2019): 033063.

[55] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondˇrejˇ Certík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E.Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, ShivamVats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R.Terrel, Štˇepán Rouˇcka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing inpython. PeerJ Computer Science, 3:e103, January 2017.

[56] A. Ho and D. Bacon. Announcing cirq: An open source framework for nisq algorithms. Google AI Blog.

[57] Schuld, Maria, Alex Bocharov, Krysta M. Svore, and Nathan Wiebe. "Circuit-centric quantum classifiers." Physical Review A 101, no. 3 (2020): 032308.

[58] Hunter, John D. "Matplotlib: A 2D graphics environment." Computing in science & engineering 9, no. 3 (2007): 90-95.

[59] Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

[60] Araujo, Israel F., Daniel K. Park, Francesco Petruccione, and Adenilton J. da Silva. "A divide-and-conquer algorithm for quantum state preparation." arXiv preprint arXiv:2008.01511 (2020).