Hybrid quantum-classical metaheuristics for automated machine learning applications

Von Dollen, D.J.

Chapter 2

Preliminaries

2.1 Machine Learning

Arthur Samuel defined machine learning as a field of computer science that gives computers the ability to learn without being explicitly programmed [180]. Tom Mitchell gave a more formal definition [138]:

Definition 2.1 (Machine Learning [138])**.** A computer program is said to learn from experience E with respect to some task T and some performance measure P_T, if its performance on T, as measured by P_T, improves with experience E.

Machine learning is an approach to artificial intelligence that applies deterministic and probabilistic models to learn patterns from data and make predictions or decisions based on this learning. This learning process begins with training a model over a dataset, where the model identifies patterns or relationships. The type of learning can be broadly categorized into supervised, unsupervised, semi-supervised, and reinforcement learning.

Here, we briefly provide an overview of these categories of machine learning.

2.1.1 Supervised Learning

Given a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ of input data $\mathbf{x}_i \in \mathcal{X}$ and associated training labels $y_i \in \mathcal{Y}$, the learning algorithm A is a procedure that takes the training dataset $\mathcal{D}$ and outputs a hypothesis h from the hypothesis space $\mathbb{H}$. Formally, $A : \{(\mathcal{X}, \mathcal{Y})\} \to \mathbb{H}$. A loss function $\mathcal{L}(\mathcal{Y}, h(\mathcal{X}))$ calculates the difference between the predicted output label $h(\mathbf{x}_i)$ and the ground truth label y_i, over the set of all predictions and labels. The goal

of the learning algorithm is to find a hypothesis h^* that minimizes the loss function, typically averaging the loss over the entire set of training data. This is also known as *empirical risk minimization*. This can be represented as follows.

$$h^* = \underset{h \in \mathbb{H}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, h(\mathbf{x}_i)) \tag{2.1}$$

Examples of supervised learning tasks include classification for categorical labels and regression for continuous labels [79].

2.1.2 Unsupervised Learning

In this category of machine learning, the data are given without labels $\mathcal{D} = \{(\mathbf{x}_i)\}_{i=1}^{n}$, and the goal of the algorithm is to find structure and identify patterns in the data. This can take the form of clustering [152], where $f : \mathcal{X} \to \{1, 2, \ldots, k_c\}$, where k_c is the number of clusters, and $f(\mathbf{x}_i)$ denotes the cluster label or assignment for a given training instance $\mathbf{x}_i$. Other examples of unsupervised learning include dimensionality reduction [222], generative modeling [79], and signal processing [76], and are often used as preprocessing techniques to improve the generalization of supervised learning [72].

2.1.3 Semi-Supervised Learning

Semi-supervised learning is considered a mix of supervised and unsupervised learning in which techniques can be combined to infer labels from a small sample of labeled data $\mathcal{D}_L = \{(\mathbf{x}_i, y_i)\}_{i=1}^{L}$ and a large sample of unlabeled data $\mathcal{D}_U = \{\mathbf{x}_j\}_{j=1}^{U}$. In this case, the structure inferred using unsupervised methods, such as clustering, can be applied to predict labels for unlabeled data [218].

2.1.4 Reinforcement Learning

In reinforcement learning (RL), a learning algorithm, or agent, learns policies to select optimal actions in the states of its environment according to a feedback mechanism defined by a reward function [208]. This framework is often modeled as a Markov Decision Process (MDP) defined by the tuple $(S, A, P, R, \gamma_{RL})$, where S is the set of all possible states, A is the set of all possible actions, $P(s' \mid s, a)$ is the state transition probability function, $R : S \times A \to \mathbb{R}$ is the reward function, and $\gamma_{RL} \in [0, 1]$ is the discount factor.

The agent selects actions $a \in A$ when in states $s \in S$ based on a policy π, which may be deterministic or stochastic. The goal is to learn an optimal policy π^* that maximizes the expected cumulative discounted reward:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma_{RL}^t R(s_t, a_t) \right] \tag{2.2}$$

Reinforcement learning has opened up many application areas and demonstrated its effectiveness in solving complex sequential decision processes. Examples of applications include game playing [199], robotics [109], operations research [129], and healthcare [245].

2.1.5 Deep Learning

Deep learning is an area of machine learning that focuses on models known as neural networks [79], which have many intermediary layers, known as hidden layers, mapping inputs to outputs. These models are typically used in supervised learning, although there are applications for other learning areas.

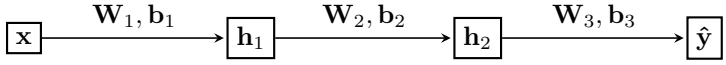

Figure 2.1: A diagram of a directed graph of a feedforward neural network.

The structure of a neural network is a graph comprised of inputs $\mathbf{x}$ (nodes), weights $\mathbf{W}$ (edges), hidden layers $\mathbf{h}$ (nodes) and an output layer $\hat{\mathbf{y}}$ (nodes). These sequences of layers are used to learn a functional mapping from $f : \mathcal{X} \to \mathcal{Y}$ or input to output. Each hidden layer's operation is given by:

$$\mathbf{h}_l = a_l(\mathbf{W}_l \mathbf{h}_{l-1} + \mathbf{b}_l) \tag{2.3}$$

where $\mathbf{h}_l$ is the output of layer l, $\mathbf{W}_l$ is the weight matrix associated with layer l, $\mathbf{b}_l$ is the bias of layer l, and a_l is the activation function applied in layer l as shown in Figure 2.1. The final output of the network $\hat{\mathbf{y}}$ is produced by the last layer. Activation functions a_l, such as *softmax*, *relu*, or *sigmoid* are introduced at each layer to inject non-linearity into the network [79]. We briefly define these activation functions here:

$$\text{sigmoid}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}} \tag{2.4}$$

$$\text{relu}(\mathbf{x}) = \max(0, \mathbf{x}) \tag{2.5}$$

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.6}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ is a vector of probabilities that sums up to 1. We show plots for the action of these activation functions in Figure 2.2.

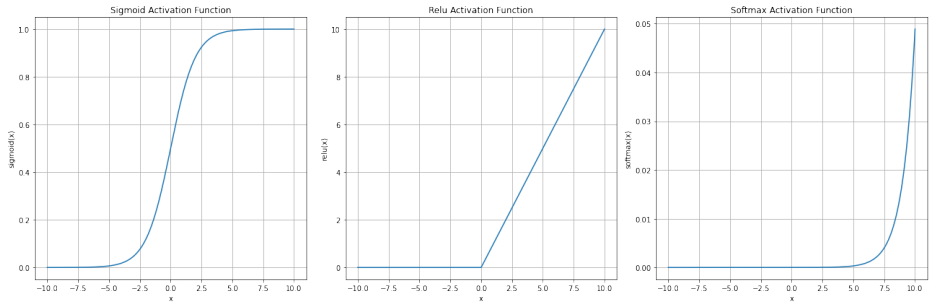

Figure 2.2: Plots for the activation functions *sigmoid*, *relu*, and *softmax*.

The network is often trained using backpropagation by updating the weights and biases of the network [42] based on predicted values vs. ground truth. The defining feature of deep neural networks is the added capacity of additional hidden layers within the network architecture and the allowance for more complex network architectures. These layers can include convolutional layers, as well as layers for max and average pooling, skip connections, and fully connected layers. An example of this type of network is ResNet [89] as shown in Figure 2.3. The rise of deep neural networks, combined with the access to large datasets, can be attributed to advancements in computer vision [118], natural language processing [116], and autonomous perception and control [35].

Figure 2.3: Diagram of ResNet Layer Architecture: Convolutional layers (Conv) Max pooling layer (MaxPool), average pooling layers (AvgPool) Residual blocks (ResBlock) and fully connected layers (FC).[89]

Other forms of machine learning are metalearning [25], transfer learning [156], online learning [192], few-shot learning [223] and zero-shot learning [240]. Machine learning applications are increasingly becoming prevalent within industry with the

rise of systems capable of processing and storing vast amounts of data. Application areas include image and speech recognition [93], social networks [211], natural language processing [51], robotics [212], autonomous vehicles [41], and recommendation systems [154].

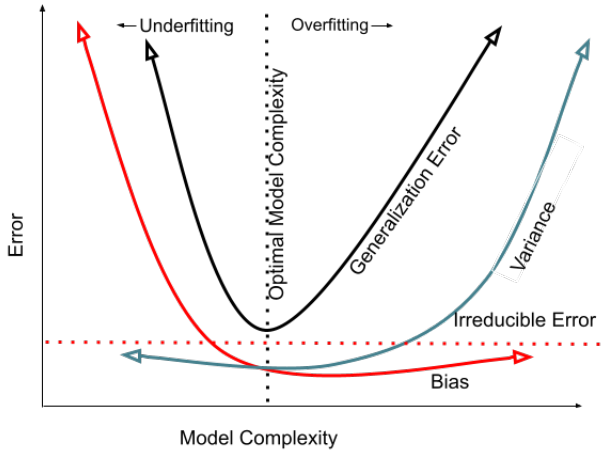

Figure 2.4: Example of the bias-variance trade off. The optimal model complexity with respect to error is shown by the dotted black line.

2.2 Bias and Variance

A fundamental concept known as the *bias-variance trade-off* may help explain the behavior of predictive models as they learn [87]. Bias refers to the approximation error when modeling a real-world problem, which may be represented by complex distributions, with a model that does not have enough representation power to capture the underlying complex distribution. In the context of supervised machine learning, high bias can cause an algorithm to misrepresent relationships between features and

labels [75]. This is commonly known as *underfitting*. Variance measures how much the predictions for a given data point vary between different parameterizations of a model. High variance can cause an algorithm to model random noise within the training data. This is known as *overfitting*. This is typically seen in very complex models that are over-parameterized relative to the number of instances or features in the training data. We show an example of bias and variance with respect to the error curves in Figure 2.4. Note that there may be inherent noise within the data that cannot be eliminated, even in the case of an optimal parameterization for a model. This inherent noise is responsible for what is known as *irreducible error*.

We may also extend *empirical risk minimization* to encompass generalization error and model complexity. We can do so by adding penalty terms to control the trade-off in model complexity and generalization error. This is known as *structural risk minimization* [220] and is given by:

$$h^* = \underset{h \in \mathbb{H}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, h(\mathbf{x}_i) + \lambda_{srm} \Omega(h)) \tag{2.7}$$

Where $\Omega(h)$ is a measure of the complexity of a hypothesis h, and λ_{srm} is a penalty term that controls the trade-off in empirical risk and model complexity. Common forms of managing this trade-off include L^1 and L^2 regularization [17].

Risk minimization frameworks can incorporate strategies such as feature selection, regularization, model selection, hyperparameter optimization [87] [110] in order to reduce model complexity and improve generalization error. Validation choices such as cross-validation or hold-out validation may influence how well the overall risk is estimated for a given model or hypothesis [14].

2.3 Optimization

Optimization is an area of applied mathematics which seeks to find the best solutions from a set of feasible solutions [67]. Although optimization and machine learning are closely related within the field of artificial intelligence, each serve distinct but complementary roles. Optimization provides the mathematical framework for finding solutions, while machine learning applies optimization algorithms to find patterns in data. Many machine learning algorithms rely on optimization subroutines for parameter estimation, model training, and hyperparameter tuning.

Definition 2.2 (Optimization Problem)**.** An optimization problem seeks to find a solution $\mathbf{x}^* \in \mathcal{X}$ that minimizes (or maximizes) an objective function $f : \mathcal{X} \to \mathbb{R}$. Formally, the optimization problem can be expressed as:

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmin}} f(\mathbf{x}) \tag{2.8}$$

where $\mathcal{X}$ is the feasible region defined by the problem constraints.

An optimization problem consists of several essential components:

- *Decision variables*: The quantities $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ that can be adjusted to influence the outcome. The set of all possible values for these variables forms the *search space*.
- *Objective function*: The function $f : \mathcal{X} \to \mathbb{R}$ to be minimized or maximized, mapping each point in the search space to a real-valued performance measure.
- *Global optimum*: A solution $\mathbf{x}^*$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$ in the minimization case (or $f(\mathbf{x}^*) \geq f(\mathbf{x})$ for maximization).
- *Local optimum*: A solution $\mathbf{x}^*$ that satisfies the optimality condition only within a neighborhood $B_\epsilon(\mathbf{x}^*) \subset \mathcal{X}$, such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in B_\epsilon(\mathbf{x}^*)$.
- *Constraints*: Conditions that define the feasible region $\mathcal{X}$ of the search space, restricting which solutions are admissible.

In optimization problems, *constraints* are conditions that solutions must satisfy, defining the feasible region of the problem domain [149, 24]. Three types of constraints are usually encountered in optimization: *equality constraints* require that specific relationships hold exactly, *inequality constraints* specify one-sided restrictions on feasible solutions, and *bound constraints* restrict each decision variable to a specific range (which could be considered special cases of inequality constraints). A solution $\mathbf{x}$ is considered *feasible* if it satisfies all constraints simultaneously, and the formulation and structure of these constraints significantly affect the tractability of the optimization problem [68].

Quadratic unconstrained binary optimization (QUBO) [130] is a form for representing combinatorial optimization problems. A QUBO problem can be formally defined as the following:

Definition 2.3 (QUBO Problem)**.** A QUBO problem seeks to find a binary vector $\mathbf{x}^* \in \{0,1\}^n$ that minimizes a quadratic objective function:

$$\mathbf{x}^* = \underset{\mathbf{x}\in\{0,1\}^n}{\operatorname{argmin}} \sum_{i=1}^{n} Q_{ii}x_i + \sum_{i=1}^{n}\sum_{j>i}^{n} Q_{ij}x_i x_j \tag{2.9}$$

where $\mathbf{x} = (x_1, \ldots, x_n)^T$ is a vector of binary decision variables with $x_i \in \{0,1\}$, and $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is a symmetric matrix in which Q_{ii} represents linear coefficients and Q_{ij} for $i \neq j$ represents quadratic interaction terms between variables.

The QUBO formulation is particularly important because many NP-hard problems can be encoded in this form, including maximum cut, graph coloring, set packing, and various scheduling problems [123]. This universality makes QUBO a valuable framework for both classical and quantum optimization approaches.

Ising models [215] are used in statistical physics to describe the behavior of magnetic spins that can be in one of two states, up or down, or (1, -1). The spin configuration is used to calculate the energy E.

$$E(\mathbf{s}) = -\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} J_{ij}s_i s_j - \sum_{i=1}^{n} h_i s_i \tag{2.10}$$

where s_i are the spin states, J_{ij} represents the interaction strength between spins, and h_i represents the external magnetic field. The binary variables in QUBO can be transformed into spin variables in the Ising model through a simple transformation: $x_i = \frac{1}{2}(s_i+1)$ [242], and are often used to encode combinatorial optimization problems for quantum computation [136, 242].

Types of optimization problems include convex optimization [24], first- and second-order optimization [149], linear and nonlinear optimization [15], combinatorial optimization (including pseudo-Boolean and quadratic binary optimization) [32], and stochastic (black-box) optimization [217]. Many of these problems are ubiquitous in real-world settings and industry [230]. Application areas include supply chain management, inventory control, resource allocation, and scheduling [175, 174, 127] . Often, these problems may be computationally difficult and not expressible as analytical functions. As such, methods known as *metaheuristics* are employed to search the problem domain. These include evolutionary algorithms, estimation of distribution algorithms, simulated annealing, and other randomization routines [23, 49, 7].

2.4 Automated Machine Learning

Automated machine learning (AutoML) [65] has emerged as an area of machine learning where the goal is to automate the construction of model architectures and pipelines for application to real-world problems and data [96]. It covers a wide range of activities including data preprocessing [74], feature engineering/selection [222], model selection [96], hyperparameter optimization [12, 203], model evaluation [64], and model deployment and monitoring [34]. AutoML optimizes complex, multidimensional landscapes for the development of machine learning models. AutoML reduces the time, effort, expense, and expertise required to produce effective machine learning models [96].

Given an unprocessed dataset $\mathcal{D}$, ($\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ in the supervised case or $\mathcal{D} = \{(\mathbf{x}_i)\}_{i=1}^n$ in the unsupervised case) a preprocessing function $P_{\mathcal{D}}$ transforms $\mathcal{D}$ into a processed dataset $\mathcal{D}'$. Preprocessing can take the form of feature selection, feature engineering, normalization, dimensionality reduction, or any other preprocessing method.

$$\mathcal{D}' = P_{\mathcal{D}}(\mathcal{D}) \tag{2.11}$$

Let $\mathbb{H}$ be the set of all possible machine learning models or a set of hypotheses. The selection of models involves choosing a model $h \in \mathbb{H}$ based on performance criteria such as accuracy, computational efficiency, and generality. Validation methods may be used to assess performance criteria. Frameworks such as empirical or structural risk minimization can be incorporated into the model selection process.

Thus:

$$h^* = \underset{h \in \mathbb{H}}{\operatorname{argmin}}\, \mathcal{L}(h(\mathcal{D}')) \tag{2.12}$$

For each model h, there is a hyperparameter space Θ_h. Hyperparameter optimization is the process of finding the optimal set of hyperparameters $\theta^* \in \Theta_h$ that minimizes the loss $\mathcal{L}$ on the preprocessed training data set.

$$\theta^* = \underset{\theta_h \in \Theta_h}{\operatorname{argmin}}\, \mathcal{L}(h^*(\theta_h, \mathcal{D}')) \tag{2.13}$$

A performance evaluation function E takes a model h^* and a test dataset $\mathcal{D}_{\text{test}}$ and returns a performance score $s \in \mathbb{R}$ which measures the generalization error over the test dataset.

$$s = E(h^*, \mathcal{D}_{\text{test}}) \tag{2.14}$$

Definition 2.4 (Automated Machine Learning)**.** Let $\mathcal{D}$ be a dataset, $\mathbb{H}$ a hypothesis space, Θ a hyperparameter space, and $\mathcal{L}$ a loss function. An AutoML system is a function $\mathcal{A} : \mathcal{D} \to \mathbb{H} \times \Theta \times \mathbb{R}$ that automatically selects and configures a machine learning pipeline:

$$(h^*, \theta^*, s) = \mathcal{A}(\mathcal{D}) \tag{2.15}$$

where $h^* \in \mathbb{H}$ is the selected model, $\theta^* \in \Theta_{h^*}$ are optimized hyperparameters, and $s \in \mathbb{R}$ is a performance metric, such that the pipeline minimizes expected generalization error without manual intervention.

The AutoML process flow typically automates the steps in deploying a machine learning model, as shown in 1.2. These steps may include *preprocessing*, *model selection*, *hyperparameter optimization* and *model evaluation*. There also exist algorithms and frameworks that combine steps, such as CASH (combined algorithm and hyperparameter optimization) [213]. Preprocessing methods such as feature selection and feature engineering may also be included as part of the AutoML process to obtain an optimal performance score or generalization error [62]. It is important to note that the search space for features, models, and hyperparameters can be very complex. The configuration spaces are often hierarchical in nature and can contain a mix of discrete and continuous variables. Some AutoML optimization processes use surrogate models to perform sequential model-based optimization. SMAC (Sequential Model-Based Algorithm Configuration) [95] is an algorithm used to optimize the hyperparameters of machine learning models and other complex systems. Also generally known as a form of Bayesian optimization, it utilizes a model-based approach, typically building a probabilistic surrogate model that estimates the performance of different configurations based on historical data. Other optimization methods used within AutoML are evolutionary and population-based methods [151], and reinforcement learning [101].

2.5 No Free Lunch

The No Free Lunch (NFL) theorems [237] are a set of results concerning optimization and machine learning that are important to the application of algorithms to various set of problems. These theorems show that no one algorithm is universally superior to all others on the set of all optimization problems.

Definition 2.5 (No Free Lunch Theorem)**.** Let $\mathcal{X}$ and $\mathcal{Y}$ be finite sets, and let $\mathcal{F}$ be the set of all functions $f : \mathcal{X} \to \mathcal{Y}$. For any optimization algorithm A, let $P_f(f, A)$

represent a performance measure of algorithm A on the function f, which may quantify the objective value achieved or the number of function evaluations required.

Define the average performance of algorithm A over all functions in $\mathcal{F}$ as:

$$\hat{P}(A) = \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} P_f(f, A) \tag{2.16}$$

Then for any two algorithms A and B:

$$\hat{P}(A) = \hat{P}(B) \tag{2.17}$$

In summary, when averaged over all possible objective functions, no optimization algorithm may outperform any other. However, there may be some sets of functions in which one algorithm may yield better performance than another [237]. This highlights the importance of selecting algorithms that may be well suited to the characteristics of the given problem. These characteristics may leverage prior knowledge of the domain of the objective function.

For machine learning, these theorems imply that no one algorithm will universally generalize equally well over all data sets [238]. This highlights the importance of empirical testing and validation of algorithm choice for a given dataset. The NFL theorems also imply that the way data is preprocessed and features are engineered can have a significant impact on the performance of learning algorithms. This aspect of machine learning may be critical to developing effective models.

It is also important to note that the NFL theorems mostly apply to a discrete set of functions. In continuous or infinite domains, the implications of the theorems may be less direct [6].

2.6 From Quantum Mechanics to Quantum Computing

The field of quantum computing has roots in the early 1900s, when a series of several groundbreaking discoveries in quantum mechanics were made by the physics community. These included the quantum hypothesis [119], the photoelectric effect [61], the model of the atom [19], the Heisenberg uncertainty principle [90] , and the Bell inequalities [9]. Schrödinger developed equations for dealing with wave mechanics that describe how a quantum system may change over time [184]. Max Born developed

a fundamental principle in quantum mechanics that gives the probability of finding a particle in a particular quantum state by squaring the amplitude of its wave function [21]. By formally being able to describe quantum states and operators, these mathematical foundations made later work in quantum computing possible.

These foundations led researchers to start exploring new theories for quantum information in the 1960s and 1970s. In 1968 Stephen Weisner invented conjugate coding for cryptography [232], and in 1976 Roman Ingarden published the first work generalizing classical information theory to quantum information theory [97]. In the 1980s, the theoretical foundation for quantum computing began to take shape. Paul Benioff proposed a quantum-mechanical model of the Turing machine [11] that led to the identification of classes of quantum computational complexity. Around the same time in the 1980s Richard Feynman proposed the idea of a quantum computer for the simulation of quantum systems [66].

2.7 Quantum Computation and Computational Complexity Classes

There may exist classes of computational problems that quantum computers may be able to solve more efficiently [13], [197]. These include *BQP* and *QMA* which are included in a wider family of computational complexity classes. We give an overview of some of the computational complexity classes with example problems here:

- *P (Polynomial Time)*: This class consists of decision problems that can be solved and verified by a deterministic Turing machine in polynomial time. Essentially, these are problems for which there is an efficient solution. An example problem in P is determining whether a number is prime.

- *NP (Nondeterministic Polynomial Time)*: NP is the class of decision problems for which, if the answer is "yes", there exists a proof that can be verified in polynomial time by a deterministic Turing machine. Crucially, it is unknown whether all NP problems can be solved in polynomial time (known as the P vs. NP problem). *NP-Complete* is the set of complete problems such that every problem in NP can be reduced to it in polynomial time. An example NP-Complete problem is the set-packing problem [123].

- *BPP (Bounded Error Probabilistic Polynomial Time)*: This class includes decision problems that can be solved by a probabilistic Turing machine in polynomial

time, with a probability of error less than 1/3 for each instance. BPP have a small chance of producing an incorrect result. Example problems in this class include Polynomial Identity Testing, and Matrix Multiplication Verification [189] [69] .

- *BQP (Bounded Error Quantum Polynomial Time)*: This class encompasses decision problems solvable by a quantum computer in polynomial time, with a bounded probability of error. BQP is relevant in the context of quantum computing, containing problems efficiently solvable on a quantum computer that may not be efficiently solvable on a classical computer. Examples of BQP problems include the discrete logarithm problem and integer factoring [197].

- *NP-Hard*: A problem *Problem A* is NP-Hard if there is a polynomial time reduction from *Problem B* in NP to *Problem A* [136]. NP-Hard problems might be even harder than NP problems and are not necessarily in NP (i.e., they might not have efficiently verifiable proofs). Solving an NP-Hard problem efficiently would solve all problems in NP efficiently. A famous NP-Hard problem is the Traveling Salesman problem [1].

- *QMA (Quantum Merlin-Arthur)*: This is the quantum analogue of NP, where the proof can be a quantum state, and the verifier is a quantum computer. Problems in QMA can be verified in polynomial time with a quantum computer, given a quantum proof. Examples of this problem class include Q-SAT, detecting insecure quantum encryption, and Hamiltonian ground state estimation [50].

- *QMA-Hard*: Similarly to NP-Hard, a problem is QMA-Hard if a problem in QMA can be reduced to it in polynomial time. These problems are at least as hard as the hardest problems in QMA. Examples of problems in this class include the interacting boson problem [231] .

We note here that it is unproven (but widely assumed) that P $\neq$ NP. There exist classes of problems which may be more quantum-efficient versus classical in the BQP problem class [13]. However, there are problems which are even more difficult when given access to a quantum system [20]. From a machine learning and optimization perspective, the existence of these complexity classes and problems provides additional motivation towards studying quantum computing and quantum machine learning so as to have tools available to deal with optimization and learning problems in these complexity classes.

2.8 Universal Gate Model Quantum Computing

In the context of quantum computing, the properties of quantum mechanics enable a different form of computation compared to classical computing. Quantum computing uses quantum bits or *qubits*. Unlike classical bits, qubits can be in a state of *superposition*, where they simultaneously exist in multiple states at the same time until measurement. This means that for a qubit representing two states 0 and 1, there are non-zero probabilities of measuring either state upon measurement. A system consisting of n qubits can exist in a superposition of 2^n states. *Entanglement* is another phenomenon in which pairs or groups of qubits become interconnected so that the results of the measurement are correlated. This phenomenon arises because the combined quantum state of the system is nonseparable and cannot be expressed as a simple tensor product of individual states. Operations on entangled qubits in superposition may allow for more complex calculations in a lower number of computing steps given quantum circuits of certain sizes and depth.

In quantum mechanics, *interference* arises due to the wavelike behavior of particles. Interference is used in quantum computing to manipulate the probabilities of qubit states. By carefully controlling the interference patterns, the probabilities for undesired quantum states can be canceled out, and the probabilities corresponding to the desired state are amplified. *Measurement* is an operator of quantum mechanics also used in quantum computing. Measurement allows for the collapse of the wave function of quantum states into basis states in the computational basis. The collapse is determined by the probability amplitudes of the quantum states. Measurement provides a way of retrieving classical information from a quantum system which may be useful for computation [158, 239].

2.8.1 Quantum Logic Gates and Circuits

Quantum computation uses quantum-mechanical effects such as superposition, entanglement, and interference. These effects are formalized in the form of logic gates and circuits, central building blocks for computation. Unlike classical gates, quantum gates manipulate qubits through operations defined by unitary matrices. When these matrices are multiplied by their complex conjugate transpose, they yield the identity matrix. This property ensures that the total probability amplitudes of a quantum state remain normalized to 1, according to the Born rule [21].

The Born rule provides a mathematical definition for calculating the probability of obtaining a specific measurement result based on the probability amplitudes. For a

qubit in the state:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.18}$$

where α and β are complex probability amplitudes satisfying the normalization condition $|\alpha|^2 + |\beta|^2 = 1$, the Born rule states that the probabilities of measuring the qubit in the states $|0\rangle$ and $|1\rangle$ are:

$$P(0) = |\alpha|^2 \tag{2.19}$$

$$P(1) = |\beta|^2 \tag{2.20}$$

This means that the probability of the qubit collapsing in state $|0\rangle$ after measurement is equal to the square of the magnitude of α, and similarly for state $|1\rangle$ with β. On the *Bloch sphere*, which is a geometric representation of the state space of a qubit, these manipulations can be visualized as rotations. Each point on the Bloch sphere represents a possible state of the qubit. The north and south poles represent the states $|0\rangle$ and $|1\rangle$. Recall that α and β are complex amplitudes corresponding to the probability that the qubit is in each state. In quantum computing, the phase of a quantum state refers to the angle of the complex numbers that represent the probability amplitudes of the state's basis components. These complex numbers can be thought of as vectors in a complex plane, and the phase specifically denotes the angle these vectors make with the positive real axis.

We shall now give an overview of some common quantum logic gates [4, 92] and describe their action in relation to the Bloch sphere. The Pauli group is a group of gates that operate on a single qubit at a time. The X-gate rotates the state of the qubit by 180 degrees around the x-axis of the Bloch sphere. If a qubit state is at the north pole of the Bloch sphere $|0\rangle$ applying an X gate will rotate it to the south pole $|1\rangle$ and vice versa. This operation is similar to the classical NOT operation.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.21}$$

The Y gate rotates the qubit state by 180 degrees around the y-axis of the Bloch sphere. This gate also flips the state, but introduces a phase. For example, if a qubit is in state $|0\rangle$ or $|1\rangle$, applying a gate Y will flip the state and multiply it by either i or

$-i$.

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{2.22}$$

The Z gate rotates the qubit state 180 degrees around the z axis. This gate changes the phase of the qubit.

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.23}$$

The Hadamard gate is used to create superposition states. This gate rotates the state vectors by 180 degrees around the axis that lies at the midpoint of the x-axis and the z-axis. For example, applying a Hadamard gate to a qubit in state $|0\rangle$ places it in an equal superposition of $|0\rangle$ and $|1\rangle$, represented by a point on the equator of the Bloch sphere.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.24}$$

The gate R_x rotates a qubit state around the x-axis of the Bloch sphere at an angle θ.

$$R_x(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \tag{2.25}$$

The gate R_y rotates a qubit state around the y-axis of the Bloch sphere at an angle θ.

$$R_y(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \tag{2.26}$$

The gate R_z rotates a qubit state around the z-axis of the Bloch sphere at an angle θ. It is important to note that R_z only changes the phase of the qubit, as shown in Figure 2.5. The phase of a qubit refers to the angle component of its complex probability amplitudes in a superposition state, representing the difference in phase between the basis states of the qubit $|0\rangle$ and $|1\rangle$.

$$R_z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix} \tag{2.27}$$

Rotation gates are important for qubit state manipulation. They enable the implementation of arbitrary single-qubit quantum gates, which are essential for quantum algorithms and quantum information processing. By adjusting the angle θ, these gates provide the flexibility to create a wide range of quantum states, facilitating complex quantum computations, beyond rotation angles of 180 degrees. These gates are essen-

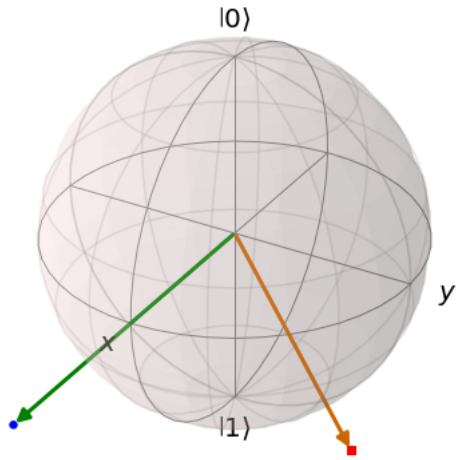

Figure 2.5: Example Visualization of the action of the R_x gate on the Bloch sphere, where the vector is rotated by 45 degrees.

tial for constructing variational quantum circuits.

There are also gates which act upon 2 or more qubit states. The Controlled NOT Gate (CNOT) entangles two qubits by flipping the second (target) qubit if the first (control) qubit is in state $|1\rangle$.

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

The CNOT gate can be used to manipulate an entangle quantum states using two or more qubits. As shown in Figure 2.6, a ladder of CNOT gates are applied sequentially to qubits after rotation gates and before measurement.

The Controlled-Z Gate CZ gate applies the Z gate to the target qubit if the control qubit is $|1\rangle$:

$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{2.29}$$

Also worth mentioning is the RZZ gate. This gate is a parametric circuit which

can be considered as a multiplexed RZ gate.

$$R_{ZZ}(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 & 0 & 0 \\ 0 & e^{i\frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & e^{i\frac{\theta}{2}} & 0 \\ 0 & 0 & 0 & e^{-i\frac{\theta}{2}} \end{pmatrix} \tag{2.30}$$

The Toffoli Gate (CCNOT) gate for quantum computing, acts as a controlled-controlled NOT gate.

$$TOFFOLI = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.31}$$

Constructing quantum circuits involves a combination of unitary operators, measurement operations, and other operators such as rotation gates as shown in Figure 2.6. The diagram illustrates a quantum circuit with four qubits, represented as q_0, q_1, q_2, and q_3. A Hadamard gate is applied to each qubit, inducing a superposition of states. Subsequently, each qubit undergoes three rotation gates: $R_x(0.5)$, $R_y(0.6)$, and $R_z(0.7)$, corresponding to rotations around the x, y, and z axes of the Bloch sphere, respectively. Controlled-Z gates are applied between the qubits q_0 and q_1, and between q_2 and q_3 to create entanglement. Finally, measurement gates are applied to each qubit, collapsing the qubit states to classical bits.

These elements form the basic building blocks of quantum algorithms and gate-model quantum computing in general. Certain sets of quantum gates are universal, meaning that they can perform any classical computation with an overhead up to a polynomial factor. The Solovay-Kitaev Theorem [108] shows that any single-qubit unitary operation can be efficiently approximated using a finite universal gate set, with the approximation error decreasing rapidly as the sequence length increases logarithmically. This theorem is foundational for quantum computing, as it confirms that quantum computers do not require an infinite number of gates to perform operations.

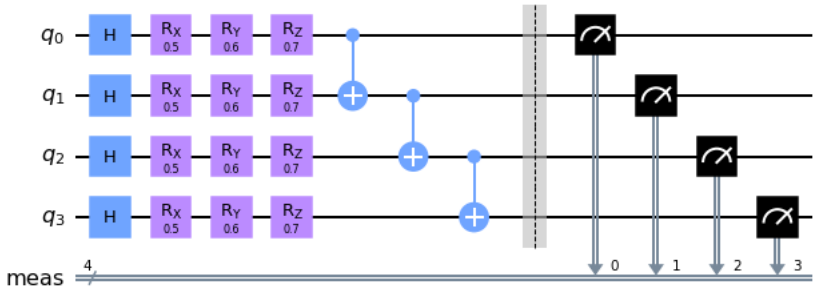

Figure 2.6: Diagram of a variational quantum circuit.

2.8.2 Hardware

Implementing the gate model quantum computing requires hardware capable of initializing qubits in a known state, performing a sequence of quantum gates, and then measuring the qubits to read out the result. At the time of this writing, these systems may be difficult to control and prone to errors due to environmental noise and system fragility and instability. As such, various techniques are currently under development towards error correction and higher operating temperatures [29]. Various physical systems are being explored for building qubits, including trapped ions [31], superconducting circuits[29], and topological materials [117]. Each of these systems has its own strengths and challenges regarding qubit fidelity, coherence time, gate speed, and scalability.

Josephson junctions [103] are critical elements in the construction of superconducting qubits, which are integral to the operation of quantum systems like those created by D-Wave Systems, Google, IBM and other quantum computing manufacturers. Josephson junctions consist of two superconductors that are separated by a thin insulating layer. This design allows for electron pairs known as Cooper pairs to quantum-mechanically tunnel through the insulator, a process that can occur without an external voltage. This process is known as the Josephson effect. The tunneling current across the junction is highly sensitive to the phase difference between the wave functions of the two superconductors. The unique properties of Josephson junctions enable the precise control required for quantum computation, allowing qubits to perform operations that are fundamental to quantum computing. This capability is crucial in the manipulation of qubits to solve complex problems.

We note here that at the time of this writing the scale of gate model systems in

terms of numbers of physical qubits remain in the low hundreds [105]. The number of actual qubits used for computation is reduced further in that a number of physical qubits are grouped to form one logical qubit, to reduce errors. These attributes along with limited accessibility, present a challenge towards developing practical applications for machine learning which demonstrate a strong quantum advantage.

2.9 Adiabatic Computing

Adiabatic quantum computing is based on the adiabatic theorem [22]: given a quantum system in its ground (minimum) energy state, if the governing Hamiltonian is changed according to slow time evolution, then the quantum system remains in its instantaneous ground state.

In quantum mechanics, a Hamiltonian H is an operator corresponding to the total energy of a system, encompassing both kinetic and potential energies. It governs the time evolution of quantum states through the Schrödinger equation [184]. In the context of adiabatic quantum computing, the Hamiltonian is time-dependent and interpolates between an driver Hamiltonian and a target Hamiltonian. The Hamiltonian encapsulates the dynamics of the quantum system and is crucial for determining how the system evolves over time.

We give a formulation for this similar to [80]:

$$H_t = A_t H_I + B_t H_f \tag{2.32}$$

subject to:

$$B_{(t=0)} = 0$$

and

$$A_{(t=T)} = 0$$

Where H_I is the driver Hamiltonian, H_f is the target Hamiltonian, and T is the evolution timescale [242].

The minimum energy gap, $g_{\min}$, is defined as the smallest difference between the energy of the ground state, $E_0(t)$ and the first energy of the excited state, $E_1(t)$, during the entire evolution process:

$$g_{\min} = \min_{0 \leq t \leq T} [E_1(t) - E_0(t)] \tag{2.33}$$

This minimum energy gap is critical in determining the time scale over which the

adiabatic evolution must occur to ensure that the system remains in its ground state, an essential condition for adiabatic quantum computation.

2.9.1 Quantum Annealing

Quantum annealing systems, which are special-purpose quantum computers for solving combinatorial optimization problems encoded as Ising Hamiltonians and QUBOs, have garnered significant attention due to their novel approach to computation. These systems utilize adiabatic computing and the natural evolution of quantum states to find the minimum of a cost function, which is often represented by a problem encoded as a Hamiltonian H_f. Quantum annealing is considered a metaheuristic for combinatorial optimization.

One of the primary strengths of quantum annealing is its ability to navigate complex energy landscapes. By leveraging an effect known as *quantum tunneling*, quantum annealers may enable escape of energy barriers and local minima in the optimization landscape for a given encoded problem, potentially finding global minima more efficiently and/or in less time than classical algorithms for certain problems [136]. This is particularly useful in combinatorial optimization problems that may have numerous local minima. Certain classical algorithms and metaheuristics may sample from or walk the landscape of the optimization problem, and may end up in local minima [60]. By following a slow time evolution for the quantum annealing process, the system may remain in a ground state and allow for quantum tunneling rather than walking the landscape of the optimization domain [136], as shown in Figure 2.7

Adiabatic quantum computing and quantum annealing are closely related in that both utilize the adiabatic theorem, allowing a quantum system to evolve slowly to remain in its ground state. Adiabatic quantum computing requires strict adiabatic conditions throughout evolution. In contrast, quantum annealing is a specialized application of adiabatic quantum computing focused on solving optimization problems by finding the ground state of a problem-specific Hamiltonian, often relaxing adiabatic conditions to achieve practical, approximate solutions more rapidly.

2.9.2 Hardware

D-Wave Systems is a company known for developing and commercializing quantum computers, historically focusing on quantum annealing systems. D-Wave's systems are built using superconducting circuits to create qubits, and arranged in unit cells called a Chimera graph as shown in Figure 2.8.

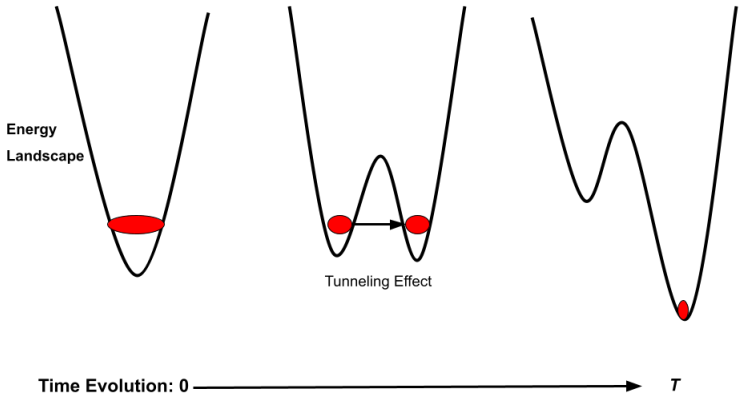

Figure 2.7: Example of quantum tunneling effect for quantum annealing of one qubit [136].

The hardware is designed to solve Quadratic Unconstrained Binary Optimization (QUBO) problems and problems encoded as Ising models, standard formats for expressing a wide range of combinatorial optimization problems. When a user wants to solve a problem using a D-Wave quantum annealer, they typically construct a QUBO or Ising model to represent their problem, where each node represents a variable and each edge represents a constraint or interaction between variables. However, the problem graph may not match the Chimera graph's structure due to differences in connectivity, as the Chimera graph is not fully connected.

Minor embedding comes into play to address this limitation [210]. It involves mapping the original problem graph onto the Chimera graph in such a way that the qubits and their connections in the hardware can represent the problem accurately. This is done by identifying a subgraph (known as the "minor" subgraph) within the Chimera graph that is isomorphic to the problem graph, allowing the problem to be represented on the hardware despite its limited connectivity.

The process of finding a minor embedding can be challenging, especially for large and fully connected problem graphs. Finding a minor embedding is NP-hard [122]. It may require representing a single logical qubit with multiple physical qubits to achieve the required connectivity, a process known as chaining. The strength of the couplings within a chain must be set carefully; if they are too weak then the chain may break,

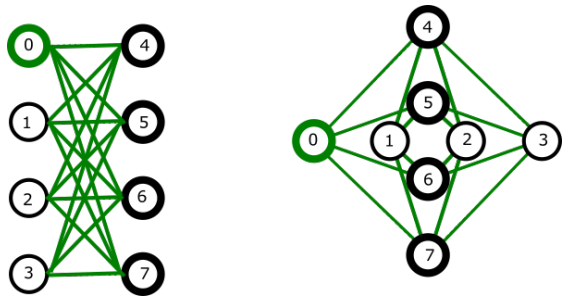

Figure 2.8: Diagram of the D-wave Chimera unit cell [210].

and if they are too strong then the chain may affect the problem's energy landscape. Both outcomes may lead to incorrect solutions.

In light of this, D-Wave provides software tools that automatically find a minor embedding for a given problem [210]. This allows users to focus on formulating their problems without needing to manually handle the intricacies of the hardware's topology. The optimization of the finding of minor embeddings and annealing offsets has also been explored in [243, 122]. At the time of this writing, quantum annealing systems use quantum processing units that contain thousands of physical qubits [210], although logical qubits may still be represented by multiple physical qubits.

2.10 Quantum Machine Learning

The field of quantum machine learning (QML) encompasses the investigation of quantum algorithms for supervised, unsupervised, and reinforcement learning regimes [59]. For the purposes of this thesis we also consider hybrid quantum classical machine learning, quantum-inspired and classical learning from quantum simulation areas of quantum machine learning. In light of this, we extend our definitions of a classical supervised learning dataset to a quantum dataset containing n quantum states:

$\mathcal{D}_Q = \{(|\psi\rangle_i, y_i)\}_{i=1}^n$ where:

- $|\psi_i\rangle$ is i^{th} quantum state of the training set
- y_i is the class membership of that quantum state in the case of classification, or $y_i \in \mathbb{R}$ in the regression case.

In the context of quantum machine learning, data might be the representation of classical information encoded into quantum states or data originating from quantum systems. Through quantum machine learning, we can study the combinations of

quantum and classical learners for quantum and classical data [2], [71]. As shown in Table 2.1, we can divide these combinations into four groups according to the class of data and the class of algorithms. Classical/classical (CC), classical/quantum (CQ), quantum/classical (QC), and quantum/quantum (QQ).

		Type of Algorithm	
		classical	quantum
Type of Data	classical	CC	CQ
Type of Data	quantum	QC	QQ

Table 2.1: Combinations of classical and quantum algorithms and data [71]. In QML, we are most concerned with the orange-highlighted squares. For the purposes of this thesis, we consider quantum inspired algorithms as part of QML. We consider classical simulation of quantum computing with quantum encoded data in the QC category. We consider quantum-inspired algorithms operating on classical data in the CQ category.

Quantum algorithms have the potential to significantly speed up the processing of complex computations involved in machine learning, such as matrix inversion [86] or finding eigenvectors and eigenvalues of operators [162, 121]. However, quantum machine learning also faces several challenges. Current quantum computers, known as Noisy Intermediate-Scale Quantum (NISQ) [165] computers, are limited in size and coherence time and prone to errors, restricting the complexity of algorithms they can reliably execute. Moreover, many quantum algorithms require a quantum computer with a high degree of fault tolerance, which is still a work in progress. Quantum random-access memory is another attribute that many quantum machine learning algorithms use to obtain theoretical speedups that have not yet been physically realized [121].

2.10.1 Quantum Kernels

Quantum kernel methods are a popular branch of quantum machine learning algorithms which present interesting characteristics, such as the potential to handle complex data distributions which may be difficult for classical methods to represent. These

models leverage reproducing kernel Hilbert spaces of quantum states for extracting patterns from data, and are dependent on the method of data encoding [185], but offer guarantees with respect to learning from the inner products of the measurements of these feature spaces [187]. Quantum kernel methods have been applied in quantum data classification tasks [181], classification of neuronal cell types [221], and solving differential equations [155].

The quantum kernel is mathematically defined as a function $kernel : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$, which quantifies the similarity between two data points $x, x' \in \mathcal{X}$ within a Hilbert space represented by density operators. The quantum kernel between two data points x and x' is defined using the trace operation as:

$$kernel(x, x') = \mathrm{Tr}\left(\rho(x)\rho(x')\right) \tag{2.34}$$

where: $\rho(x)$ and $\rho(x')$ are the density operators corresponding to the data points x and x', respectively. Tr denotes the trace operation in the quantum state space.

In practice, the quantum states are prepared using a sequence of unitary operations, or quantum gates, and the kernel is calculated using quantum circuits designed to implement these operations. The kernel effectively captures the quantum-mechanical probability amplitudes and benefits from the high-dimensional feature space of quantum states.

The advantage of quantum kernel methods has been demonstrated by constructing datasets in which the quantum computer can discern intrinsic labeling patterns [88], using the discrete log problem; a problem that can be efficiently solved on a quantum computer but may be intractable for classical computers [197]. The comparison of quantum kernels and other quantum machine learning methods is a topic of continued debate and research in the quantum machine learning community [99].

2.10.2 Variational Algorithms for Quantum Machine learning

Variational quantum circuits (VQCs) are a key component in the field of quantum machine learning, particularly within the context of NISQ devices. These circuits are constructed with parameterized rotation gates, whose parameters are optimized during a learning process, allowing the circuits to be tailored for specific tasks like classification or regression.

We provide a mathematical formulation of VQCs is as follows similar to [134, 201, 16, 37]:

Consider a quantum system with n qubits. A variational quantum circuit $U(\boldsymbol{\theta})$ in

this system is represented as a sequence of parameterized quantum gates that act on the qubits. The circuit transforms an initial state $|\psi_0\rangle$ into a final state $|\psi(\boldsymbol{\theta})\rangle$, which is given by:

$$|\psi(\boldsymbol{\theta})\rangle = U(\boldsymbol{\theta})|\psi_0\rangle \tag{2.35}$$

The unitary transformation $U(\boldsymbol{\theta})$ is composed of L layers, where each layer ℓ includes both parameterized and nonparameterized gates:

$$U(\boldsymbol{\theta}) = \prod_{\ell=1}^{L} U_\ell(\boldsymbol{\theta}_\ell) \tag{2.36}$$

Here, $\boldsymbol{\theta} = \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_L$ denotes the set of all parameters across the layers, and $U_\ell(\boldsymbol{\theta}\ell)$ is the unitary operation for layer ℓ with its specific parameters $\boldsymbol{\theta}_\ell$.

The aim of a variational algorithm is to optimize the parameters $\boldsymbol{\theta}$ such that the final state $|\psi(\boldsymbol{\theta})\rangle$ minimizes a loss function $\mathcal{L}(\boldsymbol{\theta})$, which is typically defined as:

$$\mathcal{L}(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle \tag{2.37}$$

In the right hand side of this expression, H is a Hermitian operator, often chosen to represent a problem-specific Hamiltonian in quantum mechanics or an observable in quantum machine learning tasks. The parameters θ are optimized using classical optimization routines:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\theta}) \tag{2.38}$$

The optimized parameter set $\boldsymbol{\theta}^*$ is used to prepare the quantum state at a minimized cost. In quantum machine learning, the variational circuit is trained to perform tasks such as classification, regression, or generative modeling by iteratively updating the parameters to minimize the cost function based on training data, drawing a parallel to the training process in classical neural networks.

The potential of VQCs comes from their ability to exploit quantum-mechanical effects, such as superposition and entanglement, which may provide a computational advantage over classical systems for certain problems. This makes them particularly promising for the approach to complex problems in chemistry, optimization, and machine learning.

However, there are challenges in working with VQCs, largely due to the errors and noise inherent in current quantum hardware. These errors can cause issues such

as barren plateaus [135], where the gradient of the optimization landscape vanishes exponentially in the number of qubits, making it difficult for optimization algorithms to find a global minimum. Researchers are actively developing techniques to overcome these issues, such as transfer learning methods [120], more efficient ansatz designs [157], and layer-wise learning [201]. The methods for which data are encoded into the VQC are also of importance in effective learning applications [161] [188].

The Quantum Approximate Optimization Algorithm (QAOA [63]) is another variational hybrid quantum-classical algorithm designed to solve combinatorial optimization problems. Given a cost Hamiltonian H_C, QAOA aims to approximate the ground state of H_C, corresponding to the optimal solution of the problem.

The algorithm uses a parameterized quantum circuit composed of alternating applications of two types of unitary operators:

The problem unitary $U(H_C, \gamma) = e^{-i\gamma H_C}$, which evolves the state under the cost Hamiltonian H_C for a duration γ. The mixing unitary $U(H_B, \beta) = e^{-i\beta H_B}$, with H_B typically being the transverse field Hamiltonian, and β the duration of this evolution. The QAOA circuit is constructed as follows:

$$|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = \prod_{p=1}^{P} U(H_B, \beta_p) U(H_C, \gamma_p) |\psi_0\rangle \tag{2.39}$$

Here, $|\psi_0\rangle$ is the initial state, often an equal superposition of all states, P is the number of QAOA layers, and $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_P)$, $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_P)$ are the parameters to be optimized. We give an example circuit diagram showing the structure of the QAOA circuit in Figure 2.9.

The goal of QAOA is to find optimal values of $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ that minimize the expectation value of the cost Hamiltonian:

$$C(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \langle \psi(\boldsymbol{\gamma}, \boldsymbol{\beta}) | H_C | \psi(\boldsymbol{\gamma}, \boldsymbol{\beta}) \rangle \tag{2.40}$$

Optimization of $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ is typically performed using classical optimization techniques. The optimal parameters produce a quantum state that approximates the ground state of H_C, providing a solution to the combinatorial optimization problem. QAOA is of particular interest as a path toward encoding and optimizing Ising models and related QUBO formulations.

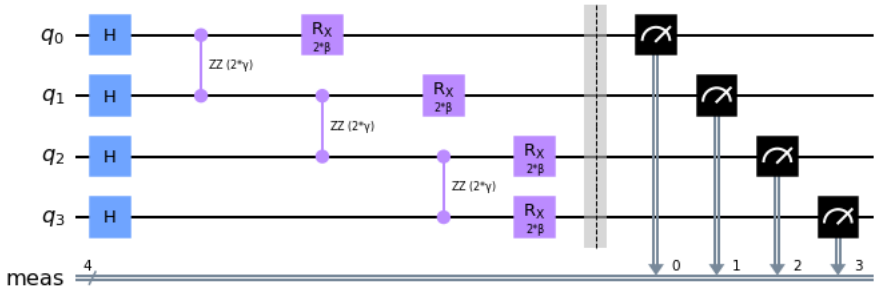

Figure 2.9: Circuit Diagram for QAOA [63].

2.11 Quantum-Inspired and Hybrid Methods

Quantum-inspired machine learning is an area of research that draws on the motivation and principles of quantum mechanics to enhance classical machine learning techniques. One of the areas of quantum-inspired machine learning centers on the use of tensor networks [168], which are mathematical structures originally developed for quantum many-body physics. These tensor networks have been applied effectively to machine learning tasks, offering a new toolkit for handling complex data and models [45].

In the realm of high-energy physics, for instance, tensor network techniques have been employed to analyze data from particle collisions [10]. By representing data samples as product states and mapping features into quantum spins, tensor networks allow the researcher to investigate scattering dynamics of quantum systems.

Tensor Train (TT) optimization is a technique based on Tensor Train decomposition [153] used for the efficient representation and manipulation of high-dimensional tensors. We define a tensor train optimization similar to [153, 179]. Consider a high-dimensional tensor A of order d with dimensions $n_1, n_2, \ldots, n_d$. This tensor can be represented as a multidimensional array $a_{i_1,i_2,\ldots,i_d}$, where $1 \leq i_k \leq n_k$ for $k = 1, 2, \ldots, d$.

In the Tensor Train format, the tensor A is decomposed into a sequence of lower-dimensional tensors, known as "cores", such that:

$$A(i_1, i_2, \ldots, i_d) = G_1(i_1) \cdot G_2(i_2) \cdots G_d(i_d) \tag{2.41}$$

Here, each $G_k(i_k)$ is a matrix of size $r_{k-1} \times r_k$, and $r_0 = r_d = 1$. The values $r_1, r_2, \ldots, r_{d-1}$ are the TT-ranks, which determine the size of the TT decomposition, and the matrices $G_k(i_k)$ are the TT-cores.

The TT format significantly reduces computational complexity and storage required for high-dimensional tensors. The optimization in the TT format involves finding TT-cores that provide the best approximation of the original tensor, often reducing its dimensions.

This technique is particularly useful in machine learning and data science, where the "curse of dimensionality" may present critical challenges to overcome. It has been applied in tasks such as tensor completion [36], tensor regression [198], and neural network compression [143].

This quantum-inspired technique for machine learning has the potential to have significant impact on the way complex datasets are analyzed and interpreted, leading to more accurate models that can be trained more efficiently. As research continues to unfold, the integration of quantum-inspired methods into mainstream machine learning promises to unlock new capabilities and insights in various domains [179].

2.12 Summary

In this chapter, we have given a broad overview of some of the topics touched upon in this thesis. We started with a refresher on the various areas of machine learning, covering some of the concepts and models that will be used later in the thesis. We cover optimization, highlighting QUBO, which is also a key ingredient in this work. After giving an introduction to AutoML, an area of machine learning that combines optimization to aid in the process of training and deploying machine learning models, we gave an overview of the no-free-lunch theorems and discussed their implications.

As the goals of this thesis are to investigate intersecting areas of quantum computing and machine learning, we provided a brief overview of the origins of quantum computing via quantum mechanics. We discussed computational complexity classes and highlighted classes where access to quantum computation might prove beneficial to a machine learning practitioner. We provide an overview of the various modes of quantum computing, including the universal gate model and quantum annealing. We discussed some of the constraints of applying quantum algorithms on near-term hardware. We give an overview of the area of quantum machine learning, discussing quantum kernel methods, which will be covered later in this thesis. We included some discussion on variational quantum circuits and the role of data encoding into quantum systems. Finally, we covered topics in quantum-inspired machine learning, which we consider a subfield of quantum machine learning, highlighting tensor train techniques to aid in dimensionality reduction in machine learning applications.