Supervised Quantum Machine Learning

Linus Ekstrøm

Fall Semester 2021

Contents

Introduction

This document will contain my thoughts and notes relevant for the course work I will be doing as a specialized curriculum in the fall semester 2021. The subject curriculum will be Unsupervised Learning with Quantum Computers, Maria Schuld and Francesco Petruccione [1]. The evaluation in the subject will consist of two turn-ins. Below is a table of the chapter topics in the aforementioned book.

Introduction: **1.1** Background; merging two disciplines. The rise of quantum machine learning. Four approaches. Quantum computing for supervised learning. **1.2** How quantum computers can classify data; the squared-distance classifier. Interference with the hadamard transformation. Quantum squared-distance classifier. Insights from the toy example.
Machine Learning: **2.1** Prediction; four example for prediction tasks. Supervised Learning. **2.2** Models; how data leads to a predictive model. Estimating the quality of a model. Bayesian learning. Kernels and feature maps. **2.3** Training; cost functions. Stochastic gradient descent. **2.4** Methods in machine learning; data fitting. Artificial neural networks. Graphical models. Kernel Models.
Quantum Information: **3.1** Introduction to quantum theory; What is quantum theory. A first taste. The postulates of quantum mechanics. **3.2** Introduction to quantum computing; what is quantum computing. Bits and qubits. Quantum Gates. Quantum parallelism and function evaluation. **3.3** An example: The Deutsch-Josza algorithm; The Deutsch algorithm. The Deutsch-Jozsa algorithm. Quantum Annealing and other computational methods. **3.4** Strategies of information encoding; basis encoding. Amplitude encoding. Qsample encoding. Dynamic encoding. **3.5** Important quantum routines; grover search. Quantum phase estimation. Matrix multiplication and inversion.
Quantum Advantages: **4.1** Computational complexity of learning. **4.2** Sample complexity; exact learning from membership queries. PAC learning from examples. Introducing noise. **4.3** Model complexity
Information Encoding: **5.1** Basis encoding; preparing superposition of inputs. Computing in basis encoding. Sampling from a qubit. **5.2** Amplitude encoding; state preparation in linear time. Qubit-efficient state preparation. Computing with amplitudes. **5.3** Qsample encoding; joining distributions. Marginalisation. Rejection sampling. **5.4** Hamiltonian encoding; polynomial time hamiltonian simulation. Qubit-efficient simulation of Hamiltonians. Density matrix exponentiation.
Quantum Computing for Inference: **6.1** Linear models; inner products with intererence circuits. A quantum circuit as a linear model. Linear models in basic encoding. Nonlinear activations. **6.2** Kernel methods; kernel and feature maps. The representer theorem. Quantum kernels. Distance based classifiers. Density gram matrices. **6.3** Probabalistic models; qsamples as probabalistic models. Qsamples with conditional independence relations. Qsamples of mean-field approximations.
Quantum Computing for Training: **7.1** Quantum blas; basic idea. Matrix inversion for training. Speedups and further applications. **7.2** Search and amplitude amplification; finding closest neighbours. Adapting Grover's search to data superpositions. Amplitude amplification for perceptron training. **7.3** Hybrid training for variational algorithms; variational algorithms. Variational quantum machine learning algorithms. Numerical optimization methods. Analytical gradients of a variational classifier. **7.4** Quantum adiabatic machine learning; quadratic unconstrained optimisation. Annealing devices as samples. Beyond annealing.
Learning with Quantum Models: **8.1** Quantum extensions of Ising-type models; the quantum Ising model. Training quantum Boltzmann machines. Quantum Hopfield models. Other probabilistic models. **8.2** Variational classifiers and neural networks; gates as linear layers. Considering the model parameter count. Circuits with a linear number of parameters. **8.3** Other approaches to build quantum models; quantum walk models. Superposition and quantum ensembles. Qboost.
Prospects for Near-Term Quantum Machine Learning: **9.1** Small Versus Big Data. **9.2** Hybrid Versus Fully Coherent Approaches. **9.3** Qualitative Versus Quantitative Advantages. What Machine Learning Can Do for Quantum Computing

Table 1: Table of contents of [1]

1 Chapter 1 Notes

"In its broadest definition, quantum machine learning summarises approaches that use synergies between machine learning and quantum information. For example, researchers investigate how mathematical techniques from quantum theory can help to develop new methods in machine learning, or how we can use machine learning to analyse measurement data of quantum experiments. Here we will use a much more narrow definition of quantum machine learning and understand it as machine learning with quantum computers or quantum-assisted machine learning." [1] p.1

1.1 Introduction to the Topic of QML

We ask ourselves the following questions and try to answer them over the course of the book. (**Note**: *Most of the information here is directly taken from [1]*)

- Does quantum information add something new to how machines recognise patterns in data?
- Can quantum computers help to solve problems faster?
- Can they learn from fewer data samples?
- Are they able to deal with higher levels of noise?
- How can we develop new machine learning techniques from the language in which quantum computing is formulated?
- What are the ingredients of a quantum machine learning (QML) algorithm?
- What are the bottlenecks in a QML algorithm?

Next we cover four different approaches of how to combine quantum computing and machine learning, depending on whether one assumes the data to be generated by a quantum (**Q**) or classical (**C**) system, and if the information processing device is (**Q**) or (**C**). This classification was introduced by [2]

First we deal with the **CC** case. In this case we mean classical data being processed classically, as in the conventional approach to machine learning. In this classification scheme it means classical machine learning based on methods borrowed from quantum information research. An example is the application of tensor networks, which have been developed for quantum many-body-systems, to neural network training.

Second, the **QC** case investigates how machine learning can help with quantum computing. An example: when we want to get a comprehensive description of the internal state of a quantum computer from as few measurements as possible we can use machine learning to analyze the measurement data. Another idea is to learn phase transitions in many-body quantum systems.

Third, **CQ** uses quantum computing to process classical datasets (Project 2 Fys4150). The datasets consist of observations from classical systems, such as text, images or time series. This requires a quantum-classical interface, which is a challenge we discuss later in the book.

Finally, the **QQ** approach looks at *quantum data* being processed by a quantum computer. This can have two different meanings. First, the data could be derived from measuring a quantum system in a physical experiment and feeding the values back into a separate quantum processing device. A much more natural setting however arises where a quantum computer is first used to simulate the dynamics of a quantum system (as investigated in the discipline of *quantum simulation* and with fruitful applications to modeling physical and chemical properties that are otherwise computationally intractable), and consequently takes the state of the quantum system as an input to a quantum machine learning algorithm executed on the very same device. The advantage of such an approach is that while measuring all information of a quantum state may require an exponential number of measurements with increasing system size, the quantum computer has immediate access to all this information and can produce the result (yes/no) directly. An exponential speedup by design.

For the remainder of the book **CQ** and **QQ** we use the term *quantum machine learning* synonymously with these. The **QQ** approach is very interesting, however there are very few results in this direction and a number of questions left to be answered.

1.2 Designing QML algorithms

From here on out we will focus on the **CQ** case for supervised learning problems. There are two chief strategies used by most of the QML researchers when designing QML algorithms. The first strategy aims at *translating* classical models into the language of quantum mechanics in the hope to harvest algorithmic speedups. The sole goal is to reproduce the results of a given model (*neural network or Gaussian process*). The second strategy, whose many potentialities are still largely unexplored, is an *exploratory* approach. Instead of translating a classical algorithm, one starts with a quantum computer and asks what type of machine learning model might fit its physical characteristics. The former approach is more akin to applied quantum computing, while the latter is a true multidisciplinary endeavor. Both these strategies will be looked into in this book.

2 Chapter 2 Notes: Machine Learning

"Machine learning originally emerged as a sub-discipline of artificial intelligence research where it extended areas such as computer perception, communication and reasoning. For humans, learning means finding patterns in previous experience which help us to deal with an unknown situation."

2.1 Four Main Concepts

This chapter is an attempt to give a quantum physicist access to major ideas in the vast landscape of machine learning research. There are four main concepts we want to introduce

- The task of supervised learning for **prediction** by generalising from labelled datasets
- How to do inference with machine learning **models**
- How to **train** models through optimization
- How well-established **methods** of machine learning combine specific models with training strategies.

2.1.1 Prediction

Almost all machine learning algorithms have one thing in common: they are fed with data and produce an answer to a question. The term *answer* can mean many different things. When future values of a time series are predicted we call it a *forecast*, when the prediction is images it is a *classification*, in case of medical predictions it is a *diagnosis*.

2.1.2 Models

The term *model* and its role in machine learning is comparable with the term of a *state* in quantum mechanics. Meaning it is a central concept with a clear definition for those who use it, but it takes practice to grasp all dimensions of it. Mathematically speaking, a map from inputs to outputs, $\mathbb{R}^N \to$ or $\{0,1\}^N \to \{0,1\}$ is a function. From a computational perspective, mapping inputs to outputs is done via an algorithm. On a more abstract level, a model specifies the rule or hypothesis that leads from input to output. One could therefore define models in supervised machine learning as *functions*, *algorithms*, or *rules that define a relationship between input data and predictions*. It is also important to distinguish between *deterministic* and *probabilistic* models.

A *deterministic model*: Let $\mathcal{X}$ be an input domain and $\mathcal{Y}$ be an output domain for a supervised learning problem. A *deterministic model* is a function

$$y = f(x;\theta) \tag{1}$$

with $x \in \mathcal{X}, y \in \mathcal{Y}$, and a set of real parameters $\theta = \{\theta_1, \cdots, \theta_D\}$. We also call f the *model function*. For general parameters Eq. (1) defines a *model family*. (**Note**: when we need the parameters in vector form it is common to use the notation $w = (w_1, \cdots, w_D)^T \in \mathbb{R}^D$ and refer to it as the *weight vector*)

Probabilistic models understand the data inputs and outputs as random variables drawn from an underlying probability distribution that approximates the ground truth. From there one can compute the probability of a certain label given an input $p(y|x)$ (a procedure called *marginalisation* which is very similar to tracing out part of density matrices in quantum theory) and translate this to a prediction. Let $\mathcal{X}$ be an input domain and $\mathcal{Y}$ be an output domain for a supervised learning problem. Let X, Y be random variables from which we can draw samples

$x \in \mathcal{X}, y \in \mathcal{Y}$, and let θ be a set of real parameters. A *probabilistic model* refers to either the *generative model distribution*

$$p(x, y; \theta)$$

or the *discriminative model distribution*

$$p(y|x; \theta)$$

over the data.

2.1.3 Training

The goal of training is to select the best model for prediction from a model family. This means we have to define an objective function that quantifies the quality of a model given a set of parameters, and training becomes an optimization problem over the objective. This objective function is called the *cost function* $C(\theta)$. The most important building blocks of a successful cost function are the *loss* $L(\mathcal{D}, \theta)$ and the *regularizer* $R(\theta)$. Regularization has been found to help against overfitting. There are lots of loss functions and regularizers, some are better suited for some tasks than others.

2.1.4 Methods

Here we briefly summarize the concept of *methods* in machine learning. A method in machine learning specifies a general ansatz for a *model function or distribution* that can be used for prediction and a *training strategy* of how to use the data to construct a specific model that generalizes from the data. In supervised learning, the training objective is to minimize the error between target outputs and predicted outputs for the training set. However, this objective serves the overarching purpose of minimizing the generalization error, which is usually measured on a test or validation set.

2.2 How Data Leads to a Predictive Model

Given some preprocessed and carefully featurized data, a generic model family has to be chosen. The model is trained by fitting the parameters and hyperparameters to the data. This means that a specific model function or distribution is chosen from the family. The training step can be of a very different nature for different approaches to machine learning. While sometimes it means to fit thousands of parameters after which the data can be discarded, it can also refer to finding parameters that weigh the data points to compute a prediction. A few models - *so called lazy learners* - derive the prediction directly from the data without generalizing from it first. Once a specific trained model is derived from the data it can be used for prediction or *inference*.

2.3 Bayesian Learning

An important alternate approach with a fundamentally different logic to other methods is *Bayesian learning*. This method looks at learning from a probabilistic perspective. Not surprisingly, this perspective considers probabilistic models and translates learning into the mathematical or computational problem of integration rather than optimization. Bayesian learning is based on Bayes rule

$$p(a|b) = \frac{p(b|a)p(a)}{p(b)}$$

where a, b are values of random variables A, B and $p(a|b)$ is the conditional probability of a given b defined by

$$p(a|b) = \frac{p(a, b)}{p(b)}$$

This rule comes with a specific terminology that has influenced probabilistic machine learning significantly. The term $p(b|a)$ is called the **likelihood**, $p(a)$ is the **prior** and $p(a|b)$ the **posterior**.

How can we use this rule in the context of learning? Given a training dataset $\mathcal{D}$ as before and understanding inputs x and outputs y as random variables, we want to find the probabilistic model $p(x, y|\mathcal{D})$ which is likely to produce the data. Formally we write

$$p(x, y|\mathcal{D}) = \int d\theta p(x, y|\mathcal{D})p(\theta|\mathcal{D}) \tag{2}$$

The first part of the integrand is the parametrized model distribution that we chose, but written as a conditional probability. The second part of the integrand is the probability that a certain set of parameters is the *right one* given the data. In a sense, this is exactly what we want to achieve by training, namely to find an optimal θ given the data.

In order to compute the unknown term $p(\theta|\mathcal{D})$ we use Bayes formula

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int d\theta p(\mathcal{D}|\theta)p(\theta)}$$

The prior describes our previous assumption as to which parameters lead to the best model *before* seeing the data. The prior is a very special characteristic of Bayesian learning which allows us to make an educated guess at the parameters without consulting the data, and under certain circumstances it can be shown to have a close relation to regularizers in an objective function. If nothing is known, the prior can be chosen to be uniform over a certain interval.

The likelihood $p(\mathcal{D}|\theta)$ is the probability of parameters θ *after* seeing the data. The Bayesian learning approach shifts the problem from finding the probability of parameters given some data to finding how likely it is to observe our dataset if we assume some parameters. This second question can be answered by consulting the model. Under the assumption that the data samples are drawn independently, one can factorize the distribution

$$p(\mathcal{D}|\theta) = \Pi_m p(x^m, y^m|\theta)$$

Since the expression on the right can be computed, we can in theory integrate over Eq. (2) to find the answer. However, for high-dimensional parameters the exact solution quickly becomes computationally intracable, and may even become difficult to approximate via sampling. It is therefore common to approach the problem with optimization which leads to the famous *maximum likelihood problem*, and considering the prior in a truly Bayesian fashion leads to the closely related **maximum a posteriori estimation**.

In conclusion, it seems that optimization and integration are two sides of the same coin, and both are hard problems to solve when no additional structure is given.

2.4 Kernels and Feature Maps

When the *similarity measure* of a model can be described as a so called **kernel**, an entire world of theory opens up which sheds a rather different light on machine learning models, and has interesting parallels to quantum computing that will be explored in further detailed later. Here we introduce the notion of *kernel methods*.

In the context of machine learning kernels are defined as follows: A (positive definite, real-valued) *kernel* is a bivariate function $\kappa : \chi \times \chi \to \mathbb{R}$ such that for any set $\chi_f = \{x^1, \cdots, x^M\} \subset \chi$ the matrix K with entries

$$K_{m,m'} = \kappa(x^m, x^{m'}), x^m, x^{m'} \in \chi_f$$

called *kernel* or *Gram matrix*, is positive semidefinite. As a consequence, $\kappa(x^m, x^{m'}) \geq 0$. A few popular kernel functions are

Name	Kernel	Hyperparameters
Linear	$x^T x'$	-
Polynomial	$(x^T x' + c)^p$	$p \in \mathbb{N}, c \in \mathbb{R}$
Gaussian	$e^{-\gamma\|\|x-x'\|\|^2}$	$\gamma \in \mathbb{R}^+$
Exponential	$e^{-\gamma\|\|x-x'\|\|}$	$\gamma \in \mathbb{R}^+$
Sigmoid	$\tanh\big(x^T x' + c\big)$	$c \in \mathbb{R}$

Table 2: Common kernels

Under certain - *but as it turns out fairly general* - conditions formalized in the so called **representer theorem** a model $f(x;\theta)$ that does not depend on the data can be rewritten in terms of kernel functions.

2.4.1 An Example: Linear kernelized model

As an illustration, consider a linear model with weights $w = (w_0, w_1, \cdots, w_N)^T$ and input $x = (1, x_1, \cdots, x_N)^T$

$$f(x; \theta = w) = w^T x$$

If we fit this model to data $\{(x^m, y^m)\}, m = 1, \cdots, M$ using a common cost function, the *representer theorem* guarantees that the optimal weight vector can be written as a linear combination of the data points

$$w_{\text{opt}} = \frac{1}{M} \sum_{m=1}^{M} \alpha_m x^m$$

with real coefficients α_m. If we insert this weight back into the linear model, we get

$$f(x; \theta = \{\alpha_m\}, \mathcal{D}) = \frac{1}{M} \sum_{m=1}^{M} \alpha_m (x^m)^T x = \frac{1}{M} \sum_{m=1}^{M} \alpha_m \kappa(x^m, x)$$

The inner product $(x^m)^T x$ is a linear kernel. We have translated the linear model with $N+1$ parameters $w_0, \cdots, w_N$ into a 'kernelized' model with M parameters $\alpha_1, \cdots, \alpha_M$. This formulation of a model is also called the *dual form*. The objective of training is now to fit the new parameters which weigh the training data points.

2.4.2 The Kernel Trick

Expressing a model in terms of a kernel function that measures distances to data points allows us to use the so called **kernel trick**:

Given an algorithm which is formulated in terms of a positive definite kernel κ, one can construct an alternative algorithm by replacing κ by another positive definite kernel κ'.

We can therefore build new models by simply exchanging one kernel with another. It makes intuitive sense that the distance to data points is a useful way to compute a prediction for a new data point. However, there is more to kernel functions than just distance measures. A kernel can be interpreted as an inner product of data that has been transformed by a *feature map* $\phi : \chi \to \mathcal{F}$

$$k(x, x') = \langle \phi(x) | \phi(x') \rangle$$

The *feature space* $\mathcal{F}$ is a Hilbert space, a complete vector space with an inner product, and ϕ a - *usually nonlinear* - transformation that projects into $\mathcal{F}$. This is interesting because the data can be easier to classify in higher dimensional spaces, for example when it becomes linearly separable.

2.4.3 Using Higher Dimensions to Separate Data

We start with an example using the XOR function. The full 'dataset' of the XOR function is given by

$$\mathcal{D} = \{((-1,-1)^T, 1), ((-1,1)^T, -1), ((1,-1)^T, -1), ((1,1)^T, 1)\}$$

and this is not separable. A feature map of the form $\phi((x_1, x_2)^T) = (x_1, x_2, x_1 x_2)^T$ allows for a hyperplane cutting through the 3-dimensional space to separate both classes. Basically we're adding a third dimension and separating the corners of the square defined by the dataset by moving in this last dimension.

In summary, many models can be expressed in terms of kernel expansions, which makes them data-dependent. This shifts the optimization problem to a different set of parameters, namely the weights of the kernel functions. By replacing one kernel function κ by another kernel function κ', we effectively change the feature map and can potentially go into very high-dimensional spaces of many different shapes, and hope that our data gets easier to classify in that space. That way simple linear models can get the power of nonlinear classifiers. Of course, designing a good model translates into finding a good feature space, which in turn means to find a good kernel.

2.5 Cost Functions and Optimization

The *cost function* $C(\theta)$ is the function that quantifies the objective according to which we train the parameters of a model. It usually consists of two building blocks: the **loss function** and the **regularizer**. Regularization has been found to help against overfitting.

2.5.1 Loss Functions

Let us have a look at some popular loss functions and regularizers. The most common loss function is the *squared loss*

$$L(\theta) = \frac{1}{2}\sum_{m=1}^{M}(f(x^m;\theta) - y^m)^2$$

The resulting optimization problem is the *least squares optimization*. The squared loss is also called *l2 loss*, and replacing the squared loss with the absolute value $|f(x^m;\theta) - y^m|$ gives rise to the *l1 loss*.

Another way to quantify the distance between outputs and targets is via the expression $y^m f(x^m;\theta)$, which is positive when the two numbers have the same sign and negative if they are different. One can use this to compute the *hinge loss*

$$L(\theta) = \sum_{m=1}^{M}\max(0, 1 - y^m f(x^m;\theta)$$

the *logistic loss*

$$L(\theta) = \sum_{m=1}^{M}\log\Big(1 + e^{-y^m f(x^m;\theta)}\Big)$$

or, if the output of the model is a probability $f(x^m;\theta) = p^m$, the *cross entropy loss*

$$L(\theta) = \sum_{m=1}^{M} -y^m\log(p^m) - (1 - y^m)\log(1 - p^m)$$

For multi-label classification with one-hot encoding, the *cross entropy* can also be used to compare the multi-dimensional target $y = (y_1, \cdots, y_D)^T, y \in \{0, 1\}$ with the output probability distribution over predictions $f(x;\theta) = (p_1, \cdots, p_D)^T$

$$L(\theta) = -\sum_{m=1}^{M}\sum_{d=1}^{D} y_d^m\log(p_d^m)$$

Estimating the parameters of a probabilistic model given a dataset is typically done through *maximum likelihood estimation*. A concept with a long-standing tradition in statistics. In fact, the square loss can be shown to be the maximum likelihood solution under the assumption of Gaussian noise. The underlying idea is to find parameters θ such that there is a high probability that the data has been drawn from the distribution $p(x, y;\theta)$. For this, we want to maximize the probability $p(\mathcal{D}|\theta)$. It is standard practice to take the logarithm of their likelihood and maximize the *log likelihood*. If the data is independently and identically distributed we have

$$\log(p(\mathcal{D}|\theta)) = \log\Big(\Pi_m p(x^m, y^m|\theta)\Big) = \sum_m \log\big(p(x^m, y^m|\theta)\big)$$

It can be shown that $p(x^m, y^m|\theta) \propto p(x^m, y^m;\theta)$ and since constant factors have no influence on the optimization problem, the maximum log-likelihood estimation is to find parameters θ^* which maximize

$$\theta^* = \max_\theta \sum_{m=1}^{M}\log\big(p(x^m, y^m;\theta)\big)$$

2.5.2 Regularizers

There are two common choices for regularizers L_1, L_2 named after the norm they are based on. If we write the parameters as a vector w for the moment and denote by $|\cdot|$ the absolute value, we can define them as

$$R_{L_1}(w) = \lambda||w||_1^2 = \sum_i |w_i|$$

and

$$R_{L_2}(w) = \lambda||w||_2^2 = \sum_i w_i^2$$

Some intuition: the L_2 regularizer adds a penalty for the length of the parameter vector and therefore favors parameters with a small absolute value. It looks at the length of the weight vector. The L_1 regularizers favors sparse parameter vectors. Meaning that it looks at the value of each parameter. The hyperparameter λ regulates how much the regularizer term contributes towards the cost function.

2.6 Stochastic Gradient Descent

Once the cost function is constructed, machine learning reduces to the mathematical problem of optimizing the cost function. Mathematical optimization theory has developed an extensive framework to classify and solve optimization problems. (I know this part pretty well already don't think I need to read this specific chapter.)

Main point is that stochastic gradient descent is preferable in pretty much every way.

2.7 Summary of Different Methods in Machine Learning

As previously stated, a method in machine learning specifies a general ansatz for a *model function or distribution* that can be used for prediction and a *training strategy* of how to use the data to construct a specific model that generalizes from the data.

Method	Model Function / Distribution
Data fitting	
Linear Regression	$f(x;w) = w^T x$
Non-linear Regression	$f(x;w) = \varphi(w,x)$
Artificial Neural Networks	
Perceptron	$f(x;w) = \varphi(w^T x)$
Feed-Forward Neural Network	$f(x;W_1,W_2,\cdots) = \cdots \varphi_2(W_2\varphi_1(W_1 x))\cdots$
Recurrent Neural Network	$f(x^{(t)};W) = \varphi(W f^{(t-1)};W)$
Boltzmann Machine	$\frac{1}{Z}\sum_h e^{-E(v,h;\theta)}$
Graphical Models Bayesian Network	$p(s) = \Pi_k p(s_k\|\pi_k;\theta)$
Hidden Markow Model	$p(V,O) = \Pi_{t=1}^T p(v^{(t)}\|v^{(t-1)})\Pi_{t-1}^T p(o^{(t)}\|v^{(t)})$
Kernel Models	
Kernel Density Estimation	$p(x\|y=c) = \frac{1}{M_C}\sum_{m\|y^m=c}\kappa(x-x^m)$
K-nearest neighbor	$p(x\|y=c) = \frac{NN_c}{k}$
Support Vector Machine	$f(x) = \sum_{m=1}^M \gamma_m y^m \kappa(x,x^m)0w_0$
Gaussian Process	$p(y\|x) = \mathcal{N}[\tilde{\kappa}^T K^{-1} y; \tilde{\tilde{\kappa}} - \tilde{\kappa}^T K^{-1}\tilde{\kappa}]$

Table 3: Summary of the model functions $f(x,w)$ and distributions $p(x)$ of machine learning methods presented in this section. Description of the variables: x is model input, s,v,h,o denote different kinds of units or random variables in probabilistic models, while w,W,θ,γ_m are learnable parameters. We denote by φ a nonlinear activation function, t a time step, c a specific class and by M_c the number of training samples from that class. $\tilde{\tilde{\kappa}}$ is a scalar kernel and $\tilde{\kappa}$ is a vector of kernel functions, and K a kernel Gram matrix. p45 [1]

2.8 Data Fitting

Most physicist are familiar with statistical methods for data fitting such a *linear* and *non-linear regression.* We briefly cover both these now.

2.8.1 Linear Regression

For **linear regression**

$$f(x;w) = w^T x + w_0$$

where $w, x \in \mathbb{R}^N$. Note that the term linear refers to linearity in the model parameters only. A nonlinear feature map on the original input space can turn linear models into powerful predictors that can very well be used to model nonlinear functions. A well known example is the feature map

$$\phi : x \in \mathbb{R} \to (1, x, x^2, \cdots, x^d)^T$$

such that

$$f(\phi(x); w) = w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d \tag{3}$$

According to the Weierstrass approximation theorem, any real single-valued function that is continuous on a real interval $[a, b]$ can be arbitrarily closely approximated by a polynomial function. Thus (3) can therefore model any function in the limit.

Learning in linear regression means to find the parameters w that fit f to the training data in order to predict new data points. A very common approach is *least squares estimation*

$$L(w) = \sum_{m=1}^{M} (w^T x^m - y^m)^2 = (Xw - y)^T (Xw - y)$$

where we have in the last step introduced matrices

$$y = \begin{bmatrix} y^1 \\ \vdots \\ y^M \end{bmatrix}, \quad X = \begin{bmatrix} x_1^1 & \cdots & x_N^1 \\ \vdots & \ddots & \vdots \\ x_1^M & \cdots & x_N^M \end{bmatrix}, \quad w = \begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix}$$

We call X *data* or *design matrix*. If $X^T X$ is of full rank (*linearly independent columns*), the estimated parameter vector can be calculated by the closed-form equation

$$w = (X^T X)^{-1} X^T y$$

To show this, write $(Xw - y)^2 = (Xw - y)^T (Xw - y) = y^T y - 2w^T X^T y + w^T X^T X w$ and calculate the derivative $\partial_w L(w) \implies -2X^T y + 2X^T X w$. At the minimum, this expression is zero. The solution to the least squares optimization problem for a linear regression model is therefore

$$w = X^+ y$$

where

$$X^+ = (X^T X)^{-1} X^T \tag{4}$$

The matrix X^+ is also called a *pseudoinverse*, a generalization of the inverse for non-square or square singular matrices. In computational terms, training a linear regression model reduces to the inversion of a $M \times K$ dimensional data matrix where usually the dimension of the feature space K is larger than the dimension of the input vectors. The fastest classical algorithms take time $\mathcal{O}(K^\delta)$, where for the best current algorithms the constant δ is bounded by $2 \leq \delta \leq 3$.

2.8.2 Singular Value Decomposition Formulation

An alternate way to solve this problem is to write the pseudoinverse as a singular value decomposition. A singular value decomposition is the generalization of an eigendecomposition $A = SDS^T$, but for general (i.e. singular) matrices. Any real matrix can be written as $A = U\Sigma V^T$, where the orthogonal real matrices U, V carry *singular vectors* $u_r, v_r, r = 1, \cdots, R$ as columns, where R is the rank of A. Σ is a diagonal matrix of appropriate dimension containing the R nonzero singular values σ_r. The inverse of A is calculated by inverting the singular values on the diagonal and taking the transpose of U, V. Using the singular value decomposition to decompose X yields

$$\begin{aligned} X^+ &= (X^T X)^{-1} X^T \\ &= (V\Sigma U^T U \Sigma V^T)^{-1} V \Sigma U^T \\ &= V \Sigma^{-2} V^T V \Sigma U^T \\ &= V \Sigma^{-1} U^T \end{aligned}$$

With this above formulation of the pseudoinverse, the solution (4) can be expressed in terms of the singular vectors and values

$$w = \sum_{r=1}^{R} \frac{1}{\sigma_r} u_r^T y v_r$$

The computational problem is now to find the singular value decomposition of X.

2.8.3 Nonlinear Regression

In **Nonlinear regression** we relax the condition that the model function has only a linear dependency on the parameters. One way of deriving nonlinear regression from the linear version is by replacing the nonlinear feature map ϕ by a nonlinear function in the inputs *and* parameters.

$$f(x) = \phi(w, x)$$

Currently, the most successful nonlinear regression models in machine learning are neural networks.

2.9 Neural Networks

From a mathematical perspective, neural networks can be seen as a nonlinear regression model with a specific choice for the model function φ, namely one where the input to the nonlinear function is given by a linear model

$$\varphi(w, x) = \varphi(w^T x)$$

an expression which then gets nested several times. Historically these models were derived from biological neural networks and they have a beautiful graphical representation reminiscent of neurons that are connected by synapses. The nonlinear function originally corresponded to the *integrate-and-fire* principle found in biological neurons, which prescribes that a neuron fires once the incoming signal surpasses a certain threshold value. Neural network research was abandoned and revived a number of times during its history. Important milestones were when Hopfield showed in 1982 that a certain type of network recovers properties of a memory, the rediscovery of the backpropagation algorithm in the late $80s$, as well as recent developments in *deep neural network* structures.

2.9.1 Perceptrons

Perceptrons are the basic building block of neural networks

$$f(x; w) = \varphi(w^T x)$$

The nonlinear function φ is called an *activation function*. One trains the perceptron model by iterating over data and updating weights according to

$$w_i^{(t+1)} = w_i^{(t)} + \eta(y^m - \varphi(x^T w^{(t)}))x_i^m$$

where η is the learning rate. Perceptrons become powerful when combined together.

2.9.2 Feed-Forward Neural Networks

Feed forward neural networks have a model function of the form

$$f(x, W_1, W_2, \cdots) = \cdots \varphi_2(W_2\varphi_1(W_1 x)) \cdots$$

here $\varphi_1, \varphi_2 \cdots$ are nonlinear activation functions between appropriate spaces. The parameters are summarized as weight matrices $W_i, i = 1, 2, \cdots$

SKIPPING A BUNCH OF STUFF TO READ UP A BIT ON QUANTUM INFORMATION THEORY FOR NOW! COME BACK AT LATER DATE TO READ THOROUGHLY!!!!!!!!!! FROM PAGE 62 IN PDF

3 Chapter 3 Notes: Quantum Information

In the following we will start with an intuition for what operators, states and unitary evolutions are, and proceed to a more rigorous introduction to quantum theory based on the Dirac notation. We then introduce quantum computing together with the conceps of gates, qubits and quantum algorithms. As a preparation for later chapters we then discuss different ways to encode information into quantum systems and present some core quantum routines used in later chapters.

3.1 A Brief History of Quantum Mechanics

The initial step towards the theory we today call quantum mechanics is commonly attributed to Max Planck in the year 1900 when he introduced the notion that energy can only take discrete values (see ultraviolet disaster). Almost in parallel, Albert Einstein made a discovery similar in spirit: the photoelectric effect.

In the following years, these early ideas of a '*theory of energy quanta*' were applied to atomic spectroscopy (Niels Bohr) and to light (Louis de Broglie). Following this Werner Heisenberg, Pasual Jordan, Max Born and, independently, Paul Dirac formulated the first mathematically consistent version of quantum theory referred to as *matrix mechanics* in 1925. Using matrix mechanics Wolfgang Pauli was able to derive the experimental results of the hydrogen spectra. Heisenberg postulated his uncertainty principle shortly after. In 1926, following a different route Erwin Schrödinger proposed his famour equation of motion for the wave function describing the state of a quantum system. Matrix mechanics and the wave function formulation were shown to be equivalent, and a more general version was proposed by Dirac and Jordan. In the following years, quantum theory branched out into many sub-disciplines such as quantum many-body systems, quantum thermodynamics, quantum optics and quantum information processing.

3.2 Intuition Building and Basic Formalism

Quantum theory can be understood as a generalization of classical probability theory to make statements about the probabilities of the outcomes of measurement on quantum systems.

3.2.1 States and Observables

Consider a set of N mutually exclusive events $\mathcal{S} = \{s_1, \cdots, s_N\}$. In physics, events usually refer to the outcomes of measurements on physical properties of a system. One can associate each event with a random variable X that can take the values $\{x_1, \cdots, x_N\}$, and define a probability distribution over these values quantifying our knowledge on how likely an event is to occur. The probabilities related to the N events, $\{p_1, \cdots, p_N\}$, are real numbers, and they fulfill $p_i \leq 0$ and $\sum_{i=1}^{N} p_i = 1$. The expectation value of the random variable is defined as

$$\langle X \rangle = \sum_{i=1}^{N} p_i x_i$$

and is the weighed average of all values the random variable can take.

From here one needs two steps in order to arrive at the description of a quantum system with N different possible configurations or measurement outcomes: **First**, summarize the probabilities and the outcomes by matrices and vectors, and **second** replace the real positive probabilities with complex amplitudes.

For the first step, we consider a vector of the square roots of probabilities $p_1, \cdots, p_N$

$$q = \begin{bmatrix} \sqrt{p_1} \\ \vdots \\ \sqrt{p_N} \end{bmatrix} = \sqrt{p_1} \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix} + \cdots + \sqrt{p_N} \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix}$$

as well as a diagonal matrix X corresponding to the random variable with the same name

$$\begin{bmatrix} x_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & x_N \end{bmatrix}$$

The expectation value can now be written

$$q^T X q = \sum_{i=1}^{N} p_i x_i$$

For simplicity, both objects q, X were expressed in the standard basis of $\mathbb{R}^N$. One can see that the basis vector $(1, 0, \cdots, 0)^T$ forms a subspace of the $\mathbb{R}^N$ that is associated with the first event. Moreover, the outer product of this basis vector with itself

$$|1 \cdots 0\rangle \langle 0 \cdots 1| = (1 \cdots 0) \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$$

is a *projector* onto this subspace. Multiplied to a vector, it '*picks out*' the first element. This is a motivation why in quantum probability theory, the measurement events are represented by projectors onto subspaces which we will see in the next section. The sum of all projectors to the subspaces of $x_1, \cdots, x_N$ is the identity. (**Note**: *you don't have to use the standard basis, but you can use any basis* $\{v_i\}$ *for which the eigenvalue equations* $Xv_i = x_i v_i$ *hold.*)

This formulation is very close to the formalism of quantum mechanics. In quantum mechanics we replace q with a *complex amplitude vector* $\alpha = (\alpha_1, \cdots, \alpha_N)^T \in \mathbb{C}^N$, and X by a complex, self-adjoint matrix $O \in \mathbb{C}^{N \times N}$. Another term for self-adjoint is *Hermitian*. Hermitian operators have the property that they are equal to their complex-conjugate transpose $O = (O^*)^T$, where in quantum mechanics the symbol for this operation is the dagger $(O^*)^T = O^\dagger$.

Let $\{v_1, \cdots, v_N\}$ be a basis of orthonormal eigenvectors of O which spans the $\mathbb{C}^N$, fulfilling eigenvalue equations $Ov_i = o_i v_i$ and normalizations $v_i^\dagger v_j = \delta_{ij}$ for all $i, j \in \{1, \cdots, N\}$. Any amplitude vector α can be written as a linear combination of this basis

$$\alpha = \alpha_1 v_1 + \cdots + \alpha_N v_N$$

The expectation value of the random variable O corresponding to the matrix O now becomes

$$\langle O \rangle = \alpha^\dagger O \alpha = \sum_{i=1}^{N} |\alpha_i|^2 v_i^\dagger O v_i = \sum_{i=1}^{N} |\alpha_i|^2 o_i$$

One can see that the $|\alpha_i|^2$ take the role of the probabilities in the classical example, and we therefore demand that the sum of these be normalized to unity. If the basis is the standard basis, then α_i will just be the entries of α and the normalization condition therefore means that the amplitude vector is of unit length.

The eigenvalues o_i of O correspond to the values of the random variable or the outcomes of measurements. For example, when measuring the energy configurations of an atom, o_i would simply be an energy value in a given unit. However, without further constraints the eigenvalues of a complex matrix can be complex, which would not make any sense when looking at physically feasible variables. This is the reason we choose O to be a Hermitian operator, since the eigenvalues of Hermitian operators are always real and therefore physical.

3.2.2 Unitary Evolutions

In quantum theory, we work with amplitude vectors with the property that the entries' sum of absolute squares has to sum up to one. Consequently, an evolution has to be described by a *unitary* matrix (*the complex equivalent of an orthogonal transformation which is known to maintain lengths*)

$$\begin{bmatrix} u_{11} & \cdots & u_{1N} \\ \vdots & \cdots & \vdots \\ u_{N1} & \cdots & u_{NN} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix} = \begin{bmatrix} \alpha'_1 \\ \vdots \\ \alpha'_N \end{bmatrix}, \qquad \sum_{i=1}^{N} |\alpha_i|^2 = \sum_{i=1}^{N} |\alpha'_i|^2 = 1$$

A unitary matrix has the property that its inverse is its complex conjugate. $U^\dagger U = 1$.

Besider unitary evolutions, one can perform *measurements* on a quantum system, which change the state. Although there are many subtleties, in their basic nature quantum measurements are equivalent to classical statistics.

3.2.3 Composite Systems and Subsystems

In classical probability theory, samples spaces are combined using a tensor product. If the probabilities do not factorize the two variables are non-separable. Creating a joint sample space in quantum mechanics is done in exactly the same way, but using amplitude vectors instead of probability vectors.

Let us look at the opposite direction of how to *marginalize* over parts of the system. Marginalizing over variables basically means to give up information about them, a change in our state of knowledge towards more uncertainty. Given a multivariate probability distribution, marginalizing means to consider *cuts* or *subspaces* of the distribution. Again, the marginalization process in quantum mechanics is structurally the same in the classical setting, and has to be modelled 'on top' of the quantum probability. For this one uses a slightly different description of quantum states than amplitude vectors.

3.2.4 Density Matrices and Mixed States

Consider an amplitude vector $\alpha = (\alpha_1, \alpha_2)^T$. We have perfect knowledge that α is the state of the system, the state is *pure*. A stochastic description of this system means that we are uncertain if our system is in state $\alpha = (\alpha_1, \alpha_2)^T$ or $\beta = (\beta_1, \beta_2)^T$, and the overall state is called a statistical *mixture* of two pure states, or a *mixed state*.

We can incorporate statistical mixtures of quantum states into the formalism of quantum mechanics by switching from amplitude vectors to so called *density matrices*. A density matrix ρ that corresponds to a pure quantum state α is given by the outer product $\rho = \alpha\alpha^T$. To statisticians, this is known as the co-variance matrix of α, which for our $2-d$ example reads

$$\rho = \alpha\alpha^\dagger = \begin{bmatrix} |\alpha_1|^2 & \alpha_1\alpha_2^* \\ \alpha_2\alpha_1^* & |\alpha_2|^2 \end{bmatrix}$$

If we do not know the quantum state of the system, but we know that the system is in state α with probability p_1 and in state β with probability p_2, the density matrix to describe such a state reads

$$\rho = p_1\alpha\alpha^\dagger + p_2\beta\beta^\dagger = \begin{bmatrix} p_1|\alpha_1|^2 + p_2|\beta_1|^2 & p_1\alpha_2\alpha_1^* + p_2\beta_2\beta_1^* \\ p_1\alpha_1\alpha_2^* + p_2\beta_1\beta_2^* & p_1|\alpha_2|^2 + p_2|\beta_2|^2 \end{bmatrix}$$

In other words, the density matrix formulation allows us to combine '*quantum statistics*' expressed by a quantum state, with '*classical statistics*' or a probabilistic state that describes our lack of knowledge. This is possible because ρ's diagonal contains probabilities and not amplitudes.

The density matrix notation lets us elegantly exclude a subsystem from the description, and the corresponding mathematical operation is to calculate the *partial trace* over the subsystem. We will later see that this is closely related to the idea of marginalization over distributions. The partial trace operation in matrix is the sum over the diagonal elements in the corresponding space. Assume a state describing two joint 2-dimensional systems in density matrix notation is given by

$$\rho_{AB} = \begin{bmatrix} \rho_{11} & \rho_{12} & \rho_{13} & \rho_{14} \\ \rho_{21} & \rho_{22} & \rho_{23} & \rho_{24} \\ \rho_{31} & \rho_{32} & \rho_{33} & \rho_{34} \\ \rho_{41} & \rho_{42} & \rho_{43} & \rho_{44} \end{bmatrix}$$

and the partial trace over the first subsystem is

$$\mathrm{tr}_A(\rho_{AB}) = \begin{bmatrix} \rho_{11} + \rho_{33} & \rho_{12} + \rho_{34} \\ \rho_{21} + \rho_{43} & \rho_{22} + \rho_{44} \end{bmatrix}$$

while the state of the second subsystem is given by

$$\mathrm{tr}_B(\rho_{AB}) = \begin{bmatrix} \rho_{11} + \rho_{22} & \rho_{13} + \rho_{24} \\ \rho_{31} + \rho_{42} & \rho_{33} + \rho_{44} \end{bmatrix}$$

3.3 The Postulates of Quantum Mechanics

In this section we switch gears into a more rigorous formulation of quantum mechanics with the help of the postulates of quantum mechanics. The main mathematical tool will be Dirac notation. The Dirac notation may be confusing to newcomers to quantum mechanics. Most topics of quantum computing can actually be expressed in terms of matrices and vectors, and we will frequently switch between the two notations. However, some quantum systems have operators with a continuous eigenbasis, where discrete and finite vectors are not very useful. Dirac notation is an abstraction that incorporates such cases, and is the standard formalism in quantum computing.

3.3.1 State Space

A quantum mechanical state lives in a Hilbert space $\mathcal{H}$, which is a complex separable vector space. As mentioned, for the purposes of this book it suffices to consider discrete and finite-dimensional Hilbert spaces. These are isomorphic to the $\mathbb{C}^n$, for $n = 2, 3, \cdots$ and a quantum state therefore has a representation as a complex vector.

Vectors in Hilbert space are usually denoted by $|\psi\rangle$. The complex vector representation of a **ket** from a discrete and finite-dimensional Hilbert space is exactly the amplitude vector $\alpha = (\alpha_1, \cdots, \alpha_N)^T$ that we introduced in the previous section. The *dual* to the ket is called the **bra**, hence Dirac notation is sometimes also referred to as braket notation (from bracket). The *bra* is the complex conjugate row vector of the *ket* $\alpha^\dagger = (\alpha_1, \cdots, \alpha_N)$. The inner product is linear in the '*ket argument*' and anti-linear in the '*bra argument*'. Meaning

$$\langle\psi_2|\psi_1\rangle^* = \langle\psi_1|\psi_2\rangle$$

The norm of a vector $|\psi\rangle \in \mathcal{H}$ is

$$||\psi|| = \sqrt{\langle\psi|\psi\rangle}$$

For every vector in $\mathcal{H}$ there is a complete orthonormal basis $\{|e_i\rangle\}$ with $\langle e_i|e_j\rangle = \delta_{ij}$, such that

$$|\psi\rangle = \sum_i |e_i\rangle \langle e_i|\psi\rangle = \sum_i \langle e_i|\psi\rangle |e_i\rangle$$

This equation describes a **basis change**, since it expresses $|\psi\rangle$ in terms of the basis $\{|e_i\rangle\}$. Furthermore, $|e_i\rangle \langle e_i|$ denotes a projector on a one-dimensional sub-space. The equation also expresses the property of **separability** of Hilbert spaces. The same equation implies that the **completeness relation** holds

$$\mathbb{I} = \sum_i |e_i\rangle \langle e_i|$$

Previously we saw the utility of **density matrices**, let us formulate these in Dirac notation. Imagine a system is in one of the states $\psi_k, k = 1, \cdots, K$, with a certain probability p_k. The density operator ρ describing this *mixed* quantum mechanical state is defined as

$$\rho = \sum_{k=1}^{K} p_K |\psi_k\rangle \langle \psi_i|$$

where all $p_k \leq 0$ and $\sum_k p_k = 1$.

3.3.2 Observables

As we saw before, observables of a quantum mechanical system are realized as self-adjoint operators O in $\mathcal{H}$ that act on $|\psi\rangle$ characterizing the system. Such self-adjoint operators have a diagonal representation

$$O = \sum_i o_i |e_i\rangle \langle e_i|$$

(**Note**: *from this form of an operator one can see that the density matrix is also an operator on Hilbert space, although it describes a quantum state*) in terms of the set of eigenvalues $\{o_i\}$ and the corresponding eigenvectors $\{|e_i\rangle\}$. This so called **spectral representation** is unique and allows to calculate analytic functions f of the observable according to the formula

$$f(O) = \sum_i f(o_i) |e_i\rangle \langle e_i|$$

In particular, if the observable is a Hermitian operator H, then for a real scalar ε

$$U = e^{i\epsilon H}$$

is a unitary operator.

In general, expectation values of a quantum mechanical observable O can be calculated as

$$\langle O\rangle = \mathrm{tr}\{\rho O\}$$

where 'tr' computes the trace of the operator. The trace operation in Dirac notation 'sandwiches' the expression inside the curly brackets by a sum over a full basis

$$\mathrm{tr}\{A\} = \sum_i \langle e_i| A |e_i\rangle$$

For a system in a pure state $\rho = |\psi\rangle \langle \psi|$, the expression for the expectation value of an observable reduces to

$$\langle O\rangle = \sum_i \langle e_i|\psi\rangle \langle \psi| O |e_i\rangle = \sum_i \langle \psi|e_i\rangle \langle e_i| O |\psi\rangle = \langle \psi| O |\psi\rangle$$

where we have used the completeness relation.

3.3.3 Time Evolution

The time evolution of a quantum mechanical system is described by the Schrödinger equation

$$i\hbar\frac{d}{dt}|\psi\rangle = H|\psi\rangle$$

In the next chapter we introduce several Hamiltonians relevant to quantum information processing. For time-dependent Hamiltonians the solutions of the equation for an initial condition $|\psi(t_0)\rangle = |\psi_0\rangle$ can be written

$$|\psi(t)\rangle = U(t)|\psi_0\rangle$$

where

$$U(t) = e^{-\frac{i}{\hbar}tH}$$

is the unitary time-evolution operator.

The time evolution of a quantum system in a mixed state described by the density operator ρ follows immediately from its definition

$$i\hbar\frac{d}{dt}\rho(t) = \frac{d}{dt}\sum_i p_i|\psi_i\rangle\langle\psi_i| = [H,\rho]$$

This equation is called the **von Neumann equation**. The formal solution of the von Neumann equation for initial condition $\rho(t_0)$ follows along similar lines to the Schrödinger equation

$$\rho(t) = U(t,t_0)\rho(t_0)U(t,t_0)^\dagger$$

In other words, to evolve a density operator, we have to apply the unitary from the left, and its complex conjugate from the right. (**Note**: *in theoretical physics and in this book it is customary to set* $\hbar = 1$)

3.3.4 Quantum Measurement

For a quantum mechanical system in a pure state $|\psi\rangle$ we consider an observable O, with a discrete spectrum, i.e.

$$O = \sum_i o_i P_i$$

where P_i denotes the projector on the eigenspace spanned by the $i-th$ eigenvector. The o_i are the measurement results of a measurement associated with projectors P_i. Such a measurement is called a *projective measurement*. According to the *Born rule*, the probability to obtain the measurement result o_n is given by

$$p(o_n) = \mathrm{tr}\{P_n|\psi\rangle\langle\psi|\} = ||P_n|\psi\rangle||^2$$

The state of the system after the measurement is given by

$$|\psi\rangle \to \frac{P_n|\psi\rangle}{\sqrt{\langle\psi|P_n|\psi\rangle}}$$

In general, if the system is described by a density matrix ρ, the probability of measuring o_n will be

$$p(o_n) = \mathrm{P_n}\rho$$

and the state of the system after the measurement will be

$$\frac{P_n\rho P_m}{\mathrm{tr}\{P_n\rho\}}$$

Consider now a measurement defined by a collection of operators $\{A_m\}$ that do not necessarily have to be projectors. This is a more general case. If, the state of the system is described by the pure state $|\psi\rangle$, the probability that the result o_m related to operator A_m occurs is given by

$$p(a_m) = \langle\psi|A_m^\dagger A_m|\psi\rangle$$

and the state after the measurement is

$$\frac{A_m |\psi\rangle}{\sqrt{\langle\psi| A_m^\dagger A_m |\psi\rangle}}$$

The measurement operators satisfy the completeness relation $\sum_m A_m^\dagger A_M = \mathbf{I}$. If we define

$$E_m = A_m^\dagger A_m$$

The operators E_m are called Positive Operator-Valued Measure (POVM) elements and their complete set is known as a Positive Operator-Valued Measurement. This is the most common framework to describe quantum measurements.

3.3.5 Composite Systems

A composite state of two joint quantum systems is a tensor product of two quantum states, which of course carries over to Dirac notation. Let us assume that our quantum system of interest Σ is composed of two sub-systems, say Σ_1, Σ_2. The sub-system $\Sigma_i, i = 1, 2$ is in a Hilbert space $\mathcal{H}_\rangle$ with $\dim(\mathcal{H}_\rangle) = N_i$ and is spanned by an orthonormal basis set $\{|e_j^i\rangle\}, j \in \mathbb{N}$. Then, the Hilbert space $\mathcal{H}$ of the total system Σ is given by the tensor product of the Hilbert spaces of the two sub-systems, i.e. $\mathcal{H}_\infty, \mathcal{H}_\in$

$$\mathcal{H} = \mathcal{H}_\infty \otimes \mathcal{H}_\in$$

The states in this joint Hilbert space can be written as

$$\sum_{j=1}^{N_1} \sum_{k=1}^{N_2}$$

and have $\dim(\mathcal{H}) = N_1 N_2$. The coefficients c_{jk} are complex scalars and satisfy $\sum_{jk} |c_{jk}|^2 = 1$. Composite quantum systems have the famous quantum feature of **entanglement**. If a state $|\psi\rangle$ can be expressed as

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$$

with $|\psi_1\rangle \in \mathcal{H}_\infty, |\psi_2\rangle \in \mathcal{H}_\in$ it is called *separable*. If the above representation is not possible, the state is called *entangled*. Separability and entanglement work similarly for density matrices. If the two subsystems Σ_1, Σ_2 are uncorrelated and described by density matrices ρ_1, ρ_2 respectively, then the total density matrix ρ of the composite system Σ is given by

$$\rho = \rho_1 \otimes \rho_2$$

Operator O acting on the total Hilbert space have the structure

$$O = O_1 \otimes O_2$$

where the two operators act on their respective subsystem. In general, the expectation value of such an operator can be calculated as

$$\langle O \rangle = \mathrm{tr}\{O\rho\} = \mathrm{tr}_1\{O_1\rho_1\}\mathrm{tr}_2\{O_2\rho_2\}$$

where $\mathrm{tr}_i, i = 1, 2$ denotes the partial trace, which only sums over basis states of the subsystem. Sometimes one is only interested in observables that operate only on one of the two subsystems i.e.

$$O = O_1 \otimes \mathbb{I}_2 \implies \langle O \rangle = \mathrm{tr}_1\{O_1\rho_1\}$$

3.3.6 Open Quantum Systems

The previous section's discussion is at the core of the theory of **open quantum systems**, where one identifies Σ_1 with the *open* system of interest and Σ_2 with its environment. The aim of the theory of open quantum systems is to find an effective dynamical description for the open system, through an appropriate elimination of the degrees of freedom of the environment. The starting point is the Hamiltonian evolution of the total system. The density matrix ρ of the total system Σ evolves according to the *von Neumann* equation

$$\frac{d}{dt}\rho = -i[H, \rho]$$

Introducing the reduced density matrix ρ_1 of the open system Σ_1 as the trace over the degrees of freedom of system Σ_2

$$\rho_1 = \mathrm{tr}_2\{\rho\}$$

the dynamics of the open system can be obtained from

$$\frac{d}{dt}\rho_1 = -i\mathrm{tr}_2\{[H,\rho]\}$$

This equation is difficult to evaluate as the dynamical equation of motion for the total system and appropriate approximations are needed to make it useful. In the case of weak-coupling between the sub-systems Σ_1, Σ_2 a situation that is typical in quantum optical applications, one can often assume that the density matrix of the total system factorizes at all times and that the density matrix of the environment is unchanged

$$\rho(t) \simeq \rho_1(t) \otimes \rho_2$$

If one further assumes that changes in the environment occur on time scales that are not resolved, the the dynamics of the open system is described by the so-called *Markovian Quantum Master equation* in Gorini-Kossakowski-Sudarshan-Lindblad form which reads

$$\frac{d}{dt}\rho_1 = -i[H,\rho] + \sum_k \gamma_k \Big(L_k \rho_1 L_k^\dagger - \frac{1}{2} L_k^\dagger L_k \rho_1 - \frac{1}{2}\rho_1 L_k^\dagger L_K \Big)$$

The operators L_k introduced above are usually referred to as Lindblad operators and describe the transition in the open system induced by the interaction with the environment. The constants γ_k are the relaxation rates for the different decay modes k of the open system. Seems this will be covered heavily in FYS4110.

3.4 Quantum Computing

Quantum computing is a subfield of quantum information processing, whose central questions are: *What is quantum information? Can we build a new type of computer based on quantum systems? How can we formulate algorithms on such machines? What does quantum theory mean for the limits of what is computable? What distinguishes quantum computers from classical ones?*

Although there are a variety of computational models that formalize the idea of quantum computations, most of them are based on the concept of a *qubit*, which is a quantum system associated with two measurable events (two-level-system, TLS). A quantum computer can be understood as a physical implementation of n qubits with a precise control on the evolution of the state. A quantum algorithm is a targeted manipulation of the quantum system with a subsequent measurement to retrieve information from the system. In this sense a quantum computer can be understood as a special sampling device: *We choose certain experimental configurations, such as the strength of a laser beam or a magnetic field, and read out a distribution over possible measurement outcomes.*

An important theorem in quantum information states that any quantum evolution can be approximated by a sequence of only a handful of elementary manipulations called *quantum gates* that only act on one or two qubits at a time. Quantum algorithms are widely formulated as *quantum circuits* of these elementary gates. A universal quantum computer consequently only has to know how to perform a small set of operators on qubits, just like classical computers are built on a limited number of logic gates. *Efficient* quantum algorithms are based on evolutions whose decomposition into a circuit of gates grows at most polynomially with the input size of the problem.

As much as a classical but is an abstract model of a binary system that can have many different physical realizations in principle, a qubit can be used as a model of many different physical quantum systems. Some current candidates for implementations are superconducting qubits, photonic setups, ion traps or topological properties of quasi-particles. Each of them has advantages and disadvantages, and it is not unlikely that future architectures use a mixture of these implementations.

3.5 Bits and Qubits

A qubit is realized as a quantum two level system and as such can be measured in two states, called the basis states. Traditionally, they are denoted by the Dirac vectors $|0\rangle, |1\rangle$, but $|-\rangle, |+\rangle, |\downarrow\rangle, |\uparrow\rangle$ are also common notations.

For our purposes it suffices to describe the transition from a classical to a quantum mechanical description of information processing by means of a very simple quantization procedure

$$0 \to |0\rangle$$
$$1 \to |1\rangle$$

The basis states form an orthonormal basis of a 2-dimensional Hilbert space which is also called the *computational basis*. As a consequence of the superposition principle of quantum mechanics the most general state of a qubit is

$$|\psi\rangle = \alpha_1 |0\rangle + \alpha_2 |1\rangle$$

where the amplitudes are complex and normalized like usual. The Hermitian conjugate, *dual*, of the above vector is

$$\langle\psi| = \alpha_1^* \langle 0| + \alpha_2^* \langle 1|$$

The above Dirac vectors have a vector representation because n-dimensional dicrete Hilbert spaces are isomorphic to the space of complex vectors $\mathbb{C}^N$. In vector notation, a general qubit is expressed as

$$|\psi\rangle = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}$$

And the Hermitian conjugate of this amplitude column vector is the transposed and conjugated row vector

$$\langle\psi| = (\alpha_1^*, \alpha_2^*) \in \mathbb{C}^2$$

Furthermore, we can represent the two computational basis states as the standard basis vectors of $\mathbb{C}^2$. Vector notation can be very useful when trying to understand the effect of quantum gates. However, as common in quantum computing, we will predominantly use Dirac notation.

It is sometimes useful to have a geometric representation of a qubit. A generic qubit in the pure state can be parametrized as

$$|\psi\rangle = e^{i\gamma}\Big(\cos\big(\frac{\theta}{2}\big)|0\rangle + e^{i\phi}\sin\big(\frac{\theta}{2}\big)|1\rangle\Big)$$

where $0 \geq \theta \geq \pi, 0 \geq \phi \geq 2\pi, \gamma$ are real numbers. The global phase factor has no observable effect and will be omitted in the following. The angles θ, ϕ have the obvious interpretation as spherical coordinates, so that the Hilbert space vector $|\psi\rangle$ can be visualized as the $\mathbb{R}^3$ vector

$$\vec{n} = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\phi)$$

pointing from the origin to the surface of a ball, the so-called **Bloch sphere**.

The Dirac notation allows also for a compact description of the inner product of two vectors in Hilbert space. We consider two vectors $|\psi_1\rangle = \alpha_1|0\rangle + \alpha_2|1\rangle, |\psi_2\rangle = \beta_1|0\rangle + \beta_2|1\rangle \in \mathbb{C}^2$ with $\alpha_i\beta_i \in \mathbb{C}$ and $\sum|\alpha_i|^2, \sum|\beta|^2 = 1$. Since the computational basis states are orthonormal, we have that

$$\langle 0|0\rangle = \langle 1|1\rangle = 1, \langle 0|1\rangle = \langle 1|0\rangle = 1$$

The inner product of $|\psi_1\rangle, |\psi_2\rangle$ is therefore given by

$$\langle\psi_1|\psi_2\rangle = \alpha_1^*\beta_1 + \alpha_2^*\beta_2$$

Similarly, the outer product of two states can be compactly written as

$$|\psi_1\rangle\langle\psi_2| = \begin{bmatrix} \alpha_1\beta_1^* & \alpha_1\beta_2^* \\ \alpha_2\beta_1^* & \alpha_2\beta_2^* \end{bmatrix}$$

which is the outer product of the amplitude vectors.

n unentangled qubits are described by a tensor product of single qubits

$$|\psi\rangle = |q_1\rangle \otimes \cdots \otimes |q_n\rangle$$

If the qubits are entangled, state $|\psi\rangle$ is not separable, and in the computational basis reads

$$|\psi\rangle = \alpha_1|0\cdots00\rangle + \alpha_2|0\cdots01\rangle + \cdots + \alpha_{2^n}|1\cdots11\rangle$$

with the same conditions on the amplitudes α_i as before. We will use the notation for a Dirac vector in the computational basis as

$$|\psi\rangle = \sum_{i=1}^{2^n} \alpha_i |i\rangle$$

The expression $|i\rangle$ for indices 1 to 2^n refers to the i-th computational basis state from the basis $\{|0\cdots 0\rangle, \cdots, |1\cdots 1\rangle\}$. The sequences of zeros and ones in the basis state at the same time corresponds to the binary representation of integer $i-1$. As a rule from now on, an index i between the Dirac bracket for a system of $n > 1$ qubits refers to the i-th computational basis state. The only exception being the single qubit state.

Using this notation, a pure density matrix is given by

$$\rho_{\text{pure}} = |\psi\rangle\langle\psi| = \sum_{i,j=1}^{N} \alpha_i^* \alpha_j |i\rangle\langle j|$$

For a general mixed state the coefficients do not factorize and we get

$$\rho_{\text{mixed}} = \sum_{i,j=1}^{N} \alpha_{ij} |i\rangle\langle j|, \qquad \alpha_{ij} \in \mathbb{C}$$

It is useful to consider the measurement of a qubit in the computational basis. Let us assume that the state being measured is the generic normalized qubit state. In this case the projectors on the two possible eigenspaces are $P_0 = |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$. The probability of obtaining the measurement outcome 0 is then

$$p(0) = \text{tr}(P_0 |\psi\rangle\langle\psi|) = \langle\psi| P_0 |\psi\rangle = |\alpha_1|^2$$

Similarly $p(1) = |\alpha_2|^2$. After the measurement, say outcome 0, the qubit is in the state

$$|\psi\rangle \to \frac{P_0 |\psi\rangle}{\sqrt{\langle\psi| P_0 |\psi\rangle}} = |0\rangle$$

A computational basis measurement can be understood as drawing a sample of a binary string of length n - where n is the number of qubits - from a distribution defined by the quantum state.

3.6 Quantum Gates

Quantum logic gates are realized by unitary transformations. Don't feel like taking notes from this nothing new when reading it, and it doesn't concern Dirac notation which I need to get better at reading.

3.7 Quantum Parallelism and Function Evaluation

As a first larger example of a quantum algorithm we want to construct a quantum logic circuit that evaluates a function $f(x)$. This simple algorithm will already exhibit one of the salient features of quantum algorithms: *quantum parallelism*. Roughly speaking, we will see how a quantum computer is able to evaluate many different values of the function $f(x)$ at the same time, or in superposition

To be specific we consider a very simple function $f(x)$ that has a single bit as input and a single bit as output, i.e. a function with a one bit domain and range.

$$f(x) : \{0,1\} \to \{0,1\}$$

Examples of such a function are the identity, constant, bit-flip functions. The idea is to construct a unitary transformation U_f such that

$$(x, y) \xrightarrow{U_f} (x, y \oplus f(x))$$

In the above equation the symbol $\oplus$ denotes mod 2 addition. It is easy to verify that U_f is unitary, we just check that $U_f^2 = \mathbb{I}$

$$(x, [y \oplus f(x)]) \to (x, [y \oplus f(x)] \oplus f(x)) = (x, y)$$

because $f(x) \oplus f(x) = 0$ for any x. Expressing U_f in operator notation, this reads

$$U_f(|x\rangle \oplus |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle$$

and we obtain the useful notation $U_f(|x\rangle \otimes |0\rangle) = |x\rangle \otimes |f(x)\rangle$. We are now in a position to demonstrate quantum parallelism. We consider a quantum circuit of 2 qubits. The circuit acts in the following way

1. We apply the Hadamard gate to the first qubit in the state $|0\rangle$ to obtain

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

2. If the second qubit is in the state $|0\rangle$, the state $|\psi\rangle$ after the application of the unitary U_f is

$$|\psi\rangle = U_f(H|0\rangle \otimes |0\rangle) = U_f \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |0\rangle)$$
$$\frac{1}{\sqrt{2}}(|0\rangle \otimes |f(0)\rangle + |1\rangle \otimes |f(1)\rangle)$$

It is now evident that the state $|\psi\rangle$ contains simultaneously the information on $f(0)$ and $f(1)$. The circuit has produced a superposition in one single step state that contains both $f(0), f(1)$.

It is important to realize that the above procedure does not yet give us an advantage over classical computation. Although $|\psi\rangle$ is a superposition of $f(0), f(1)$ in order to access the information we need to perform a measurement. If we measure

$$\sum_{x=0,1} |x\rangle |f(x)\rangle$$

with a computational basis measurement, we obtain only one value of x and $f(x)$. In fact, we are only able to get a value of our function at a random argument. The real power of quantum algorithms will be made evident in the next example.

3.7.1 The Deutsch Algorithm

The Deutsch algorithm exploits what we have learned so far to obtain information about a global property of a function $f(x)$. A function of a single bit can either be *constant* ($f(0) = f(1)$), or *balanced* ($f(0) \neq f(1)$). These properties are *global*, because in order to establish them we need to calculate both $f(0)$ and $f(1)$ and compare the results. As we will see a quantum computer can do better.

Essentially, the Deutsch algorithm computes

$$|\psi_3\rangle = (H \otimes \mathbb{I}) U_f (H \otimes H)|01\rangle$$

for an initial state $|\psi_0\rangle = |01\rangle$. Step by step

1. First we calculate $|\psi_1\rangle$

$$|\psi_1\rangle = (H \otimes H)|01\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$
$$= \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$$

2. Next we apply U_f to $|\psi_1\rangle$. It is convenient to write $|\psi_1\rangle$ as

$$|\psi_1\rangle = \frac{1}{2}\Big(\sum_{x=0}^{1} |x\rangle\Big) \otimes (|0\rangle - |1\rangle)$$

In order to evaluate the action of U_f on $|\psi_1\rangle$ we need to separately consider the case when $f(x) = 0, f(x) = 1$. For the first case we find

$$U_f(|x\rangle \otimes (|0\rangle - |1\rangle)) = |x, 0 \oplus f(x)\rangle - |x, 1 \oplus f(x)\rangle = |x, 0 \oplus 0\rangle - |x, 1 \oplus 1\rangle$$
$$|x\rangle (|0\rangle - |1\rangle)$$

Similarly for $f(x) = 1$ we find

$$U_f(|x\rangle \otimes (|0\rangle - |1\rangle)) = -|x\rangle (|0\rangle - |1\rangle)$$

The two above cases can elegantly be summarized as

$$|\psi_2\rangle = U_f|\psi_1\rangle = \frac{1}{2}\Big(\sum_{x=0}^{1}(-1)^{f(x)}|x\rangle\Big) \otimes (|0\rangle - |1\rangle) = |\phi\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

In other words, the net result after the application of U_f on the input register is

$$|\phi\rangle = \frac{1}{\sqrt{2}}\big((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\big)$$

3. In the last step, just before the measurement, we apply a Hadamard gate to the input qubit.

$$|\psi_3\rangle = (H \otimes \mathbb{I})\Big(|\phi\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\Big)$$

It is easy to calculate

$$H|\phi\rangle = \frac{1}{2}\big((-1)^{f(0)} + (-1)^{f(1)}\big)|0\rangle + \frac{1}{2}\big((-1)^{f(0)} - (-1)^{f(1)}\big)|1\rangle$$

Thus if the measurement of the qubit gives the state $|0\rangle$, then we know for sure that $f(0) = f(1)$ as the coefficient in front of $|1\rangle$ vanishes. Therefore the function $f(x)$ is constant. Alternatively, if the measurement gives the state we know for certain that $f(0) \neq f(1)$ and the function is balanced!

Quantum parallelism has allowed the calculation of the global properties of a function without having to evaluate explicitly the values of the function! (**Note**: *The next sectoin in [1] is the natural continuation onto the Deutsch-Jozsa algorithm. Understanding the previous one makes the next one simple, and I already have plenty of notes on it from MAT3420. Skipping.*)

3.8 Quantum Annealing and Other Computational Models

Very short read. Might do notes on later

3.9 Strategies of Information Encoding

This section is redundant-ish with chapter 5. Just introduces stuff which will be covered deeper in that chapter. Might do notes on this later too.

3.10 Important Quantum Routines

This is also something which I have knowledge / notes on from before. These should be looked at again when writing about further algorithms to make sure they're understood. Listing the ones mentioned:

- Grover Search (Amplitude amplification)
- Quantum Phase Estimation (with Quantum Fourier Transform)
- Quantum Matrix Multiplication and Inversion (**NOT STUDIED / NOTES ON**). This uses quantum phase estimation to multiply matrices. Apparently central to some QML algos (Pretty long explanation)

4 Chapter 4 Notes: Quantum Advantages

Before coming to the design of quantum machine learning algorithms, this chapter is an interlude to discuss how quantum computing can actually assist machine learning. Although quantum computing researchers often focus on asymptotic computational speedups, there is more than one measure of merit when it comes to machine learning. We will discuss three dimensions here, namely **computational complexity**, **sample complexity** and the **model complexity**. While the section on computational complexity allows us to establish the terminology already used in previous chapters with more care, the section on sample complexity ventures briefly into quantum extensions of statistical learning theory. The last section on model complexity provides arguments towards what has been called the *exploratory* approach to quantum machine learning, in which quantum physics is used as a resource to build new types of models altogether.

4.1 Computational Complexity of Learning

The concept of runtime complexity and speedups has already been used in the previous sections and is the most common figure of merit when accessing potential contributions of quantum computing to machine learning. Quantum machine learning inherited this focus on runtime speedups from quantum computing where researchers mostly resort to proving theoretical runtime bounds to advertise the power of their algorithms. Let us briefly introduce the basics of asymptotic computational complexity.

The runtime of an algorithm on a computer is the time it takes to execute the algorithm, in other words the number of elementary operations multiplied by their respective execution times. Computational complexity theory looks at the *asymptotic complexity* or the rate of growth of the runtime with the size n of the input. 'Asymptotic' thereby refers to the fact that one is interested in laws for sufficiently large inputs only. If the resources needed to execute an algorithm or the number of elementary operations grow polynomially with the size of the input n, it is **tractable** and the problem is in theory *efficiently solvable*. Exponential growth makes an algorithm *intractable* and the problem *hard*.

When considering quantum computers, estimating the actual runtime of an algorithm is even more problematic. Not only do we not have a unique implementation of qubits yet that gives us a set of elementary gates as well as a technology, it is nontrivial to decompose quantum algorithms into this set. In many cases, we can claim that we know there is such a sequence of elementary gates, but it is by no means clear how to construct it. The vast majority of authors in quantum information processing are therefore interested in the asymptotic complexity of their routines, and how they compare to classical algorithms. In a qubit-based quantum computer, the input is considered to be the number of qubits n, which we already hinted at by the notation. To avoid confusion when looking at different concepts for the input, we will call polynomial algorithms regarding the number of qubits **qubit-efficient**. Algorithms which are efficient with respect to the number of amplitudes will be referred to as **amplitude-efficient**.

The field of **quantum complexity theory** has been developed as an extension to the classical case, and is based on the question whether quantum computers are in principle able to solve computational problems fast in relation to the runtime complexity.

A collaboration of researchers at the forefront of benchmarking quantum computers came up with a useful typology for the term 'quantum speedup' which shows the nuances of the term

1. A *provable quantum speedup* requires a proof that there can be no classical algorithm that performs as well or better than the quantum algorithm

2. A *strong quantum speedup* compares the quantum algorithm with the best classical algorithm.

3. If we relax the term 'best classical algorithm' to the 'best available classical algorithm' we get the definition of a *common quantum speedup*

4. The fourth category of *potential quantum speedup* relaxes the conditions further by comparing two specific algorithms and relating the speedup to this instance only

5. Lastly, and useful when doing benchmarking with quantum annealers, is the *limited quantum speedup* that compares to 'corresponding' algorithms such as quantum and classical annealing

4.2 Sample Complexity

Besides the asymptotic computational complexity, the so called *sample complexity* is an important figure of merit in classical machine learning, and offers a wealth of theoretical results regarding the subject of learning, summarized under the keyword *statistical learning theory*. It also offers a door to explore more fundamental questions of quantum machine learning, for example what it means to learn in a quantum setting, or how we can formulate a quantum learning theory.

Sample complexity refers to the number of samples needed to generalize from data. Samples can either be given as training instances drawn from a certain distribution (*examples*) or by computing outputs to specifically chosen inputs (*queries*). For the supervised pattern recognition problem we analyzed so far, the dataset is given as examples and no queries can be made. One can easily imagine a slightly different setting where queries are possible, for example when a certain experiment results in input-output pairs, and we can choose the settings for the experiment to generate data.

Considerations about sample complexity are usually based on binary functions $f : \{0,1\}^N \to \{0,1\}$. The sample complexity of a machine learning algorithm refers to the number of samples that are required to learn a *concept*

from a given *concept class*. A concept is the rule f that divides the input space into subsets of the two class labels 0 and 1.

There are two important settings in which sample complexity is analyzed

1. In *exact learning from membership queries*, one learns the function f by querying a membership oracle with inputs x and receives the answer whether $f(x)$ evaluates to 1 or not.

2. The framework of **Probably Approximately Correct (PAC)** learning was introduced by Valiant and asks how many examples from the original concept are needed in the worst case to train a model so that the probability of an error ϵ (*the probability of assigning the wrong label*) is smaller than $1-\delta$ for $0 \leq \delta \leq 1$. The examples are drawn from an arbitrary distribution via an example oracle. This framework is closely related to the Vapnik-Chervonenkis- (VC) dimension of a model

To translate these two settings into a quantum framework, a quantum membership oracle as well as a quantum example oracle are introduced. They are in a sense parallelized versions of the classical sample generators, and with quantum interference of amplitudes we can hope to extract more information from the distribution than possible in the classical case. Rather surprisingly, it turns out that the classical and quantum sample complexity are polynomially equivalent. (**Note**: *this only concerns the sample complexity, there are examples of efficiently learnable quantum algorithms in terms of computational complexity, while the best classical algorithm is intractable.*)

4.2.1 Exact Learning from Membership Queries

Sample complexity in relation to queries in closely related to the concept of 'quantum query complexity' which is an important figure of merit in quantum computing in the oracular setting. A quantum oracle can be described as a unitary operation

$$U : |x, 0\rangle \rightarrow |x, 0 \oplus f(x)\rangle$$

where $x \in \{0,1\}^n$ is encoded in the computational basis.

Two famous quantum algorithms that demonstrate how for specific types of problems only a single quantum query can be sufficient, are the Deutsch-Josza algorithm as well as the Bernstein-Vazirani algorithm. They are both based on the principle of applying the quantum oracle to a register in uniform superposition, thereby querying all possible inputs in parallel. Writing the outcome into the phase and interfering the amplitudes then reveals information on the concept, for example if it was a balanced or constant function. Note that this does not mean the function itself is learnt and is therefore not sufficient as an example to prove theorems on general quantum learnability.

In 2003, **Hunziker et al** introduced a larger framework which that call 'impatient learning' and proposed the following two conjectures on the actual number of samples required in the asymptotic quantum setting:

1. For any family of concept classes $C = \{C_i\}$ with $|C| \rightarrow \infty$, there exists a quantum learning algorithm with membership oracle query complexity $\mathcal{O}(\sqrt{|C|})$

2. For any family of concept classes $C = \{C_i\}$ containing $|C| \rightarrow \infty$ concepts, there exists a quantum learning algorithm with membership oracle query complexity $\mathcal{O}(\frac{\log|\mathcal{C}|}{\sqrt{\gamma}})$, where $\gamma \leq 1/3$ is a measure of how easy it is to distinguish between concepts, and small γ indicate a harder class to learn

While the first conjecture was proven by Ambainis et al., the second conjecture was resolved in a series of contributions (These are all sourced in [1] p.133). The classical upper bound for exact learning from membership queries is given by $\mathcal{O}(\frac{\log|C|}{\gamma})$.

Servedio and Gortler compared these results and found that if any class of boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$ is learnable from Q quantum membership queries, it is then learnable by at most $\mathcal{O}(nQ^3)$ classical membership queries. This result shows that classical and quantum learnability have at most a polynomial overhead. It becomes apparent that no exponential speedup can be expected from quantum sample complexity for exact learning with membership queries.

4.2.2 Probably Approximately Correct (PAC) Learning from Examples

It is a well-established fact from classical learning theory that a (ϵ, δ)-PAC learning algorithm for a non-trivial concept class C of Vapnik-Chervonenkis-dimension d requires at least $\Omega(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{d}{\epsilon})$ examples, but can be learnt with at most $\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{d}{\epsilon}\log\frac{1}{\epsilon})$ examples.

A first contribution to quantum PAC learning was made by Bshouty and Jackson in 1998, who defined the important notion of a quantum example oracle. The quantum example oracle is equivalent to what has been introduced earlier as **qsample**

$$\sum_i \sqrt{p(x)}\,|x, f(x)\rangle$$

where the probabilities of measuring the basis states in the computational basis reflects the distribution $p(x)$ from which the examples are drawn. They show that a certain class of functions (so called 'polynomial-size disjunctive normal form expressions') are efficiently learnable by a quantum computer from quantum example oracles which draw examples from a uniform distribution. This speedup is achieved by interfering the amplitudes of the qsample with a quantum Fourier transform. However, the PAC model requires learnability under *any* distribution, and the results of Bshouty and Jackson do therefore not lead to a statement about the general quantum PAC case. It can be shown that the equivalence of classical and quantum learning extends to the PAC framework. It has been proven (*sources* [**?**] *p.134*) that if any class C of boolean functions $f\{0,1\}^n \to \{0,1\}$ is learnable from Q evaluations of the quantum example oracle, it is then learnable by $\mathcal{O}(nQ)$ classical examples.

Improvements in a later paper by Atici and Servedio prove a lower bound on quantum learning that is close to the classical setting. After a few more improvements it was finally shown that in the PAC setting, quantum and classical learners require the same amount of examples up to a constant factor. This also holds true for the *agnostic* learning framework, which loosens PAC learning by looking for a hypothesis that is sufficiently close to the best one.

As a summary under the quantum formulation of classical learning theory based on oracles we are not expecting any exponential advantages from quantum computing.

4.2.3 Introducing Noise

Even though the evidence suggest that classical and quantum sample complexity are similar up to at most polynomial factors, and interesting observation derives from the introduction of noise into the model. Noise refers to corrupted query results or examples for which the value $f(x)$ is flipped with probability μ. For the quantum example oracle, noise is introduced by replacing the oracle with a mixture of the original qsample and a corrupted qsample weighted by μ. In the PAC setting with a uniform distribution investigated by Bshouty and Jackson, the quantum algorithm can still learn the function by consuming more examples, while noise is believed to render the classical problem unlearnable. A similar observation is made by Cross, Smith and Smolin, who consider the problem of learning n-bit parity functions by membership queries and examples, a task that is easy both classically and quantumly, and in the sample as well as time complexity sense. However, Cross et al. find that in the presence of noise, the classical case becomes intractable while the quantum samples required only grow logarithmically. To ensure a fair comparison, the classical oracle is modelled as a dephased quantum channel of the membership oracle and the quantum example oracle respectively. As a toy model the authors consider a slight adaptation of the Bernstein-Vazirani algorithm. These observations might be evidence enough that a fourth category of potential quantum advantages, the robustness against noise, could be a fruitful avenue for further research.

4.3 Model Complexity

Lastly, we will come to the third potential advantage that quantum computing could offer machine learning. The term 'model complexity' is a wide concept that refers to the flexibility, capacity, richness or expressive power of a model. For example, a model giving rise to linear decision boundaries is less flexible than a model with more complex decision boundaries. A linear fit in regression is less flexible than a higher-order polynomial. Roughly speaking, more complex model families offer a larger space of possible trained models and therefore have a better chance to fit the training data.

Of course, this does not mean that more flexible models have a higher generalization power. In fact, they are much more prone to overfitting, which is quantified by Vapniks upper bound for generalization error ϵ_{gen}. With probability of at least $1-\delta$ for some $\delta > 0$ the bound is given by

$$\epsilon_{\text{gen}} \leq \epsilon_{\text{emp}} + \sqrt{\frac{1}{M}\Big(d\Big(\log\big(\frac{2M}{d}\big)+1\Big)+\log\big(\frac{4}{\delta}\big)\Big)}$$

The generalization error is bounded from above by the 'empirical error' on the training set, plus an expression that depends on the VC-dimension d. The VC-dimension of a model is closely related to the model complexity and

is defined for binary models. It refers to the maximum number of data points M assigned to two classes for which a hypothesis or *trained version* of the model $f(x;\theta)$ makes no classification error, or *shatters* the data.

(**Note**: *There are interesting points made in the conclusion of this subsection about potential stuff to look into during master thesis work.*)

5 Chapter 5 Notes: Information Encoding

If we want to use a quantum computer to learn from classical data we have to think about how to represent features by a quantum system. We furthermore have to design a recipe for 'loading' data from a classical memory into the quantum computer. In quantum computing this process is called **state preparation**.

Classical machine learning textbooks rarely discuss matters of data representation and data transfer to the processing hardware. For quantum algorithms, these questions cannot be ignored. The strategy of how to represent information as a quantum state provides the context of how to design the quantum algorithm and what speedups one can hope to harvest. The actual procedure of encoding data into the quantum system is part of the algorithm and may account for a crucial part of the complexity.

5.1 Basis Encoding

Assume we are given a binary dataset $\mathcal{D}$ where each pattern $x^m \in \mathcal{D}$ is a binary string of the form $x^m = (b_1^m, \cdots, b_N^m)$ with $b_i^m \in \{0,1\}$ for $i = 1, \cdots, N$. We can prepare a superposition of basis states $|x^m\rangle$ that qubit-wise correspond to the binary input patters,

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} |x^m\rangle$$

The amplitude vector corresponding to this above state has entries $1/\sqrt{M}$ for basis states that are associated with a binary pattern from the dataset, and zero entries otherwise. Since the total number of amplitudes is much larger than the number of nonzero amplitudes M, basis encoded datasets generally give rise to sparse amplitude vectors.

5.1.1 Preparing Superpositions of Inputs

An elegant way to construct such 'data superpositions' in time linear in M and N has been introduced by Venture, Martinez and others, and will be summarized here as an example of basis encoded state preparation. The following is the circuit for one step in the routine

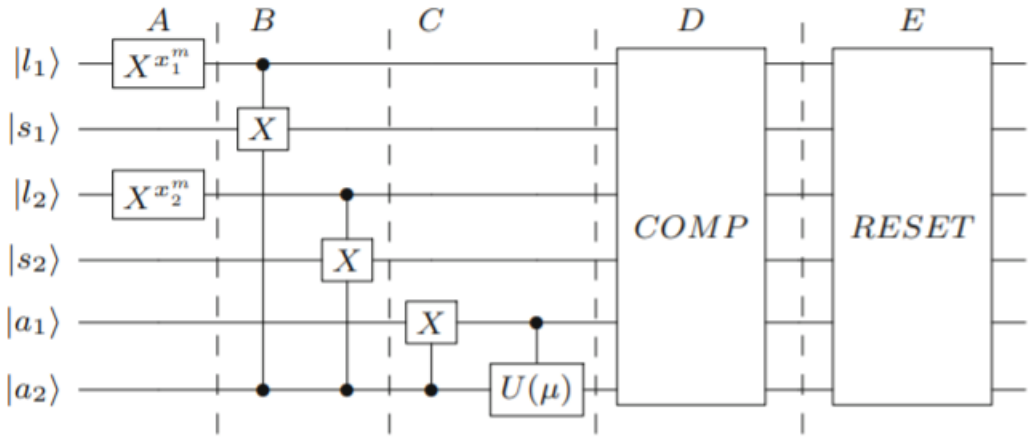

Figure 1: Taken from [1] p.142

We will simplify things by considering binary inputs in which every bit represents one feature, or $\tau = 1$. We require a quantum system

$$|l_1, \cdots, l_N; a_1, a_2; s_1, \cdots, s_N\rangle$$

with three registers; a **loading register** of N qubits, the **ancilla register** with two qubits and the N-qubit **storage register**. We start in the ground state and apply a Hadamard to the second ancilla qubit to get

$$\frac{1}{\sqrt{2}}\underbrace{|0,\cdots,0;0,0;0,\cdots,0\rangle}_{\text{MemoryBranch}}+\frac{1}{\sqrt{2}}\underbrace{|0,\cdots,0;0,1;0\cdots,0\rangle}_{\text{StorageBranch}}$$

The left term, flagged with $a_2=0$, is called the **memory branch**, while the right term, flagged with $a_2=1$ is the **processing branch**. The algorithm iteratively loads patterns into the loading register and 'breaks away' the right size of terms from the processing branch to add it to the memory branch. This way the superposition of patterns is 'grown' step by step.

To explain how one iteration works, assume that the first m training vectors have already been encoded after iterations $1,\cdots,m$ of the algorithm. This leads to the state

$$\left|\psi^{(m)}\right\rangle=\frac{1}{\sqrt{M}}\sum_{k=1}^{m}\underbrace{|0,\cdots,0;00;x_1^k,\cdots,x_N^k\rangle}_{\text{MemoryBranch}}+\sqrt{\frac{M-m}{M}}\underbrace{|0,\cdots,0;01;0\cdots,0\rangle}_{\text{StorageBranch}}$$

The memory branch stores the first m inputs in its storage register, while the storage register of the processing branch is in the ground state. In both branches the loading register is also in the ground state.

To execute the $(m+1)$-th step of the algorithm, write the $(m+1)$-th pattern $x^{m+1}=(x_1^{m+1},\cdots,x_N^{m+1})$ into the qubits of the loading register. This can be done by applying an X-gate to all qubits that correspond to nonzero bits in the input pattern. Next, in the processing branch the pattern get copied into the storage register using a CNOT-gate on each of the N qubits. To limit the execution to the processing branch we only have to control the CNOTs with the second ancilla being in $a_2=1$. This leads to

$$\frac{1}{\sqrt{M}}\sum_{k=1}^{m}\left|x_1^{m+1},\cdots,x_N^{m+1};00;x_1^k,\cdots,x_N^k\right\rangle+\sqrt{\frac{M-m}{M}}\left|x_1^{m+1},\cdots,x_N^{m+1};01;x_1^{m+1},\cdots,x_N^{m+1}\right\rangle$$

Using a CNOT gate, we flip $a_1=1$ if $a_2=1$, which is only true for the processing branch. Afterwards apply the single qubit unitary

$$U_{a_2}(\mu)=\begin{bmatrix}\sqrt{\frac{\mu-1}{\mu}} & \frac{1}{\sqrt{\mu}}\\ \frac{-1}{\sqrt{\mu}} & \sqrt{\frac{\mu-1}{\mu}}\end{bmatrix}$$

with $\mu=M+1-(m+1)$ to qubit a_2 but controlled by a_1. On the full quantum state, this operation amounts to

$$\mathbb{I}_{\text{loading}}\otimes c_{a_1}U_{a_2}(\mu)\otimes\mathbb{I}_{storage}$$

This splits the processing branch into two subbranches, one that can be 'added' to the memory branch and one that will remain the processing branch for the next step,

$$\begin{aligned}\frac{1}{\sqrt{M}}\sum_{k=1}^{m}\left|x_1^{m+1},\cdots,x_N^{m+1};00;x_1^k,\cdots,x_N^k\right\rangle+\frac{1}{\sqrt{M}}\left|x_1^{m+1},\cdots,x_N^{m+1};10;x_1^{m+1},\cdots,x_N^{m+1}\right\rangle\\ +\frac{\sqrt{M-(m+1)}}{\sqrt{M}}\left|x_1^{m+1},\cdots,x_N^{m+1};11;x_1^{m+1},\cdots x_N^{m+1}\right\rangle\end{aligned}$$

To add the subbranch marked by $|a_1a_2\rangle=|10\rangle$ to the memory branch, we have to flip a_1 back to 1. To confine this operation to the desired subbranch we can condition on an operation that compares if the loading and storage register are in the same state, and that $a_2=1$. Also, the storage register of the processing branch as well as the loading register of both branches has to be reset to the ground state by reversing the previous operations, before the next iteration begins. After the $(m+1)$-th iteration we start with a state similar to before but this time with $m\to m+1$. The routine requires $\mathcal{O}(MN)$ steps, and succeeds with certainty.

5.1.2 Computing in Basis Encoding

Acting on binary features encoded as qubits gives us the most computational freedom to design quantum algorithms. In principle, each operation on bits that we can execute on a classical computer can be done on a quantum computer as well.

Once data inputs have been encoded into a superposition like above, the data inputs can be processed in quantum parallel. For example, if a routine is known to implement a machine learning model f

$$\mathcal{A}(|x\rangle \otimes |0\rangle) \rightarrow |x\rangle \otimes |f(x)\rangle$$

and write the binary prediction $f(x)$ into the state of a qubit, we can perform inference in parallel on the data superposition

$$\mathcal{A}\Big(\frac{1}{\sqrt{M}} \sum_{m=1}^{M} |x^m\rangle \otimes |f(x^m)\rangle\Big)$$

From this superposition one can extract statistical information, for example by measuring the last qubit. Such a measurement reveals the expectation value of the prediction of the entire dataset.

5.1.3 Sampling from a Qubit

As the previous example shows, the result of a quantum machine learning model can be basis encoded as well. For binary classification this only requires one qubit. If the qubit is in the state $|f(x)\rangle = |1\rangle$ the prediction is 1 etc. A superposition can be interpreted as a probabilistic output that provides information on the uncertainty of the result.

In order to read out the state of the qubit we have to measure it, and we want to briefly address how to obtain the prediction estimate from measurements, as well as what number of measurements are needed for a reliable prediction. The field of reconstructing a quantum state from measurements is called *quantum tomography*, and there are very sophisticated ways in which samples from these measurements can be used to estimate the density matrix that describe the state. Here we consider a much simpler problem, namely to estimate the probability of measuring basis state $|0\rangle$ or $|1\rangle$. In other words, we are just interested in the diagonal elements of the density matrix, not in the entire state. Estimating the output of a quantum model in basis encoding is therefore a 'classical' rather than a 'quantum task'

Let the final state of the output qubit be given by the density matrix

$$\rho = \begin{bmatrix} \rho_{00} & \rho_{01} \\ \rho_{10} & \rho_{11} \end{bmatrix}$$

We need the density matrix formalism here because the quantum computer may be in a state where other qubits are entangled with the output qubit, and the single-qubit-state is therefore a mixed state. We assume that the quantum algorithm is error-free, so that repeating it always leads to previsely the same density matrix ρ to take measurements from. The diagonal elements ρ_{00} and ρ_{11} fulfil $\rho_{00} + \rho_{11} = 1$ and give us the probability of measuring the qubit in state $|0\rangle$ or $|1\rangle$ respectively. We associate ρ_{11} with the probabilistic output $f(x) = p$ which gives us the probability that model f predicts $y = 1$ for the input x.

To get an estimate $\hat{p}$ of p we have to repeat the entire algorithm S times and perform a computational basis measurement on the output qubit in each run. This produces a set of samples $\Omega = \{y_1, \cdots, y_S\}$ of outcomes, and we assume the samples stem from a distribution that returns 0 with probability $1 - p$ and 1 with probability p. Measuring a single qubit is therefore equivalent to sampling from a Bernoulli distribution.

An important question is how many samples from the single qubit measurement we need to estimate p with error ϵ. In physics language, we want an 'error bar' ϵ of our estimation $\hat{p} \pm \epsilon$, or the *confidence interval* $[\hat{p} - \epsilon, \hat{p} + \epsilon]$.

5.2 Amplitude Encoding

An entire branch of quantum machine learning algorithms encode the dataset into the amplitudes of a quantum state.

$$|\psi_{\mathcal{D}}\rangle = \sum_{m=1}^{M} \sum_{i=1}^{N} x_i^m |i\rangle |m\rangle = \sum_{m=1}^{M} |\psi_{x^m}\rangle |m\rangle$$

This quantum state has an amplitude vector of dimension NM that is constructed by concatenating all training inputs $\alpha = (x_1^1, \cdots, x_N^1, \cdots, x_1^M, \cdots, x_N^M)^T$. The dataset has to be normalized so that the absolute square of the amplitude vector is one. The training outputs can either be basis encoded in an extra qubit $|y^m\rangle$ entangled with the $|m\rangle$ register, or encoded in the amplitudes of a separate quantum register.

$$|\psi_y\rangle = \sum_{m=1}^{M} y^m |m\rangle$$

Amplitude encoding of datasets therefore requires the ability to prepare an arbitrary state

$$|\psi\rangle = \sum_i \alpha_i |i\rangle$$

both efficiently and robustly. The main advantage of amplitude encoding is that we only need $n = \log NM$ qubits to encode a dataset of M inputs with N features each.

If the quantum machine learning algorithm is polynomial in n, it has a logarithmic run-time dependency on the data set size. Promises of exponential speedups from qubit-efficient quantum machine learning algorithms sound strange to machine learning practitioners, because simple loading the NM features form the memory hardware takes time that is linear in NM. And indeed, the promises only hold if the state preparation can also be done qubit-efficiently. This is in fact possible in some cases that exhibit a lot of structure.

5.2.1 State Preparation in Linear Time

Given an n-qubit quantum computer, the theoretical lower bound of the depth of an arbitrary state preparation circuit is known to be $\frac{1}{n}2^n$. Current algorithms perform slightly worse with slightly less than 2^n parallel operations, most of which are expensive 2-qubit gates.

We'll take a look at one routine proposed by Möttönen et al. ([1] p.150). They consider the reverse problem, to map an arbitrary state to the ground state.

The basic idea is to control a rotation on qubit q_s by all possible states of the previous qubits $q_1, \cdots, q_{s-1}$ using sequences of so called *multi-controlled rotations*. In other words, we explicitly do a different rotation for each possible branch of the superposition, the one in which the previous qubits were in state $|0\cdots0\rangle$ to the branch in which they are in $|1\cdots1\rangle$. This is comparable to tossing $s-1$ coins and manipulating the sth coin differently depending on the measurement outcome via a look-up table.

A full sequence of multi-controlled rotations around vectors v^i with angles β_i consists of the successive applications of the 2^{s-1} gates

$$\begin{aligned}
c_{q_1=0}\cdots c_{q_{s-1}=0} R_{q_s}(v^1, \beta_1) |q_1\cdots q_{s-1}\rangle |q_s\rangle \\
c_{q_1=0}\cdots c_{q_{s-1}=0} R_{q_s}(v^2, \beta_2) |q_1\cdots q_{s-1}\rangle |q_s\rangle \\
\vdots \\
c_{q_1=0}\cdots c_{q_{s-1}=0} R_{q_s}(v^{2^{s-1}}, \beta_{2^{s-1}}) |q_1\cdots q_{s-1}\rangle |q_s\rangle
\end{aligned}$$

¨ For example, for $s = 3$ a full sequence consists of the gates

$$\begin{aligned}
c_{q_1=0}c_{q_2=0} R_{q_3}(v^1, \beta_1) |q_1q_2\rangle |q_3\rangle \\
c_{q_1=0}c_{q_2=1} R_{q_3}(v^2, \beta_2) |q_1q_2\rangle |q_3\rangle \\
c_{q_1=1}c_{q_2=0} R_{q_3}(v^3, \beta_3) |q_1q_2\rangle |q_3\rangle \\
c_{q_1=1}c_{q_2=1} R_{q_3}(v^4, \beta_4) |q_1q_2\rangle |q_3\rangle
\end{aligned}$$

¨ which rotate q_3 in a different way for all four branches of the superposition of q_1, q_2.

For a general quantum state one has to apply two *cascades* of such operations where each cascade is a sequence of multi-controlled rotations that run through all qubits q_1 to q_n. The first cascade uses R_Z rotations (*which fixes the rotation axis* v), and the second cascade applies R_Y rotations. Since we are usually interested in real amplitude vectors, we can neglect the first cascade.

The choice of rotation angles β is related to the amplitudes of the original state as follows

$$\beta_j^s = 2\arcsin\Big(\frac{\sqrt{\sum_{l=1}^{2^{s-1}} |\alpha_{(2j-1)2^{s-1}+l}|^2}}{\sqrt{\sum_{l=1}^{2^s} |\alpha_{(j-1)2^s+l}|^2}}\Big)$$

The above routine is always possible to use for amplitude-efficient state preparation, but it requires an exponential number of operations regarding the number of qubits. In the gate model of quantum computing, a number of proposals have been brought forward to prepare specific classes of states qubit-efficiently, or in $\log(MN)$.

Skipping notes on a few sections here.

5.2.2 Computing with Amplitudes

In contrast to basis encoding, amplitude encoding radically limits the type of computations we can perform. There are two general operations for the manipulation of amplitudes in the formalism of quantum theory: unitary transformations and measurements.

Unitary transformations are linear transformations of the amplitude vector that preserve its length. Even if we consider subsystems, we still get linear dynamics. A projective measurement is clearly a nonlinear operation, but its outcome is stochastic and if averaging over several runs it will reflect the distribution of amplitudes. But not all is lost: An important way to introduce nonlinearities on the level of amplitudes are post-selective measurements that we used in the branch selection scheme of the previous section. In a post-selective measurement of a single qubit, the state of the qubit is measured in the computational basis and the algorithm is only continued if this qubit is found in a particular state **ASK STIAN ABOUT THIS TOMORROW!**. This condition works like an 'if' statement in programming. Post-selection has the effect of setting the amplitudes in a branch of superpositions to zero, namely the branch that is entangled with the qubit in the state of an unsuccessful measurement. (**Note**: *in most cases we can push the post-selection to the end of the algorithm and reject final measurement outcomes if the result of said qubit is in the desired state*). A second, closely related idea is to use so called *repeat-until-success circuits*. Here the unsuccessful measurement does not result in repeating the entire algorithm, but prescribes a small correction which restores the state before the postselective measurement. Postselection can therefore be repeated until success is observed.

5.3 Qsample Encoding

Qsample encoding has been introduced to associate a real amplitude vector $(\sqrt{p_1}, \cdots, \sqrt{p_N})^T$ with a classical discrete probability distribution $p_1, \cdots, p_N$. This is in a sense a hybrid case of basis and amplitude encoding, since the information we are interested in is represented by amplitudes, but the N features are encoded in the qubits. An amplitude-efficient quantum algorithm is therefore exponential in the input dimension, while a qubit-efficient algorithm is polynomial in the input. For a given probability distribution, state preparation works the same way as in amplitude encoding. In some cases the distribution may be a discretization of a parametrized countinuous and efficiently integrable distribution $p(x)$, in which case the implicit Grover-Rudolph scheme can be applied.

5.3.1 Joining Distributions

When joining two quantum systems representing a qsample

$$|\psi_1\rangle = \sum_{i=1}^{2^n} \sqrt{u_i}\,|i\rangle$$

and

$$|\psi_2\rangle = \sum_{j=1}^{2^n} \sqrt{s_j}\,|j\rangle$$

the joint state of the total system is described as a tensor product of the two states

$$|\psi_1\rangle \otimes |\psi_2\rangle = \sum_{i,j}^{2^n} \sqrt{u_i s_j}\,|i\rangle\,|j\rangle$$

Sampling the binary string ij from this state is observed with probability $u_i s_j$ which is a product of the two original probabilities. In other words, the qsample of two joint qsample is a product distribution.

5.3.2 Marginalization

Given a qsample

$$|\psi\rangle = \sum_{i=1}^{2^n} \sqrt{p_i}\,|i\rangle$$

The mathematical operation of tracing over a qubit q_k corresponds to omitting the k-th qubit from the description, leading to a statistical ensemble over all possible states of that qubit. The resulting state is in general a mixed state.

To write down the effect of a trace operation is awkward and hides the simplicity of the concept, but we will attempt to do it anyway in order to show the equivalence to marginalization. Remember that the i-th computational basis state $|i\rangle$ represents a binary sequence, more precisely the binary representation of $i-1$

This is too much right now, come back later

5.4 Hamiltonian Encoding

While the approaches discussed so far encode features explicitly into quantum states we can choose an implicit encoding which uses the features to define the evolution of the quantum state. Instead of preparing a quantum state which contains the features or a distribution in its mathematical description, they now define an evolution applied to a state.

Hamiltonian encoding associates the Hamiltonian of a system with a matrix that represents the data, such as the design matrix X containing the feature vectors as rows, or the Gram matrix X^TX. We may have to use preprocessing tricks that make this matrix Hermitian. Similar to amplitude encoding, the general case will yield amplitude-efficient state preparation algorithms, while for some limited datasets we can get qubit-efficient schemes. This is most prominently the case for sparse data.

To use Hamiltonian encoding we need to be able to implement an evolution

$$|\psi'\rangle = e^{-iH_At}\,|\psi\rangle$$

on a quantum computer. The Hamiltonian H_A 'encodes' a Hermitian matrix A of the same dimensions, which means that the matrix representation of H_A is entry-wise equivalent to A. The initial quantum state $|\psi\rangle$ describes a system of n qubits, and one can think of the final state $|\psi'\rangle$ as a quantum state that now 'contains' the information encoded into the Hamiltonian.

The process of implementing a given Hamiltonian evolution on a quantum computer is called *Hamiltonian simulation*. Since we only consider time-independent Hamiltonians, any unitary transformation can be written as e^{-iH_At}. Hamiltonian simulation on a gate-based quantum computer is therefore very closely related to the question of how to decompose a unitary into quantum gates. However, an operator-valued exponential function is not trivial to evaluate, and given H there are more practical ways than computing the unitary matrix $U = e^{-iH_At}$ and decomposing it into gates.

5.4.1 Polynomial Time Hamiltonian Simulation

Consider a Hamiltonian H that can be decomposed into a sum of several 'elementary' Hamiltonians $H = \sum_{k=1}^{K} H_k$ so that each H_k is easy to simulate. For non-commuting H_k, we cannot apply the factorization rule for scalar exponentials, or

$$e^{-\sum_k H_k t} \neq \prod_k e^{-iH_k t}$$

An important idea introduced by Seth Lloyd in his 1996 seminal paper was to use the first-order Suzuki-Trotter formula instead

$$e^{-\sum_k H_k t} = \prod_k e^{-iH_k t} + \mathcal{O}(t^2)$$

The Trotter formula states that for small t the factorization rule is approximately valid. This can be leveraged when we write the evolution of H for time t as a sequence of small time-steps of length Δt

$$e^{-iHt} = (e^{-iH\Delta t})^{\frac{1}{\Delta t}}$$

While for the left side, the Trotter formula has a large error, for each small time step Δt the error becomes negligible. Of course, there is a trade-off: The smaller Δt, the more often the sequence has to be repeated. But overall, we have a way to simulate H by simulating the terms H_k.

5.4.2 Qubit-Efficient Simulation of Hamiltonians

There are special classes of Hamiltonians that can be simulated in time that is logarithmic in their dimension. If the Hamiltonian encodes a data matrix, this means that the runtime is also logarithmic in the dimension of the dataset. The most prominent example are Hamiltonians that only act on a constant number of qubits, so called *strictly local Hamiltonians*. This is not surprising the we remember that a local Hamiltonian can be expressed by a constant number of terms constructed from Pauli operators.

More generally, it has been shown that *sparse* Hamiltonians can be simulated qubit-efficiently.

5.4.3 Density Matrix Exponentiation

There are special conditions under in which we can guarantee qubit-efficiency for densely populated Hamiltonians. The most important in the context of quantum machine learning is the technique of *density matrix exponentiation*. This technique can be used for more general amplitude-efficient simulations, but becomes qubit-efficient for low-rank Hamiltonians. Instead of querying an oracle, the data is initially encoded in a density matrix, for which we can apply techniques from previous sections. Although density matrix exponentiation relies on many technical details and its understanding requires a high level of quantum computing expertise, we want to try to briefly sketch the outline of the idea. This section is consequently suited for more advanced readers.

The goal of density matrix exponentiation is to simulate a Hamiltonian $e^{iH_\rho t}$ that encodes a non-sparse density matrix ρ and apply it to a quantum state σ. This is only possible because both operators are Hermitian, and every density matrix has an entry-wise equivalent Hamiltonian. It turns out that simulating H_ρ is approximately equivalent to simulating a swap operator S, applying it to the state $\rho \otimes \sigma$ and taking a trace operation. A swap operator is sparse and its simulation therefore efficient. It also means that whatever data we can encode in the entries of a density matrix, we can approximately encode into a Hamiltonian

$$\begin{aligned}\mathrm{Tr}_2\{e^{-iS\Delta t}(\sigma\otimes\rho)e^{iS\Delta t}\} &= \sigma - i\Delta t[\rho,\sigma] + \mathcal{O}(\Delta t^2)\\ &\simeq e^{-\rho\Delta t}\sigma e^{i\rho\Delta t}\end{aligned}$$

Note that $\sigma - i\Delta[\rho,\sigma]$ are the first terms of the exponential series $e^{-i\rho\Delta t}\sigma e^{i\rho\Delta t}$. In other words, simulating the swap operator that 'exchanges' the state of the qubits of state ρ,σ for a short time Δt, and taking the trace over the second quantum system results in an effective dynamic as if exponentiating the second density matrix.

To **prove** the relation, consider two general mixed states

$$\begin{aligned}\rho &= \sum_{i,i'=1}^{N} a_{ii'}\,|i\rangle\langle i'|\\ \sigma &= \sum_{j,j'=1}^{N} b_{jj'}\,|j\rangle\langle j'|\end{aligned}$$

where $\{|i\rangle, |j\rangle\}$ are the computational bases in the Hilbert spaces of the respective states, which have the same dimension. Now write the operator-valued exponential functions as a series

$$\mathrm{Tr}_2\{e^{-iS\Delta t}(\sigma\otimes\rho)e^{iS\Delta t}\} = \mathrm{Tr}_2\Big\{\sum_{k,k'=0}^{\infty}\frac{(-i)^k\Delta t^k i^{k'}\Delta t^{k'}}{k!k'!}S^k(\sigma\otimes\rho)S^{k'}\Big\}$$

and apply the swap operators. Since $S^2 = 1$, higher order terms in the series only differ in the prefactor, and for $\Delta t << 1$ the higher orders quickly vanish. We therefore only write the first few terms explicitly and summarize the vanishing tail of the series in a $\mathcal{O}(\Delta t^2)$ term,

$$\begin{aligned}\mathrm{Tr}_2\Big\{\Big(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}\,|j\rangle\langle j'|\otimes|i\rangle\langle i'|\Big) + i\Delta t\Big(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}\,|j\rangle\langle j'|\otimes|i\rangle\langle i'|\Big)S\\ - i\Delta t S\Big(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}\,|j\rangle\langle j'|\otimes|i\rangle\langle i'|\Big) + \mathcal{O}(\Delta t^2)\Big\}\end{aligned}$$

Next, apply the swap operator and the trace operation. Tracing out the second system means to 'sandwich' the entire expression by $\sum_k \langle k|\cdot|k\rangle$, where $\{|k\rangle\}$ is a full set of computational basis vectors in the Hilbert space of the second system. We get

$$\sum_{jj'}b_{jj'}\,|j\rangle\langle j'| + i\Delta t\sum_{ij}\sum_{k}a_{ki}b_{jk}\,|j\rangle\langle i| - i\Delta t\sum_{ij}\sum_{k}a_{ik}b_{kj}\,|i\rangle\langle j| + \mathcal{O}(\Delta t^2)$$

Which is in fact the same as $\sigma - i\Delta t(\sigma\rho - \rho\sigma) + \mathcal{O}(\Delta t^2)$

Density matrix exponentiation is commonly used in combination with phase estimation. Once a density matrix containing data is exponentiated, we can 'apply' it to some amplitude encoded state and extract eigenvalues of ρ through a phase estimation routine using the inverse quantum Fourier transform.

Density matrix exponentiation is qubit-efficient for low-rank matrices, given that the state ρ can be prepared qubit-efficiently. To see this, we first have to note that the desired accuracy is not necessarily constant, but depends on the size of H_ρ, especially when we are interested in the eigenvalues of ρ.

6 Chapter 6 Notes: Quantum Computing for Inference

After the discussion of classical-quantum interfaces, we are now ready to dive into techniques that can be used to construct quantum machine learning algorithms. As laid out in the introduction, there are two strategies to solve learning tasks with quantum computers. First, one can try to translate a classical machine learning method into the language of quantum computing. The challenge here is to combine quantum routines in a clever way so that the overall quantum algorithm reproduces the results of the classical model. The second strategy is more exploratory and creates new models that are tailor-made for the working principles of a quantum device. Here the numerical analysis of the model performance can be more more important than theoretical promises of asymptotic speedups.

This chapter will focus on the first approach and present building blocks for quantum algorithms for *inference*, which we understand as algorithms that implement an input-output map $y = f(x)$ or a probability distribution $p(x|y)$ of a machine learning model. We will introduce a number of techniques, tricks, subroutines and concepts that are commonly used as building blocks in different branches of the quantum machine learning literature.

6.1 Linear Models

A linear model is a parametrized model function mapping N-dimensional inputs $x = (x_1, \cdots, x_N)^T$ to K-dimensional outputs $y = (y_1, \cdots, y_K)^T$. The function is linear in a set of weight parameters that can be written as a $K \times N-$dimensional weight matrix W

$$f(x; W) = Wx$$

The output can be the multi-dimensional result of a regression task, or it can be interpreted as a one-hot encoding to a multi-label classification task. When the output is a scalar, we have that $K = 1$ and the linear model becomes an inner product between the input and a weight vector $w = (w_1, \cdots, w_N)^T$

$$f(x; w) = w^T x + w_0 = w_0 + w_1 x_1 + \cdots + w_N x_N$$

As mentioned before, it is common to include the bias w_0 in the inner product by extending the parameter vector and considering the extended input $x = (1, x_1, \cdots, x_N)^T$

6.2 Inner Products with Interference Circuits

A natural way to represent a scalar-valued linear model on a quantum computer is to encode the inputs x and parameters w in quantum state vectors via amplitude encoding and compute their inner product

$$\langle \psi_w | \psi_x \rangle = f(x; w) = w^T x$$

However, while playing a prominent role in the mathematical description of quantum theory, it is actually not trivial to measure the inner product between two quantum states. The most well-known of the inner product interference routines is the so called *swap test* and returns the absolute value of the inner product of two separate quantum states.

6.2.1 The Swap Test

We consider two qubit registers in the respective quantum states $|\psi_a\rangle, |\psi_b\rangle$ that encode the real vectors a and b. The swap test is a common trick to *write* the absolute square of their inner product $|\langle \psi_a | \psi_b \rangle|^2$ into the probability of measuring an ancilla qubit in a certain state. To achieve this, one creates a superposition of the ancilla qubit and swaps the quantum states in one branch of the superposition, after which they are interfered.

Starting with a state $|0\psi_a\psi_b\rangle$ a Hadamard on the ancilla qubit leads to

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\,|\psi_a\psi_b\rangle$$

We now apply a Swap operator on the two registers ψ_a, ψ_b which is conditional on the ancilla in state $|1\rangle$. This operation swaps the states $\psi_a\psi_b \to \psi_b\psi_a$ in the branch marked by the first qubit being in state $|1\rangle$

$$\frac{1}{\sqrt{2}}(|0\psi_a\psi_b\rangle + |1\psi_b\psi_a\rangle)$$

Another Hadamard applied to the ancilla results in the state

$$\frac{1}{2}|0\rangle \otimes (|\psi_a\psi_b\rangle + |\psi_b\psi_a\rangle) + \frac{1}{2}|0\rangle \otimes (|\psi_a\psi_b\rangle - |\psi_b\psi_a\rangle)$$

This computes two branches of a superposition, one containing a sum between the 'unswapped' and 'swapped' states of the two registers, and the other containing their difference.

The probability of measuring the ancilla qubit in state $|0\rangle$, i.e. the acceptance probability is

$$p_0 = |\langle 0|\psi\rangle|^2 = \frac{1}{2} - \frac{1}{2}|\langle\psi_a|\psi_b\rangle|^2$$

and reveals the absolute value of the inner product through

$$p_0 = \frac{1}{2} - \frac{1}{2}\mathrm{Tr}\{\rho_a\rho_b\}$$

Note that here the expression $\rho_a\rho_b$ is not an abbreviation of the tensor product, but a matrix product.

With a little trick in the way information is encoded into the amplitudes it is possible to reveal the sign of the inner product using the swap test. One simply has to extend the vectors $a \to a'$ and $b \to b'$ by one extra dimension $N+1$, $a', b' \in \mathbb{R}^{N+1}$. The entry of the extra dimension is set to the constant value 1. To renormalize we then have to multiply the entire $N+1$-dimensional vector with $\frac{1}{\sqrt{2}}$.

With the extra constant dimension, the result of the swap test between $|\psi_{a'}\rangle, |\psi_{b'}\rangle$ will be

$$\begin{aligned} p(0) &= \frac{1}{2} - \frac{1}{2}|\langle\psi_{a'}|\psi_{b'}\rangle|^2 \\ &= \frac{1}{2} - \frac{1}{2}|\frac{1}{2}a_1b_1 + \cdots + \frac{1}{2}a_Nb_N + \frac{1}{2}|^2 \\ &= \frac{1}{2} - \frac{1}{2}|\frac{1}{2}a^Tb + \frac{1}{2}|^2 \end{aligned}$$

Since $a^Tb \in [-1,1]$ the expression $|\frac{1}{2}a^Tb + \frac{1}{2}|$ is guaranteed to lie in the positive interval $[0,1]$. Hence, we can extract the inner product of the original vectors via

$$a^Tb = 2\sqrt{1-2p_0} - 1$$

and since $p_0 \in [+, \frac{1}{2}]$, this value does indeed lie in the interval $[-1,1]$.

6.3 A Quantum Circuit as a Linear Model

We still deal with inputs in amplitude encoding $|\psi_x\rangle$, but now look at the case where the model parameters w are 'dynamically encoded' in a unitary, or more general, in a quantum circuit.

6.3.1 Unitary Linear Model

Rather trivially, a linear unitary transformation can be interpreted as a special linear model

$$|\psi_{x'}\rangle = U|\psi_x\rangle$$

that maps a N-dimensional complex vector onto a N-dimensional complex vector.

6.3.2 Non-Unitary Linear Model

An interesting twist appears when we look at linear models that map from $\mathbb{R}^N$ to a lower-dimensional vector space $\mathbb{R}^K, K < N$ to implement the layer of a neural net. If we want to allow for arbitrary concatenation of linear layers, the input encoding of the quantum algorithm has to be of the same type as the output encoding. If we also want to reduce the dimensionality in the map, we have to consider a subsystem of the qubits as the output of the evolution. This necessitates the use of the density matrix formulation.

Consider a density matrix ρ_x that encodes an input x on its diagonal, i.e. $\mathrm{diag}\{\rho_x\} = (x_1, \cdots, x_N)^T$. For example, we could encode the square roots of the features in amplitude encoding in a $n = \log N$ qubit quantum

system $\left|\psi_{\sqrt{x}}\right\rangle = \sum_i \sqrt{x_i}\left|i\right\rangle$, and computing the density matrix of this pure state $\rho_x = \left|\psi_{\sqrt{x}}\right\rangle\left\langle\psi_{\sqrt{x}}\right|$ results in the desired encoding. A unitary evolution in this language is formally written as

$$\rho_{x'} = U\rho_x U^\dagger$$

We then trace out the first $1, \cdots, n-k$ of the n qubits so that we end up with only the last k qubits, reducing the Hilbert space from $N = \log n$ dimensions to $K = \log k$ dimensions. This yields the 'hidden layer density matrix'

$$\rho_h = \mathrm{Tr}_{1,\cdots,n-k}\{\rho_{x'}\}$$

We are not restricted to reducing the dimension, we can add additional zero initialized qubits and extending the further evolution to these new qubits to increase the dimension of the Hilbert space again.

6.4 Nonlinear Activations

A linear model is not very powerful when it comes to data that is not linearly separable. We therefore need nonlinearities in the model function. To focus the discussion, we will look at how to introduce nonlinearities from the perspective of neural networks where they enter as nonlinear activation functions φ mapping a net input v to $\varphi(v)$.

6.4.1 Amplitude Encoding Case

Nonlinear transformations of amplitudes are very likely to be unphysical. However, there is one element in quantum theory that allows for nonlinear updates of the amplitudes: measurement. Measurement outcomes are to begin with non-deterministic, however we can use branch selection to make them deterministic.

In the next section we will look at an alternative strategy to give amplitude encoding more power, namely through so called *kernel methods*.

6.4.2 Basis Encoding Case

In basis encoding, nonlinear transformations of the form

$$\left|v\right\rangle\left|0\right\rangle \to \left|v\right\rangle\left|\varphi(v)\right\rangle$$

can be implemented deterministically. One can always take a classical algorithm to compute φ on the level of logical gates, translate it into a reversible routine and use this as a quantum algorithm. Depending on the desired non-linearity, there may be much more efficient options.

6.4.3 Angle Encoding

Before finishing this section we want to have a look at a third type of encoding. Assume that by some previous operations, the net input $\theta = w_0 + w_1x_1 + \cdots + w_Nx_N$ is written into the angle of an ancilla or 'net input qubit', which is entangled with an output qubit in some arbitrary state $\left|\psi\right\rangle$

$$R_y(2v)\left|0\right\rangle \otimes \left|\psi\right\rangle_{\mathrm{out}}$$

As a reminder, the rotation about the y axis is defined as

$$R_y(2v) = \begin{bmatrix} \cos v & -\sin v \\ \sin v & \cos v \end{bmatrix}$$

The net input qubit is thus rotated around the y axis by an angle v, and $R_y(2v)\left|0\right\rangle = \cos v\left|0\right\rangle + \sin v\left|1\right\rangle$. The goal is now to rotate the output qubit by a nonlinear activation φ which depends on v. One way to accomplish this task is the *repeat-until-success* circuits.

6.5 Kernel Methods

In this section we will show that the feature spaces of **kernel methods** and quantum Hilbert spaces have a similar mathematical formalism and how this can be used for quantum machine learning.

One can understand a positive semi-definite kernel as a distance measure on the input space, and use it to compute the distance between each training input and the new input we aim to classify, and favour the class of 'closer' training data when the decision for the prediction is taken. Such a kernel corresponds to an inner product of data points mapped to a higher dimensional *feature space*. The *representer theorem* shows how a large class of trained models can be expressed in terms of kernels. Finally, the *kernel trick* allows us to construct a variety of models that are formulated in terms of a kernel function by replacing it with another kernel function.

The following section will review these concepts with more mathematical foundation, after which we show how to understand state preparation and information encoding as a 'quantum feature map'. Finally, we discuss how to compute kernels or distance measures on input space of a quantum computer and show how to construct density matrices that correspond to a kernel Gram matrix.

6.5.1 Kernels and Feature Maps

Let be a non-empty set of patterns or inputs, as a reminder a positive semi-definite kernel was defined as a map $\kappa : \times \to \mathbb{C}$ where for any subset of the input set $x_1, \cdots, x_M \in$ with $M \geq 2$ the Gram matrix with entries

$$K_{ij} = \kappa(x_i, x_j)$$

is positive definite. Kernels fulfil $\kappa(x, x) \geq 0$ and $\kappa(x, x') = \kappa(x', x)^*$

We can always construct a feature map from a kernel. A feature map was introduced before as a map $\phi :\to \mathcal{F}$ from the input space into a feature space. Previously we looked at some examples where the feature space was a real vector space $\mathbb{R}^K, K > N$ for example the map from

$$(x_1, x_2)^T \to (x_1, x_2, x_1^2 + x_2^2)^T$$

The feature map that can always be constructed from a kernel κ maps from the input set to complex-valued functions on the input set

$$\phi :\to \mathbb{C}^, \qquad x \to \kappa(., x)$$

Here, the feature vectors in $\mathbb{C}$ are themselves functions that map from the input space to the space of complex numbers $\to \mathbb{C}$. The functions in feature space are kernels with one 'open slot'

From this feature map we can construct a feature space that is a vector space with an inner product. The vector space contains linear combinations of $\{\kappa(., x_m)\}$, where $\{x^1, \cdots, x^M\} \subseteq$ is an arbitrary set of inputs

$$f(\cdot) = \sum_{m=1}^{M} \nu_m \kappa(., x_m)$$

with coefficients $\nu_m \in \mathbb{R}$. Using a second vector of the same form

$$g(\cdot) = \sum_{m'=1}^{M'} \mu_{m'} \kappa(., x_{m'})$$

The inner product can be defined as

$$\langle f, g \rangle = \sum_{m=1}^{M} \sum_{m'=1}^{M'} \nu_m^* \mu_{m'} \kappa(x_m, x_{m'})$$

With this inner product, the feature map has the property

$$\langle \phi(x) | \phi(x') \rangle = \langle \kappa(\cdot, x), \kappa(\cdot, x') | = \kappa(x, x')$$

Read from the right to left, this relationship is a core idea of the theory of kernel methods: A kernel of two inputs x and x' computes the inner product of the same inputs mapped to a feature space. As a mere technicality in this context, one can extend the inner product space by the norm $||f|| = \sqrt{\langle \cdot, \cdot \rangle}$ and the limit points of Cauchy series under this norm, and get a Hilbert space called the *Reproducing Kernel Hilbert Space*.

One can also argue vice versa and construct a kernel from a vector space with an inner product. It follows from the above that given a mapping $\phi_\to \mathcal{H}$, one can define a kernel via

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$$

6.5.2 The Representer Theorem

Kernel methods use models for machine learning that are based on kernel functions that measure distances between data inputs. They are not simple a specific ansatz for a model. The wide scope of kernel methods is revealed by the *representer theorem*.

Consider an input domain , a kernel $\kappa : \times \to \mathbb{R}$, a data set $\mathcal{D}$ consisting of data pairs $(x^m, y^m) \in \times\mathbb{R}$ and a class of model functions $f :\to \mathbb{R}$ that can be written in the form

$$f(x) = \sum_{l=1}^{\infty} \mu_l \kappa(x, x^l)$$

with $x, x^l \in$ and $\mu_l \in \mathbb{R}$ and $||f|| < \infty$. Furthermore, assume a cost function $C : \mathcal{D} \to \mathbb{R}$ that quantifies the quality of a model by comparing predicted outputs $f(x^m)$ with targets y^m, and which has a regularization term of the form $g(||f||)$ where $g : [0, \infty) \to \mathbb{R}$ is a strictly monotonically increasing function.

The representer theorem says that any function f minimizing the cost function C can be written as

$$f(x) = \sum_{m=1}^{M} \nu_m \kappa(x, x^m)$$

Kernelized Linear Model Take a standard linear model function $f(x) = w^T x$. If we choose $= \mathbb{R}^N, \mu_l = w_l$ for $l = 1, \cdots, N$ and $\mu_l = 0$ for $l > N$, as well as $x_l = \hat{e}_l$ to be the standard basis and κ to be an inner product kernel:

$$\sum_{l=1}^{\infty} \mu_l \kappa(x, x_i) = \sum_{i=1}^{N} w_i x^T \hat{e}_i = w^T x$$

Considering a square loss objective with l_1 regularization and $g(||f||) = ||f||$ with a standard norm, we can apply the representer theorem. According to the theorem, the optimal linear model with regards to the objective can be expressed as

$$f(x) = \sum_{m} \nu_m \kappa(x, x^m)$$

If we choose $\kappa(x, x^m) = x^T x^m$ (a linear kernel), this means that the weight vector har effectively been expanded in terms of the training inputs, $w = \sum_m \nu_m x^m$. In short, the optimal weight vector lies in the subspace spanned by the data. The kernelized version of a model is now amenable to the kernel trick. By exchanging κ for another kernel κ', we can change the feature space in which the data gets effectively mapped.

6.5.3 Quantum Kernels

Quantum States - just like feature vectors - also live in Hilbert spaces and allow for a very natural definition of a *quantum kernel*. Input encoding maps x to a vector in a Hilbert space. The main ingredient of a quantum kernel is simply to interpret the process of encoding an input $x \in$ into a quantum state $|\phi(x)\rangle$ as a feature map.

If the quantum state $|\psi(x)\rangle$ is interpreted as a feature vector in amplitude encoding (which we call a *quantum feature state*), the inner product of two such quantum feature states can be associated with a quantum kernel

$$\kappa(x, x') = \langle \phi(x) | \phi(x') \rangle$$

Hence, an inner product of quantum states produced by an input encoding routine is always a kernel. Any quantum computer that can compute such an inner product can therefore be used to estimate the kernel, and the estimates can be fed into a classical kernel method. If the result of this inner product is impossible to simulate on a classical computer, the kernel intractable. A challenge is therefore to find quantum kernels for which we have a quantum advantage and which at the same time prove interesting for machine learning purposes.

The quantum kernel itself depends solely on the input encoding routine, which is the state preparation circuit that writes the input into the quantum state $|\phi(x)\rangle$. Different encoding strategies give rise to different kernels. To illustrate this, we look at five different input encoding strategies and present their associated quantum kernel.

Basis Encoding . If the input patterns x are binary strings of length n with integer representation i, and we chose basis encoding, the feature map maps the binary string to a computational basis state

$$\phi : i \to |i\rangle$$

The computational basis state corresponds to a standard basis vector in a 2^n-dimensional Hilbert space. The associated quantum kernel is given by the Kronecker delta

$$\kappa(i, j) = \langle i|j\rangle = \delta_{ij}$$

which is of course a very strict similarity measure on the input space, since it only yields a nonzero value for inputs that are exactly the same.

Amplitude Encoding . If the input patterns $x \in \mathbb{R}^N$ are real vectors of length $N = 2^n$ and we chose amplitude encoding, we get the feature map

$$\phi : x \to |\psi_x\rangle$$

If the input x is normalized to one, its dimension is a power of 2, $n = \log_2 N$ and we only use the real subspace of the complex vector space, this map is simply the identity. The kernel is the inner product

$$\kappa(x, x') = \langle \psi_x|\psi_{x'}\rangle = x^T x'$$

which is also known as a linear kernel.

Copies of Quantum States. We can map an input $x \in \mathbb{R}^N$ to copies of an amplitude encoded quantum state

$$\phi : x \to |\psi_x\rangle \otimes \cdots \otimes |\psi_x\rangle$$

and we get the homogeneous polynomial kernel

$$\kappa(x, x') = \langle \psi_x|\psi_{x'}\rangle \otimes \cdots \otimes \langle \psi_x|\psi_{x'}\rangle = (x^T x')^d$$

If the original inputs are extended by some constant features, for example when padding the input to reach the next power of 2, the constant features can play a similar role to the offset c of a general polynomial kernel $\kappa(x, x') = (x^t x + c)^d$

Angle Encoding . Given $x = (x_1, \cdots, x_N)^T \in \mathbb{R}^N$ once more, we can encode one feature per qubit. To recap, the feature x_i is encoded in a qubit as $|q(x_i)\rangle = \cos(x_i)\,|0\rangle + \sin(x_i)\,|1\rangle$. We get the feature map

$$\phi : x \to \begin{bmatrix}\cos x_1 \\ \sin x_1\end{bmatrix} \otimes \cdots \otimes \begin{bmatrix}\cos x_N \\ \sin x_N\end{bmatrix}$$

and the corresponding kernel is a cosine kernel

$$\begin{aligned}\kappa(x, x') &= \begin{bmatrix}\cos x_1 \\ \sin x_1\end{bmatrix}^T \begin{bmatrix}\cos x'_1 \\ \sin x'_1\end{bmatrix} \otimes \cdots \otimes \begin{bmatrix}\cos x_N \\ \sin x_N\end{bmatrix}^T \begin{bmatrix}\cos x'_N \\ \sin x'_N\end{bmatrix} \\ &= \Pi_{i=1}^N (\sin x_i \sin x'_i + \cos x_i \cos x'_i) \\ &= \Pi_{i=1}^N \cos(x_i - x'_i)\end{aligned}$$

Coherent States . *Coherent states* can be used to explicitly compute so called *radial basis functions kernels*

$$\kappa(x, x') = e^{-\delta|x-x'|^2}$$

for the case $\delta = 1$.

Coherent states are known in the field of quantum optics as a description of light modes. Formally, they are superpositions of so called *Fock states*, which are basis states from an infinite-dimensional basis $\{|0\rangle, |1\rangle, |2\rangle, \cdots\}$

A coherent state has the form

$$|\alpha\rangle = e^{-\frac{|\alpha|^2}{2}} \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle$$

for $\alpha \in \mathbb{C}$. Encoding a real scalar input $c \in \mathbb{R}$ into a coherent state $|\alpha = c\rangle = |\alpha_c\rangle$ induces a feature map to an infinite dimensional space

$$\phi : c \to |\alpha_c\rangle$$

We can encode a real vector $x = (x_1, \cdots, x_N)^T$ in N joint coherent states

$$|\alpha_x\rangle = |\alpha_{x_1}\rangle \otimes \cdots \otimes |\alpha_{x_N}\rangle$$

For two coherent states α_x, $|\alpha_{x'}\rangle$, the kernel corresponding to this feature map is

$$\kappa(x, x') = e^{-\left(\frac{|x|^2}{2} + \frac{|x'|^2}{2} - x^\dagger x'\right)}$$

and its absolute value is given by

$$|\kappa(x, x')| = |\langle \alpha_x | \alpha_{x'} \rangle| = e^{-\frac{1}{2}|x - x'|^2}$$

Since products of kernels are also kernels, we can construct a new kernel

$$\tilde{\kappa}(x, x') = \kappa(x, x') \cdot \kappa(x, x')^*$$

which is the desired basis function or Gaussian kernel.

According to the generalized definition of coherent states, one of their characteristics is that inner products of basis states are not orthogonal, which means that their inner product is not zero. Generalized coherent states therefore allow the feature map to map inputs to basis states, while still producing more interesting kernels than a simple delta function as in the basis encoding example above. It is interesting to note that one cannot only use the quantum computer to estimate the kernel function, but apply a quantum circuit to state $|\phi(x)\rangle$ to process the input in the 'feature Hilbert space'. For example, a trainable or *variational circuit* can learn how to process the feature quantum state and compute a prediction from it. Such an approach has been termed the *explicit approach* to build a quantum classifier from feature maps, since it explicitly computes in feature space instead of implicitly using kernel functions.

7 Chapter 7 Notes: Quantum Computing for Training

The previous chapter looked into strategies of implementing inference algorithms on a quantum computer, or how to compute the prediction of a model using a quantum instead of a classical device. This chapter will be concerned with how to optimize models using quantum computers, a subject targeted by a large share of the quantum machine learning literature.

We will look at four different approaches to optimization in quantum machine learning: linear algebra calculus, search, hybrid routines for gradient descent and adiabatic methods.

7.1 Quantum B.L.A.S.

The quantum computing community developed a rich collection of quantum algorithms for **basic linear algebra subroutines** or *quantum blas*, in analogy to the linear algebra libraries of various programming platforms. Quantum blas include routines such a matrix multiplication, matrix inversion and singular value decomposition. These can be used to solve optimization problems, and heavily rely on the idea of amplitude encoding. The quantum machine learning algorithms based on quantum blas are rather technical combinations of the subroutines introduced in previous chapters.

7.2 Basic Idea

A number of learning algorithms contain linear algebra routines such as inverting a matrix or finding a matrix' eigenvalues and eigenvectors. The matrices are usually constructed from the training set, and therefore grow with the dimension N and/or number of training vectors M. The basic idea of the linear algebra approach in quantum machine learning is to use quantum systems for linear algebra calculus, where the design matrix is represented by the Hamiltonian of the system via dynamic encoding.

Quantum systems are natural eigendecomposers in the sense that the result of measurements are eigenvalues of operators to certain eigenstates. One can say that the dynamics of the quantum system *emulates* a linear algebra calculation. This can be used for matrix inversion as demonstrated in the algorithm for linear systems by Harrow,

Hassidim and Lloyd. In this algorithm one prepares a special quantum system whose evolution is defined by the matrix that is to be inverted, and the physical evolution allows us to invert the eigenvalues of that matrix and read out some desired information via measurements.

The crux of this idea is the same as in amplitude encoding. Many quantum blas require a number of manipulations that is polynomial in the number n of qubits, which means that they are qubit-efficient. Algorithms composed of such subroutines depend only logarithmic on the number $N = 2^n$ of amplitudes, and the same usually applies to the number of training vectors M. A logarithmic dependency is a significant speedup.

For example: a training set of one billion vectors that each have dimension one million can be represented in a Hamiltonian (*using the previous trick of extending the rectangular design matrix to a* $2bn \times 2bn$ *Hermitian matrix*) of a $n = 31$ qubit system.

Of course, there has to be a caveat: similar to state preparation, simulating such a Hamiltonian in general may take of the order of 4bn operations. But we saw interesting exceptions when the Hamiltonian is sparse or low-rank. This obviously poses restrictions on the data one can deal with, and only little has been done to find out which datasets could fulfil the requirements of qubit-efficient processing. Still, the promises of simulating linear algebra calculus with quantum systems for big data applications are impressive, and therefore worth investigating.

7.3 Matrix Inversion for Training

This section presents some selected examples of classical machine learning algorithms that rely on eigenvalue decomposition or matrix inversion, and for which quantum algorithms have been proposed.

Classical Method	Computation	Strategy
Data Matrix Inversion		
Linear Regression (sparse)	$(X^\dagger X)X^\dagger y$	HHL^-, HHL
Linear Regression (low-rank)	$\sum_{r=1}^{R} \sigma_r^{-1} u_r v_r^T y$	HHL^{DME}
Kernel Matrix Inversion		
Support Vector Machine	$K^{-1}y$	HHL^{DME}
Gaussian Process	$\kappa^T K^{-1}$ and $\kappa^T K^{-1}\kappa$	HHL + SWAP
Adjacency Matrix Inversion		
Hopfield Network		HHL + HHL^{DME}

Table 4: Simplified overview of the quantum blas based quantum machine learning algorithms presented in this chapter. The design matrix X is composed of the training inputs, and y is a vector of training outputs. u_r, v_r, σ_r are singular vectors / values of $X^T X$ and K is a kernel matrix. Abbreviations: HHL - quantum matrix inversion routine, HHL^- - quantum matrix multiplication routine (omitting the inversion of the eigenvalues), HHL^{DME} - quantum matrix inversion routine with density matrix exponentiation to simulate the required Hamiltonian, SWAP - interference circuit to evaluate inner products of quantum states.

7.3.1 Inverting Data Matrices

The first suggestion to use HHL's quantum matrix inversion technique for statistical data analysis was proposed by Wiebe, Braun and LLoyd. Closely related to machine learning their goal was to 'quantize' linear regression for data fitting, in which the best model parameters of a linear function for some data points have to be found. It is known that basic linear regression reduces to finding a solution to

$$w = (X^T X)^{-1} X^T y$$

where w is a N-dimensional vector containing the model parameters, y is a M-dimensional vector containing the target outputs from the dataset, and the rows of the data matrix X are the N-dimensional training inputs. The most demanding computational problem behind linear regression is the inversion of the square of the data matrix $(X^T X)^{-1}$

As a rough reminder, the HHL routine solves a linear system of equations $Az = b$ by amplitude encoding b into a quantum state and applying the evolution $e^{iH_A t} |\psi_b\rangle$, where A is encoded into the Hamiltonian. Quantum phase

estimation can extract the eigenvalues of H_A and store them in basis encoding, and some quantum post-processing writes them as amplitudes and inverts them, which effectively computes the quantum state $|\psi_{A^{-1}b}\rangle$. Omitting the inversion step multiplies the matrix A as it is $|\psi_{Ab}\rangle$. Therefore, we have to apply HHL twice to perform linear regression as it consists of both a matrix inversion and a multiplication.

The runtime of the routine grows with the condition number κ of the data matrix with $\mathcal{O}(\kappa^6)$, where the Hamiltonian simulation as well as the conditional measurement in the branch selection are each responsible for a term $\mathcal{O}(\kappa^3)$. Hamiltonian simulation furthermore contributes a linear dependency on the sparsity s of the data matrix. The runtime grows with the inverse of the error as $\mathcal{O}(1/\log\epsilon)$ when modern Hamiltonian simulation methods are used. Reading out the parameters through measurements requires a number of repetitions that is of the order N.

A slight variation on the linear regression algorithm allows us to replace the sparsity condition with the requirement that the design matrix is of low rank, or close to a low rank matrix, and requires only one step of the matrix inversion technique. A low rank means that the data is highly redundant and reducible to only a few vectors. The basic idea is to express the design matrix as a singular value decomposition, which leads to a slightly different expression for the solution w in terms of the singular values σ_r and singular vectors u_r, v_r of the data design matrix

$$w = \sum_{r=1}^{R} \sigma_r^{-1} u_r v_r^T y$$

The singular vectors of X are the eigenvectors of $X^T X$, while the singular values of X are the square roots of the eigenvalues of $X^T X$. We therefore only need to perform one eigendecomposition of $X^T X$, thereby 'saving' the matrix multiplication with X^T above. But there is another advantage to taking the route of the SVD. Since $X^T X$ is always a positive definite matrix, we can encode it in a density matrix $\rho_{X^T X}$ and use the technique of density matrix exponentiation to apply $e^{i\rho_{X^T X}t}$ for a time t. Density matrix exponentiation takes the role of Hamiltonian simulation in the original HHL routine. As discussed before, density matrix exponentiation can yield exponential speedups for eigenvalue extraction if $X^T X$ can be approximated by a low-rank matrix.

7.3.2 Inverting Kernel Matrices

The square of the design matrix $X^T X$ is an example of a larger class of positive definite matrices, namely kernel Gram matrices. The idea to use density matrix exponentiation for optimization was in fact first explored in the context of kernel methods by Rebentrost, Mohseni and Lloyd's quantum SVM. It was also the firts proposal that applied quantum blas to machine learning in the stricter sense.

While support vector machines do not directly lead to a matrix inversion problem, a version called *least-squares support vector machine* turns the convex quadratic optimization into least squares optimization. In short by replacing the inequality constraints with equalities, the support vector machine (with a linear kernel) becomes equivalent to a regression problem of the form

$$\begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & & & \\ \cdots & K & & \\ 1 & & & \end{bmatrix} \begin{bmatrix} w_0 \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ y \end{bmatrix}$$

with the Gram matrix K of entries $(K)_{mm'} = (x^m)^T x^{m'}$, the Lagrangian parameters $\gamma = (\gamma_1, \cdots, \gamma_M)^T$, and a scalar bias $w_0 \in \mathbb{R}$. To obtain the γ_i one has to invert the kernel matrix. Note that we simplified the formalism by ignoring so called 'slack parameters' that cater for non-linearly separable datasets.

After preparing the kernel matrix as a density matrix, one can use the density matrix exponentiation technique to simulate $e^{i\rho_K t}$ and proceed with the HHL algorithm to apply K^{-1} to a quantum state $|\psi_y\rangle$ to obtain $|\psi_{K^{-1}y}\rangle$. We call this version of HHL with density matrix exponentiation (DME) for the Hamiltonian simulation, in short the HHL^{DME} routine.

The outcome of the algorithm is a quantum state that encodes the bias w_0 as well as the Lagrangian parameters $\gamma_1, \cdots, \gamma_M$ as

$$|\psi_{w_0,\gamma}\rangle = \frac{1}{w_0^2 + \sum_m \gamma_m^2} \Big(w_0 |0\cdots 0\rangle + \sum_{m=1}^{M} \gamma_m |m+1\rangle \Big)$$

With the help of an interference circuit, this state can be used to classify new inputs.

7.3.3 Inverting Adjacency Matrices

Amongst the problems that can be mapped to matrix inversion is also the Hopfield model. The Hopfield model is a recurrent neural network with certain restrictions on the connectivity, which can be used for associative memory. To 'train such a model, once can set the weighing or adjacency matrix of the graph to

$$W = \frac{1}{MN}\sum_{m=1}^{M} x^m (x^m)^T - \frac{1}{N}\mathbb{1}$$

a technique known as the *Hebbian learning rule*. This rule foresees that every weight w_{ij} connecting two units i, j is chosen as the average of the product $x_i x_j$ of features over the entire dataset. The famous thumb rule is 'neurons that fire together wire together'. **Won't go into the details now**

7.4 Search and Amplitude Amplification

The second line of approaches in using quantum computing for optimization is based on Grover search and amplitude amplification. These algorithms typically promise a quadratic speedup compared to classical techniques.

7.4.1 Finding Closest Neighbors

Dürr and Høyer developed a quantum subroutine using Grover's algorithm to find the minimum of a function $C(x)$ over binary strings $x \in \{0,1\}^n$. Let

$$\frac{1}{\sqrt{N}}\sum_x |x\rangle\,|x_{\text{curr}}\rangle$$

to be a superposition where we want to search through all possible basis states $|x\rangle$. In each step an oracle marks all states $|x\rangle$ that encode an input x so that $C(x)$ is smaller than $C(x_{\text{curr}})$. To perform the comparison, x_{curr} is saved in an extra register. The marked amplitudes get amplified, and the $|x\rangle$ register measured to draw a sample x'. If the result x' is indeed smaller than x_{curr}, one replaces the current minimum by the newly found one. The routine gets repeated until the oracle runs empty, at which point we have found the desired minimum.

In principle, this routine could be used to find the set of model parameters $|w\rangle$ that minimizes a given cost function, and brute force search could be improved by a quadratic speedup. However, it is not difficult to see that this search over all possible sets of parameters is hard in the first place. If we have D parameters and each is discretized with prevision τ, we have to search over $2^{D\tau}$ binary strings, and even an improvement to $\sqrt{2^{D\tau}}$ would be hopeless.

However, the Dürr and Høyer routine can be applied to the task of finding the closest neighbor in clustering and nearest neighbor methods. For example given a data superposition

$$|[\tilde{x}]\rangle \sum_{m=0}^{M-1} |m\rangle\,|[x^m]\rangle\,|0\cdots 0\rangle$$

where $[\tilde{x}], [x^m]$ indicates that the encoding of the new input and the training inputs is arbitrary. To find the nearest neighbor of the new input $\tilde{x}$ amongst the training inputs x^m, one has to compute the distance between the new input and all training points in superposition and write it into the last register.

$$|[\tilde{x}]\rangle \sum_{m=0}^{M-1} |m\rangle\,|[x^m]\rangle\,|[\text{dist}(\tilde{x}, x^m)]\rangle$$

This 'distance register' stores the 'cost' of a training point and can serve as a lookup table for the cost function. Iteratively reducing the subspace of possible solutions to the training points that have a lower cost than some current candidate x^l will eventually find the closest neighbor.

7.4.2 Adapting Grover's Search to Data Superpositions

In some contexts it can be useful to restrict Grover's search to a subspace of basis vectors. For example, the data superposition in basis encoding

$$\frac{1}{\sqrt{M}}\sum_{m=1}^{M} |x^m\rangle$$

corresponds to a sparse amplitude vector that has entries $\frac{1}{\sqrt{M}}$ for basis states that correspond to a training input, and zero else. Standard Grover search rotates this data superposition by the average, and thereby assigns a non-zero amplitude to training inputs that were not in the database. This can decrease the success probability of measuring the desired result significantly.

Common Grover search We want to amplify the amplitude of search string 0110 in a sparse uniform superposition

$$|\psi\rangle = \frac{1}{\sqrt{6}}\big(\,|0000\rangle + |0011\rangle + |0110\rangle + |1001\rangle + |1100\rangle + |1111\rangle\,\big)$$

which in vector notation corresponds to the amplitude vector

$$\frac{1}{\sqrt{6}}(1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1)^T$$

In conventional Grover search, the first step is to mark the target state by a negative phase. This is often formulated in terms of an oracle reflection U_f to our state $|\psi\rangle$

$$\frac{1}{\sqrt{6}}(1,0,0,1,0,0,-1,0,0,1,0,0,1,0,0,1)^T$$

after which every amplitude α_i is rotated via $\alpha_i \to -\alpha_i + 2\bar{\alpha}$ where $\bar{\alpha}$ is the average of all amplitudes. Another way of formulating this is a second oracle reflection about our state $U_s = 2\,|\psi\rangle\,\langle\psi| - 1$. After 'one' step in the Grover algorithm we have performed $U_s U_f\,|\psi\rangle$. We then have

$$\frac{1}{2\sqrt{6}}(1,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1)^T$$

In the second iteration we mark our target again

$$\frac{1}{2\sqrt{6}}(1,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1)^T$$

and each amplitude gets updated again

$$\frac{1}{8\sqrt{6}}(5,3,3,5,3,3,13,3,3,5,3,3,5,3,3,5)^T$$

As we see, the basis states that are not part of the data superposition end up having a non-negligible probability to be measured.

Ventura and Martinez therefore introduce a simple adaptation to the amplitude amplification routine that maintains the zero amplitudes. After the desired state is marked and the Grover operator is applied for the first time rotating all amplitudes by the average, a new step is inserted which marks *all* states that were originally in the database superposition. The effect gives all states but the desired one the same phase and absolute value, so that the search can continue as if starting in a uniform superposition of all possible states.

Ventura, Martinez search Going back to the previous example and applying the Venture-Martinez trick, starting with the same initial state, marking the target, and 'rotating' the amplitudes for the first time

$$\frac{1}{2\sqrt{6}}(1,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1)^T$$

the adapted routine 'marks' all amplitudes that correspond to states in the data superposition

$$\frac{1}{2\sqrt{6}}(1,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1)^T$$

followed by another rotation

$$\frac{1}{4\sqrt{6}}(1,1,1,1,1,1,9,1,1,1,1,1,1,1,1,1)^T$$

From now on we can proceed with Grover search as usual and it is obvious from this example that the amplitude amplification process is much faster, i.e. leads to a much larger probability of measuring the target state than without the adaptation.

7.4.3 Amplitude Amplification for Perceptron Training

Another straight forward application of amplitude amplification to machine learning targets the training of perceptrons. Perceptron models have the advantage that rigorous bounds for training are known. Using the standard learning algorithm and training vectors with unit norm assigned to two classes that are each separated by a positive margin γ, the training is guaranteed to converge to a zero classification error on the training set in $\mathcal{O}(\frac{1}{\gamma^2})$ iterations over the training inputs. A slightly different approach to training in combination with amplitude amplification allows us to improve this to $\mathcal{O}(\frac{1}{\sqrt{\gamma}})$

7.5 Variational Algorithms

The canon of quantum computing has been extended by a variety of variational algorithms. We'll go through two examples here, the *variational eigensolver* and the *quantum approximate optimization algorithm.*

7.5.1 Hybrid Training for Variational Algorithms

With full-blown fault-tolerant quantum computers still in the future, a class of hybrid classical-quantum algorithms has become popular to design near-term applications for quantum devices of the first generation. The idea of hybrid training of variational algorithms is to use a quantum device - possibly together with some classical processing - to compute the value of an objective function $C(\theta)$ for a given set of classical parameters θ. A classical algorithm is then used to optimize over the parameters by making queries to the quantum device.

A parametrized circuit can be thought of as a family of circuits where the parameters define one particular member of the family. For simplicity we will only consider unitary circuits here, but the concepts are easily extended to the evolution of subsystems and mixed states.

Measurements on the final state $|\psi(\theta)\rangle$ return estimates of expectation values. These expectations depend on the circuit parameters θ. A cost function $C(\theta)$ uses the expectation values to define how good θ is in a given problem context. The goal of the algorithm is to find the circuit parameters θ of the variational circuit $U(\theta)$ that minimize $C(\theta)$.

To find the optimal circuit parameters, a classical algorithm iteratively queries the quantum device. These queries can either be the expectation values that define the cost, or - as we will see below - different expectation values that reveal the gradients of the variational circuit. Since training is a joint effort by a quantum and a classical algorithm, the training is called a 'hybrid' scheme.

The variational circuit $U(\theta)$ is an ansatz that defines a set of all possible states $|\psi(\theta)\rangle$ it is able to prepare. It is typically much smaller than the space of all unitaries, since that would require a number of parameters that is quadratic in the Hilbert space dimension, which quickly becomes prohibitive with a growing number of qubits. Similar to the task of choosing a good model in machine learning, a fundamental challenge in variational algorithms lies in finding an ansatz that is rich enough to allow the parametrized state to approximate interesting solutions to the problem with as few parameters as possible. An interesting point is that for qubit-efficient circuits $U(\theta)$ that are not classically simulable, the overall training scheme exhibits an exponential quantum speedup, because the objective function could not be computed efficiently on a classical computer.

The great appeal of variational schemes is that they are suitable for near-term quantum technologies for the following reasons. Firstly, they do only require a fraction of the overall algorithm to run coherently. Second, there are many possible ansätze for the circuit which means that it can be designed based on the strength of the device. Third, the fact that the circuit is learned can introduce robustness against systematic errors - if for example a certain quantum gate in the device systematically over-rotates the state, the parameters adjust themselves to correct for the error. Fourth, compared to other types of algorithms, optimization as an iterative scheme can work with noise, for example in the estimations of the objective function terms.

7.5.2 Variational Eigensolvers

Variational algorithms were initially proposed as a prescription to find ground states - that is, lowest energy eigenstates - of quantum systems, where the cost is simply the energy expectation value. The variational principle of quantum mechanics tells us that the ground state $|\psi\rangle$ minimizes the expectation

$$\frac{\langle\psi| H |\psi\rangle}{\langle\psi|\psi\rangle}$$

where H is the Hamiltonian of the system. The best approximation $|\psi(\theta^*)\rangle$ to the ground state given an ansatz $|\psi(\theta)\rangle$ minimizes over all sets of parameters θ. Assuming the kets are normalized, the cost function of the variational algorithm is therefore given by

$$C(\theta) = \langle\psi(\theta)| H |\psi(\theta)\rangle$$

The quantum device estimates $C(\theta)$ for an initial parameters set and hybrid training iteratively lowers the energy of the system by minimizing the cost function.

In theory, we can perform measurements on the state $|\psi(\theta)\rangle$ to get an estimate of the expectation value of H. However, for general Hamiltonians this can involve a prohibitive number of measurements.

Luckily, in many practical cases H can be written as a weighted sum of local operators $H = \sum_k h_k H_k$ with $h_k \in \mathbb{R} \forall k$. The overall expectation is the given by the sum

$$C(\theta) = \sum_k h_k \langle\psi(\theta)| H_k |\psi(\theta)\rangle$$

of the estimates of 'local' expectation values. The local estimates are multiplied by the coefficients h_k and summed up on the classical device. These 'local' energy expectations are much easier to estimate, which reduces the number of required measurements dramatically. If the number of local terms in the objective function is small enough, i.e. it only grows polynomially with the number of qubits, estimating the energy expectation through measurements is qubit-efficient.

To give one example, remember that the Hamiltonian of a qubit system can always be written as a sum over Pauli operators

$$H = \sum_{i,\alpha} h^i_\alpha \sigma^i_\alpha + \sum_{i,j,\alpha,\beta} h^{ij}_{\alpha,\beta} \sigma^i_\alpha \sigma^j_\beta + \cdots$$

and the expectation value becomes sum of expectations

$$\langle H \rangle = \sum_{i,\alpha} h^i_\alpha \langle\sigma^i_\alpha\rangle + \sum_{i,j,\alpha,\beta} h^{ij}_{\alpha,\beta} \langle\sigma^i_\alpha \sigma^j_\beta\rangle + \cdots$$

where the superscripts i, j denote the qubit that the Pauli-operator acts on, while the subscripts define the Pauli operator $\alpha, \beta = 1, x, y, z$. From this representation we see that the energy expectation becomes qubit-efficient if the Hamiltonian can be written as a sum of only a few (i.e. tractably many) terms, each involving only a few Pauli operators. This is common in quantum chemistry, where Hamiltonians describe electronic systems under Born-Oppenheimer approximations, as well as in many-body physics and the famous Ising/Heisenberg model.

Generally speaking, variational quantum eigensolvers minimize the expectation value of an operator, here the Hamiltonian, using a quantum device to estimate the expectation. Eigensolvers are particularly interesting in cases where the estimation is qubit-efficient on a quantum device, while simulations on a classical computer are intractable.

7.5.3 Quantum Approximate Optimization Algorithm

Another popular variational algorithm has been presented by Farhi and Goldstone, and in its original version solves a combinatorial optimization problem. For us, the most important aspect is the ansatz for $|\psi(\theta)\rangle$ for a given problem that will be adapted to prepare Gibbs states in the quantum machine learning algorithm below.

Consider an objective function that counts the number of statements from a predefined set of statements $\{C_k\}$ which are satisfied by a given bit string $z = \{0, 1\}^{\otimes n}$

$$C(z) = \sum_k C_k(z)$$

A statement can for instance be '$(z_1 \wedge z_2) \vee z_3$' which is fulfilled for $z = z_1 z_2 z_3 = 000$ but violated by $z = 010$. If statement k is fulfilled by $z, C_k(z) = 1$, and else it is 0. We can represent this objective function by the expectation values $\langle z| C |z\rangle$ of a quantum operator that we also call C. In matrix representation, the operator contains on its diagonal the number of statements satisfied by a bit string z, whereby the integer representation i of z defines the element $H_{i+1,i+1} = C(z)$ of the Hamiltonian. The matrix has only zero off-diagonal elements.

Example 7.4 Consider an objective function defined on bit strings of length $n = 2$ with statements $C_1(z_1, z_2) = (z_1 \wedge z_2), C_1(z_1, z_2) = (z_1 \vee z_2)$, the matrix expression of operator $C = C_1 + C_2$ would be given by

$$\begin{bmatrix} \langle 00| C |00\rangle & \langle 00| C |01\rangle & \langle 00| C |10\rangle & \langle 00| C |11\rangle \\ \langle 01| C |00\rangle & \langle 01| C |01\rangle & \langle 01| C |10\rangle & \langle 01| C |11\rangle \\ \langle 10| C |00\rangle & \langle 10| C |01\rangle & \langle 10| C |10\rangle & \langle 10| C |11\rangle \\ \langle 11| C |00\rangle & \langle 11| C |01\rangle & \langle 11| C |10\rangle & \langle 11| C |11\rangle \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

and contains the number of fulfilled statements on the diagonal.

If we interpret a quantum state $|\psi\rangle = \sum_z \alpha_z |z\rangle$ as a qsample that defines a probability distribution over z-strings, then $\langle\psi| C |\psi\rangle$ is the expectation value of operator C under this distribution. Measuring $|\psi\rangle$ in the computational basis means to draw samples from the space of all bit strings.

We now define the ansatz $|\psi(\theta)\rangle$ of the variational algorithm. For this we need a second operator, a sum of Pauli σ_x or 'flip operators' on all bits

$$B = \sum_{i=1}^{n} \sigma_x^i$$

Together, B and C define the ansatz for the parametrized state preparation scheme.

Consider the two parametrized unitaries

$$U(B, \beta) = e^{-i\beta B}, \qquad U(C, \gamma) = e^{-i\gamma C}$$

Starting with a uniform superposition $|s\rangle = \frac{1}{\sqrt{2.}} \sum_z |z\rangle$ and alternately applying $U(B, \beta), U(C, \gamma)$ for short times t prepares the parametrized state

$$|\psi(\theta)\rangle = U(B, \beta_K)U(C, \gamma_K) \cdots U(B, \beta_1)U(C, \gamma_1) |s\rangle$$

The set of parameters θ consists of the $2K$ parameters $\beta_1, \cdots, \beta_K, \gamma_1, \cdots, \gamma_K$.

Farhi and Goldstein have shown that for $K \to \infty$, the maximum expectation value over all θ is equal to the maximum of the objective function

$$\lim_{K\to\infty} \max_{\theta} \langle\psi(\theta)| C |\psi(\theta)\rangle = \max_z C(z)$$

Let θ^* be the parameter set that maximises $\langle\psi(\theta)| C |\psi(\theta)\rangle$. The above suggests that sampling computational basis states from $|\psi^*\rangle$ will reveal good candidates for z, since the state assigns more probability to basis states that correspond to z values which fulfil a lot of statements.

7.6 Variational Quantum Machine Learning Algorithms

We discuss how the variational eigensolvers can be extended to implement a binary classifier in which the expectation value of a quantum observable is interpreted as the output of a machine learning model.

7.6.1 Variational Classifiers

The idea of a variational eigensolver can be extended for supervised learning tasks by interpreting the expectation value of an observable O as the output of a classifier

$$f(x; \theta) = \langle\psi(x; \theta)| O |\psi(x; \theta)\rangle$$

The resulting model is what we call a *variational classifier*. (**Note**: *the variational circuit also has to depend on the model inputs* x).

One way to associate an expectation value with a binary model output $f(x; \theta)$ is to define

$$f(x; \theta) = \langle\psi(x; \theta)| \sigma_z^j |\psi(x; \theta)\rangle$$

where the right side is the expectation value of the σ_z operator applied to the j-th qubit. The σ_z operator measures whether a qubit is in state 0 or 1, and the expectation value is equivalent to measuring the probability of this qubit being in state 1. We can translate the expectation value into a probability of measuring the qubit in state 1 by performing the scaling shift

$$p(q_j = 1) = 0.5\langle\langle\psi| \sigma_z^j |\psi\rangle\rangle + 1$$

When defining the model output as an expectation value, the standard squared loss cost function, which measures the difference between the model outputs and the targets y^m for each training sample $m = 1, \cdots, M$ becomes

$$C(\theta) = \sum_{m=1}^{M} \left(\langle \psi(x^m;\theta)| \, \sigma_z^j \, |\psi(x^m;\theta)\rangle - y^m \right)^2$$

Compared to variational eigensolvers, the cost function is no longer the expectation value itself, but a function of the expectation value. The model outputs are evaluated by the quantum device, while the classical device helps to update the parameters θ with respect to the cost $C(\theta)$.

7.7 Numerical Optimization Methods

We have so far ignored the issue of how to optimize the classical parameters θ in the hybrid training scheme. Since the quantum algorithm provides measurement samples for the expectation of an operator, we only have an estimation of the output, whose precision can be increased by repeating the algorithm and measurement.

7.7.1 Numerical Gradient-Based Methods

Gradient-based methods also have two sub-categories; *numerical* and *analytical*. If only black-box access to the cost function is provided, one can use finite-differences to compute the gradient numerically

$$\frac{\partial C(\theta)}{\partial \theta_l} = \frac{C(\theta_1, \cdots, \theta_D) - C(\theta_1, \cdots, \theta_l + \Delta\theta_l, \cdots, \theta_D)}{\Delta\theta_l} + \mathcal{O}(\Delta\theta_l^2) + \mathcal{O}(\frac{\epsilon}{\Delta\theta_l})$$

7.7.2 Analytical Gradient-Based Methods

The second type of gradient-based methods uses analytical gradients. Consider a quantum circuit

$$U(\theta) = G(\theta_L) \cdots G(\theta_l) \cdots G(\theta_1)$$

that consists of L parametrized 'elementary unitary blocks' G. These blocks are each defined by a set of parameters $\theta_l, l = 1, \cdots, L$. Let μ be an arbitrary parameters in θ. The analytical gradient $\nabla_\mu C(U(\theta))$ of a cost function that depends on the variational circuit usually consists of the 'derivative of the circuit', $\partial_\mu U(\theta)$. We define the partial derivative of a matrix here as the matrix that results from taking the partial derivative of each element.

The problem with the expression for the derivative is that these 'sub-derivatives' are not necessarily unitaries, and therefore no quantum circuits. That means we cannot estimate any expectations from these circuits with the quantum device. However, one can often use tricks to estimate $\partial_\mu U(\theta)$ with the device, and thereby use the analytical gradients in a hybrid training scheme. If possible, this is much preferred over numerical gradients that are less robust.

Assume that μ only appears in the set of parameters θ_l (although the calculation becomes only slightly more complex when several parameters are tied to each other). Then

$$\partial_\mu U(\theta) = G(\theta_L) \cdots \partial G(\theta_l) \cdots G(\theta_1)$$

This is significant: Due to its linearity, the 'derivative of the circuit' is exactly the same as the original circuit except for one block.

A useful observation arises for cases in which

$$\partial_\mu G(\theta_l) = \sum_i a_i G_i(\theta_l)$$

holds. This means that the derivative of the unitary block can be computed as the weighted sum of other blocks. Below we will discuss two cases: one where $G_i(\theta_l) = G(r_i(\theta_l))$ (*which means that only the parameters are transformed by a set of known and fixed functions* r_i), and the other the G_i form a decomposition of the *generator* O of the gate $G = e^{i\theta_l O}$ for a single parameter θ_l

7.7.3 Derivatives of Gates

To give a concrete example of this still rather abstract notion of estimating gradients of circuits, consider a circuit constructed from elementary blocks that are general single qubit gates

$$G(\alpha, \beta, \gamma) = \begin{bmatrix} e^{i\beta}\cos\alpha & e^{i\gamma}\sin\alpha \\ -e^{-i\gamma}\sin\alpha & e^{-i\beta}\cos\alpha \end{bmatrix}$$

General in this context means that any single qubit gate can be represented in this form. The derivative of such a gate with respect to the parameters $\mu = \alpha, \beta, \gamma$ are

$$\begin{aligned} \partial_\alpha G(\alpha, \beta, \gamma) &= G(\alpha + \frac{\pi}{2}, \beta, \gamma) \\ \partial_\beta G(\alpha, \beta, \gamma) &= \frac{1}{2}G(\alpha, \beta + \frac{\pi}{2}, 0) + \frac{1}{2}G(\alpha, \beta + \frac{\pi}{2}, \pi) \\ \partial_\gamma G(\alpha, \beta, \gamma) &= \frac{1}{2}G(\alpha, 0, \gamma + \frac{\pi}{2}) + \frac{1}{2}G(\alpha, \pi, \gamma + \frac{\pi}{2}) \end{aligned}$$

From this we can see that the transformations $r_i(\theta)$ of the parameters consists of shifting some parameters by $\frac{\pi}{2}$ and setting others to a constant. Controlled single qubit gates can be decomposed into linear combinations of unitaries in a similar fashion.

As a second example, consider a variational circuit that is parametrized as Pauli matrices, so that the building blocks read

$$G(\mu) = e^{i\mu O}$$

with O being a tensor product of Pauli operators inclusive the identity, each acting on one of the n qubits. The derivative of a gate $e^{i\mu O}$ is formally given by

$$\partial_\mu e^{i\mu O} = iOe^{i\mu O}$$

and since we can apply the Pauli gates as unitaries, we can simply add O as a further gate to construct the derivative circuit. If O was a non-unitary Hermitian operator, we could try to decompose it into a linear combination of unitaries.

8 Chapter 8 Notes: Learning with Quantum Models

The last two chapters were mainly concerned with the translation of known machine learning models and optimization techniques into quantum algorithms in order to harvest potential runtime speedups known from quantum computing. This chapter will look into 'genuine' quantum models for machine learning which either have no direct equivalent in classical machine learning, or which are quantum extensions of classical models with a new quality of dynamics. A quantum model as we understand it here is a model function or distribution that is based on the mathematical formalism of quantum theory, or naturally implemented by a quantum device.

This chapter sheds some light onto what has been earlier called the *exploratory* approach of quantum machine learning. If focuses less on speedups, but is interested in creating and analyzing new models based on quantum mechanics, that serve as an addition to the machine learning literature. Although still in its infancy, the great appeal of this approach is that it is suited for near-term or noisy as well as non-universal quantum devices, because of the change in perspective: We start with a quantum systems and ask whether it can be used for supervised learning.

8.1 Quantum Extensions of Ising-Type Models

Hopfield networks and Boltzmann machines are both important machine learning methods that are based on one of the most successful family of models in physics, so called *Ising models*. The original Ising model describes ferromagnetic properties in statistical mechanics, namely the behaviour of a set of two-dimensional spin systems that interact with each other via certain nearest-neighbor interactions. We use the term in a wider sense here, to denote models of interacting units or particles of any connectivity and weight.

The Ising model is also no stranger to machine learning. Hopfield networks - which have a historical significance in reconnecting artificial neural networks with brain research - have in fact been introduced by a physicist, James

Hopfield, who transferred the results from statistical physics to recurrent neural networks. Boltzmann machines can be understood as a probabilistic version of Hopfield networks.

In both cases, the spins or particles are associated with binary variables or neurons, while the interactions correspond to the weights that connect the units. The physical equivalence gives rise to a well-defined quantum extension of these two methods.

Hopfield networks and Boltzmann machines are both a special case of recurrent neural networks consisting of G binary variables $s = s_1 \cdots s_G$ with $s_i \in \{-1, 1\}$ or $s_i \in \{0, 1\}$ for all $i = 1, \cdots, G$. The units can be distinguished into a set of visible and hidden units $s = vh = v_1 \cdots v_{N_v} h_1 \cdots h_{N_h}$ with $N_v + N_h = N$. We can define an Ising-type energy function that assigns a real value to each configuration of the system

$$E(s) = -\sum_{i,j} w_{ij} s_i s_j - \sum_i b_i s_i$$

with parameters $\{w_{ij}, b_i\}$ and $i, j = 1, \cdots, G$

In Hopfield models the weights are symmetric $w_{ij} = w_{ji}$ and non-self-connecting ($w_{ii} = 0$), and there is no constant field $b_i = 0$. With this architecture, the state of each unit is successively (synchronously or chronologically) updated according to the perceptron rule.

$$s_i^{(t+1)} = \big(\sum_j w_{ij} s_j^{(t)}\big)$$

The updates can be shown to lower or maintain the energy of the state in every time step $t \to t + 1$, until it converges to the closest local minimum. These dynamics can be interpreted as an associative memory recall of patterns saved in minima.

The recall algorithm is similar to a Monte Carlo algorithm at zero temperature. In every step a random neuron s_i is picked and the state is flipped to $\bar{s_i}$ if it does not increase the energy, that means if

$$E(s_1 \cdots \bar{s_i} \cdots s_N) \leq E(s_1 \cdots s_i \cdots s_N)$$

A generalized version of the Hopfield model allows for a finite temperature and probabilistic updates of units in the associative memory recall. If one defines the probability of updating unit s_i to go from $s = s_1 \cdots s_i \cdots s_N$ to $\bar{s} = s_1 \cdots \bar{s_i} \cdots s_N$ as

$$p(\bar{s}) = \frac{1}{1 + e^{-2E(\bar{s})}}$$

we arrive at Boltzmann machines (with the temperature parameters set to 1). Here the binary units are random variables and the energy function defines the probability distribution over states $\{s\}$ via

$$p(s) = \frac{1}{Z} e^{-E(s)}$$

Where Z is the partition function

$$Z = \sum_s e^{-E(s)}$$

For *restricted* Boltzmann machines, the parameters w_{ij} connecting two hidden or two visible units are set to zero.

8.2 The Quantum Ising Model

In the transition to quantum mechanics, the interacting units of the Ising model are replaced by qubits, the Hamiltonian energy function becomes an operator

$$H = -\frac{1}{2} \sum_{ij} w_{ij} \sigma_z^i \sigma_z^j - \sum_i b_i \sigma_z^i$$

by replacing the values s_i by Pauli-σ_z operators acting on the i-th or j-th qubit. When we express the Hamiltonian as a matrix in the computational basis, it is diagonal. The reason for this is that the computational basis $\{|k\rangle\}$ is an eigenbasis to the σ_z operator, which means that the off-diagonal elements disappear, or $\langle k| \sigma_z^i |k'\rangle = 0$ and $\langle k| \sigma_z^i \sigma_z^j |k'\rangle = 0$ for all i, j if $k \neq k'$. The eigenvalues on the diagonal are the values of the energy function.

The quantum state of the system defines a probability distribution over the computational basis states. For an Ising model it is given by

$$\rho = \frac{1}{Z} e^{-H}$$

with the partition function being elegantly expressed by a trace

$$Z = \mathrm{Tr}\{e^{-H}\}$$

For diagonal matrices H, the matrix exponential e^{-H} is equal to a matrix carrying exponentials of the diagonal elements on its diagonal. Consequently, ρ is diagonal with

$$\mathrm{diag}\{\rho\} = \big(\frac{1}{Z} e^{-E(s=0\cdots 0)}, \cdots, \frac{1}{Z} e^{-E(s=1\cdots 1)}\big)$$

The diagonal elements of the density matrix define the Boltzmann probability distribution for all different possible states of the system's spins or neurons, and the operator formalism is fully equivalent to the classical formalism.

Having formulated the Ising model in the language of quantum mechanics, it is not difficult to introduce quantum effects by adding terms with the other two Pauli operators which do not commute with σ_z. Non-commuting operators do not share an eigenbasis, and the resulting operator is not diagonal in the computational basis. It is therefore not any more a different mathematical formalism for the classical Ising model, but the non-zero off-diagonal elements give rise to genuine quantum dynamics.

It is common to only introduce a few 'quantum terms', such as the *transverse* or *x-field*

$$H = -\frac{1}{2} \sum_{ij} w_{ij} \sigma_z^i \sigma_z^j - \sum_i b_i \sigma_z^i - \sum_i c_i \sigma_x^i$$

where the Pauli operator σ_x^i is applied to the i-th qubit with a strength c_i. Since the Hamiltonian for the transverse Ising model is no longer diagonal in the computational basis, superpositions of basis states are eigenstates of H, and the system exhibits different dynamics compared to the classical system.

The strategy of extending the Hamiltonian illustrates a more general recipe to design quantum versions of machine learning models based on statistical physics

1. Formulate the classical model in the language of quantum mechanics, for example using quantum state ρ to describe the probability distribution, and using operators to describe the 'evolution' of the system
2. Add terms that account for quantum effects, such as a transverse field in the Hamiltonian or a coherent term in a master equation
3. Analyze the resulting quantum model for learning and characterize effects deriving from the extension

Skipping some detailed discussion of these models to focus on variational circuits.

8.3 Variational Classifiers and Neural Networks

Learning models based on variational circuits are a prime example for the exploratory approach to quantum machine learning. The quantum device is used as a computational (sub-)routine in a mathematical model. It is treated as a black-box that produces predictions when fed with inputs. Hence one works with a modle that can be naturally implemented by a quantum device.

In the following we will again look at how to interpret a quantum gate as a linear layer in the language of neural networks. This may help to tap into the rich theory for neural nets in order to develop and analyze quantum models based on variational circuits. It also raises interesting questions about the number of parameters needed to build a sufficiently powerful classifier. We finally look at some proposals from classical machine learning to decompose $N \times N$-dimensional unitaries into matrices which can be computed in time $\mathcal{O}(N)$ or below.

8.3.1 Gates as Linear Layers

Let us consider a variational circuit that consists of parametrized gates as defined previously. Insted of trigonometric functions, we can use two complex numbers $z, u \in \mathbb{C}$ with each a real and an imaginary value as parameters

$$G(z, v) = \begin{bmatrix} z & u \\ -u^* & z^* \end{bmatrix}$$

To neglect the global phase we can include the condition that $|z|^2 + |c|^2 = 1$. An advantage of this parametrization is that it does not introduce non-linear dependencies of the circuit parameters on the model output, while a disadvantage in practice is that during training the normalization condition has to be enforced.

Let us have a look at the structure of the overall matrix representation of a controlled single qubit gate. As a reminder, a single qubit gate applied to the i-th qubit of an n qubit register has the following structure

$$G_{q_i} = \mathbf{1}_1 \otimes \cdots \otimes \underbrace{G(z,u)}_{\text{position } i} \otimes \cdots \otimes \mathbf{1}_n$$

The $2^n \times 2^n$ matrix G_{q_i} is sparse and its structure depends on which qubit i the gate acts on.

As a product of matrices, a variational circuit

$$U(\theta) = G(\theta_L) \cdots G(\theta_1)$$

with $\theta_1, \cdots \theta_L \subset \theta$, can be understood as a sequence of linear layers of a neural network where the layers have a constant size. This perspective allows us to use graphical representations to visualize the connectivity power of a single qubit gate and helps to understand the mechanics of the unitary circuit as a classifier. The position of the qubit as well as the control qubit determine the architecture of each layer, i.e. which units are connected and which weights are tied (have the same value at all times).

8.3.2 Considering the Model Parameter Count

The number of parameters or weights trained in a conventional neural network is at least as large as the input, since each input unit has to be connected to the following layer. In a variational quantum circuit, we typically have much fewer parameters than inputs. Mathematically speaking, a single qubit gate implements a matrix multiplication that transforms *all* N amplitudes, however large N is, controlled by only three parameters. Of course, this matrix multiplication has a lot of structure or 'strongly tied weights'

It is an interesting question what representation power a quantum classifier based on a variational circuit has if we keep the number of parameters poly-logarithmic in the input dimension of the data. For example, imagine a 10 qubit system where the gate matrices act on 2^{10}-dimensional vectors, while the circuit has only a depth of 100 gates and hence of the order of 300 parameters. This is reminiscent of compact representations of weight matrices with tensor networks, to which some rich connections can be made. First investigations on quantum classifiers based on variational circuits with a poly-logarithmic number of parameters in the size of the Hilbert space are promising, but there are yet many open questions.

Keeping the parameter count low is especially important if we use the feature maps discussed earlier to introduce non-linearities into the model. If the feature map maps an input to an infinite-dimensional input layer which the circuit is applied to, we necessarily have much fewer parameters than 'hidden neurons' in the layer.

On the other end of the spectrum, there is a situation in which a parameter count of variational circuits can be of the order of the input. While so far we considered the variational circuit to define the quantum model that is trained by a classical algorithm, one can also hybridize inference - in other words the model - by using *quantum-assisted* or *hybrid architectures*. A hybrid architecture of a neural network has a 'quantum layer', while all remaining layers are standard classical neural network layers. Typically, the quantum layer would be of low-width and in the 'deep end' of a deep neural network, where it only has to deal with low-dimensional features. This circumvents the need for a resource-intense classical-quantum interface to feed the data into the quantum device. If the input dimension to the quantum layer is low enough, we can also easily allow for a number of parameters that is polynomial in this dimension. The idea of hybrid architectures has been originally proposed for probabilistic models, where one samples the states of some hidden units from a quantum device and computes classical samples for the next hidden and visible layer from the quantum samples. Whether this approach shows significant advantages in real applications is still mostly an open research question.

9 Chapter 9 Notes: Prospects for Near-Term Quantum Machine Learning

In order to run the quantum machine learning algorithms presented in this document (book) [1] we often assumed to have a **universal, large-scale, error-corrected** quantum computer available. *Universal* means that the computer can implement any unitary operation for the quantum system it is based on. *Large-scale* refers to the

fact that we have a reasonably high number of qubits (or alternative elementary quantum systems). *Error-corrected* means that the outcomes of the algorithm are exactly described by the theoretical equations of quantum theory, in other words, the computer does not make any errors in the computation besides those stemming from numerical instability.

As a rule of thumb, the quantum community currently speak of intermediate-term algorithms if we use of the order of 100 qubits and on the order of 1000 gates. An important question is therefore what approaches to quantum machine learning are actually feasible in noisy intermediate-term devices. In other words, *are there quantum machine learning algorithms for real-life problems that use only about 100 qubits, have a circuit depth of about 1000 gates and are robust to some reasonable levels of noise?*

This is an active research question, and we will now discuss some aspects of this question.

References

[1] M. Schuld and F. Petruccione, *Supervised Learning with Quantum Computers*. Springer Publishing Company, Incorporated, 1st ed., 2018.

[2] G. S. Aïmeur E., Brassard G., "Machine learning in a quantum world.," 2006.