UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

Jogi Suda Neto

Parkinson, Chaos & Quantum Computing

São José do Rio Preto
2023

Jogi Suda Neto

Parkinson, Chaos & Quantum Computing

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação junto ao Programa de Pós-Graduação Interunidades em Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho” - Campus de São José do Rio Preto.

Orientador: Prof Dr Rodrigo Capobianco Guido

São José do Rio Preto
2023

S943p

Suda Neto, Jogi
Parkinson, Chaos & Quantum Computing / Jogi Suda Neto. -- São José do Rio Preto, 2023
80 p.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientador: Rodrigo Capobianco Guido

1. Inteligencia artificial. 2. Computadores quânticos. 3. Doença de Parkinson. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Jogi Suda Neto

Parkinson, Chaos & Quantum Computing

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação Interunidades em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho" - Campus de São José do Rio Preto.

Orientador: Prof Dr Rodrigo Capobianco Guido

Comissão Examinadora

Prof Dr Rodrigo Capobianco Guido
UNESP - Campus de São José do Rio Preto
Orientador

Profª Dra. Renata Spolon Lobato
UNESP - Campus de São José do Rio Preto

Prof Dr Paulo Rogério Scalassara.
Universidade Tecnológica Federal do Paraná, Câmpus Cornélio Procópio

São José do Rio Preto
06 de setembro de 2023

DEDICATÓRIA

Ao meu tio.

ABSTRACT

Signal processing models and artificial intelligence algorithms have benefited from the advance of Quantum Computing. Nevertheless, just modest possibilities have been seen in that sense for speech and voice analysis. Thus, the intention of the author of this dissertation is to investigate whether Quantum Machine Learning models might perform well in classifying Parkinsonian speech. Particularly, since the speech database used in the experiments produced just a few features, available current quantum models are feasible for the proposed approach. The results, up to this point, stimulate further investigation and suggest prominent perspectives.

Key-words: Quantum machine learning. Parkinson's disease. Digital signal processing. Voice analysis

RESUMO

Modelos de processamento de sinais e algoritmos de inteligência artificial se beneficiaram com o avanço da Computação Quântica. No entanto, apenas possibilidades modestas foram vistas nesse sentido para a análise da fala e da voz. Assim, a intenção do autor desta dissertação é investigar se os modelos de Aprendizado de Máquina Quântica podem ter um bom desempenho na classificação da fala parkinsoniana. Particularmente, como o banco de dados de fala utilizado nos experimentos contém poucas características, os modelos quânticos atuais disponíveis são viáveis para a abordagem proposta. Os resultados, até aqui, estimulam futuras investigações e sugerem perspectivas proeminentes.

palavras-chave: Aprendizado de máquina quântica. Doença de Parkinson. Processamento de sinais digitais. Análise de voz

FIGURES

TABLES

SUMMARY

SUMMARY

Chapter 1

Introduction

Since Parkinson's disease was discovered in 1817 by James Parkinson, the first doctor who profoundly studied it, there has been a tremendous progress. As mankind's scientific knowledge gradually evolved, neuroscience has been able to incorporate different ideas coming from multiple areas. With novel digital speech processing (DSP) and machine learning models, new and non-invasive diagnoses for neurodegenerative diseases have become possible, as shown in [1, 2, 3]. In those papers, we observe that handcrafted feature extraction strategies can detect Parkinson successfully.

Concomitantly with the deep learning revolution, the new field of quantum physics emerged, originating the quantum computation and the quantum information theory. They expanded previous possibilities, including new ways of transmitting, storing, and processing information [4, 5, 6]. In spite of the considerable growth of that field, processing large amounts of quantum information is still a difficult task. Indeed, environments interact with data and, thus, sustainably blocking undesirable interactions for thousands of qubits is still a pursuit of the scientific community. Given this scenario, a new promising possibility have arisen recently: the noisy intermediate-scale quantum (NISQ) devices [7]. In contrast to early quantum algorithms, when dealing with NISQ devices, only hundreds of noisy qubits are required. Although universal quantum computing for NISQ devices does not yet exist, specific computing tasks can be operated on such platforms. An important example is the method adopted in this dissertation, i.e., the variational quantum algorithms (VQA) [8] that can deal with practical problems such as combinatorial optimization and machine learning. VQA is built through a properly parameterized operation whose parameterization is obtained by means of a

classical computer.

Consequently, the author of this dissertation is inspired by recent results suggesting a connection between alpha-synuclein (αsyn) [9, 10] and deterioration of some speech parameters, considering a zebra finch (a bird breed) neural model [11], which is similar to the human pathway. This could lead to the possibility of early Parkinson's diagnosis with non-invasive methods that require only ordinary speech recordings, by using smartphones, for instance. We specifically discuss how VQAs compare against classical machine learning methods, when features coming from nonlinear dynamical systems theory are used.

Chapter 2

Theoretical Background

2.1 Parkinson's Disease (PD)

Parkinson's Disease (PD) was first medically studied in 1817 by James Parkinson, where pathological symptoms were described as a "shaking palsy" since they were characterized by involuntary shaking movements, diminished muscular power, and even affected intellectual functions, just to cite a few [12]. Nowadays, we know it is a neurodegenerative disease associated with the loss of dopaminergic neurons [13].

One major symptom of PD is bradykinesia [14], characterized by slowness or reduction in both automatic and intentional movements, such as blinking and trying to grab a glass of water. Another known problem is dyskinesia [15], which by definition is the erroneous involuntary motion in people, however, unlike bradykinesia, dyskinesia is not a symptom of the disease itself but a consequence of some medications used for treatment of the disease.

In this dissertation, speech impairment symptoms are the focus. As to be explained in subsection 2.2.1, new studies have found evidence of linkings between early-stage Parkinson's disease and speech disorders [11]. From this and the development of both ML and QML, in the future, a first screening stage could be performed in fast and non-invasive ways, just requiring speech recordings, such as those coming from smartphones.

2.1.1 Causes of Parkinson

Frédéric Lewy discovered in 1997 a major breakthrough: Lewy bodies [16]. They are abnormal protein aggregates that develop inside nerve cells, primarily in the brain. The principal consitutuent of Lewy bodies is a protein called α-synuclein, which is found in all neurons and is involved in maintaining synaptic function [16], although its exact functions are still unknown. However, when α-synuclein accumulates and forms aggregates, it can lead to the formation of Lewy bodies, which are in turn associated with several neurodegenerative diseases, including, PD and dementia with Lewy bodies (DLB) [17]. Depending on where those bodies accumulate, different symptoms may occur. In general:

1. sleep problems, depression, bladder disregulation, unstable temperature control, and blood pressure problems are linked to the presence of Lewy bodies in the *brain stem*.

2. the traditional motor symptoms such as tremor, muscle rigidness or slowness are associated with the presence of Lewy bodies in the *substantia nigra*.

3. memory, spatial navigation, planning, multitasking, delusions, and anxiety disorders can arise when Lewy bodies aggregate in the *cortex*, which is responsible for cognition.

Also, several lines of evidence suggest a deep connection between Lewy bodies and Parkinson's disease:

1. Genetic factors: Mutations in the SNCA gene [18], which encodes the α-synuclein protein, have been linked to familial forms of PD [10, 18]. These mutations can lead to an overproduction of α-synuclein or the production of a misfolded protein, both of which can promote the formation of Lewy bodies [9].

2. Spreading of pathology: Recent studies have shown that α-synuclein aggregates can spread from one neuron to another, leading to the propagation of Lewy body pathology throughout the brain [19]. This spreading may contribute to the progression of PD symptoms and the involvement of different brain regions over time.

3. Cellular dysfunction: The accumulation of α-synuclein and the formation of Lewy bodies can disrupt various cellular processes, such as protein degradation, mitochondrial function, and synaptic transmission, ultimately leading to neuronal dysfunction and death [20].

4. Inflammation: The presence of Lewy bodies can trigger an inflammatory response in the brain, which may contribute to the neurodegenerative process in PD [21].

2.1.2 α-synuclein and speech impairment

Studying vocal impairments in PD has been challenging due to the lack of well-defined circuitry in rodent models. For this reason, in a recent research work [11], a male zebra finch bird model was used to study vocal changes due to the overexpression of SNCA human gene in the α-fetoprotein (AFP) region of the brain. The major results are:

1. soluble αsyn expression is positively correlated with song production.

2. αsyn overexpression changes the acoustic structure of song depending on syllable type.

3. αsyn overexpression leads to changes in self-similarity of syllables depending on syllable type.

In other words, it was hypothesized that overexpression of the human wild αsyn gene in the so-called area X of the male birds - which is located in the basal ganglia and is directly linked to their singing ability [22] - leads to poorer quality syllables, similar to vocal abnormalities observed in individuals with PD. This motivates our work to develop non-invasive computational diagnostic methods that can be used in the future to assess patients with potentially early stages of the disease.

2.1.3 The main treatments used in the XIX century

Since the first essay from James Parkinson until almost a century later, the main therapies for parkinsonian patients were physical-based [12]. Jean-Martin Charcot was

a doctor from the 1800s that extensively studied and spread information about Parkinson's disease, making important contributions to our understanding of the pathology [23]. He had tried testing his patients with multiple pharmacological compounds [23], without success, leading him to state, *ipsis litteris*:

> *Everything, or almost everything, has been tried against this disease. Among the medicinal substances that have been extolled and which I have myself administered to no avail, I need only enumerate a few.*

Having given up on trying medicinal substances for finding an optimal strategy against that disease, Charcot decided to focus on physical treatments. He noticed that patients who went through horse and train rides had diminished symptoms afterwards. This lead him to design an electrically-powered vibratory device that gave periodic mechanical stimulus. Later, a vibratory helmet was proposed by one of his students, Gilles de la Tourette, as in Figure 1. Some other treatments used at that time also included spa therapy and hydrotherapy [12].

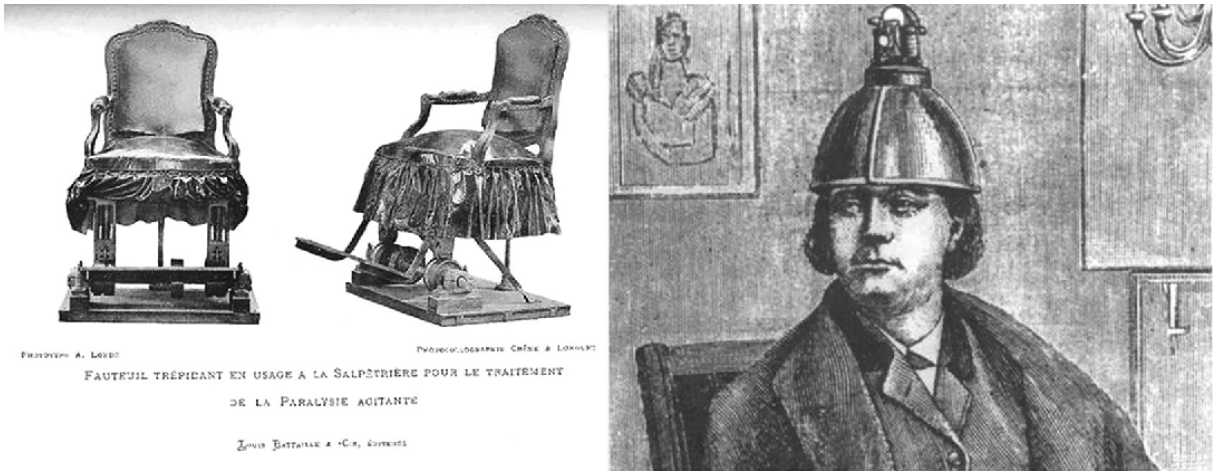


Figure 1: [left]: the chair proposed by Charcot to emulate a rhythmic movement in the Parkinsonian patients; [right]: a vibratory Helmet proposed by Guilles de la Tourette, one of his students. Image found in [24]

.

2.1.4 Levodopa and dopamine-based therapies

The first production of dopamine dates back to 1910. It was synthesized by G. Barger and J. Ewens [12]. Later, however, Arvid Carlsson [25] discovered it to be an independent neurotransmitter in the brain - an observation that would make a

tremendous contribution to neuroscience as we know today. In the late 1950s, Carlsson et. al [26] also discovered that akinetic effects of reserpine - a chemical that induces parkinsonian features [27, 28] - could be **reversed** through intravenous injection of the dopamine precursor: 3,4-dihydroxyphenylalanine (DOPA).

Hornykiewicz, in 1959, together with Herbert Ehringer, conducted studies [29] on the dopamine distribution in post-mortem brains, concluding that dopamine was severely reduced in the caudate and putamen in patients with PD, compared to the healthy ones. With these observations in hand, and having received a supply of Levodopa from Hornykiewicz, Birkmayer, for the first time, started trials in parkinsonians, injecting it in them intravenously, and then stated [29]:

> *Bed-ridden patients who were unable to sit up, patients who could not stand up when seated, and patients who when standing could not start walking performed all these activities with ease after L-dopa [levodopa]. They walked around with normal associated movements and they could even run and jump. The voiceless, aphonic speech, blurred by pallilalia and unclear articulation, became forceful and clear as in a normal person (Birkmayer and Hornykiewicz, 1961)*.

The interest of researchers in dopamine at that time, however, would only grow when George Cotzias proposed, in 1967, the first clinically efficient oral treatment of DOPA that is still used nowadays. On a side note, it is also interesting to see how early-stages medicine might also have benefited from Levodopa, as Masabaldi Pacana - a plant that contains beans of *Mucuna Pruriens* which naturally contains Levodopa - was the main therapy used at that time by the Indians [30] for treating a condition thought to potentially be what we know today as PD.

Finally, the motivation of this work is to study how quantum machine learning can be used for non-invasive diagnosis of PD through speech. Given the introduction to PD and its pathological effects on speech, classical machine learning first will be discussed in the next section, before discussing further quantum computing and quantum machine learning.

2.2 Classical Machine Learning

For all models explained in this section, we shall mathematically represent the dataset as $D = \{(x_i, y_i) \mid x_i \in \mathbb{R}^n, y_i \in \{0, 1\}\}$. Here, x_i is known as a feature vector, with n dimensions, where each of its components are its features; in the same context, y_i represents the class (binary, in this case) of x_i. Finally, the task at hand is to classify each new data point x (patient's speech extracted features) as being either 0 (healthy) or 1 (pathological).

2.2.1 K-Nearest Neighbors (KNN)

Starting with one of the most traditional machine learning algorithms, the KNN is presented here [31]. Given a new data point x_j from a dataset S, the model calculates its closest K nearest neighbors, using some distance function (usually Euclidean):

$$d(x_i, x_j) = \sqrt{\sum_{p=1}^{n} (x_i^p - x_j^p)^2} \tag{2.1}$$

After computing all distances $d(\cdot, j)$, the closest K neighbors classes to x_j are selected:

$$f : \mathbb{N} \to S, \quad f(k) = \begin{cases} \max S, & k = 1, \\ \max\{s : s \in S, s < f(k-1)\}, & k > 1 \end{cases} \tag{2.2}$$

$$g : \mathbb{N} \to S, \quad g(k) = \begin{cases} \arg\max S, & k = 1, \\ \arg\max\{s : s \in S, s < f(k-1)\}, & k > 1 \end{cases} \tag{2.3}$$

$$\psi = \{y_{g(i)}\}, \quad i \in \bigcup_{i=1}^{K} \{g(i)\}, \tag{2.4}$$

where x_i^p represents the p-th component of x_i. The y_j label is predicted, then, according to the majority boolean function:

$$\langle y_1, \ldots, y_n \rangle = \text{Majority}\,(y_1, \ldots, y_n) = \left\lfloor \frac{1}{2} + \frac{(\sum_{i=1}^{n} y_i)}{n} \right\rfloor. \tag{2.5}$$

In this function, in case of a draw, it favors ones.

2.2.2 Logistic Regression

Logistic Regression (LR) is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome [32]. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). It is used to predict a binary outcome (1 / 0, yes / no, parkinsonian / healthy) given a set of independent variables.

The logistic regression model is given by the following equation:

$$P(Y = 1|X) = \frac{1}{1 + e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+...+\beta_n X_n)}},$$

where:

• $P(Y = 1|X)$ is the probability of the outcome being 1 (success) given the values of the independent variables $X_1, X_2, ..., X_n$.

• β_0 is the intercept term, which represents the log-odds of the outcome when all independent variables are equal to zero.

• $\beta_1, \beta_2, ..., \beta_n$ are the coefficients of the independent variables $X_1, X_2, ..., X_n$, which represent the change in the log-odds of the outcome for a one-unit increase in the corresponding independent variable.

The logistic function can be written as:

$$f(x) = \frac{1}{1 + e^{-x}}.$$

The logit function, which is the inverse of the logistic function, is given by:

$$\text{logit}(P) = \log \frac{P}{1 - P} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n.$$

To estimate the coefficients $\beta_0, \beta_1, ..., \beta_n$, we use the method of maximum likelihood estimation (MLE). The likelihood function for logistic regression is given by:

$$L(\beta) = \prod_{i=1}^{n} P_i^{y_i}(1 - P_i)^{1-y_i}$$

where:

• $P_i = P(Y = 1|X_i)$ is the probability of the outcome being 1 for the i-th observation.

• y_i is the actual outcome for the i-th observation (either 0 or 1).

The goal of MLE is to find the values of $\beta_0, \beta_1, ..., \beta_n$ that maximize the likelihood function. This is typically done using numerical optimization techniques, such as gradient descent [33] or the Newton-Raphson method [34].

In our tests, we use the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS - quasi-Newton) algorithm [35]. This is an unconstrained non-linear optimization algorithm, and the main idea is to approximate the Hessian matrix using a limited amount of memory, instead of storing the entire matrix. This makes it more scalable and efficient for high dimensional problems.

2.2.3 Multilayer Perceptron (MLP)

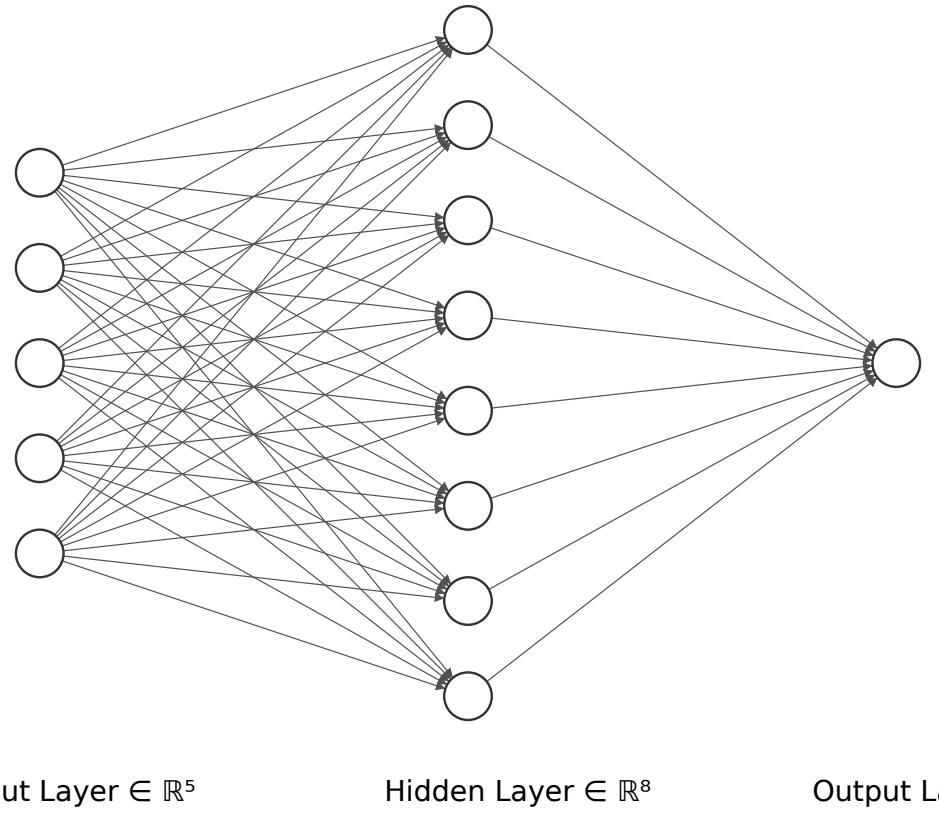


Figure 2: Multilayer Perceptron with one input layer, one hidden layer, and one output layer. Source: created by the author.

A Multilayer Perceptron (MLP) [32], a class of feedforward artificial neural networks, is a finite acyclic graph composed of multiple layers (neurons, or nodes) linked by weighted connections, which nowadays uses backpropagation [36] and optimization algorithms [37, 38] to find its optimal weights and biases during training by minimizing

the cost function. It has flexible architecture consisting of an input layer, one or more hidden layers, and an output layer. These models have been widely and successfully used in many applications, e.g.: in image and speech recognition [39, 40, 41], natural language processing [42, 43], among many other. The equations for an MLP can be described as follows:

1. Weighted sum of inputs:

   Each neuron computes a weighted sum of its inputs. Let x_i be the input values, w_{ij}^l be the weights connecting layer $l-1$ to layer l; z_j^l, the weighted sum for neuron j in the l-th layer; and finally, b_j^l, the bias associated to neuron j in layer l. Then, the weighted sum can be represented as:

   $$z_j^l = \sum_{i=1}^{n} w_{ij}^l x_i + b_j^l.$$

2. Activation function:

   An activation function is applied to the weighted sum to introduce non-linearity into the model. This is an extremely important step needed for MLPs to have universal function approximation capacity [44]. Common activation functions include the sigmoid function, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU). Let $\hat{y}_j^l$ be the output of neuron j in layer l after applying the activation function $f(\cdot)$. Then, the following holds:

   $$\hat{y}_j^l = f(z_j^l).$$

3. Loss function:

   To train the MLP in a supervised fashion, a loss function is used to measure how good the model's predictions are, usually by some difference between the predicted output and the true target values. Standard loss functions include the mean squared error (MSE) for regression tasks and the cross-entropy loss for classification tasks. Let t_k be the true target value for output neuron k, and L be the loss function. If some given MLP has l layers, then, the empirical loss can be represented as:

$$L = \sum_{j=1}^{c} L(\hat{y}_j^l, t_j),$$

where j is the number of output neurons in the last layer l.

4. Backpropagation:

   To update the weights of the MLP, the backpropagation algorithm is used. It computes the gradient of the loss function with respect to each weight by using the chain rule of calculus. The weights are then updated using gradient descent or other optimization algorithms. Formally, the weight correction Δw_{ij}^l can be written as [32]:

   $$\Delta w_{ij}^l = \eta \, \delta_j(n) \, \hat{y}^l i(n), \tag{2.6}$$

   where $\delta_j(n)$ is called the local gradient, and can be found by applying the chain rule [32].

This process is repeated for multiple epochs (iterations through the entire dataset) until the network converges to a solution with minimal test error.

A very interesting property in MLPs is explained by the Universal Approximation Theorem [44], which states that any continuous function can be approximated to an arbitrary degree of precision by some neural network with a single hidden layer, although it does not provide any orientation on finding the optimal network parameters, nor does it specify the number of neurons required in the hidden layer. It just states that such a network exists. Achieving better convergence, generalization, choosing the right inductive bias in the architecture, are all current topics of research on their own.

2.2.4 Support Vector Machine (SVM)

The main idea behind Support Vector Machines [45] is to find the optimal hyperplane that separates the data points of different classes with the maximum margin. The margin is defined as the distance between the hyperplane and the closest data points, which are called support vectors. The larger the margin, the better the generalization of the classifier.

Given a set of training data points $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ is a feature vector and $y_i \in \{-1, 1\}$ is the corresponding class label, the goal of SVM is to find the optimal

hyperplane that separates the data points of different classes. The hyperplane can be represented as:

$$\mathbf{w}^T\mathbf{x} + b = 0$$

where $\mathbf{w} \in \mathbb{R}^d$ is the weight vector and $b \in \mathbb{R}$ is the bias term, which controls the shift from the origin. The decision function for a new data point $\mathbf{x}$ is given by:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b). \tag{2.7}$$

To find the optimal hyperplane, we need to maximize the margin, which is given by:

$$\text{margin} = \frac{2}{\|\mathbf{w}\|}. \tag{2.8}$$

Maximizing the margin is equivalent to minimizing the norm of the weight vector $\|\mathbf{w}\|$. This can be formulated as the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, N \end{aligned}$$

This is a convex quadratic optimization problem with linear constraints, which can be solved using the method of Lagrange multipliers. The Lagrangian function is given by:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1], \tag{2.9}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)$ are the Lagrange multipliers. To find the optimal solution, we need to minimize $\mathcal{L}$ with respect to $\mathbf{w}$ and b, and maximize it with respect to $\boldsymbol{\alpha}$. Our original problem can be restated as:

$$p^* = \min_{\mathbf{w},b} \max_{\boldsymbol{\alpha}\geq 0} \quad \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}). \tag{2.10}$$

Where p^* is a reference to the optimal solution of the primal problem (our original optimization problem). It happens that by changing the order of this min-max problem, we have a new perspective:

$$d^* = \max_{\boldsymbol{\alpha}\geq 0} \min_{\mathbf{w},b} \quad \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}). \tag{2.11}$$

Here, d^* now is the solution to what is called the dual problem, and when the primal problem obeys the so-called Slater's conditions [46] (which happens for the SVM primal objective), then strong duality holds, and we say that $d^* = p^*$. Essentially, we are left with an optimization problem in terms of α at the end, and this brings advantages, in this new point-of-view: the use of kernels, as we will see. Taking the derivatives of $\mathcal{L}$ with respect to $\mathbf{w}$ and b, and setting them to zero, we get the following conditions:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i = 0,$$
$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{N} \alpha_i y_i = 0.$$

Substituting these conditions back into the Lagrangian, we obtain the dual problem:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & \alpha_i \geq 0, \quad i = 1, \ldots, N. \end{aligned}$$

This is a convex quadratic optimization problem with linear constraints, which can be solved using quadratic programming techniques, like the Sequential Minimal Optimization (SMO) [47]. Once the optimal Lagrange multipliers $\boldsymbol{\alpha}^*$ are found, the optimal weight vector and bias term can be computed as follows:

$$\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* y_i \mathbf{x}_i$$

$$b^* = y_k - \mathbf{w}^{*T} \mathbf{x}_k \quad \text{for any} \quad k \quad \text{such that} \quad \alpha_k^* > 0.$$

The support vectors are the data points $\mathbf{x}_i$ for which $\alpha_i^* > 0$. The decision function for a new data point $\mathbf{x}$ can be written in terms of the support vectors as:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i^* y_i \mathbf{x}_i^T \mathbf{x} + b^* \right).$$

In the case of non-linearly separable data, the dual formulation allows for a clever trick, so that the SVM algorithm can be extended by using feature maps $\phi : \mathbb{R}^n \to \mathbb{R}^d$ (where usually $d >> n$) to map the input data points into a higher-dimensional space, where they become linearly separable. In the primal form, these high dimensional vectors have to be computed explicitly, but this demands too much complexity, becoming unfeasible in pratice. In fact, sometimes ϕ might be an infinite-dimensional feature map, like the Gaussian Radial Basis Function (RBF) Kernel [45]. In the dual formulation, however, this is mitigated by the use of a kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ to implicitly compute the inner product between the transformed data points $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$ in a higher-dimensional space without ever having to calculate the feature maps directly. This is very powerful, and some popular kernel functions, like the Gaussian, are defined as:

1. Gaussian RBF Kernel:

   Feature map: $\phi(x) = e^{-\gamma \|x\|^2}$, where $\gamma > 0$ is a parameter.

   Kernel function: $K(x, y) = e^{-\gamma \|x-y\|^2}$

2. Laplacian Kernel:

   Feature map: $\phi(x) = e^{-\gamma \|x\|}$, where $\gamma > 0$ is a parameter.

   Kernel function: $K(x, y) = e^{-\gamma \|x-y\|}$

3. Sigmoid Kernel:

   Feature map: $\phi(x) = \tanh(\gamma x + c)$, where $\gamma > 0$ and c is a constant.

   Kernel function: $K(x, y) = \tanh(\gamma \langle x, y \rangle + c)$

The dual problem can then be reformulated, in terms of the kernels, as:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & \alpha_i \geq 0, \quad i = 1, \ldots, N. \end{aligned}$$

And the decision function for a new data point $\mathbf{x}$ in the kernelized SVM is given by:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}) + b^* \right).$$

Now that classical machine learning models were briefly discussed, we present some popular optimization algorithms in the next subsection.

2.2.5 Optimization methods

Here we will discuss briefly some supervised optimization methods used both in classical machine learning and quantum machine learning:

Stochastic Gradient Descent (SGD): to start with, the SGD [37] is an optimization algorithm used to minimize an objective function that is written as a sum of differentiable functions. It is widely used to this day for training neural networks. The main idea of this technique is to update the model's parameters iteratively using a random subset of the training data, instead of the entire dataset, which requires less memory complexity, hence making it computationally more efficient than the standard Gradient Descent (GD) algorithm, especially for large-scale and high-dimensional problems.

The main equations for SGD are the objective function, the gradient approximation, and the update rule: first of all, let's consider an objective function $J(\theta)$ that we want to minimize, where θ represents the model's parameters. The objective function can then be written as a sum of differentiable functions:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i, \theta)).$$

Here, N is the number of training examples, $L(y_i, f(x_i, \theta))$ is the loss function that mea-

sures the discrepancy between the true label y_i and the model's prediction $f(x_i, \theta)$ for the i-th training example (x_i, y_i).

In Gradient Descent, we update the model's parameters by moving in the direction of the negative gradient of the objective function with respect to the parameters:

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t)$$

where η is the learning rate, and $\nabla J(\theta_t)$ is the gradient of the objective function at the current parameter values θ_t. Computing the gradient requires calculating the average of the gradients of the loss function for all training examples:

$$\nabla J(\theta_t) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(y_i, f(x_i, \theta_t)).$$

However, in Stochastic Gradient Descent, we approximate the true gradient by computing the gradient of the loss function for a random subset of the training data, called a mini-batch:

$$\nabla J(\theta_t) \approx \frac{1}{M} \sum_{i=1}^{M} \nabla L(y_i, f(x_i, \theta_t)),$$

where M is the size of the mini-batch, and $M \ll N$. The mini-batch is randomly sampled at each iteration, which introduces some noise in the gradient estimation. This noise can help escape local minima and converge faster to the global minimum.

The update rule for Stochastic Gradient Descent is:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{M} \sum_{i=1}^{M} \nabla L(y_i, f(x_i, \theta_t)).$$

Adaptive Moment Estimation (ADAM): The ADAM [38] method is an adaptive learning rate optimization algorithm designed for training deep neural networks, and has succesfully been used in many works, like in large language model training (GPT-3) [42], in vision transformers [41] and also in split-attention networks [39]. It combines the advantages of two popular optimization techniques: AdaGrad and RMSProp. This method computes adaptive learning rates for each parameter by considering the first and second moments of the gradients.

Given the loss function $L(\theta)$, where θ_t is the parameter vector at discrete-time t,

then the moments are estimated as follows:

1. First moment estimation:

   ADAM maintains an exponentially decaying average of past gradients, which is similar to momentum. The first moment m_t is updated as follows:

   $$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\frac{\partial L}{\partial \omega_t} \tag{2.12}$$

   Here, m_t and m_{t-1} are the first moments at time steps t and $t-1$, respectively, and β_1 is the exponential decay rate for the first moment estimates.

2. Second moment estimation:

   ADAM also maintains an exponentially decaying average of past squared gradients, which is similar to RMSProp. The second moment v_t is updated as follows:

   $$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\left(\frac{\partial L}{\partial \omega_t}\right)^2 \tag{2.13}$$

   Where v_t and v_{t-1} are the second moments at time steps t and $t-1$, respectively, and β_2 is the exponential decay rate for the second moment estimates.

3. Bias correction:

   Since both m_t and v_t are initialized as zero vectors, they are biased towards zero, especially during the initial time steps. To counteract this bias, we perform bias correction for both first and second moment estimates:

   $$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{2.14}$$

   $$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{2.15}$$

   Now, $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected first and second moment estimates, respectively, and t is the current time step.

4. Parameter update:

Finally, we update the parameters using the bias-corrected first and second moment estimates:

$$\theta_{t+1} = \theta_t - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}.$$

Here, θ_{t+1} is the updated parameter vector, α is the learning rate, and ϵ is a small constant to prevent numerical errors, that is, division by zero (usually set to 10^{-8}).

In summary, ADAM adapts the learning rate for each parameter by considering the first and second moments of the gradients. It combines the advantages of AdaGrad and RMSProp, making it suitable for training deep neural networks.

2.2.6 Validation metrics

Having reliable and robust statistical measures of the machine learning model performance is necessary, and for a medical dataset, specifically, the right measures should be selected very carefully, since true/false positives/negatives have each different relevances depending on the pathology in analysis. To put into context:

1. **Cost of False Positives:** in medical applications, false positive predictions can have significant consequences, such as unnecessary treatments, interventions, or further testing. These can lead to increased healthcare costs, patient anxiety, and potential harm to patients.

2. **Cost of False Negatives:** on the other hand, false negatives predictions can also be devastating: it is enough to imagine that, by a false negative mistake, a classifier sends home a patient with some specific cancer which, if treated right, leads to a complete cure, but if not, leads to certain death.

3. **Imbalanced Datasets:** medical datasets often exhibit class imbalance, where the number of instances of one class (e.g., pathological patients) is significantly lower than the number of instances of the other class (healthy ones, for example). In such cases, "naive" statistical measures, like accuracy alone, are bad indicators of performance, since it does not look into true/false positive/negative rates, as will be explained further.

Thus, having the right measures is extremely important for medical applications, as it avoids common statistical pitfalls. These measures are known as validation metrics in the machine learning jargon. Here is an overview of such metrics used in this study:

1. Accuracy: accuracy is a simple metric that calculates the proportion of correct predictions out of the total predictions made. In medical dataset classification tasks, it is vital to ensure the model can accurately identify the existence or nonexistence of a medical condition. However, it can be deceptive in cases with imbalanced datasets, where the number of instances of one class significantly outweighs the other. In this case, a high accuracy might indicate that a model is biased towards the majority class.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.16}$$

2. Recall: Also known as sensitivity or true positive rate (TPR), recall measures the proportion of true positive predictions out of all actual positive instances. In medical dataset classification tasks, recall is essential as it evaluates the model's ability to correctly identify positive cases, such as patients with a specific disease, minimizing false negatives. Due to the severe consequences of false negatives in medical applications, recall is very often prioritized over other metrics. Formally, it is written as:

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{2.17}$$

3. Specificity: the true negative rate (TNR), or specificity, is the ratio of the number of negative instances correctly classified to the total number of actual negative instances. It measures how good is the capacity of a model to identify negative instances (e.g., healthy patients). Mathematically:

$$\text{Specificity} = \frac{TN}{TN + FP}. \tag{2.18}$$

Although recall is often overlooked in medical classification tasks, a model with high TNR is also desirable, to avoid "false alarms" and unnecessary treatments.

4. Precision: defined as the ratio of true positive predictions (TP) to the total number of positive predictions made by the classifier, which includes both true positive predictions (TP) and false positive predictions (FP), precision is important to minimize the cost of false positive predictions. It is mathematically defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{2.19}$$

5. F1-score: F1-score is the harmonic mean of precision and recall, providing a single metric that balances both false positives and false negatives. It is particularly valuable when dealing with imbalanced datasets. In medical dataset classification tasks, F1-score is an important metric as it helps assess the trade-off between precision and recall, ensuring a good balance between identifying true positive cases and minimizing false positives and false negatives. The F1-score is defined as:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}. \tag{2.20}$$

F1-score is particularly useful in situations where there is an uneven class distribution, such as medical datasets, as it seeks a balance between precision and recall. A high F1-score indicates that the model has both high precision and high recall, a good indicator. So, in our experiments, for simplicity, we make use both of accuracy and F1, to capture more nuanced information about our models.

2.2.7 K-fold cross validation

The main idea of K-fold cross validation is to have a more robust estimation of a model's predictive performance, especially when the dataset size is small [48]. This technique enhances the reliability of performance estimation by dividing the dataset into K equally sized subsets or "folds" and training and evaluating the model K times, using different folds as validation sets and the remaining K-1 folds as training sets.

The K-fold cross validation process consists of the following steps:

1. Partition the dataset into K equally sized subsets (folds): The dataset is randomly divided into K subsets, ensuring each data point belongs to only one fold. The value of K depends on the dataset size and the balance between computational cost and estimation accuracy, with common choices being 5 or 10.

2. Perform K iterations of model training and evaluation: For each iteration ($i = 1, 2, ..., K$), the model is trained using data from $K - 1$ folds and evaluated on the remaining fold, which acts as the validation set. This ensures each data point is used once in the validation set and $K - 1$ times in the training set.

3. Compute performance metrics: Metrics such as accuracy, precision, recall, or F1-score are calculated for each iteration based on the model's predictions on the validation set, estimating the model's generalization to unseen data.

4. Average the performance metrics: After completing all K iterations, the performance metrics are averaged to obtain a single performance estimate for the model. This average performance is considered a more reliable indicator of the model's true performance, as it is less influenced by the specific choice of training and validation sets.

5. Model selection and hyperparameter tuning: K-fold cross validation can be employed to compare different models or fine-tune hyperparameters of a single model. By comparing average performance metrics across various models or hyperparameter settings, the best-performing model or settings can be selected.

The primary benefit is a more accurate and reliable performance estimate compared to a single train-test split, as it minimizes the effect of random data variations. However, this method also increases computational costs, as the model must be trained and evaluated K times. Despite this limitation, K-fold cross validation remains a popular and widely used technique for model evaluation and selection in machine learning and statistics.

After the explained introduction to the classical machine learning methods used in this work, and before diving further into the realm of quantum machine learning, a brief introduction to the principles of quantum mechanics and quantum computing are given in the next section.

2.3 Introduction to Quantum Computing

Qubits - quantum bits - are two-level systems that leverage quantum effects, such as superposition and entanglement, to store and process information [6]. Just like regular bits are the atomic units of classical computation, qubits are the analogous for quantum computing. But, different from classical bits, they can exist in infinite states of superposition, that is, like being both 0 and 1 units of classical information at the same time. To begin our discussion, we introduce the mathematical background needed to describe these systems.

2.3.1 Dirac notation

A discrete quantum system with n distinguishable states is represented by a unit vector in a complex vector space $\mathbb{C}^n$, where the n distinguishable states form an orthonormal base in this space. From now on, we also introduce the **Dirac's braket notation**, a specific way of representing vectors [6]. In this notation, we can say that a vector $a \in V$, where V is a vector space, can be represented as :

$$|\, a\rangle = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}, \tag{2.21}$$

which is also called the *ket*. On the other hand, *bras* are linear maps from V to $\mathbb{C}$, and they belong to the space V^* dual to V. A simple way of interpreting the *bras*, is that they are the transpose conjugate of their corresponding *kets*, i.e:

$$\langle a\,| = (a_1^*, a_2^* \ldots, a_n^*)\,. \tag{2.22}$$

2.3.2 Matrix operations

To define some important operations on matrices, we start with the following: adjoint($A^\dagger$) and trace($Tr(A)$), where $A \in \mathcal{M}(\mathbb{C}^d)$ is a complex matrix. The adjoint opera-

tion, then, is defined as taking the transpose of the complex conjugate of A:

$$A^*(i,j) = (A(i,j))^*, \quad A^T(i,j) = A(j,i), \quad A^\dagger = (A^*)^T, \tag{2.23}$$

while the trace is defined as the linear map $Tr : \mathcal{L}(\mathbb{C}^d) \to \mathbb{C}$. This operation sums the diagonal entries of A:

$$Tr(A) = \sum_{i=1}^{d} A(i,i). \tag{2.24}$$

One interesting property of the trace that often comes in hand is that it is cyclic: for example, for three matrices $A, B, C \in \mathcal{M}(\mathbb{C}^d)$, then $Tr(ABC) = Tr(CAB) = Tr(BCA)$. From this, it holds that even if A and B do not commute ($AB \neq BA$), the traces of their products are equal: $Tr(AB) = Tr(BA)$.

2.3.3 Inner & outer product

In quantum computing, inner and outer products play a crucial role in the manipulation and understanding of quantum states and operations. These products are used to describe the relationships between quantum states, as well as to perform calculations on quantum systems. Here, we will elaborate on the importance of inner and outer products in quantum computing.

Inner Product:

The inner product is a binary operation that takes two vectors and returns a scalar. In quantum computing, it is used to calculate the probability amplitudes and the probability of measuring a specific state. Given two vectors $|\psi\rangle$ and $|\phi\rangle$, the inner product is denoted as $\langle\psi|\phi\rangle$ and is defined as:

$$\langle\psi|\phi\rangle = \sum_i \psi_i^* \phi_i, \tag{2.25}$$

where ψ_i^* is the complex conjugate of the i-th component of the vector $|\psi\rangle$.

The importance of the inner product in quantum computing lies in its ability to:

1. Determine the probability amplitudes of states and transitions: The inner product of two quantum states gives the probability amplitude of transitioning from one state to another. This is important for understanding the dynamics of quantum systems.

2. Determine the orthogonality of quantum states: If the inner product of two

quantum states is zero, the states are orthogonal, which means they are independent and can be used as a basis for a quantum system.

3. Measure the similarity between quantum states: The inner product is also used to quantify the similarity between two quantum states. A larger inner product indicates a higher degree of similarity.

Outer Product:

The outer product is a binary operation that takes two vectors and returns a matrix. In quantum computing, for example, it is very often used to describe quantum gates and also density matrices, which are used to describe mixed quantum states and are essential for understanding the behavior of open quantum systems. Given two vectors $|\psi\rangle$ and $|\phi\rangle$, the outer product is denoted as $|\psi\rangle\langle\phi|$ and is defined as:

$$|\psi\rangle\langle\phi| = \begin{pmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_n \end{pmatrix} \begin{pmatrix} \phi_1^* & \phi_2^* & \cdots & \phi_n^* \end{pmatrix} \tag{2.26}$$

Essentially, inner and outer products are fundamental in the context of quantum computing, as they allow for the manipulation and understanding of quantum states, similarities and operations.

2.3.4 Eigenvectors & eigenvalues

For any given matrix $A \in \mathcal{L}(\mathbb{C}^d)$, an eigenvector is a special nonzero type of vector that satisfies:

$$A|\psi\rangle = \lambda|\psi\rangle. \tag{2.27}$$

Where $\lambda \in \mathbb{C}$ is known as its corresponding eigenvalue. Eigenvalues and eigenvectors play a crucial role in quantum computing, as they are fundamental to the understanding of quantum systems and their behavior. Eigenvalues and eigenvectors of operators (matrices) provide valuable insights into the properties of quantum systems and the transformations they undergo. As an example, the eigenvalues of a unitary matrix have a magnitude of 1, which means that the action of operator preserves the norm of the state vector. This property is crucial for maintaining the probabilistic inter-

pretation of quantum states.

As will be discussed further in the postulates of quantum mechanics, the state of a quantum system is represented by a vector $|\psi\rangle$ in a complex Hilbert space. The evolution of this state is governed by the Schrödinger equation [49]:

$$i\hbar\frac{d}{dt}|\psi(t)\rangle = H|\psi(t)\rangle, \tag{2.28}$$

where H is the Hamiltonian operator, which represents the total energy of the system, and $\hbar$ is the reduced Planck constant. The Hamiltonian is a Hermitian matrix, which means that its eigenvalues are real and its eigenvectors form an orthonormal basis for the Hilbert space.

Eigenvalues and eigenvectors of the Hamiltonian also have a direct physical interpretation: the eigenvalues represent the possible energy levels of the system, and the eigenvectors represent the corresponding stationary states. When a quantum system is in a stationary state, its energy is constant, and its state vector evolves only by a phase factor:

$$|\psi(t)\rangle = e^{-iE_n t/\hbar}|\psi_n\rangle, \tag{2.29}$$

where E_n is the eigenvalue corresponding to the eigenvector $|\psi_n\rangle$.

2.3.5 Spectral decomposition

Given the review on outer products, eigenvectors and eigenvalues at hand, then any normal matrix (when $A^\dagger A = AA^\dagger$) can be decomposed in terms of its eigenvalues and eigenvectors:

$$A = \sum_{i=1}^{d} \lambda_i |\lambda_i\rangle\langle\lambda_i|. \tag{2.30}$$

This is known as the spectral decomposition (or eigendecomposition, alternatively).

2.3.6 Operators

In the context of Quantum Mechanics, the operators are linear maps that act on the quantum states, represented by complex vectors [6]. These maps have an

equivalent matricial representation. There are three types of operators:

1. **Hermitian operators**: In quantum mechanics, a Hermitian operator is a linear operator that satisfies a specific condition known as Hermiticity. Let $\hat{A}$ be a Hermitian operator. Then, the Hermiticity condition states that the operator must satisfy the following property:

$$\hat{A}^{\dagger} = \hat{A},$$

where $\hat{A}^{\dagger}$ represents the Hermitian adjoint of $\hat{A}$. The Hermitian adjoint of an operator is obtained by taking the complex conjugate of its transpose. That is: $\hat{A}^{\dagger} = A^{T*}$. This condition ensures that the eigenvalues of a Hermitian operator are real, and the corresponding eigenvectors are orthogonal. In other words, if $\hat{A}$ is a Hermitian operator and λ is an eigenvalue of $\hat{A}$, then λ is a real number, and the eigenvectors corresponding to different eigenvalues are orthogonal.

Hermitian operators represent physical observables [6], like position, momentum, energy, and angular momentum. The eigenvalues of these operators correspond to the possible outcomes of measurements of the corresponding observables, while the eigenvectors represent the states of the system into which the wavefunction collapses after the measurement. As will be seen in the fundamental postulates of quantum mechanics, each eigenstate has an associated eigenvalue, which is outcome observed with some probability after the measurement.

Another important property to say is that the Hermiticity of operators ensures that the expectation values of observables are real. The expectation value of an observable $\hat{A}$ in a quantum state ψ is given by:

$$\langle \hat{A} \rangle = \langle \psi | \hat{A} | \psi \rangle.$$

Since the Hermitian adjoint of $\hat{A}$ is equal to $\hat{A}^{\dagger}$, the expectation value can be written as:

$$\langle \hat{A} \rangle = \langle \psi | \hat{A}^{\dagger} | \psi \rangle = \langle \psi | \hat{A} | \psi \rangle^{*}.$$

This shows that the expectation value is a real number.

2. **Positive semi-definite operators**: these are Hermitian operators that have non-negative eigenvalues [6]. They play a fundamental role in the description of physical systems. Formally, an operator A acting on a Hilbert space $\mathcal{H}$ is positive semi-definite if it satisfies the following conditions:

   (a) Hermiticity: $A = A^{\dagger}$.

   (b) Non-negativity of eigenvalues: All eigenvalues λ_i of A are non-negative, i.e., $\lambda_i \geq 0$.

   The positive semi-definite property of an operator A has important implications in quantum mechanics. For instance, it guarantees that the expectation value $\langle A \rangle$ of A in any quantum state is non-negative. This property arises from the non-negativity of eigenvalues, as the expectation value can be expressed as a sum of the eigenvalues weighted by the probabilities of measuring those eigenvalues.

   Furthermore, positive semi-definite operators are closely related to observables in quantum mechanics. An observable is represented by a Hermitian operator, and if this operator is positive semi-definite, it corresponds to a physical quantity that can be measured in a quantum system. The eigenvalues of the operator represent the possible outcomes of the measurement, and the eigenvectors correspond to the states in which the measurement yields a definite value.

   Positive semi-definite operators also find applications in the study of entanglement and quantum information theory. For example, the density matrix, which describes the statistical properties of a quantum system, is a positive semi-definite operator. The positivity of the density matrix ensures that the probabilities of different outcomes are non-negative, and it allows for the calculation of various entanglement measures and quantum correlations.

3. **Unitary operators**: Unitary operators preserve the lengths of vectors upon which they act in a vector space. They are used to describe quantum states and transformations. In the context of quantum mechanics, unitary operators are crucial because they preserve the probability interpretation of quantum states. The evolution of any closed quantum system over time is described by a unitary operator, for example. Mathematically, an operator U is said to be unitary if it satisfies two conditions:

(a) It is invertible, meaning there exists another operator U^{-1} such that $UU^{-1} = U^{-1}U = I$, where I is the identity operator.

(b) Its inverse is equal to its adjoint (or conjugate transpose), i.e., $U^{-1} = U^{\dagger}$.

Unitary operators also have the property that their eigenvalues are complex numbers of absolute value 1, and different eigenvectors corresponding to distinct eigenvalues are orthogonal.

2.3.7 States & Hilbert spaces

From this, the inner-product between two vectors $|a\rangle, |b\rangle \in V$ allows for the notion of angles between vectors, and norms, and it is given by:

$$\langle a \mid b\rangle = a_1^* b_1 + a_2^* b_2 + \ldots a_n^* b_n. \tag{2.31}$$

More formally, an inner-product on a complex vector space V $(\cdot,\cdot) : V \times V \to \mathbb{C}$ that, for each $u, v, w \in V$ and $\alpha, \beta \in \mathbb{C}$:

1. $(\vec{v}, \vec{v}) \geq 0$, and $(\vec{v}, \vec{v}) = 0$ if and only if $\vec{v} = \overrightarrow{0}$;
2. $(\alpha\vec{u} + \beta\vec{v}, \vec{w}) = \alpha(\vec{u}, \vec{w}) + \beta(\vec{v}, \vec{w})$;
3. $(\vec{v}, \vec{w}) = \overline{(\vec{w}, \vec{v})}$
4. $\vec{v}, \vec{w}$ are said to be orthogonal if the inner product $(\vec{v}, \vec{w}) = \langle v|w\rangle = 0$.

Such a vector space, equipped with an inner product, is an inner product space. A Hilbert space, then, is just an inner product space which is complete under its induced norm.

Back to the qubits, considering it is physically realised as an electron in a simple hydrogen atom, then we know it can be in the ground-state, represented by $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, in the excited state, $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, and in infinite possible linear combinations of $|0\rangle, |1\rangle$, that is:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \alpha, \beta \in \mathbb{C} \quad \text{and} \quad |\alpha|^2 + |\beta|^2 = 1,$$

where $|\psi\rangle$ is said to be the state of the qubit, and α and β are called *probability amplitudes*. Also, in note how $\{|0\rangle, |1\rangle\}$ form an orthonormal basis in $\mathbb{C}^2$. In general, the principal desirable properties of qubits are *interference*, *superposition*, *entanglement* and *measurement*.

2.3.8 Classical probabilities

In Kolmogorov's original formulation of 1933 [50], the mathematical theory of probability gained a unified formulation, since a universal structure of a *probability space* $(\Omega, \Sigma, \mathbb{P})$ was introduced, along with its axioms. In this framework, Ω is a set, Σ is a σ-algebra of subsets of sets of Ω, where Ω itself is an element, and $\mathbb{P}$ is a function $\Sigma \to [0, 1]$ satisfying the following criteria:

1. *normalisation*:

$$\mathbb{P}(\Omega) = 1 \tag{2.32}$$

2. *additivity*:

$$A \cap B = \emptyset \quad \Longrightarrow \quad \mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) \tag{2.33}$$

3. *continuity*:

$$A_1 \subset A_2 \subset A_3 \subset \cdots \quad \Longrightarrow \quad \mathbb{P}\left(\bigcup_{n=1}^{\infty} A_n\right) = \lim_{n\to\infty} \mathbb{P}\left(A_n\right). \tag{2.34}$$

In this framework, Ω is interpreted as the set of all possible *outcomes* from the experiment; Σ is the collection of *events*, or statements - a set of yes/not questions - about the outcomes; given any event A, then, $\mathbb{P}(A)$ is the *probability* that A will occur. If $(A_i)_{i \in I}$ is said to be an independent collection of events when, for all finite $J \subset I$, the following holds:

$$\mathbb{P}\left(\bigcap_{i\in J} A_i\right) = \prod_{i\in J} \mathbb{P}\left(A_i\right) \tag{2.35}$$

Finally, the **Weak Law of Large Numbers** states that, if $\{A_1, A_2, ..., A_N\}$ are all independent with probability p, then for any $\epsilon > 0$:

$$\begin{gathered} \lim_{n\to\infty} \mathbb{P}\left[\left|\frac{1}{n} K_n - p\right| < \varepsilon\right] = 1, \\ K_n(\omega) := \#\left\{j \in \{1, 2, \ldots, n\} \mid \omega \in A_j\right\}. \end{gathered} \tag{2.36}$$

The interpretation is that, after many trials of the same experiment, the success rate, i.e, the 'yes' rate, for A becomes each time closer to its true probability p. This allows us to empirically measure probabilities.

2.3.9 Quantum Probabilities

One of the most important discoveries in Quantum Theory was made by John Bell in his famous paper [51], while working at CERN. His paper confirms Quantum Theory, showing that there is no classical description for it, and eliminating hidden variable theories. Hence, the classic kolmogorovian definition of probabilities has to be redefined to fit the framework of quantum mechanics:

1. the sample space η is now the set of all possible one-dimensional projections of a *-algebra $\mathcal{A}$ of operators on a Hilbert space $\mathcal{H}$. So, each event is a closed subspace of $\mathcal{H}$.

2. the probability function $P : \eta \rightarrow [0, 1]$ must be σ-additive.

Essentially, a quantum probability distribution is described by a density operator, which is a Hermitian, positive semidefinite operator with unitary trace. This is also known as a quantum state. To see the analogy between these properties and classical distributions, let's see the properties:

1. **Hermiticity**: as was seen, the hermiticity condition ensures that the operator is equal to its transpose conjugate. As a consequence, its eigenvalues are always real. This is analogous to the fact the probabilities are always real values, not complex.

2. **Positive semidefinite**: remembering that an operator $f : V \rightarrow V$ is positive semidefinite if $\langle v, fv \rangle \geq 0, v \in V$, which intuitively means that f does not cause vectors to point in an opposite direction, then it preserves the probabilistic interpretation of states. In other words, probabilities are always greater than or equal to zero.

3. **Unitary trace**: finally, the trace of an operator is equal to the sum of its diagonal entries, and also of its eigenvalues. If trace must be unitary, then the analogy is that classical probabilities in a distribution always sum up to 1.

The interesting part is that every classical probability distribution defines a quantum state, and also every quantum state defines a classical distribution.

2.3.10 Quantum Gates

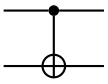
Quantum gates are the building blocks of parametrized quantum circuits, in the sense that they perform the most elementary operations on the qubits, just like classical logic gates operate on classical bits. The main differences are the use of quantum effects, as explained before, which have no classical counterpart: *superposition* and *entanglement*. Another difference lies on the fact that these operations are reversible, differently from the classical regime. It is a consequence of Schrodinger's equation that the time evolution of a quantum system is always governed by *unitary linear operators*. This means that if U is such an operator, then $U^{\dagger} = U^{-1}$. One simple example is the Pauli-X gate, which is the analogous to the classical NOT gate. It is written as:

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

This has the effect of changing a spin-up to a spin-down and vice-versa. Other extremely important gate is the Hadamard, which puts the initialised qubits into a state of equal superposition, and is written as:

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

Besides these, there exists *Toffoli* gates, Pauli-Y and Pauli-Z gates, among many others. It is important to mention the *universality* of quantum computing: this means that any unitary transformation can be approximated to an arbitrary error degree. A very simple example of a 2-qubit gate is the CNOT:



Here, the CNOT gate is represented by a control qubit (the top line with a dot) and a target qubit (the bottom line with a plus inside a circle). The control qubit is connected

to the target qubit by a vertical line, indicating that the state of the target qubit is flipped depending on the state of the control qubit.

2.3.11 Universality

The Solovay-Kitaev theorem is a fundamental result in quantum computing that provides a method for approximating any unitary operation using a finite set of universal quantum gates [52]. This theorem is important because it shows that any quantum computation can be efficiently approximated using a small set of gates, which is crucial for the practical implementation of quantum algorithms on quantum hardware.

The theorem can be stated as follows:

Theorem (Solovay-Kitaev): Let $\mathcal{G}$ be a finite set of unitary gates that generates a dense subset of the unitary group $U(d)$. Given any unitary operator $U \in U(d)$ and any $\epsilon > 0$, there exists an algorithm that outputs a sequence of gates $g_1, g_2, \ldots, g_m \in \mathcal{G}$ such that

$$\left\| U - \prod_{i=1}^{m} g_i \right\| \leq \epsilon, \tag{2.37}$$

where $\|\cdot\|$ denotes the operator norm. Moreover, the length of the sequence m is polynomial in $\log(1/\epsilon)$ and $\log d$.

The proof of the Solovay-Kitaev theorem is based on the idea of recursive approximation. The algorithm starts with a rough approximation of the target unitary U using the gates in $\mathcal{G}$, and then iteratively refines the approximation by concatenating and simplifying sequences of gates. The key insight is that the error in the approximation decreases exponentially with the number of iterations, while the length of the sequence grows only polynomially.

The Solovay-Kitaev algorithm can be summarized as follows:

1. Find a rough approximation V_0 of U using the gates in $\mathcal{G}$.
2. For each iteration $k = 1, 2, \ldots, n$:
   (a) Find a sequence of gates S_k that approximates the operator $UV_{k-1}^{\dagger}$.
   (b) Set $V_k = S_k V_{k-1}$.

3. Output the final sequence V_n.

The main technical challenge in the algorithm is to efficiently find the sequence S_k in step 2a. This can be done using a combination of techniques, such as nearest-neighbor search in the Cayley graph of $\mathcal{G}$ and precomputed lookup tables for short sequences.

The Solovay-Kitaev theorem has important implications for the design of quantum computers and the development of quantum algorithms. It shows that any quantum computation can be efficiently approximated using a small set of universal gates, which simplifies the task of building quantum hardware and designing quantum algorithms. Moreover, the theorem provides a concrete method for compiling quantum circuits, which is an essential step in the process of implementing quantum algorithms on real quantum devices.

Now that the introduction to quantum computing was given, it is equally important to understand the fundamental postulates of quantum mechanics before proceeding.

2.4 The Postulates of Quantum Mechanics

2.4.1 Postulate 1: Individual Quantum Systems

The first postulate of Quantum Mechanics states that the state of a quantum system is completely described by a wave function, denoted as $\psi(\mathbf{r}, t)$, which is a complex-valued function of position $\mathbf{r}$ and time t. The wave function contains all the information about the system and its evolution in time. The square of the magnitude of the wave function, $|\psi(\mathbf{r}, t)|^2$, represents the probability density of finding the particle at position $\mathbf{r}$ and time t. Mathematically, this can be written as:

$$P(\mathbf{r}, t) = |\psi(\mathbf{r}, t)|^2. \tag{2.38}$$

In the context of Quantum Computing, the first postulate is applied to qubits, which are the basic units of quantum information. A qubit is a quantum system that can exist in a superposition of two basis states, usually denoted as $|0\rangle$ and $|1\rangle$ - known

as the computational basis. The state of a qubit can be represented by a wave function as follows:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle. \tag{2.39}$$

Here, α and β, known as probability amplitudes, are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. The probabilities of measuring the qubit in the $|0\rangle$ and $|1\rangle$ states are given by $|\alpha|^2$ and $|\beta|^2$, respectively.

Although the theory of quantum computing, so far, uses qubits in its majority, it is also possible to define other entities of quantum information. A qutrit is a three-level quantum system, for example, for which its state can be represented as follows:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle + \gamma|2\rangle, \tag{2.40}$$

where $|0\rangle$, $|1\rangle$, and $|2\rangle$ are the basis vectors of the associated Hilbert (state) space, and α, β, and γ are its corresponding probability amplitudes, such that normalisation requires that $|\alpha|^2 + |\beta|^2 + |\gamma|^2 = 1$.

It is possible to go even further by defining a qudit as a d-level quantum system, which can be represented by a d-dimensional Hilbert space. The state of a qudit can be written as:

$$|\psi\rangle = \sum_{i=0}^{d-1} \alpha_i |i\rangle, \tag{2.41}$$

and the same conditions of normalisation required for qubits and qutrits follow here.

Quantum computing exploits the principles of quantum mechanics, such as superposition and entanglement, to perform computations that are infeasible for classical computers. Quantum algorithms are designed to manipulate qubits and their wave functions in such a way that the desired computational result can be obtained with high probability upon measurement. The first postulate of quantum mechanics is crucial in understanding and designing quantum algorithms, as it provides the foundation for representing and manipulating quantum states.

2.4.2 Postulate 2: Quantum Operations & Observables

One natural question to ask is: what kind of operations can be realised on a qubit? It happens that nature only allow for unitary transformations on qubits - except

for measurements. The second postulate describes the nature of quantum mechanical operators and how they relate to observable systems:

every measurable physical quantity is mathematically described by an operator acting on the state space of the corresponding system, and this operator is known as an Observable. This operator is always linear and Hermitian.

Physical Observable	Operator
Position	$\hat{x}$
Momentum	$\hat{p} = -i\hbar\frac{\partial}{\partial x}$
Energy	$\hat{H} = \frac{\hat{p}^2}{2m} + V(\hat{x})$
Angular Momentum	$\hat{L} = \hat{x} \times \hat{p}$
Spin	$\hat{S}$
Time Reversal	$\hat{T}$

Table 2.1: Some examples of physical observables and their corresponding operators.

To put into context, the table 2.1 shows some examples of physical observables and their associated operators. So, an operator is a mathematical entity that pulls information from the wavefunction, revealing properties of the quantum mechanical system. The fact that an observable must be Hermitian comes from the fact that the resultt of a measurement in real life are real values. This leads to the next postulate.

2.4.3 Postulate 3: Measurement

The third postulate states that only the eigenvalues of the quantum mechanical operator are possible to be measured. So, if the eigenstates of an observable M are $\psi_1, \psi_2, \ldots, \psi_n$ with real eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$, then the Quantum Theory states that only $\{\lambda_i,\ i \in \{1, \ldots, n\}\}$ can be physically observed in an experiment.

General formalism

Quantum measurements are a collection of operators $\{M_m\}$ that act on the state space of the system to be measured, where m represents the outcome that may occur in the experiment. Letting $|\psi\rangle$ be the state of the system right before the measurement,

then the probability that result m is observed is given by:

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle \tag{2.42}$$

and the state of the system after the measurement is:

$$|\psi'\rangle = \frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}. \tag{2.43}$$

It is important to note that all measurement operators must satisfy the completeness relation:

$$\sum_m M_m^\dagger M_m = \mathbb{I}, \tag{2.44}$$

which comes from the fact that probabilities sum to one. To contextualize, if a single qubit on a state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ is measured in the computational basis, then the measurement operators corresponding to each observable, respectively, are given by $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$. If one wanted to know the probability of having 1 as outcome, then:

$$p(1) = \langle\psi|M_1^\dagger M_1|\psi\rangle = \langle\psi|M_1 M_1|\psi\rangle = \langle\psi|M_1|\psi\rangle = |\beta|^2,$$

and similarly it is for calculating the probability of a collapse in the $|0\rangle$ state.

Projective measurements

A projective measurement (PVM) [6] is a set of projectors $B = \{\Pi_i\}_{i=0}^m$ such that $\sum_{i=0}^m \Pi_i = \mathbb{I}$, where this last condition is the completeness relation. When each Π_i is rank one, that is, $\Pi_i = |\psi_i\rangle\langle\psi_i|$, then it is said that B models a measurement in the basis $\{|\psi_i\rangle\}$. In the context of quantum machine learning, very often measurements are made in the computational basis $B = \{|0\rangle\langle 0|, |1\rangle\langle 1|\}$.

Positive-Operator Valued Measure (POVM)

A Positive-Operator Valued Measure (POVM) [6] is a set of operators $\{E_i\}$ on a Hilbert space $\mathcal{H}$ that satisfies the following conditions:

1. Each E_i is a positive semi-definite operator, i.e., for all $\psi \in \mathcal{H}$, $\langle\psi|E_i|\psi\rangle \geq 0$.

2. The operators $\{E_i\}$ sum up to the identity operator on $\mathcal{H}$, i.e., $\sum_i E_i = I$.

In the context of quantum mechanics, a POVM can be seen as a generalization of a PVM. One important difference is that the number of measurement operators N in PVMs is at most the dimension of the Hilbert space associated with the corresponding quantum mechanical system, since they are all orthogonal. This is not necessarily true for POVMs, since they need not be orthogonal.

2.4.4 Postulate 4: Expected values

The sheer act of observing a quantum system irreversibly collapses its state, and it does so in a probabilistic way (with quantum probabilities, as we will see). Suppose we have the same state $|\psi_i\rangle$ as in equation (2.39). Then, as already explained, the coefficients α and β are complex numbers $\mathbb{C}$, called probability amplitudes, and they must obey the following relation:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.45}$$

Now, The probability of collapse into the state $|0\rangle$ is given by $|\alpha|^2$, and alternatively, the collapse into $|1\rangle$ has probability $|\beta|^2$. In general, the following postulate holds:

When a measurement of some observable A in a generic state $|\psi\rangle$ is made, the probability of seeing one of its eigenvalues λ_i is given by the inner product between $|\psi\rangle$ and the associated eigenstate $|\lambda_i\rangle$, that is, $|\langle\lambda_i|\psi\rangle|^2$.

2.4.5 Postulate 5: Evolution of a system in time

The evolution of a closed system (isolated) from the state $|\psi_0\rangle$ at time t_0 to the state $|\psi\rangle$ at time t is described by a unitary operator U_{t,t_0}:

$$|\psi\rangle = U_{t,t_0}|\psi_0\rangle. \tag{2.46}$$

The temporal evolution operator is a matrix that describes how the quantum state evolves over time. This same matrix is unitary, and hence it is always true that

$U_{t,t_0}U_{t,t_0}^{\dagger} = \mathbb{I}$, being $\mathbb{I}$ the identity matrix. In an isolated system, the temporal evolution operator is described in terms of another operator, called the Hamiltonian H. The Hamiltonian is an observable whose eigenvalues correspond to the allowed energies of the system. When H is time-independent, then the temporal evolution operator is given by $U(t,t_0) = e^{\frac{-i}{\hbar}H(t-t_0)}$. This solution is found by solving the well-known Schrödinger's equation:

$$H\Psi(\mathbf{r},t) = i\hbar\frac{\partial\Psi}{\partial t}. \tag{2.47}$$

This is a linear partial differential equation that describes how a quantum state evolves over time.

2.4.6 Postulate 6: Composite Systems

The state space of a composite physical system is given by the tensor product of the state space of each component (subsystem) in the system. So, for n systems enumerated from 1 to n, having the system i in the state $|\psi_i\rangle$, the state space of the composite system is written as:

$$|\psi_0\rangle \otimes |\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle = |\psi_0\rangle|\psi_1\rangle \ldots |\psi_n\rangle = |\psi_0\psi_1 \ldots \psi_n\rangle \tag{2.48}$$

When a quantum system is in a mixture of states, there is no state vector that correctly describes the system, and instead the density operator formalism is used. Instead, suppose that the system is in one of many possible states $|\psi_i\rangle$ with probability p_i. Then the set $\{p_[, |\psi_i\rangle\}$ is called an ensemble of pure states. In this case, the system is described by the density matrix as:

$$\rho = \sum_i p_i|\psi_i\rangle \langle \psi_i|$$

From this, it is also possible to reformulate the postulates in terms of the density operator formalism.

2.5 Quantum Hardware

2.5.1 DiVicenzo's criteria

Before going into further details about types of quantum hardware, it is important to observe that, in 2000, DiVicenzo made a list of desirable features an ideal Quantum Computer should display [5]. They are:

1. *a scalable physical system with well-characterized qubits*: as we already saw, qubits are two-level systems that leverage quantum effects. The first criterion for a good quantum computer is that these systems should be well-defined, and if the qubit has more than two possible levels - i.e. besides the ground-sate and the first excited sate - these should be designed such that the probability of going into a state higher than the first excited one is very small.

2. *qubit initialization to a simple fiducial state, like* $|000...\rangle$: the capacity of qubits to be initialized in the same state repeatedly is desired. The first reason is straightforwards, since we wish the registers to have a known value before any computation. Another reason for this lies in the need for quantum error, since it needs a continuous supply of low-entropy states, like $|0\rangle$.

3. *long coherence times*: quantum systems are extremely sensitive, such that their desired quantum effects can only hold for so long in the presence of external noise. The best Quantum Computer should have its qubits holding these quantum properties as long as it can, so more quantum operations can be done. Also, as argued in the introduction of this work, as of today, we are living on the NISQ era. So, this is still a challenge given the current quantum hardware architectures available today, but we have seen some promising works in this direction.

4. *universal set of gates*: quantum operations are the building block of quantum algorithms, and consequently of all Quantum Machine Learning models. Generally, as explained before, quantum algorithms consist of a sequence of unitary transformations, and it is desired that in a Quantum Computer, one can approximate any unitary transformation up to an arbitrary error degree. For this purpose, we say a *universal set of gates* should exist in such an architecture.

5. *Measurement of individual qubits*: there should be a way to efficiently measure the output of the quantum computations.

With this criteria in hands, we briefly illustrate next a traditional example of quantum hardware, the so-called ion traps.

2.5.2 Ion traps

Physically, quantum circuits can be realised through different technologies. The most classical one is that of ion trapping. What makes this an attractive option is that ions can be confined to a small location in space using electromagnetic fields. The first ion-trap was proposed in 1953 by Wolfgang Paul [53], which is nowadays also known as a Paul-trap. Ion trapping is also widely using in fabrication of high-precision atomic clocks. In theory, an ion should be trapped in a potential. However, *Earnshaw's theorem* [54] states that it is impossible to confine a charge in three dimensions using static potentials. Instead, the potential becomes a saddle, where the ion is trapped in one direction, but ends up scaping in the perpendicular direction. What's possible to do, however, is to rotate this potential at the right frequency, so that the wall 'generated' confines the ion in all directions. It can be shown that the correct rotating electric potential is given by:

$$\Phi = \frac{1}{2}\left(u_x x^2 + u_y y^2 + u_z z^2\right) + \frac{1}{2}\left(v_x x^2 + v_y y^2 + v_z z^2\right)\cos(\omega t + \phi), \tag{2.49}$$

where $u_x, u_y, u_z, v_x, v_y, v_z$ are potential energies in x, y, z; ϕ is a phase factor; and finally, all these variables have to be tuned accordingly to the ion's mass and charge, and the potential's angular frequency ω.

Once the ion system is trapped, to perform operations on it, then, photons are shot at the system, so it can go from the ground-state to the excited state. From the first DiVicenzo's criterion, we would also like as many qubits as possible in a Quantum Computer, so one ion alone is not enough. To realise this, ionic chains (which is a one-dimensional array) can be used. The problem is that shooting photons may cause relative motion between the ions, so undesired interactions may happen. To overcome this, we can leverage the **Mossbauer effect**: ions placed in a sufficiently spaced array are cooled down until their motion becomes quantized. When photons are shot, all

ions recoil together [6], eliminating the unwanted interactions.

The Hamiltonian that describes the electron in the trapped ion is approximately given by:

$$\hat{H} = \frac{\hbar\Omega}{2}\left(S_+ e^{i\varphi} + S_- e^{-i\varphi}\right) \tag{2.50}$$

where S_+ and S_- are matrices described by:

$$S_+ = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \quad S_- = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}. \tag{2.51}$$

For the time-independent hamiltonian, a qubit starting in the ground state evolves to the following time-dependent state, according to Schrödinger's equation, seen before:

$$|\psi(t)\rangle = e^{-i\hat{H}t/\hbar}|g\rangle. \tag{2.52}$$

By solving this equation, it can be shown that the unitary transformation produced by a laser pulse of amplitude B, duration t, and phase ϕ on an ion with magnetic moment μ_m is given by:

$$U(\Omega, \varphi, t) = \begin{pmatrix} \cos\left(\frac{\Omega t}{2}\right) & -i\sin\left(\frac{\Omega t}{2}\right)e^{-i\varphi} \\ -i\sin\left(\frac{\Omega t}{2}\right)e^{i\varphi} & \cos\left(\frac{\Omega t}{2}\right) \end{pmatrix}, \tag{2.53}$$

where ω is called the **Rabi frequency**. From the equation above, we can see that any arbitrary X and Y rotations can be done by setting $\varphi = 0$ or $\varphi = \frac{\pi}{2}$. Thus, the fourth criterion is met, and a universal set of single-qubit gates can be created by adjusting the phase and duration of the light pulses.

2.6 Quantum Machine Learning

2.6.1 Quantum Neural Network (QNN)

Variational circuits, also known as "parameterized quantum circuits", are quantum algorithms that depend on free parameters. Most of hybrid quantum-classical algorithms rely on these circuits. The key idea of the variational circuit is to optimize those parameters according to a cost function [55]. Like standard quantum circuits,

they consist in three ingredients:

1. preparation of a fixed initial state.
2. a quantum circuit parametized by a set of free parameters θ.
3. the measurement of an observable at the output, that may be made up from local observable itens for each or a subset of wires in the circuit.

The general structure of a Variational Quantum Circuit is shown in Figure 3, where $U(x)$ is the data encoding block which is predefined and static. It should be designed considering the nature of the problem and, thus, it is a crucial part in the architecture. $\Phi(\theta)$ represents the *tunable* part, which will be optimized, usually based on a classical optimization algorithm such as gradient descent, stochastic gradient descent or particle swarm optimization. These parameters can be seen as the *weights* in the classical neural networks. In this work, gradient descent was used.

A variety of results has shown that Quantum Variational Circuits are more expressive when compared to conventional neural networks in respect to the number of parameters [56, 57, 58]. Furthermore, successful experiments have shown that these circuits have multiple applications, such as function approximation [59], classification [60, 61, 62], generative modeling [63, 64], deep reinforcement learning [65], and transfer learning [66].

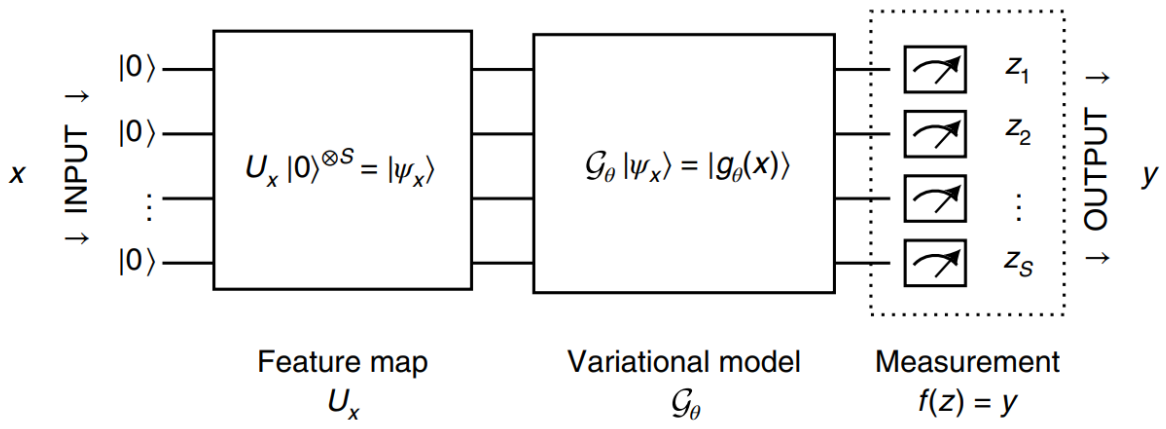


Figure 3: Simple quantum variational circuit architecture. Diagram taken from [67].

A number of researchers are still trying to understand, however, in which cases

can QNNs display better generalization over classical machine learning models. In this direction, some interesting studies are starting to pave the way [68, 67].

Chapter 3

The Proposed Approach

Here we will go through the data used in this study, along with all the extracted features available, that are further embedded in our quantum circuits for classification.

3.1 Data & Acoustic features

In this work we use the same data as in [69]. It is composed of 195 sustained vowel phonation recordings from 31 subjects, being 23 affected by PD and 8 healthy. On average, six phonations were recorded for each subject, ranging from 1 to 36 seconds in time-length. The subjects ages ranged from 46 to 85 years, and time of diagnosis, from 0 to 36 years.

The handcrafted acoustic features used in the proposed approach are now briefly explained. The main motivation to choose handcrafted features over deep learned ones is due to the fact that at the time this dissertation is being written, we are in the NISQ era, as explained in subsection 2.5. Since we still have very few number of qubits to run experiments with, we limit ourselves to a fine selection of handcrafted features only.

To begin with, [70] considers we have three types of speech signals: type I are those quasi-periodic; type II includes those which show some degree of dynamical changes, such as different frequencies, and in type III we have signals that have no kind of periodic pattern. This original classification was designed considering signals generated by deterministic processes. In [71], however, the authors argue that speech

samples in clinical settings are often quite noisy, and thus it is better to treat them as stochastic. In [72], a new category is introduced for these signals, which they call type IV.

As one of the most traditional features, jitter is measured as the perturbation in pitch period or, equivalently, changes that occur in the fundamental frequency (F_0). Differently, shimmer is another measure based on the perturbation in the signal amplitude. One challenge with these features is that they assume a deterministic (and periodic) generating process, thus only suited for type I voice signals, and also cannot capture breath noise, a common characteristic among pathological speech found in clinical settings. Nonetheless, they have been widely used in the literature [73, 74, 75, 76] for discriminating pathological from normal speech, and thus we also consider them in this study.

Other traditional features used in this study were average, maximum, and minimum in F_0; noise-to-harmonic and harmonic-to-noise ratios, abbreviated as NHR and HNR, respectively. These last features capture the proportion of noise, i.e., breathiness, for instance, in the content of the voice signal.

Other methods that also rely on linear assumptions about the signals include the Discrete Fourier Transform and cepstral analysis, Specifically, they assume the signal has (quasi)periodicity, linearity, determinism or Gaussianity. Although they have their use, for clinical data, there is a better mathematical framework to work with: the theory of Nonlinear Dynamical Systems. Here, measures of chaos have been proposed as features for discrimination between healthy and disordered speech. Well-known examples are Lyapunov exponents [77], correlation dimension (D2) [78]. These tools are excellent, as they are suited to handle nonlinearities (which is most observed in clinical data), but another challenge is that for very noisy and breathy speech samples, these measures can fail, as they require the attractor - a region in phase space to which our dynamical system tends to evolve in time, regardless of the initial conditions - not to be destroyed by the noise.

To capture both deterministic and stochastic effects of speech, the authors of [71] model this problem by using *Langevin dynamics* in order to describe the equation of motion generating the speech signal. At any time $t \in \mathcal{R}$, the state of the system is given by the vector $\boldsymbol{u}(t) \in \mathbb{R}^d$. The Langevin equation [78] reads as:

$$\dot{\boldsymbol{u}} = \boldsymbol{f}(\boldsymbol{u}(t)) + \epsilon(t), \tag{3.1}$$

where ϵ is a stochastic-forcing vector, representing the breathiness effects - caused by the inability of proper closing the vocal folds, so that air has to escape through a partial opening, causing turbulent airflow[79] - and $f(u)$ effectively models the deterministic governing dynamics of the speech. From this, the concepts of periodicity and aperiodicity can be seen as special cases of recurrence in the phase space, where recurrent orbits are those that return to a specific region of the space after some time. Mathematically, phase space recurrence can be defined as:

$$\boldsymbol{u}(t) \subset \boldsymbol{B}(\boldsymbol{u}(t + \delta t), r). \tag{3.2}$$

Where $\boldsymbol{B}(\boldsymbol{u}, r)$ is a ball of radius r around point u in phase space, and δt is the period it takes a trajectory to return to this region, called the recurrence time. From this, periodicity is seen as a special case when $r = 0$ and δt is always the same for all $t \in \mathbb{R}$; that is, a trajectory returns to the exact same point after some regular period of time δt. For aperiodic but recurrent signals, we relax this condition by saying their trajectories return to the same region after δt, but not necessarily the same point; that is, $r > 0$.

Although the reconstructed state space has rich structural information about the governing dynamics, all we have to start with is a time-series measurement, that is, the speech itself. Now, Takens' theorem states that information in hidden variables of a dynamical system can be preserved in a time-series measurement [80], and by obeying the necessary conditions, a diffeomorphism between a delay embedded attractor and the attractor in original coordinates can be obtained. This means that information about the governing dynamics is preserved through the time-series, and thus we can use this information to classify parkinsonian and healthy subjects. This motivates the use of features based on recurrence analysis, such as the recurrence period density entropy

(RPDE), introduced in [71]. To start with, say there is some measurement function $h : \mathbb{R}^d \rightarrow \mathbb{R}$ that makes the signal, such that:

$$s_n = h(\boldsymbol{u}(n\Delta t)). \tag{3.3}$$

This assumes that at each instant Δt, there is a corresponding d-dimensional state vector $\boldsymbol{u}(n\Delta t)$ that is projected to a single real scalar via $h(u)$, where n is the discrete-time step. There are different methods for reconstructing the state space. The one used in this research work and in [71] was introduced by the authors of [81], optimizing m and τ. These parameters have important effects: for very small τ, for instance, they lead to noisy recurrences, whereas too large values end up making recurrences difficult to detect. Thus values should be carefully selected. There are also other methods, such as False Nearest Neighbors, which is not used since it needs deterministic assumptions [78, 81]. From this, a time delayed embedding vector of size m can be built as follows:

$$s_n = [s_n, s_{n-\tau}, s_{n-\tau}, ..., s_{n-(m-1)\tau}]^T, \tag{3.4}$$

where τ is the time delay, and m is the embedding dimension.

To further extract the features from the signal time recurrence, the *method of close returns* [82] is used, considering both deterministic and stochastic speech signals. Here, a closed ball $B(s_{n_0}, r)$ is centered around s_{n_0}, with $r > 0$. The trajectory of the initial state vector s_{n_0} is forwarded in time until it leaves the ball, i.e., $s_{n_0}, s_{n_0+1}, ..., s_{n_0+k}$, where $n_0 + k$, with $k > 0$, is the discrete-time step where it leaves B - or formally: $\|s_{n_0} - s_{n_0+k}\|_{l2} > r$. Finally, the recurrence time T is one such that $\|s_{n_0} - s_{n_0+T}\|_2 \leq r$, i.e., the number of discrete-time steps it takes for the trajectory to return to the ball. The recurrence time is taken for all s_n, and then a histogram is formed as $R(T)$, so that by normalising it, we have:

$$P(T) = \frac{R(T)}{\sum_{i=1}^{T_{\max}} R(i)}, \tag{3.5}$$

where $T_{\max}$ is the maximum recurrence time found in the embedded space. In the end, we have a probability distribution P of n discrete values. This is known as the

recurrence time probability density. From this, we can exploit one idea: to extract the uncertainty contained in this probability distribution of recurrence times - on average, we expect pathological signals to fail periodicity more and exhibit chaotic effects, thus very different recurrence times δt should appear in the density. On the other hand, healthy signals are expected to have more regular δt, thus less information is present in their recurrence time distribution. From the well-known Shannon's definition of entropy, the RPDE can then be defined as:

$$RPDE = \sum_{i=1}^{n} P(i) \ln P(i). \tag{3.6}$$

This measure is good because it basically ranks different types of signals based on their period uncertainty, ranging from the most regular ones - type I - to those that are very breathy, and thus corrupted by noise - type IV.

Finally, the last features used the speech signals are the Detrended Fluctuation Analysis[71] (DFA), used for capturing the the stochastic self-similarity of the speech, related to the noise component in our Langevin equation, $\epsilon(t)$, and the Pitch-Period Entropy (PPE) [69], which measures the extent of non-Gaussian fluctuations in the sequence of relative semitone pitch period variations.

To summarize, in practice, the speech signals lie on a spectrum: on one hand, almost perfectly periodic signals, and on the other, very noisy, breathy, and stochastic ones. Since there is no perfect measure suited for all types mentioned, we use all the ones explained before, so multiple types of signals are covered, as listed before. The detailed list of these features is listed in Table 3.1.

Feature	Description
F0	Average vocal fundamental frequency
Fhi	Maximum vocal fundamental frequency
Flo	Minimum vocal fundamental frequency
Jitter (%)	jitter as a percentage
Jitter (Abs)	absolute jitter in microseconds
Jitter (DDP)	Average absolute difference of differences between cycles, divided by the average period
RAP	Relative Amplitude Perturbation
PPQ	five-point Period Perturbation Quotient
Shimmer	local shimmer
Shimmer (dB)	local shimmer in decibels
Shimmer (APQ3)	Three point Amplitude Perturbation Quotient
Shimmer (APQ5)	Five point Amplitude Perturbation Quotient
Shimmer (DDA)	Average absolute difference between consecutive differences between the amplitudes of consecutive periods
APQ	11-point Amplitude Perturbation Quotient
NHR	Noise-to-Harmonics ratio
HNR	Harmonics-to-Noise ratio
RPDE	Recurrence Period Density Entropy
DFA	Detrended Fluctuation Analysis
D2	Correlation dimension
PPE	Pitch Period Entropy

Since the number of features is very high, and we use angle encoding for the parametrized circuits - which requires a number of qubits proportional to that of features - we first fixed a number of 6 qubits in our experiments, and then performed a sliding window over the features presented above (given the respective order that the features were presented).

3.2 Implementation

All the quantum models were implemented in Pennylane [83]. The first block of the code for the variational circuit used is shown below on a bottom-up approach, by first defining the types of layers that will be used in out circuit - Hadamard layers, y-rotations and entangling layers:

```
import pennylane as qml
import numpy as np
import torch
import torch.nn as nn

def H_layer(nqubits):
    """Layer of single-qubit Hadamard gates.
    """
    for idx in range(nqubits):
```

```
        qml.Hadamard(wires=idx)


def RY_layer(w):
    """Layer of parametrized qubit rotations around the y axis.
    """
    for idx, element in enumerate(w):
        qml.RY(element, wires=idx)


def entangling_layer(nqubits):
    """Layer of CNOTs followed by another shifted layer of CNOT.
    """
    # In other words it should apply something like :
    # CNOT  CNOT  CNOT  CNOT...  CNOT
    #   CNOT  CNOT  CNOT...  CNOT
    for i in range(0, nqubits - 1, 2):  # Loop over even indices:
     ↪  i=0,2,...N-2
        qml.CNOT(wires=[i, i + 1])
    for i in range(1, nqubits - 1, 2):  # Loop over odd indices:
     ↪  i=1,3,...N-3
        qml.CNOT(wires=[i, i + 1])
```

The next part is the quantum node, which is the heart of our model, as it performs all the quantum operations based on the layer definitions showed in the last step:

```
@qml.qnode(dev, interface="torch")
def quantum_net(q_input_features, q_weights_flat, q_depth,
 ↪  n_qubits):
    """
    The variational quantum circuit.
    """

    # Reshape weights
```

```
    q_weights = q_weights_flat.reshape(q_depth, n_qubits)

    # Start from state |+> , unbiased w.r.t. |0> and |1>
    H_layer(n_qubits)

    # Embed features in the quantum node
    RY_layer(q_input_features)

    # Sequence of trainable variational layers
    for k in range(q_depth):
        entangling_layer(n_qubits)
        RY_layer(q_weights[k])

    # Expectation values in the Z basis
    exp_vals = [qml.expval(qml.PauliZ(position)) for position in
     ↪  range(n_qubits)]
    return tuple(exp_vals)
```

Then, we dress our quantum circuit in a torch-like format:

```
class PQC(nn.Module):
    """
    Torch module implementing the *dressed* quantum net.
    """

    def __init__(self, q_depth, n_qubits, q_delta):
        """
        Definition of the *dressed* layout.
        """

        super().__init__()
        self.n_qubits = n_qubits
        self.q_depth = q_depth
```

```
        self.q_params = nn.Parameter(q_delta *
        ↪ torch.randn(q_depth * n_qubits))
        self.post_net = nn.Linear(n_qubits, 2)

    def forward(self, input_features):
        """
        Defining how tensors are supposed to move through the
    ↪ *dressed* quantum
        net.
        """

        # obtain the input features for the quantum circuit
        # (normalization required)
        q_in = torch.tanh(input_features) * np.pi / 2.0

        # Apply the quantum circuit to each element of the batch
        ↪ and append to q_out
        q_out = torch.Tensor(0, n_qubits)
        q_out = q_out.to(device)
        for elem in q_in:
            q_out_elem = quantum_net(elem, self.q_params,
            ↪ self.q_depth, self.n_qubits).float().unsqueeze(0)
            q_out = torch.cat((q_out, q_out_elem))

        # return the two-dimensional prediction from the
        ↪ postprocessing layer
        return self.post_net(q_out)
```

Finally, we have our training function:

```
import torch
import numpy as np
from sklearn.metrics import f1_score

```

```
5  device = 'cpu'
6
7  def train_model(model, dataloaders, dataset_sizes, batch_size,
   ↪  criterion, optimizer, num_epochs):
8      # best_model_wts = copy.deepcopy(model.state_dict())
9      best_acc = 0.0
10     best_loss = 10000.0  # Large arbitrary number
11     best_acc_train = 0.0
12     best_loss_train = 10000.0  # Large arbitrary number
13     # print("Training started:")
14
15     for epoch in range(num_epochs):
16
17         # Each epoch has a training and validation phase
18         for phase in ["train", "validation"]:
19             if phase == "train":
20                 # Set model to training mode
21                 model.train()
22             else:
23                 # Set model to evaluate mode
24                 model.eval()
25             running_loss = 0.0
26             running_corrects = 0
27
28             # Iterate over data.
29             n_batches = dataset_sizes[phase] // batch_size
30             it = 0
31             for inputs, labels in dataloaders[phase]:
32                 batch_size_ = len(inputs)
33                 inputs = inputs.to(device)
34                 labels = labels.to(device)
35                 optimizer.zero_grad()
36
```

```
37                  # Track/compute gradient and make an optimization
                     ↪  step only when training
38                  with torch.set_grad_enabled(phase == "train"):
39                      outputs = model(inputs)
40                      _, preds = torch.max(outputs, 1)
41                      loss = criterion(outputs, labels)
42                      if phase == "train":
43                          loss.backward()
44                          optimizer.step()
45
46                  # Print iteration results
47                  running_loss += loss.item() * batch_size_
48                  batch_corrects = torch.sum(preds ==
                     ↪  labels.data).item()
49                  running_corrects += batch_corrects
50                  print(
51                      "Phase: {} Epoch: {}/{} Iter: {}".format(
52                          phase,
53                          epoch + 1,
54                          num_epochs,
55                          it + 1
56                      ),
57                      end="\r",
58                      flush=True,
59                  )
60                  it += 1
61
62              # Print epoch results
63              epoch_loss = running_loss / dataset_sizes[phase]
64              epoch_acc = running_corrects / dataset_sizes[phase]
65              epoch_f1 = f1_score(labels.cpu(), preds.cpu(),
                 ↪  average='weighted')
66              print(
```

```
                "Phase: {} Epoch: {}/{} Loss: {:.4f} Acc: {:.4f}
                 ↪  F1: {:.4f}       ".format(
                    "train" if phase == "train" else "validation
                     ↪  ",
                    epoch + 1,
                    num_epochs,
                    epoch_loss,
                    epoch_acc,
                    epoch_f1
                )
            )

            # Check if this is the best model wrt previous epochs
            if phase == "validation" and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_f1 = epoch_f1
            if phase == "validation" and epoch_loss < best_loss:
                best_loss = epoch_loss
            if phase == "train" and epoch_acc > best_acc_train:
                best_acc_train = epoch_acc
            if phase == "train" and epoch_loss < best_loss_train:
                best_loss_train = epoch_loss

            Update learning rate
            if phase == "train":
                scheduler.step()

    return best_acc, best_f1
```

With this, we define our quantum model, which will have 1 qubit per feature, and all the results of our experiments are presented in the next section.

Chapter 4

Tests and Results

In table 4.1, the results of the performance metrics are shown for all models used in this study.

Table 4.1: Average results on hold-out test set for the aforementioned models against the dataset. The abbreviated feature names mean - where some features have MDVP, meaning that they were extracted using the Kay Pentax multidimensional voice program [69]: F_0: "MDVP:Fo(Hz)", F_1: "MDVP:Fhi(Hz)", F_2: "MDVP:Flo(Hz)", F_3: "MDVP:Jitter(%)", F_4: "MDVP:Jitter(Abs)", F_5: "MDVP:RAP", F_6: "MDVP:PPQ", F_7: "Jitter:DDP", F_8: "MDVP:Shimmer", F_9: "MDVP:Shimmer(dB)", F_10: "Shimmer:APQ3", F_11: "Shimmer:APQ5", F_12: "MDVP:APQ", F_13: "Shimmer:DDA", F_14: "NHR", F_15: "HNR", F_16: "RPDE", F_17: "DFA", F_18: "spread1", F_19: "spread2", F_20: "D2", F_21: "PPE"

Algorithm	Accuracy	F1-score	Features
Support Vector Machine	0.841026	0.811661	F_1, F_2, F_3, F_4, F_5, F_6

Continued on next page

Table 4.1: Average results on hold-out test set for the aforementioned models against the dataset. The abbreviated feature names mean: F_0: "MDVP:Fo(Hz)", F_1: "MDVP:Fhi(Hz)", F_2: "MDVP:Flo(Hz)", F_3: "MDVP:Jitter(%)", F_4: "MDVP:Jitter(Abs)", F_5: "MDVP:RAP", F_6: "MDVP:PPQ", F_7: "Jitter:DDP", F_8: "MDVP:Shimmer", F_9: "MDVP:Shimmer(dB)", F_10: "Shimmer:APQ3", F_11: "Shimmer:APQ5", F_12: "MDVP:APQ", F_13: "Shimmer:DDA", F_14: "NHR", F_15: "HNR", F_16: "RPDE", F_17: "DFA", F_18: "spread1", F_19: "spread2", F_20: "D2", F_21: "PPE"

Algorithm	Accuracy	F1-score	Features
K-Nearest Neighbors	0.869231	0.866572	F_1, F_2, F_3, F_4, F_5, F_6
Multilayer Perceptron	0.764103	0.673225	F_1, F_2, F_3, F_4, F_5, F_6
Logistic Regression	0.835897	0.803514	F_1, F_2, F_3, F_4, F_5, F_6
Quantum Neural Network	0.753846	0.675395	F_1, F_2, F_3, F_4, F_5, F_6
Support Vector Machine	0.828205	0.791630	F_2, F_3, F_4, F_5, F_6, F_7
K-Nearest Neighbors	0.887179	0.882961	F_2, F_3, F_4, F_5, F_6, F_7
Multilayer Perceptron	0.753846	0.655925	F_2, F_3, F_4, F_5, F_6, F_7
Logistic Regression	0.820513	0.779972	F_2, F_3, F_4, F_5, F_6, F_7
Quantum Neural Network	0.758974	0.663761	F_2, F_3, F_4, F_5, F_6, F_7
Support Vector Machine	0.735897	0.639130	F_3, F_4, F_5, F_6, F_7, F_8
K-Nearest Neighbors	0.802564	0.791208	F_3, F_4, F_5, F_6, F_7, F_8
Multilayer Perceptron	0.743590	0.634238	F_3, F_4, F_5, F_6, F_7, F_8
Logistic Regression	0.743590	0.634238	F_3, F_4, F_5, F_6, F_7, F_8
Quantum Neural Network	0.769231	0.685551	F_3, F_4, F_5, F_6, F_7, F_8
Support Vector Machine	0.748718	0.649504	F_4, F_5, F_6, F_7, F_8, F_9
K-Nearest Neighbors	0.810256	0.798252	F_4, F_5, F_6, F_7, F_8, F_9

Continued on next page

Table 4.1: Average results on hold-out test set for the aforementioned models against the dataset. The abbreviated feature names mean: F_0: "MDVP:Fo(Hz)", F_1: "MDVP:Fhi(Hz)", F_2: "MDVP:Flo(Hz)", F_3: "MDVP:Jitter(%)", F_4: "MDVP:Jitter(Abs)", F_5: "MDVP:RAP", F_6: "MDVP:PPQ", F_7: "Jitter:DDP", F_8: "MDVP:Shimmer", F_9: "MDVP:Shimmer(dB)", F_10: "Shimmer:APQ3", F_11: "Shimmer:APQ5", F_12: "MDVP:APQ", F_13: "Shimmer:DDA", F_14: "NHR", F_15: "HNR", F_16: "RPDE", F_17: "DFA", F_18: "spread1", F_19: "spread2", F_20: "D2", F_21: "PPE"

Algorithm	Accuracy	F1-score	Features
Multilayer Perceptron	0.743590	0.634238	F_4, F_5, F_6, F_7, F_8, F_9
Logistic Regression	0.743590	0.634238	F_4, F_5, F_6, F_7, F_8, F_9
Quantum Neural Network	0.748718	0.654244	F_4, F_5, F_6, F_7, F_8, F_9
Support Vector Machine	0.743590	0.634238	F_5, F_6, F_7, F_8, F_9, F_10
K-Nearest Neighbors	0.748718	0.740761	F_5, F_6, F_7, F_8, F_9, F_10
Multilayer Perceptron	0.743590	0.634238	F_5, F_6, F_7, F_8, F_9, F_10
Logistic Regression	0.743590	0.634238	F_5, F_6, F_7, F_8, F_9, F_10
Quantum Neural Network	0.761538	0.689904	F_5, F_6, F_7, F_8, F_9, F_10
Support Vector Machine	0.743590	0.634238	F_6, F_7, F_8, F_9, F_10, F_11
K-Nearest Neighbors	0.741026	0.726394	F_6, F_7, F_8, F_9, F_10, F_11
Multilayer Perceptron	0.743590	0.634238	F_6, F_7, F_8, F_9, F_10, F_11
Logistic Regression	0.743590	0.634238	F_6, F_7, F_8, F_9, F_10, F_11
Quantum Neural Network	0.758974	0.678940	F_6, F_7, F_8, F_9, F_10, F_11
Support Vector Machine	0.743590	0.634238	F_7, F_8, F_9, F_10, F_11, F_12
K-Nearest Neighbors	0.758974	0.748896	F_7, F_8, F_9, F_10, F_11, F_12
Multilayer Perceptron	0.743590	0.634238	F_7, F_8, F_9, F_10, F_11, F_12

Continued on next page

Table 4.1: Average results on hold-out test set for the aforementioned models against the dataset. The abbreviated feature names mean: F_0: "MDVP:Fo(Hz)", F_1: "MDVP:Fhi(Hz)", F_2: "MDVP:Flo(Hz)", F_3: "MDVP:Jitter(%)", F_4: "MDVP:Jitter(Abs)", F_5: "MDVP:RAP", F_6: "MDVP:PPQ", F_7: "Jitter:DDP", F_8: "MDVP:Shimmer", F_9: "MDVP:Shimmer(dB)", F_10: "Shimmer:APQ3", F_11: "Shimmer:APQ5", F_12: "MDVP:APQ", F_13: "Shimmer:DDA", F_14: "NHR", F_15: "HNR", F_16: "RPDE", F_17: "DFA", F_18: "spread1", F_19: "spread2", F_20: "D2", F_21: "PPE"

Algorithm	Accuracy	F1-score	Features
Logistic Regression	0.743590	0.634238	F_7, F_8, F_9, F_10, F_11, F_12
Quantum Neural Network	0.743590	0.634238	F_7, F_8, F_9, F_10, F_11, F_12
Support Vector Machine	0.743590	0.634238	F_8, F_9, F_10, F_11, F_12, F_13
K-Nearest Neighbors	0.766667	0.741828	F_8, F_9, F_10, F_11, F_12, F_13
Multilayer Perceptron	0.743590	0.634238	F_8, F_9, F_10, F_11, F_12, F_13
Logistic Regression	0.743590	0.634238	F_8, F_9, F_10, F_11, F_12, F_13
Quantum Neural Network	0.743590	0.647926	F_8, F_9, F_10, F_11, F_12, F_13
Support Vector Machine	0.743590	0.634238	F_9, F_10, F_11, F_12, F_13, F_14
K-Nearest Neighbors	0.779487	0.763256	F_9, F_10, F_11, F_12, F_13, F_14
Multilayer Perceptron	0.743590	0.634238	F_9, F_10, F_11, F_12, F_13, F_14
Logistic Regression	0.743590	0.634238	F_9, F_10, F_11, F_12, F_13, F_14
Quantum Neural Network	0.751282	0.665970	F_9, F_10, F_11, F_12, F_13, F_14
Support Vector Machine	0.743590	0.634238	F_10, F_11, F_12, F_13, F_14, F_15
K-Nearest Neighbors	0.751282	0.752653	F_10, F_11, F_12, F_13, F_14, F_15
Multilayer Perceptron	0.743590	0.634238	F_10, F_11, F_12, F_13, F_14, F_15
Logistic Regression	0.756410	0.662281	F_10, F_11, F_12, F_13, F_14, F_15

Continued on next page

Table 4.1: Average results on hold-out test set for the aforementioned models against the dataset. The abbreviated feature names mean: F_0: "MDVP:Fo(Hz)", F_1: "MDVP:Fhi(Hz)", F_2: "MDVP:Flo(Hz)", F_3: "MDVP:Jitter(%)", F_4: "MDVP:Jitter(Abs)", F_5: "MDVP:RAP", F_6: "MDVP:PPQ", F_7: "Jitter:DDP", F_8: "MDVP:Shimmer", F_9: "MDVP:Shimmer(dB)", F_10: "Shimmer:APQ3", F_11: "Shimmer:APQ5", F_12: "MDVP:APQ", F_13: "Shimmer:DDA", F_14: "NHR", F_15: "HNR", F_16: "RPDE", F_17: "DFA", F_18: "spread1", F_19: "spread2", F_20: "D2", F_21: "PPE"

Algorithm	Accuracy	F1-score	Features
Quantum Neural Network	0.746154	0.652346	F_10, F_11, F_12, F_13, F_14, F_15
Support Vector Machine	0.743590	0.634238	F_11, F_12, F_13, F_14, F_15, F_16
K-Nearest Neighbors	0.776923	0.767036	F_11, F_12, F_13, F_14, F_15, F_16
Multilayer Perceptron	0.743590	0.634238	F_11, F_12, F_13, F_14, F_15, F_16
Logistic Regression	0.735897	0.656995	F_11, F_12, F_13, F_14, F_15, F_16
Quantum Neural Network	0.746154	0.642767	F_11, F_12, F_13, F_14, F_15, F_16
Support Vector Machine	0.743590	0.634238	F_12, F_13, F_14, F_15, F_16, F_17
K-Nearest Neighbors	0.851282	0.847908	F_12, F_13, F_14, F_15, F_16, F_17
Multilayer Perceptron	0.743590	0.634238	F_12, F_13, F_14, F_15, F_16, F_17
Logistic Regression	0.733333	0.645755	F_12, F_13, F_14, F_15, F_16, F_17
Quantum Neural Network	0.743590	0.638180	F_12, F_13, F_14, F_15, F_16, F_17
Support Vector Machine	0.846154	0.827473	F_13, F_14, F_15, F_16, F_17, F_18
K-Nearest Neighbors	0.869231	0.865369	F_13, F_14, F_15, F_16, F_17, F_18
Multilayer Perceptron	0.746154	0.643789	F_13, F_14, F_15, F_16, F_17, F_18
Logistic Regression	0.805128	0.759730	F_13, F_14, F_15, F_16, F_17, F_18
Quantum Neural Network	0.743590	0.634238	F_13, F_14, F_15, F_16, F_17, F_18

Continued on next page

Table 4.1: Average results on hold-out test set for the aforementioned models against the dataset. The abbreviated feature names mean: F_0: "MDVP:Fo(Hz)", F_1: "MDVP:Fhi(Hz)", F_2: "MDVP:Flo(Hz)", F_3: "MDVP:Jitter(%)", F_4: "MDVP:Jitter(Abs)", F_5: "MDVP:RAP", F_6: "MDVP:PPQ", F_7: "Jitter:DDP", F_8: "MDVP:Shimmer", F_9: "MDVP:Shimmer(dB)", F_10: "Shimmer:APQ3", F_11: "Shimmer:APQ5", F_12: "MDVP:APQ", F_13: "Shimmer:DDA", F_14: "NHR", F_15: "HNR", F_16: "RPDE", F_17: "DFA", F_18: "spread1", F_19: "spread2", F_20: "D2", F_21: "PPE"

Algorithm	Accuracy	F1-score	Features
Support Vector Machine	0.871795	0.863158	F_14, F_15, F_16, F_17, F_18, F_19
K-Nearest Neighbors	0.920513	0.920371	F_14, F_15, F_16, F_17, F_18, F_19
Multilayer Perceptron	0.741026	0.637262	F_14, F_15, F_16, F_17, F_18, F_19
Logistic Regression	0.807692	0.784380	F_14, F_15, F_16, F_17, F_18, F_19
Quantum Neural Network	0.748718	0.650302	F_14, F_15, F_16, F_17, F_18, F_19
Support Vector Machine	0.884615	0.879381	F_15, F_16, F_17, F_18, F_19, F_20
K-Nearest Neighbors	0.905128	0.903961	F_15, F_16, F_17, F_18, F_19, F_20
Multilayer Perceptron	0.741026	0.640477	F_15, F_16, F_17, F_18, F_19, F_20
Logistic Regression	0.820513	0.809241	F_15, F_16, F_17, F_18, F_19, F_20
Quantum Neural Network	0.746154	0.642767	F_15, F_16, F_17, F_18, F_19, F_20

From this, we can see that the so-called Quantum Neural Networks can achieve good performance as discriminators for pathological speech, comparable to well-established machine learning models, even when limited to a small number of qubits. While the K-Nearest Neighbors model slightly outperformed the Quantum Neural Network in terms of accuracy at 0.77 vs 0.75, the Quantum Neural Network achieved a higher F1-score of 0.66 compared to 0.63 for the other models. It is important to notice that our aim in this study was not to try to demonstrate any kind of "quantum supremacy" over classical

machine learning models, since quantum machine learning is still in the early stages of its development, but rather to show that it is possible to use this class of models in medical applications. In our case, it is the classification of impaired parkinsonian speech, and the results are comparable to classical models, as stated before.

Chapter 5

Conclusion and Future Work

In this work, quantum machine learning for parkinsonian speech detection was explored with good and comparable results to classical machine learning. In modern approaches, the speech feature extraction is usually automated using deep neural networks, trading many times explainability for generalization. Despite the limitations of current quantum computers, the results represent a first step, demonstrating useful medical applications of this new class of models. As quantum computing technologies continue to scale up the number of high fidelity qubits, improving critical factors, like coherence time and error correction, we posit hope that this new generation of models will have great results, potentially surpassing classical ones.

Regarding the main difficulties, since the field of quantum computing is in its birth and large-scale access to quantum hardware is not still available, we have to stick to simulators, which require high time complexity, and hence a low number of qubits for the experiments. On the good side, lots of frameworks are constantly being developed, which began with low-level ones - just like the beginning of classical computing, when Assembly and C were the top languages, used for almost every application - and now other frameworks that allow for higher-level implementations, like Pennylane.

For conclusion and future directions of work, one interesting approach to see is when a post-NISQ era comes, and with this, robust hardware with large number of qubits to use. Then, much wider/deeper quantum models could be used for the new task at hand, possibly removing the need for handcrafted feature engineering. Other line of research is to tackle possible symmetries in speech data as an effective inductive bias in the parametrized circuits, or even to discover hidden symmetries with them.

Bibliography

[1] Maria Eugenia Dajer, Paulo Rogério Scalassara, Jamille Lays Marrara, and JC Pereira. Voice analysis of patients with neurological disorders using acoustical and nonlinear tools. In 2012 IEEE International Workshop on Machine Learning for Signal Processing, pages 1–6. IEEE, 2012.

[2] Federica Amato, Luigi Borzì, Gabriella Olmo, and Juan Rafael Orozco-Arroyave. An algorithm for parkinson's disease speech classification based on isolated words analysis. Health Information Science and Systems, 9(1):1–15, 2021.

[3] Rahul Gupta, Theodora Chaspari, Jangwon Kim, Naveen Kumar, Daniel Bone, and Shrikanth Narayanan. Pathological speech processing: State-of-the-art, current challenges, and future directions. In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 6470–6474. IEEE, 2016.

[4] Charles H Bennett and David P DiVincenzo. Quantum information and computation. Nature, 404(6775):247–255, 2000.

[5] David P DiVincenzo. The physical implementation of quantum computation. Fortschritte der Physik: Progress of Physics, 48(9-11):771–783, 2000.

[6] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[7] John Preskill. Quantum computing in the nisq era and beyond. Quantum, 2:79, 2018.

[8] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. Nature Reviews Physics, 3(9):625–644, 2021.

[9] AB Singleton, M Farrer, J Johnson, A Singleton, S Hague, J Kachergus, M Hulihan, T Peuralinna, A Dutra, R Nussbaum, et al. α-synuclein locus triplication causes parkinson's disease. Science, 302(5646):841–841, 2003.

[10] Mihael H Polymeropoulos, Christian Lavedan, Elisabeth Leroy, Susan E Ide, Anindya Dehejia, Amalia Dutra, Brian Pike, Holly Root, Jeffrey Rubenstein, Rebecca Boyer, et al. Mutation in the α-synuclein gene identified in families with parkinson's disease. Science, 276(5321):2045–2047, 1997.

[11] Cesar A Medina, Eddie Vargas, Stephanie J Munger, and Julie E Miller. Vocal changes in a zebra finch model of parkinson's disease characterized by alpha-synuclein overexpression in the song-dedicated anterior forebrain pathway. PloS one, 17(5):e0265604, 2022.

[12] Christopher G Goetz. The history of parkinson's disease: early clinical descriptions and neurological therapies. Cold Spring Harbor perspectives in medicine, 1(1):a008862, 2011.

[13] Werner Poewe, Klaus Seppi, Caroline M Tanner, Glenda M Halliday, Patrik Brundin, Jens Volkmann, Anette-Eleonore Schrag, and Anthony E Lang. Parkinson disease. Nature reviews Disease primers, 3(1):1–21, 2017.

[14] Alfredo Berardelli, John C Rothwell, Philip D Thompson, and Mark Hallett. Pathophysiology of bradykinesia in parkinson's disease. Brain, 124(11):2131–2146, 2001.

[15] John G Nutt. Motor fluctuations and dyskinesia in parkinson's disease. Parkinsonism & related disorders, 8(2):101–108, 2001.

[16] Maria Grazia Spillantini, Marie Luise Schmidt, Virginia M-Y Lee, John Q Trojanowski, Ross Jakes, and Michel Goedert. α-synuclein in lewy bodies. Nature, 388(6645):839–840, 1997.

[17] Michel Goedert, Maria Grazia Spillantini, Kelly Del Tredici, and Heiko Braak. 100 years of lewy pathology. Nature Reviews Neurology, 9(1):13–24, 2013.

[18] Wei Xu, Lan Tan, and Jin-Tai Yu. The link between the snca gene and parkinsonism. Neurobiology of aging, 36(3):1505–1518, 2015.

[19] Kelvin C Luk, Victoria Kehm, Jenna Carroll, Bin Zhang, Patrick O'Brien, John Q Trojanowski, and Virginia M-Y Lee. Pathological α-synuclein transmission initiates parkinson-like neurodegeneration in nontransgenic mice. Science, 338(6109):949–953, 2012.

[20] C Warren Olanow and Patrik Brundin. Parkinson's disease and alpha synuclein: is parkinson's disease a prion-like disorder? Movement Disorders, 28(1):31–40, 2013.

[21] Etienne C Hirsch and Stéphane Hunot. Neuroinflammation in parkinson's disease: a target for neuroprotection? The Lancet Neurology, 8(4):382–397, 2009.

[22] Constance Scharff and Fernando Nottebohm. A comparative study of the behavioral deficits following lesions of various parts of the zebra finch song system: implications for vocal learning. Journal of Neuroscience, 11(9):2896–2913, 1991.

[23] Jean Martin Charcot. Lecons sur, les maladies du système nerveux, volume 1. Lecrosnier et Babé, 1886.

[24] Christopher G Goetz. The history of parkinson's disease: early clinical descriptions and neurological therapies. Cold Spring Harbor Perspectives in Medicine:, 1(1), 2011.

[25] A Carlsson. Thirty years of dopamine research. Advances in neurology, 60:1–10, 1993.

[26] Arvid Carlsson, Margit Lindqvist, and TOR Magnusson. 3, 4-dihydroxyphenylalanine and 5-hydroxytryptophan as reserpine antagonists. Nature, 180(4596):1200–1200, 1957.

[27] E Lorenc-Koci, K Ossowska, J Wardas, and S Wolfarth. Does reserpine induce parkinsonian rigidity? Journal of neural transmission-Parkinson's disease and dementia section, 9(2-3):211–223, 1995.

[28] Valéria S Fernandes, José R Santos, Anderson HFF Leão, André M Medeiros, Thieza G Melo, Geison S Izídio, Alicia Cabral, Rosana A Ribeiro, Vanessa C Abílio, Alessandra M Ribeiro, et al. Repeated treatment with a low dose of reserpine as a progressive model of parkinson's disease. Behavioural brain research, 231(1):154–163, 2012.

[29] Andrew J Lees, Eduardo Tolosa, and C Warren Olanow. Four pioneers of l-dopa treatment: Arvid carlsson, oleh hornykiewicz, george cotzias, and melvin yahr. Movement Disorders, 30(1):19–36, 2015.

[30] Bala V Manyam. Paralysis agitans and levodopa in "ayurveda": ancient indian medical treatise. Movement disorders: official journal of the Movement Disorder Society, 5(1):47–48, 1990.

[31] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. IEEE transactions on information theory, 13(1):21–27, 1967.

[32] Simon Haykin. Neural networks: a comprehensive foundation. Prentice Hall PTR, 1998.

[33] Sebastian Ruder. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747, 2016.

[34] Tjalling J Ypma. Historical development of the newton–raphson method. SIAM review, 37(4):531–551, 1995.

[35] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. Mathematical programming, 45(1-3):503–528, 1989.

[36] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. nature, 323(6088):533–536, 1986.

[37] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning, 2018.

[38] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[39] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander Smola. Resnest: Split-attention networks, 2020.

[40] Lucas Rafael Stefanel Gris. Reconhecimento de voz utilizando wav2vec 2.0 para o português brasileiro. B.S. thesis, Universidade Tecnológica Federal do Paraná, 2021.

[41] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[42] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[43] OpenAI. Gpt-4 technical report, 2023.

[44] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366, 1989.

[45] Corinna Cortes and Vladimir Vapnik. Support-vector networks. Machine learning, 20:273–297, 1995.

[46] Jonathan Borwein and Adrian Lewis. Convex Analysis. Springer, 2006.

[47] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[48] Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. Journal of the royal statistical society: Series B (Methodological), 36(2):111–133, 1974.

[49] David J Griffiths and Darrell F Schroeter. Introduction to quantum mechanics. Cambridge university press, 2018.

[50] Andreĭ Nikolaevich Kolmogorov and Albert T Bharucha-Reid. Foundations of the theory of probability: Second English Edition. Courier Dover Publications, 2018.

[51] John S Bell. On the einstein podolsky rosen paradox. Physics Physique Fizika, 1(3):195, 1964.

[52] Christopher M Dawson and Michael A Nielsen. The solovay-kitaev algorithm. arXiv preprint quant-ph/0505030, 2005.

[53] Wolfgang Paul. Electromagnetic traps for charged and neutral particles. Reviews of modern physics, 62(3):531, 1990.

[54] Samuel Earnshaw. On the nature of the molecular forces which regulate the constitution of the luminiferous ether. Transactions of the Cambridge Philosophical Society, 7:97, 1848.

[55] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. New Journal of Physics, 18(2):023023, 2016.

[56] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, and Dacheng Tao. Expressive power of parametrized quantum circuits. Physical Review Research, 2(3):033125, 2020.

[57] Trevor Lanting, Anthony J Przybysz, A Yu Smirnov, Federico M Spedalieri, Mohammad H Amin, Andrew J Berkley, Richard Harris, Fabio Altomare, Sergio Boixo, Paul Bunyk, et al. Entanglement in a quantum annealing processor. Physical Review X, 4(2):021041, 2014.

[58] Sukin Sim, Peter D Johnson, and Alán Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. Advanced Quantum Technologies, 2(12):1900070, 2019.

[59] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. Physical Review A, 98(3):032309, 2018.

[60] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta. Supervised learning with quantum-enhanced feature spaces. Nature, 567(7747):209–212, 2019.

[61] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. arXiv preprint arXiv:1802.06002, 2018.

[62] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. Quantum Science and Technology, 4(4):043001, 2019.

[63] Seth Lloyd and Christian Weedbrook. Quantum generative adversarial learning. Physical review letters, 121(4):040502, 2018.

[64] Shouvanik Chakrabarti, Huang Yiming, Tongyang Li, Soheil Feizi, and Xiaodi Wu. Quantum wasserstein generative adversarial networks. Advances in Neural Information Processing Systems, 32, 2019.

[65] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational quantum circuits for deep reinforcement learning. IEEE Access, 8:141007–141024, 2020.

[66] Andrea Mari, Thomas R Bromley, Josh Izaac, Maria Schuld, and Nathan Killoran. Transfer learning in hybrid classical-quantum neural networks. Quantum, 4:340, 2020.

[67] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. Nature Computational Science, 1(6):403–409, 2021.

[68] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R McClean. Power of data in quantum machine learning. Nature communications, 12(1):2631, 2021.

[69] Max Little, Patrick McSharry, Eric Hunter, Jennifer Spielman, and Lorraine Ramig. Suitability of dysphonia measurements for telemonitoring of parkinson's disease. Nature Precedings, pages 1–1, 2008.

[70] Ingo R Titze. Summary statement. In Workshop on acoustic voice analysis, National center for voice and speech, 1995.

[71] Max Little, Patrick Mcsharry, Stephen Roberts, Declan Costello, and Irene Moroz. Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection. Nature Precedings, pages 1–1, 2007.

[72] Alicia Sprecher, Aleksandra Olszewski, Jack J Jiang, and Yu Zhang. Updating signal typing in voice: addition of type 4 signals. The Journal of the Acoustical Society of America, 127(6):3710–3716, 2010.

[73] André A Spadoto, Rodrigo C Guido, Felipe L Carnevali, André F Pagnin, Alexandre X Falcão, and João P Papa. Improving parkinson's disease identification through evolutionary-based feature selection. In 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pages 7857–7860. Ieee, 2011.

[74] Ronald J Baken and Robert F Orlikoff. Clinical measurement of speech and voice. Cengage Learning, 2000.

[75] Jean Schoentgen. Jitter in sustained vowels and isolated sentences produced by dysphonic speakers. Speech Communication, 8(1):61–79, 1989.

[76] Dirk Michaelis, Matthias Fröhlich, and Hans Werner Strube. Selection and combination of acoustic features for the description of pathologic voices. The Journal of the Acoustical Society of America, 103(3):1628–1639, 1998.

[77] Alan Wolf, Jack B Swift, Harry L Swinney, and John A Vastano. Determining lyapunov exponents from a time series. Physica D: nonlinear phenomena, 16(3):285–317, 1985.

[78] Holger Kantz and Thomas Schreiber. Nonlinear time series analysis, volume 7. Cambridge university press, 2004.

[79] Michael H Krane. Aeroacoustic production of low-frequency unvoiced speech sounds. The Journal of the Acoustical Society of America, 118(1):410–427, 2005.

[80] Floris Takens. Detecting strange attractors in turbulence. In Dynamical systems and turbulence, Warwick 1980, pages 366–381. Springer, 1981.

[81] Mario Ragwitz and Holger Kantz. Markov models from data by simple nonlinear time series predictors in delay embedding spaces. Physical Review E, 65(5):056201, 2002.

[82] Daniel P Lathrop and Eric J Kostelich. Characterization of an experimental strange attractor by periodic orbits. Physical Review A, 40(7):4028, 1989.

[83] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M Sohaib Alam, Guillermo Alonso-Linaje, B AkashNarayanan, Ali Asadi, et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. arXiv preprint arXiv:1811.04968, 2018.