Technical University of Crete

Department of

Electrical and Computer Engineering

Quantum Computing for Generative Modeling and Applications

by

Dimitrios A. Komninos

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DIPLOMA OF
ELECTRICAL AND COMPUTER ENGINEERING

December 5, 2023

THESIS COMMITTEE

Professor Dimitrios G. Angelakis, *Thesis Supervisor*

Associate Professor Vasilios Samoladas

Professor Georgios Chalkiadakis

Abstract

This thesis dives into the intersection of quantum computing and generative modeling by exploring their relationship and potential applications across various domains, with a primary focus on finance. The journey begins with a comprehensive analysis of the mathematical framework behind quantum mechanics. Then, classical generative modeling techniques are presented, specifically restricted Boltzmann machines (RBMs) for data reconstruction and denoising and generative adversarial networks (GANs) for the generation of synthetic data, using the popular MNIST dataset as a benchmark.

Building on this foundational knowledge, we transition into the realm of quantum machine learning. The struggles of implementing a fault-tolerant quantum computer for learning tasks is presented and how we can approach such pieces of work through different angles with current available technology. We introduce parameterized quantum circuits (PQCs) and quantum circuit Born machines (QCBMs), two essential components of quantum computing for generative modeling and machine learning tasks in general. A key highlight of this section is the training of various topologies of Born machines on a simple dataset, showcasing the ability to effectively learn the underlying data distribution through quantum processes.

We then discuss how the above classical and quantum approaches can be used in the financial sector. Leveraging the power of generative modeling, a Wasserstein GAN with gradient penalty is employed to generate realistic financial time series data, using the S&P 500 index closing values as a benchmark. This marks a critical step towards synthesizing financial data for various analytical and predictive purposes.

At last, we introduce and study a quantum Wasserstein GAN (QWGAN) in the financial domain. Here, the traditional WGAN generator is replaced by a parameterized quantum circuit featuring diverse architectures. This novel approach not only has the potential to enrich the generative capabilities, but also harnesses the inherent quantum advantages for more, possibly, efficient and accurate data generation.

Key Words: *quantum mechanics, quantum computation, machine learning, generative modeling, finance*

Acknowledgements

First and foremost, I express my deepest appreciation to my thesis advisor, Professor Dimitrios Angelakis, whose unwavering guidance, expertise and patience have been indispensable throughout this journey. Your mentorship has not only shaped this thesis but also enriched my understanding of quantum computing and generative modeling. Also, I would like to thank my family for their constant support and encouragement. Your belief in my abilities and unwavering encouragement have been the driving force behind my pursuit of knowledge. I also extend my gratitude to my friends and peers who provided both moral support and stimulating discussions throughout this academic endeavor. Finally, I would like to express my appreciation to all those whose names may not appear in this section but whose contributions, however small, have been integral to the completion of this thesis.

Contents

1 Introduction

1.1 Quantum Computing Now and Beyond

Since 1981, it has been widely recognized that quantum theory can be utilized to simulate physics on a machine whose operation and behavior deviate significantly from classical mechanics, a concept introduced by Richard P. Feynman through the quantum computer model [1]. Quantum computers have proven to be really robust on dealing, or not, with a wide variety of problems and reach to a solution exponentially faster than their classical counterparts, although this is not the case for every class of tasks. As Feynman states, "It's not a Turing machine, but a machine of a different kind".

With the prospect of achieving (exponential) acceleration in demanding computational tasks, the competition to develop a quantum computer that is both scalable and resilient is in full swing [2]. A key milestone in this field will be when a universal quantum computer performs a computational task that is beyond the capability of any classical computer, an event known as *quantum supremacy*.

Several technology companies and research institutions have developed quantum hardware and quantum programming languages, making quantum computing more accessible than ever before. Quantum supremacy, demonstrated by Google's quantum processor in 2019, marked a significant milestone, showcasing quantum computers' computational advantage for specific tasks. However, it's crucial to recognize that quantum computers are still in the Noisy Intermediate-Scale Quantum (NISQ) era, where they are error-prone and have limited qubits [3]. This current phase necessitates substantial advancements in error correction, qubit stability and hardware scalability to fully unlock the transformative potential of quantum computing. Researchers and organizations around the world are tirelessly working to address these challenges, making the current state of quantum computing an exciting and dynamic landscape poised for groundbreaking developments in the near future.

1.2 Prospects of Quantum Generative Modeling

The integration of quantum computing into the realm of generative modeling offers a promising avenue for reshaping the landscape of artificial intelligence and machine learning. Quantum computing, grounded in the principles of quantum mechanics, presents unique opportunities and advantages that could potentially revolutionize the field of generative modeling in several

transformative ways. One of the most notable advantages lies in the realm of computational speed. Quantum computers have the potential to deliver exponential speedups over classical counterparts for specific computational tasks. This heightened processing power holds the promise of generating high-quality data samples at a remarkably accelerated pace, opening doors to real-time or near-real-time generative applications that were previously unattainable.

Additionally, quantum computing can significantly enhance the process of sampling from complex probability distributions, a fundamental requirement in generative modeling. Models such as Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) stand to benefit from this quantum advantage, enabling the generation of data samples that closely match intricate underlying distributions.

The synthesis of synthetic data is another area poised for transformation. Quantum computers have the potential to generate more diverse and realistic synthetic data, thereby facilitating improved data augmentation for training machine learning models. This, in turn, can lead to enhanced model generalization and performance, particularly in situations where training data is scarce. Furthermore, the combination of quantum machine learning techniques with generative modeling holds the potential to yield more potent and adaptable generative models capable of handling intricate data distributions. Quantum generative modeling holds promise for financial applications by leveraging quantum computing's ability to process complex data and perform rapid simulations, enabling more accurate risk assessments and portfolio optimization. Its capacity to handle high-dimensional financial data and generate probabilistic distributions can enhance decision-making processes and advance algorithmic trading strategies.

The proficiency in data compression and feature selection can also be harnessed to reduce the computational complexity of generative models. This promises more efficient and compact data representations, potentially alleviating resource-intensive model training processes. Moreover, quantum randomness, an inherent characteristic of quantum systems, can inject a new dimension of stochasticity. This expanded randomness can lead to more creative and diverse data generation, making quantum-enhanced generative models particularly valuable in creative applications like art generation and content creation. Finally, the security and privacy of generative models can be strengthened through the application of quantum cryptography. In domains where data confidentiality and integrity are paramount, such as healthcare and finance, quantum-enhanced security offers a robust safeguard.

1.3 Outline

In this work, we will present some generative models and their extended versions in terms of quantum computation, focusing on generative adversarial networks (GANs). The classical GAN framework blends beautifully with the notion of quantum-classical hybrid learning models in the NISQ era, as it can be scaled and adjusted by integrating a parameterized quantum circuit (PQC) whose parameters are optimized with a classical algorithm. We tackle with some common applications and focus primarly on the financial sector. A gentle outline of the thesis follows.

In Chapter 2, a thorough analysis of the mathematical framework behind quantum computing is presented. We present basic aspects of linear algebra such as vector spaces, orthonormal basis sets, unitary and hermitian operators. Quantum information, gates, circuits and properties such as superposition and entanglement are respresented by utilizing those aspects. The reader may skip this section if confident with quantum computing basics.

In Chapter 3, we dive into machine learning basics and focus on generative modeling. We present a restricted Boltzmann machine (RBM) and the math behind the model as a gentle introduction and train it on the popular MNIST dataset for the tasks of data reconstruction and denoising. Then, we deal with a more complex architecture known as generative adversarial network (GAN). Using the same dataset, we try to generate synthetic data from random noise inputs.

In Chapter 4, quantum machine learning (QML) is introduced. The focus lies in hybrid quantum-classical variational models that leverage the current resources in the NISQ era. A quantum circuit Born machine (QCBM) is thoroughly analyzed and trained on a simple dataset to show that quantum circuits have the ability to learn the underlying data distribution effectively with the help of current state-of-the-art classical optimization techniques. Also, quantum generative adversarial networks (QGANs) are introduced and an implementation follows in the the last chapter.

In the last and main chapter of this thesis, we concentrate on the financial sector, a promising domain for near-term applications of quantum generative modeling. First, we introduce financial time series and their properties, where the example at hand is the S&P 500 index. We present some applications of GANs regarding finance by providing various references. For this work, we chose to implement a more advanced classical model known as Wasserstein GAN that uses a gradient penalty term as an additional regularization technique. Research is conducted on how synthetic financial data can be generated that also exhibit real-world properties and the complexity of such a task. The idea is then extended by integrating a parameterized quan-

tum circuit (PQC) in the WGAN architecture. We introduce the Quantum Wasserstein GAN (QWGAN) model and make points on how it should be approached in order to generate realistic data. Motivated by another work, we try to generate financial time series and discuss the results.

2 Fundamentals of Quantum Computation

So, what is quantum computation ? The most fascinating fact about this question is that there are endless possibe explanations, from using relatively simple terms to extreme mathematical aspects starting from elementary abstract algebra. Our purpose is to present the preliminaries of quantum computation without neglecting the basic properties and definitions by utilizing aspects of linear algebra. For a thorough studying of quantum computing and information, the reader is refered to [4].

2.1 Qubits: Quantum States as Complex Vectors

In quantum mechanics, a *quantum state* is represented by a complex vector in a vector space we call the Hilbert space and is n-dimensional. The following definition follows the so-called *Dirac* notation for such vectors;

Definition 2.1.1 (*Complex Vector: Ket-Bra Notation*)**.** Let $u \in \mathbb{C}^n$. The *Ket* notation is a *quantum state* representation of u as a n-dimensional column vector as

$$|u\rangle = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} \tag{1}$$

where $u_1, u_2, ..., u_n \in \mathbb{C}$. The *Bra* notation is used to represent the *conjugate transpose* of $|u\rangle$ as a n-dimensional row vector as

$$\langle u| = \begin{pmatrix} u_1^* \\ u_2^* \\ \vdots \\ u_n^* \end{pmatrix}^T = \begin{pmatrix} u_1^* & u_2^* & \dots & u_n^* \end{pmatrix} \tag{2}$$

where $u_1^*, u_2^*, ..., u_n^* \in \mathbb{C}$. We will denote the conjugate transpose of $|u\rangle$ with the *dagger* symbol as $|u\rangle^\dagger = \langle u|$.

Generally, we are interested in *qubits*, which are vectors in $\mathbb{C}^2$. In correspondence to the the basic unit of information in classical computing which is the *bit* and takes the values 0 or 1, the *quantum bit* or *qubit* is the basic unit of information in quantum computing and defined as

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{3}$$

.

The following definitions encapsulate the essentials of quantum states.

Definition 2.1.2 (*Linear Combination*)**.** A vector $|u\rangle \in \mathbb{C}^n$ is a *linear combination* of vectors $|v_1\rangle$, $|v_2\rangle$, ..., $|v_m\rangle \in \mathbb{C}^n$, if $|u\rangle$ can be expressed as

$$|u\rangle = \sum_{i=1}^{m} a_i |v_i\rangle \tag{4}$$

where $a_i \in \mathbb{C}$.

Later on, we will see that a quantum system can be in a superposition of states and this is expressed as a linear combination of vectors. Superposition is a fundamental principal in quantum mechanics, not present in classical physics, where if we take a single electron as an example of a quantum system, the phenomenon states that the particle can be on all possible *spin states* in any direction when it is not observed (measured). Any quantum algorithm utilizes this effect during evaluation.

Definition 2.1.3 (*Spanning Set of Vectors*)**.** A *spanning set* of vectors $\{|v_i\rangle\}$, where $i = 1, 2, ..., m$ for $\mathbb{C}^n$, is a set in terms of which any other vector $|u\rangle \in \mathbb{C}^n$ can be written as a *linear combination*.

In this case, we say that the set $\{|v_i\rangle\}$ spans the Hilbert space. Also, we should note that a spanning set for a given vector space is not unique.

Definition 2.1.4 (*Linear Independence*)**.** We say that a set of non-zero vectors $|v_1\rangle$, $|v_2\rangle$, ..., $|v_m\rangle \in \mathbb{C}^n$ is *linearly independent* if

$$\sum_{i=1}^{m} a_i |v_i\rangle = 0 \;\Leftrightarrow\; a_i = 0,\ \forall\ i = 1, 2, ..., m \tag{5}$$

If one vector of the set can be written as a linear combination of other vectors in the set, the we say that the set is *linearly dependent*.

Definition 2.1.5 (*Basis*)**.** A *basis* in $\mathbb{C}^n$ is any set of vectors that is a *spanning set* and *linearly independent*.

As one may notice, because a spanning set of vectors is not unique in a given vector space, the same applies for a basis. Any basis in $\mathbb{C}^n$ consists of n vectors, called the *basis vectors*. Quantum states in n dimensions are straightforward generalizations of qubits in terms of basis vectors.

Definition 2.1.6 (*Quantum State - Born Rule*)**.** A *quantum state* is a vector $|\psi\rangle \in \mathbb{C}^n$ that is a *linear combination* of a *basis set* $\{|v_i\rangle\}$, $i = 1, 2, ..., n$, with coefficients $c_i \in \mathbb{C}$ as

$$|\psi\rangle = \sum_{i=1}^{n} c_i |v_i\rangle \tag{6}$$

The scalars $c_i \in \mathbb{C}$ associated with the basis vectors are called *probability amplitudes*, as in quantum mechanics they give the probabilities of projecting the respective state into a basis vector when the appropriate measurement is performed. This is called the *Born rule*. Thus, $c_i \in \mathbb{C},\ \forall i = 1, 2, ..., n$, correspond to a *probability distribution* and must obey

$$\sum_{i=1}^{n} |c_i|^2 = \sum_{i=1}^{n} c_i c_i^* = 1 \tag{7}$$

The probability that a system $|\psi\rangle$ is in state $|v_i\rangle$ is $|c_i|^2$.

Definition 2.1.7 (*Inner Product*)**.** Let $|u\rangle, |v\rangle \in \mathbb{C}^n$. The *inner product* between the two vectors is defined as

$$|u\rangle^\dagger \cdot |v\rangle = \langle u|v\rangle \tag{8}$$

with $\langle u|v\rangle \in \mathbb{C}$.

The inner product between complex vectors is not commutative in general, as opposed to the dot product between two vectors in Euclidean space $\mathbb{R}^n$.

Definition 2.1.8 (*Norm of a Vector*)**.** The *norm* of a vector $|u\rangle \in \mathbb{C}^n$ is defined as

$$\||u\rangle\| = \sqrt{\langle u|u\rangle} \tag{9}$$

and in an abstract way, it represents the length of $|u\rangle$.

Definition 2.1.9 (*Unit Vector - Normalization*)**.** Let $|u\rangle \in \mathbb{C}^n$, with $|u\rangle \neq \vec{0}$. Vector $|u\rangle$ is *normalized* if $\langle u|u\rangle = 1$, so its *norm* is unity. We call $|u\rangle$ a *unit vector* if

$$\||u\rangle\| = 1 \tag{10}$$

Definition 2.1.10 (*Mutually Orthogonal Vectors*)**.** Let $|u\rangle, |v\rangle \in \mathbb{C}^n$, with $|u\rangle, |v\rangle \neq \vec{0}$. The vectors are *mutually orthogonal* if $\langle u|v\rangle = \langle v|u\rangle = 0$.

Definition 2.1.11 (*Orthonormal Basis*)**.** A *basis* is *orthonormal* if all of its vectors are *unit vectors* and *mutually orthogonal*.

The very well-known *Gram-Schmidt Orthogonalization* method is used to construct an orthonormal basis $\{|v_i\rangle\}$ from an arbitrary basis $\{|u_i\rangle\}$ set of vectors. We are not going further on this, as it does not serve any purpose on this work. From now on, orthonormality is implied a priori. The basis

of interest on almost any quantum computing problem is the *computational basis*, consisting of the 2-dimensional vectors $|0\rangle$ and $|1\rangle$, where

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{11}$$

It is trivial to show that those vectors obey all of the above definitions. Other basis sets may also be used. In the next section, we will see how we can move between different basis sets of vectors with the use of *unitary operators*.

Interestingly, we can visualize the qubit as a point in the 3-d unit sphere, which is so-called the Bloch sphere. The quantum state of a system with one qubit w.r.t the computational basis is

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle\,, \quad a, b \in \mathbb{C}$$

As a point in the 3-d unit sphere (Bloch sphere), it is defined as

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle\,, \quad 0 < \theta \leq \pi,\ 0 \leq \phi \leq 2\pi \tag{12}$$

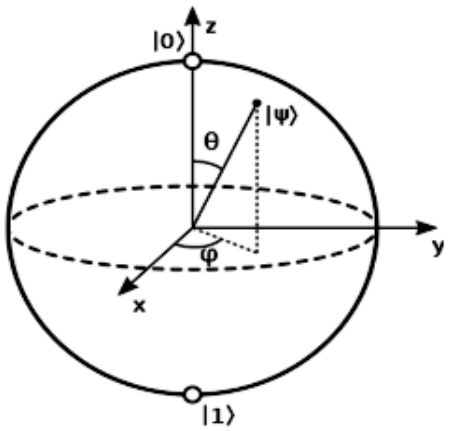


Figure 1: One-qubit quantum state on the Bloch sphere. While a classical bit can take only two values (0 or 1), the qubit can be in a superposition of basis states, a fact that can be exploited for exponential speedup.

In quantum computing, an orthonormal basis contains all the possible states that a system may be found in after measurement. The meaning of measurement will be clear later on. Following *Definition 2.1.6*, we want to use *normalized states* and *orthonormal basis* sets of vectors in order to respect

the probabilities and statistics of quantum mechanics. We also need to be able to distinguish the states after they are observed (measured).

Now that we have a well-defined structure for a quantum state, we would like to construct linear operators with certain properties, known as quantum gates, to act on states by preserving the norm. If the inner product is different after an action on the state, this would lead to a totally abstract basis. Such operators are the equivalent of logic gates in classical computing, with some examples being *NOT, AND, OR, XOR*. Next section is about quantum operations on states.

2.2 Quantum Gates as Complex Operators

Definition 2.2.1 (*Linear Operator*)**.** A *linear operator* $A : V \to W$ of dimensions $(n \times n)$ is a *transformation* or *mapping* from n-dimensional vector space V to vector space W of the same dimension and acts on vectors, namely quantum states.

Let $|\psi\rangle \in \mathbb{C}^n$ and a basis set $\{|v_i\rangle\}$, $i = 1, 2, ..., n$. Then:

$$A\,|\psi\rangle = A \cdot \sum_{i=1}^{n} c_i\,|v_i\rangle = \sum_{i=1}^{n} c_i A\,|v_i\rangle\,,\ c_i \in \mathbb{C} \tag{13}$$

We will only deal with *square* operators, as we want to map onto a space of equal dimension. The terms *linear operator* and *quantum gate* will be used interchangeably from now on.

Definition 2.2.2 (*Hermitian Operator*)**.** A linear operator $A : \mathbb{C}^n \to \mathbb{C}^n$ is *Hermitian if and only if* it is equal to its conjugate transpose denoted by $A^\dagger$, namely

$$A = A^\dagger \tag{14}$$

where $A^\dagger$ is obtained by interchanging the rows and columns of A and taking the conjugate of all the elements.

In quantum mechanics, operators that represent physical observables are Hermitian. This is because of certain properties that specify these operators:

- The *eigenvalues* are *real numbers*. We will see that this is mandatory as the only quantities that we can measure in a quantum system are the eignevalues of such operators. *Eigenvalues* and *eigenvectors* will be defined shortly.
- The *eigenvectors* constitute an *orthonormal basis set* for the given vector space.

Definition 2.2.3 (*Unitary Operator*)**.** A linear operator $U : \mathbb{C}^n \to \mathbb{C}^n$ is *unitary if and only if*

$$UU^\dagger = U^\dagger U = \mathbb{I} \tag{15}$$

where $\mathbb{I}$ is the identity matrix and the action preserves the norm of a given vector state.

By definition, it is easy to verify that a unitary operator is *orthogonal*, *normal* ($UU^\dagger = U^\dagger U$) and *invertible* ($U^\dagger = U^{-1}$). Thus, a crucial difference to the classical logics gates is that unitary quantum gates can be inverted. In classical computing, this is not true, as if we come up with some output of a logic gate, there is no way to find out the respective input. In quantum computing, this is feasible by constructing the conjugate transpose of the respective operator.

Unitary operators describe the time evolution of a quantum state. Some basic properties of unitary operators include:

- U is *diagonalizable*, meaning that it can be decomposed as $U = VDV^\dagger$, where V is unitary and D is a *diagonal matrix*.
- $|\det(U)| = 1$
- *Eigenspaces* are *orthogonal*.
- $U = e^{iH}$, the *matrix exponential* with $i = \sqrt{-1}$ and H a *Hermitian operator*. We will see how we can obtain this later on.
- The *eigenvalues* are complex numbers with *modulus 1*.
- If the eigenvalues are *non-degenerate*, then the eigenvectors are *mutually orthogonal*.

Unitary transformations are also used in order to transform the matrix representation of an operator in one basis to a representation of another basis. For simplicity, let us consider the Hilbert space for qubits $\mathbb{C}^2$ and two orthonormal basis sets $\{|u_i\rangle\}$, $\{|v_i\rangle\}$, $i = 1, 2$. The change of basis matrix from $\{|u_i\rangle\}$ to $\{|v_i\rangle\}$ is given by

$$U = \begin{pmatrix} \langle v_1|u_1\rangle & \langle v_1|u_2\rangle \\ \langle v_2|u_1\rangle & \langle v_2|u_2\rangle \end{pmatrix} \tag{16}$$

A state vector $|\psi\rangle$ in $\{|u_i\rangle\}$ is a state vector $|\psi'\rangle$ in $\{|v_i\rangle\}$, as $|\psi'\rangle = U|\psi\rangle$. An operator $A \in \mathbb{C}^2$ expressed in terms of $\{|u_i\rangle\}$ is the operator A' in basis $\{|v_i\rangle\}$, as $A' = UAU^\dagger$.

Definition 2.2.4 (*Commutator*)**.** The *commutator* of two operators $A, B : \mathbb{C}^n \to \mathbb{C}^n$ is defined as

$$[A, B] = AB - BA \tag{17}$$

If $[A, B] = 0$, then operators A, B *commute*. It is trivial to prove that if two operators commute, then they share common eigenvectors.

The most famous commutator relation is that of the position operator X and momentum operator P:

$$[X, P] = i\hbar\mathbb{I}$$

where $\hbar$ is the *reduced Planck constant*. This is what the *Heisenberg Acertainty* states, that position and momentum of a particle cannot be determined simultaneously. Also, we can define the *anticommutator* as $[A, B] = AB + BA$.

The *Pauli operators* is a set of 2×2 complex matrices which are *Hermitian* and *unitary* and defined as

$$\sigma_1 = \sigma_X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\sigma_2 = \sigma_Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

$$\sigma_3 = \sigma_Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Thus, these operators correspond to realizable quantum gates acting on qubits. Let us provide their main properties:

- $\sigma_1^2 = \sigma_2^2 = \sigma_3^2 = -i\sigma_1\sigma_2\sigma_3 = \mathbb{I}$
- $\det \sigma_j = -1,\ j = 1, 2, 3$
- $tr\sigma_j = 0,\ j = 1, 2, 3$
- The *eigenvalues* of all Pauli operators are ±1

The *normalized eigenvectors* are, respectively for $\sigma_j,\ j = 1, 2, 3$:

$$|\psi_{x+}\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}, \quad |\psi_{x-}\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$

$$|\psi_{y+}\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\i\end{pmatrix}, \quad |\psi_{y-}\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-i\end{pmatrix}$$

$$|\psi_{z+}\rangle = \begin{pmatrix}1\\0\end{pmatrix}, \quad |\psi_{z-}\rangle = \begin{pmatrix}0\\1\end{pmatrix}$$

Next, we provide basic information on *eigenvectors* and *eigenvalues*.

Definition 2.2.5 (*Eigenvectors and Eigenvalues*)**.** A given vector $|u\rangle \in \mathbb{C}^n$ is an *eigenvector* of an operator $A \in \mathbb{C}^{n\times n}$, if the following equation is satisfied:

$$A\,|u\rangle = \lambda\,|u\rangle \tag{18}$$

where $\lambda \in \mathbb{C}$ is the corresponding *eigenvalue*.

The number of eigenvalues coincides with the dimensions of the corresponding linear operator. To find the eigenvalues, we solve the *characteristic equation*

$$\det(A - \lambda\mathbb{I}) = 0 \tag{19}$$

where $\det(\cdot)$ denotes the *determinant* of matrix $A - \lambda\mathbb{I}$. If each of the eigenvectors of an operator is associated with a *unique* eigenvalue, we say that they are *non-degenerate*.

Later on, we will see that the eigenvalues of operators are the only realizable measurables in a quantum system. Also, we will see their connection to the probabilities of aj system, as well as for *gate decomposition* and matrices transforming to other orthonormal basis sets.

Definition 2.2.6 (*Outer Products*)**.** Let $|u\rangle, |v\rangle \in \mathbb{C}^n$. The *outer product* is defined as $|u\rangle\langle v|$ and it respresents a different notation for some operator $A \in \mathbb{C}^{n\times n}$.

Definition 2.2.7 (*Spectral Decomposition*)**.** A linear operator $A \in \mathbb{C}^{n\times n}$ is *normal* if it has *mutually orthogonal eigenvectors* and their normalized versions $\{|u_i\rangle\}$ can diagonalize the operator as

$$A = \sum_i \lambda_i\,|u_i\rangle\langle u_i| \tag{20}$$

where λ_i are the corresponding eigenvalues. Then:

$$A = VDV^\dagger$$

where V is a unitary matrix containing the eigenvectors of A in its columns and D is a diagonal matrix whose main diagonal elements are the eigenvalues of A.

Every quantum gate is unitary and by definition, normal. So, they can be decomposed using the spectral decomposition theorem. If we recall the eigenvalues and eigenvectors of the Pauli matrices, by the definition above we see that they reconstruct the same operators. This theorem provides the means to construct quantum gates with respect to other gates. Also, notice the interrelation with unitary transformations.

Next, we provide a very important relation which is useful in manipulating expressions that often occur in quantum computation. Additionally, sometimes we may know only the action of some operator on basis vectors and we would like to find the matrix respresentation. The *closure relation* is the tool.

Definition 2.2.8 (*Closure Relation*)**.** Given a basis set $\{|u_i\rangle\}$ in $\mathbb{C}^n$, the identity operator can be expressed as

$$\sum_{i=1}^{n} |u_i\rangle \langle u_i| = \mathbb{I}_{n\times n} \tag{21}$$

Let a state vector $|\psi\rangle \in \mathbb{C}^n$ with respect to the basis $\{|u_i\rangle\}$. If we denote the inner product $\langle u_i|\psi\rangle = c_i$, the arbitrary state $|\psi\rangle$ can be expanded in terms of basis $\{|u_i\rangle\}$ as

$$|\psi\rangle = \mathbb{I}\,|\psi\rangle = \sum_{i=1}^{n} |u_i\rangle \langle u_i|\psi\rangle = \sum_{i=1}^{n} c_i\,|u_i\rangle \tag{22}$$

For an operator A with respect to basis $\{|u_i\rangle\}$ we have

$$A = \mathbb{I}A\mathbb{I} = (\sum_{i=1}^{n} |u_i\rangle \langle u_i|)A(\sum_{j=1}^{n} |u_j\rangle \langle u_j|) \tag{23}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} \langle u_i|\,A\,|u_j\rangle\,|u_i\rangle \langle u_j| \tag{24}$$

where we see that $A_{ij} = \langle u_i|\,A\,|u_j\rangle$.

Definition 2.2.9 (*Trace of an Operator*)**.** Let an operator $A \in \mathbb{C}^{n\times n}$ with respect to a basis set $\{|u_i\rangle\}$ in $\mathbb{C}^n$. The *trace* of A is defined as the sum of its diagonal elements

$$tr(A) = \sum_{i=1}^{n} A_{ii} = \sum_{i=1}^{n} \langle u_i| A |u_i\rangle \tag{25}$$

Definition 2.2.10 (*Expectation Value of an Operator*)**.** Let an operator $A \in \mathbb{C}^{n\times n}$ and a quantum state $|\psi\rangle \in \mathbb{C}^n$. The *average value of measurement results* on the quantum state $|\psi\rangle$ after it is prepared multiple times is defined as

$$\langle A\rangle = \langle\psi| A |\psi\rangle \tag{26}$$

This value is not actually measured! We measure only the eigenvalues of operator A. This is purely a statistical result. For higher statistical moments, we have $\langle A\rangle, \langle A^2\rangle, ...$ The standard deviation as we know from elementary statistics is

$$\Delta A = \sqrt{\langle A^2\rangle - \langle A\rangle^2} \tag{27}$$

At last, we will see some introductory material on *projection operators* just because this section has to do with operators in general. More information on *projective measurements* later on this chapter.

For simplicity, let us consider an arbitrary qubit state $|\psi\rangle \in \mathbb{C}^2$ in terms of the computational basis $\{|0\rangle, |1\rangle\}$. The projection operator is defined as

$$P = |\psi\rangle \langle\psi| \tag{28}$$

and is Hermitian. If $|\psi\rangle$ is normalized, then $P^2 = P$. Also, if P_1, P_2 are two projection operators that commute, i.e $[P_1, P_2] = 0$, then their product $P_1P_2 = P_2P_1$ is also a projection operator. Notice that the spectral decomposition theorem allows us to write an operator A in terms of projection operators. The projection operator $P_i = |u_i\rangle \langle u_i|$ projects onto the subspace defined by the eignevalue λ_i. The eigenvalue represents a given measurement result for the operator A. So, projection operators represent a type of *measurement* described by quantum theory. Since the basis states satisfy the closure relation, we easily see that

$$\sum_i P_i = \mathbb{I} \tag{29}$$

The probability of finding the i-th outcome when a measurement is made on a system prepared in state $|\psi\rangle$ is $\langle\psi| P_i |\psi\rangle$.

2.3 The Postulates Of Quantum Mechanics

Now, we have all the tools to start working and modeling quantum systems. In this brief section, we provide the so-called *postulates of quantum mechanics*, where various interpretations have been presented throughout the years. I will state it as 4 laws that govern quantum theory.

1. **The State of a System as a Complex Vector**
   The state of a quantum system is represented by a complex vector $|\psi(t)\rangle$ at time t in a Hilbert space that contains all the information we can obtain about the system. We work with *normalized states*, such that $\langle\psi|\psi\rangle = 1$, which we call *state vectors*. The basic unit of information is the *qubit* in Hilbert space $\mathbb{C}^2$ and it is a state vector $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$, where $a, b \in \mathbb{C}$ and $|a|^2 + |b|^2 = 1$.

2. **Observable Quantities as Complex Operators**
   To every dynamical variable A that is physically a measurable quantity, there corresponds a *Hermitian* operator A whose eigenvectors form a *complete orthonormal basis* of the Hilbert space.

3. **Measurements as Eigenvalues**
   The possible results of measurement of a dynamical variable A are the *eigenvalues* a_i of the corresponding Hermitian operator A. Using the spectral decomposition theorem, we can write the operator A in terms of its eigenvalues and corresponding projection operators as

$$A = \sum_i a_i P_i \tag{30}$$

   where $P_i = |u_i\rangle\langle u_i|$ with respect to the basis $\{|u_i\rangle\}$. The probability of obtaining measurement result a_i is

$$Pr(a_i) = \langle\psi|\,P_i\,|\psi\rangle = tr(P_i\,|\psi\rangle\langle\psi|) \tag{31}$$

   The probability amplitude $c_i = \langle u_i|\psi\rangle$ gives us the probability of obtaining measurement result a_i as

$$Pr(a_i) = \frac{|c_i|^2}{\langle\psi|\psi\rangle} \tag{32}$$

   where usually $\langle\psi|\psi\rangle = 1$. A measurement result causes the *collapse of the wave function*, meaning that the system is left in state $|u_i\rangle$. The *post-measurement* state of the system is

$$\frac{P_i\,|\psi\rangle}{\sqrt{\langle\psi|\,P_i\,|\psi\rangle}} \tag{33}$$

4. **Time Evolution of the System - Schrödinger Equation**
   The time evolution of a *closed* (physically isolated) quantum system is governed by the *Schrödinger Equation*

$$i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = H|\psi(t)\rangle \tag{34}$$

   where H is an operator called the *Hamiltonian* of the system and corresponds to the total energy of the quantum system. The possible energies are the eigenvalues of H. The state at a later time t is

$$|\psi(t)\rangle = e^{-iHt/\hbar}|\psi(0)\rangle \tag{35}$$

   where $|\psi(0)\rangle$ is the initial state. Therefore, $U = e^{-iHt/\hbar}$ is the unitary operator that governs the time evolution of a quantum system. The action of operators on states is nothing else than aplying the Schrödinger equation.

2.4 Multiparticle Systems and Tensor Products

So far, we were concerned with quantum systems of a single particle, a qubit. In order to construct and analyze systems with more than one qubit, called *composite systems*, we make use of the *Kronecker product*, frequently seen as the *tensor product*.

Without loss of generality, we consider the two-particle case. Let two Hilbert spaces $H_1 = \mathbb{C}^{d_1}$ and $H_2 = \mathbb{C}^{d_2}$. With the tensor product, we can construct a larger Hilbert space H with dimensions $d_1 \cdot d_2$ as

$$H = H_1 \otimes H_2$$

where $H = \mathbb{C}^{d_1 d_2}$. For $d_1 = d_2 = 2$, the case of qubits, the Hilbert space of their composite system is $\mathbb{C}^4$. In general, if we have n qubits $\{|\psi_i\rangle\}_{i=1,\dots,n}$ then the composite system $|\psi\rangle$ is

$$|\psi\rangle = \bigotimes_{i=1}^{n}|\psi_i\rangle \tag{36}$$

and the Hilbert space is $\mathbb{C}^{2^n}$. The dimensions grow exponentially with respect to the number of qubits.

One of the basic properties regarding the tensor product between vectors $|x\rangle$, $|x_1\rangle$, $|x_2\rangle$, $|\phi\rangle$, $|\phi_1\rangle$, $|\phi_2\rangle$ is *linearity*, meaning that

$$|\phi\rangle \otimes (|x_1\rangle + |x_2\rangle) = |\phi\rangle \otimes |x_1\rangle + |\phi\rangle \otimes |x_2\rangle$$
$$(|\phi_1\rangle + |\phi_2\rangle) \otimes |x\rangle = |\phi_1\rangle \otimes |x\rangle + |\phi_2\rangle \otimes |x\rangle$$

and

$$|\phi\rangle \otimes (a\,|x\rangle) = a\,|\phi\rangle \otimes |x\rangle$$

where $a \in \mathbb{C}$. The basis of the composite system is the tensor product of basis sets of the corresponding qubits. Also, the tensor product is *noncommutative*

$$|\phi\rangle \otimes |x\rangle \neq |x\rangle \otimes |\phi\rangle$$

In column vector notation, if $|\phi\rangle = \begin{pmatrix} a \\ b \end{pmatrix}$ and $|x\rangle = \begin{pmatrix} c \\ d \end{pmatrix}$ then

$$|\phi x\rangle = \begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a \begin{pmatrix} c \\ d \end{pmatrix} \\ b \begin{pmatrix} c \\ d \end{pmatrix} \end{pmatrix} = \begin{pmatrix} ac \\ ad \\ bc \\ bd \end{pmatrix}$$

where often we omit the symbol $\otimes$ and we use the notation $|\phi\rangle\,|x\rangle$, or more simply $|\phi x\rangle$.

Tensor products are defined similarly for operators. Let $|\phi\rangle \in \mathbb{C}^{d_1}$ and an operator $A \in \mathbb{C}^{d_1 \times d_1}$. Let also $|x\rangle \in \mathbb{C}^{d_2}$ and an operator $B \in \mathbb{C}^{d_2 \times d_2}$ and the composite system

$$|\psi\rangle = |\phi\rangle \otimes |x\rangle$$

in $\mathbb{C}^{d_1 \times d_2}$. Then:

$$(A \otimes B)\,|\psi\rangle = A\,|\phi\rangle \otimes B\,|x\rangle$$

The operator $A \otimes B$ acts on Hilbert space $\mathbb{C}^{d_1 \times d_2}$ and its dimensions are $d_1 d_2 \times d_1 d_2$. Some properties include:

- If A, B are Hermitian, then $A \otimes B$ is also Hermitian.
- If A, B are projection operators, then $A \otimes B$ is a projection operator.
- If A, B are unitary operators, then $A \otimes B$ is unitary.
- If A, B are positive operators, then $A \otimes B$ is positive.
- $(A \otimes B)^\dagger = A^\dagger \otimes B^\dagger$
- $(A \otimes B)(C \otimes D) = AC \otimes BD$, where the dimension of rows of C is d_1 and the dimension of rows of D is d_2.

In matrix notation, consider $A, B \in \mathbb{C}^{2\times 2}$:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Then:

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{pmatrix}$$

In many cases of practical interest, rather than considering a single quantum system, we need to study a large number or collection of systems called an *ensemble*. For this purpose, the *density operator* is used, a well-defined tool for describing quantum channels with statistics. As this is out of the scope of this thesis, the reader is refered to [4] for further information if interested.

2.5 Measurement Theory

We have seen how we can define multi-qubit systems and manipulate them by using quantum gates to perform computations. Now, we wish to devise a way in order to extract information out of such systems. In the third postulate of quantum mechanics, we talked about measurement and wave function collapse. In contrast to the classical systems, it turns out that measurement does have a profound impact on a quantum mechanical system, altering its state in an *irreversible* way.

Let us consider a general qubit in the computational basis

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle$$

where $a, b \in \mathbb{C}$. After a measurement is made, the qubit will be forced into state $|0\rangle$ or state $|1\rangle$. Superposition of states is gone after the measurement. Thomas Young demonstrated this in 1802 with the famous double-slit experiment. The measurement of a quantum system involves some type of interaction or *coupling* of the system with the corresponding measuring device. A system *coupled* to an environment is called an *open system*, in conjuction with *closed systems* we discussed up to this point.

When we talked about quantum gates, we provided some information on *projective measurements*, also called *Von Neumann measurements* described by operators. These in turn, describe mutually exclusive possible states. Let us summarize the key properties below:

- $P = P^\dagger$, *Hermitian*
- $P = P^2$

- Projection operators are mutually orthogonal, so $P_i P_j = \delta_{ij} P_i = 0$, $\forall\ i \neq j$. With 0 we denote the zero matrix. Notice that this is a necessary and sufficient condition for the sum of such two or more operators to be also a projection operator.

- They form a complete set of orthogonal projection operators, so

$$\sum_i P_i = \mathbb{I}$$

- The number of projection operators in some Hilbert space is less, or equal to the dimensions of the respective space.

Let the Hilbert space $\mathbb{C}^n$ and consider a set of mutually orthogonal projection operators $\{P_1, P_2, ..., P_n\}$. Let also a system in an arbitrary state $|\psi\rangle$. The probability of finding the i-th outcome when a measurement is made is

$$Pr(i) = |P_i |\psi\rangle|^2 = \langle\psi| P_i^2 |\psi\rangle = \langle\psi| P_i |\psi\rangle = tr(P_i |\psi\rangle \langle\psi|)$$

Also, notice how the spectral decomposition theorem allows us to write general operators with respect to projection operators. Let some general operator $A \in \mathbb{C}^{n\times n}$ with eigenvalues $\{\lambda_i : i = 1, ..., n\}$ and eigenvectors $\{|u_i\rangle : i = 1, ..., n\}$. Then

$$A = \sum_{i=1}^{n} \lambda_i |u_i\rangle \langle u_i| = \sum_{i=1}^{n} \lambda_i P_i$$

The expectation value of A is thus

$$\langle A\rangle = \langle\psi| A |\psi\rangle = \sum_{i=1}^{n} \lambda_i \langle\psi| P_i |\psi\rangle$$

Now, consider a system after measurement. We say that the *wave function* $|\psi\rangle$ *collapses*, meaning that the system is found on some basis state and quantum properties have vanished. The system after measurement is

$$\frac{P_i |\psi\rangle}{\sqrt{\langle\psi| P_i |\psi\rangle}}$$

where the division is conducted to ensure that the state is normalized.

When we want to measure composite systems with more than two state vectors, we simply use the tensor product among the operators, combined

with the identity operator acting on states that we do not want to measure. For example, let two qubits of a composite system with respect to the computational basis be in a superposition of states as

$$\frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2}$$

If we want to measure only the first qubit, we use the operators $P_0 \otimes \mathbb{I}$ and $P_1 \otimes \mathbb{I}$. Similarly, for the second qubit only we would have $\mathbb{I} \otimes P_0$ and $\mathbb{I} \otimes P_1$. For both qubits simultaneously, we tensor the projection operators respectively.

The measurement theory can be generalized for general measurements with different properties. Also, the so-called *positive operator valued measures (POVM)* can be constructed by operators that are not projection operators in general. This allows for the construction of more general measurements where projective measurements do not apply. A common example includes the detection of a photon in a laboratory, where a POVM allows to describe the system without regard to the post-measurement state.

2.6 Entanglement

Quantum entanglement is a phenomenon that lies at the very heart of quantum mechanics, defying classical intuitions and challenging our understanding of the fundamental nature of the universe. At its core, entanglement represents a profound connection between particles, where the state of one becomes intrinsically linked to the state of another, regardless of the physical separation between them. This connection is so remarkable that, as Einstein famously stated, "spooky action at a distance" becomes a simple description. Entangled particles, whether they are electrons, photons, or any other quantum entities, exhibit correlations that defy classical logic and their behavior remains one of the most enigmatic and captivating aspects of quantum physics.

In this section, we will see some mathematical notation to represent quantum entanglement. Again, for further investigation regarding more advanced topics on entanglement refer to [4].

Definition 2.6.1 (*Bipartite Systems*)**.** A *bipartite system* $|\psi\rangle \in \mathbb{C}^{d_1 d_2}$ consists of two quantum states $|\phi\rangle \in \mathbb{C}^{d_1}$, $|x\rangle \in \mathbb{C}^{d_2}$ with basis sets $\{|\phi_i\rangle : i = 1, ..., d_1\}$ and $\{|x_j\rangle : j = 1, ..., d_2\}$ as

$$|\psi\rangle = \sum_{i=1}^{d_1} \sum_{j=1}^{d_2} c_{ij} |\phi_i\rangle \otimes |x_j\rangle \tag{37}$$

where $c_{ij} \in \mathbb{C}$ with

$$\sum_{i=1}^{d_1} \sum_{j=1}^{d_2} |c_{ij}|^2 = 1$$

.

For simplicity, let us consider the two-qubit system with respect to the computational basis, where

$$|\psi\rangle = c_{00}\,|00\rangle + c_{01}\,|01\rangle + c_{10}\,|10\rangle + c_{11}\,|11\rangle$$

with probability amplitudes $c_{ij} \in \mathbb{C}$.

Now that we have a notation for bipartite systems, let us see how an entangled vector looks like. For an example of a basis for a bipartite system, consider the *Bell basis*. The members of this basis are called the *Bell states* or the *EPR states*, they constitute an orthonormal basis in $\mathbb{C}^4$ and they are identified by the vectors

$$\left|\Phi^{\pm}\right\rangle = \frac{|00\rangle \pm |11\rangle}{\sqrt{2}}, \qquad \left|\Psi^{\pm}\right\rangle = \frac{|01\rangle \pm |10\rangle}{\sqrt{2}} \tag{38}$$

In quantum physics, $|\Psi^+\rangle$ is the *triplet* state or *spin-1 state* and $|\Psi^-\rangle$ is the *singlet* state or *spin-0* state.

One may write the Bell states more compactly as

$$|\beta_{xy}\rangle = \frac{|0y\rangle + (-1)^x\,|1\bar{y}\rangle}{\sqrt{2}} \tag{39}$$

where $\bar{y}$ denotes "*not* y"and y is called the *parity bit*. All Bell states are *maximally entangled* states.

In general, if there are some states $|\phi\rangle$, $|x\rangle$, such that $|\psi\rangle = |\phi\rangle \otimes |x\rangle$, meaning that $|\psi\rangle$ can be *factorized*, we say that state $|\psi\rangle$ is *separable* and there is no *entanglement*. About the "amount" of entanglement, one may consider the Shannon entropy as a measure of uncertainty by using the probability amplitudes of the corresponding state. The entropy of the Bell states is equal to 1, meaning maximum entanglement.

2.7 Quantum Gates, Circuits and Algorithms

In the last section of this chapter, we are going to present fundamentals of quantum computation, quantum circuit representation and aspects in designing quantum algorithms.

2.7.1 Single-Qubit Gates

Quantum gates can be thought of as an abstraction that represents information processing. As we have already seen, quantum gates are unitary operators represented by matrices. A remarkable fact emerging from this, is that quantum gates are reversible, where this is not the case for classical gates. Also, the number of inputs is equal to the number of outputs in quantum gates, contrary to the classical evaluation.

A quantum gate of n inputs and n outputs is represented by a square matrix of degree 2^n. The Pauli operators are single qubit gates and their action on the computational basis states is

$$X|0\rangle = |1\rangle, \quad X|1\rangle = |0\rangle \tag{40}$$

$$Y|0\rangle = i|1\rangle, \quad Y|1\rangle = -i|0\rangle \tag{41}$$

$$Z|0\rangle = |0\rangle, \quad Z|1\rangle = -|1\rangle \tag{42}$$

The Pauli X matrix is identified as the U_{NOT} operator, as shown by its action above. Notice that we are with respect to the computational basis.

Consider the basis $\{|+\rangle, |-\rangle\}$ where

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \qquad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

These vectors are the eigenvectors of the Pauli X matrix, so they constitute an orthonormal basis set for $\mathbb{C}^2$ because X is unitary. In order to find the representation of the Pauli operators in this basis, we use a unitary transformation. The change of basis matrix in this case is

$$U = \begin{pmatrix} \langle +|0\rangle & \langle +|1\rangle \\ \langle -|0\rangle & \langle -|1\rangle \end{pmatrix}$$

which is identified by the *Hadamard* matrix

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{43}$$

For this reason, we sometimes refer to the basis $\{|+\rangle, |-\rangle\}$ as the *Hadamard basis*. The Hadamard gate creates *superposition* when acted on the computational basis states, transforming them to the Hadamard basis vectors.

Projection operators are also single qubit gates, as measurement is applied by an action of some projection operator on a state vector.

2.7.2 Multi-Qubit Gates

In the case of a composite system, we use the tensor product as defined earlier in order to act multiple gates on some system by composing a gate out of single-qubit gates. By using the identity operator we can act on specific qubits, or with different operators upon a system all in parallel by expanding the operation.

So far, we have discussed about gates that can be expressed with a single tensor product. Let us give a definition concerning multi-qubit gates;

Definition 2.7.1 (*Local Operators*)**.** A unitary operator representing a quantum gate is said to be *local*, if it can be factorized to a single tensor product.

Examples of local operators in $\mathbb{C}^{4\times4}$ are $H\otimes H$, $X\otimes\mathbb{I}$, $\mathbb{I}\otimes Z$ and so on. Similarly, we expand the tensor product for larger systems by placing operators to desired positions as operands.

However, we want to create effects such as entanglement and this is not feasible with local operators. In this situation, we use *non-local* gates. The action of such gates on some qubit depends on the state of another qubit. For this reason, a non-local unitary operator cannot be factorized down to a single tensor product.

The most basic example of non-local operators are the *controlled* operators. Let us consider the *controlled-NOT* operator, which we denote by $U_{CNOT}\in\mathbb{C}^{4\times4}$. The action of this operator on the computational basis in $\mathbb{C}^4$ is

$$\begin{aligned} U_{CNOT}\,|00\rangle &= |00\rangle \\ U_{CNOT}\,|01\rangle &= |01\rangle \\ U_{CNOT}\,|10\rangle &= |11\rangle \\ U_{CNOT}\,|11\rangle &= |10\rangle \end{aligned}$$

We see that the second qubit is reversed when the first qubit is at state $|1\rangle$. We also need some measurement to determine the state of the first qubit. Using the closure relation, we can find the matrix elements of U_{CNOT} as

$$U_{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \left(\begin{array}{c|c} \mathbb{I} & \mathbf{0} \\ \hline \mathbf{0} & X \end{array}\right) \tag{44}$$

$$= P_0\otimes\mathbb{I} + P_1\otimes X \tag{45}$$

With the same manner, we can construct multiple qubit controlled gates by inserting more controls and actions, for example, CCX, CZ, XC, CH e.t.c.

The action of U_{CNOT} on two qubit system $|x\rangle \otimes |y\rangle$ can be written more compactly as

$$U_{CNOT} |x\rangle \otimes |y\rangle = |x\rangle \otimes |x \oplus y\rangle \tag{46}$$

where $\oplus$ denotes the *exclusive or* operation, where $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$. In general, the output is 1 if the number of 1's in the input is odd.

2.7.3 Circuits & Algorithms

We now move on to the construction and representation of quantum circuits. In general, a quantum circuit is a unitary operator composed by the product of local and non-local operators, the quantum gates. Let us consider the Bell basis vectors

$$\left|\Phi^{\pm}\right\rangle = \frac{|00\rangle \pm |11\rangle}{\sqrt{2}}, \qquad \left|\Psi^{\pm}\right\rangle = \frac{|01\rangle \pm |10\rangle}{\sqrt{2}} \tag{47}$$

and we work on the computational basis in $\mathbb{C}^4$. In order to create the EPR pairs, we act with the operator

$$U_{BELL} = U_{CNOT}(H \otimes \mathbb{I}) \tag{48}$$

in each of the basis states $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. The representation of a quantum circuit with input qubits $|0\rangle \otimes |0\rangle$ and output the entangled state

$$\left|\Phi^{+}\right\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

is shown below:

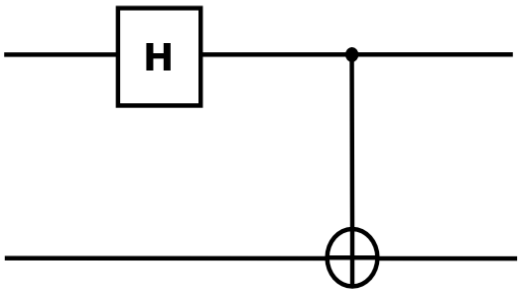


Figure 2: Bell Circuit creates entanglement between qubits in the computational basis.

The Bell vectors can be obtained by providing $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ as input states in the circuit above.

In general, quantum circuits are represented by n horizontal lines for n inputs and n outputs, local unitary actions are shown with squares on the respective qubit and non-local operations are identified with vertical lines showing the corresponding qubits. Lines are assumed to be connected by the tensor product. Also, the actions are written in reverse order of the mathematical expression and this is clear above where $H \otimes \mathbb{I}$ acts first on the inputs and U_{CNOT} comes right after. A last thing to mention, the so called *gate decomposition* has to do with decomposing gates into a series of single-qubit gates and U_{CNOT} operations. The latter is a *universal quantum gate*, meaning that any other gate can be decomposed in controlled-NOT gates and single qubit gates.

A quantum algorithm is a quantum circuit for some specific computation. We know that a quantum state of n qubits exists in a Hilbert space $\mathbb{C}$ of 2^n dimensions, with the basis vectors needed being also 2^n. This exponential growth of the domain dimensions, as a result of the linear increase in the number of qubits, is the main reason for the exponential increase in speed of quantum computations. Also, the fact that a quantum system may be in a superposition of states allows us to do simultaneous parallel computations that cannot be done, even in principle, on any classical computer.

More advanced algorithms take advantage of entanglement to fulfill their cause. Remarkable mentions include *Quantum Teleportation* as a network protocol for communication and *Superdense Coding* as a coding method for messages during communication. A detailed presentation along with quantum programming may be found in [5].

Other algorithms include the *Quantum Fourier Transform*, *Phase Estimation* algorithm, *Order Finding* algorithm and *Shor's* algorithm. The last utilizes all of the previous algorithms with some extra operations and the problem it tries to solve is to find the prime numbers that factorize a very large number. Again, see [5] for further investigation. With this algorithm, every current encryption protocol will be useless. However, Shor's algorithm is very sensitive to *quantum noise* and the interaction with the environment and needs really advanced methods from the field of *quantum error correction.*

We are mainly focused in the *noisy intermediate-scale quantum (NISQ)* devices with a few hundreds of qubits [3]. Such devices are very sensitive to the environment and may lose their quantum state due to *quantum decoherence.* They are not designed with advanced error-correction methods. However, a lot of achievements have been made on NISQ devices.

3 Deep Learning & Generative Models

3.1 Introduction

The concept of Artificial Intelligence, or AI for abbreviation, goes back to the 1930s since Alan Turing published his first work, one of the most important and influential papers in the history of computer science [6]. Research upon the subject grew rapidly in the following years and massively the last decade where we see various and complex techniques utilized in industry and influencing our lives every day.

More terms have been introduced throughout the years including *Machine Learning*, *Neural Networks* and *Deep Learning*. They should not be confused with one another. AI is a general concept and the others are just parts of it. The following picture and points help to clarify the matter.

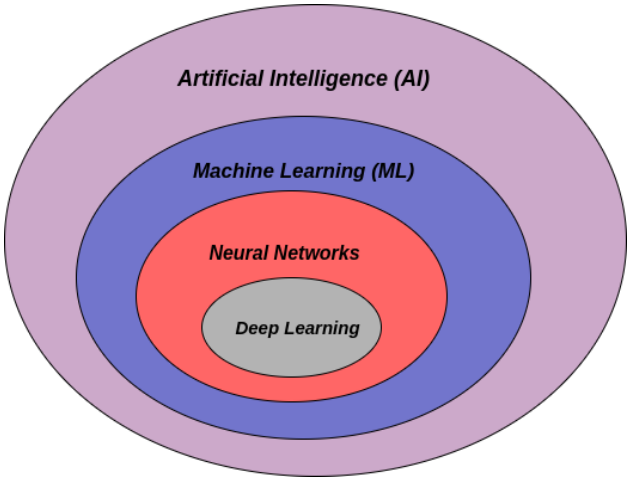


Figure 3: ML is not pure AI.

- **Artificial Intelligence**: A machine accomplishes a task that requires human intelligence.
- **Machine Learning**: The art of designing an AI model based on data.
- **Neural Networks**: A family of architectures of ML algorithms.
- **Deep Learning**: Neural networks with multiple layers of computation.

Every machine learning model is categorized first and foremost based on feedback and then, based on the task it is intended to execute. The next graph provides a visual categorization of ML systems.

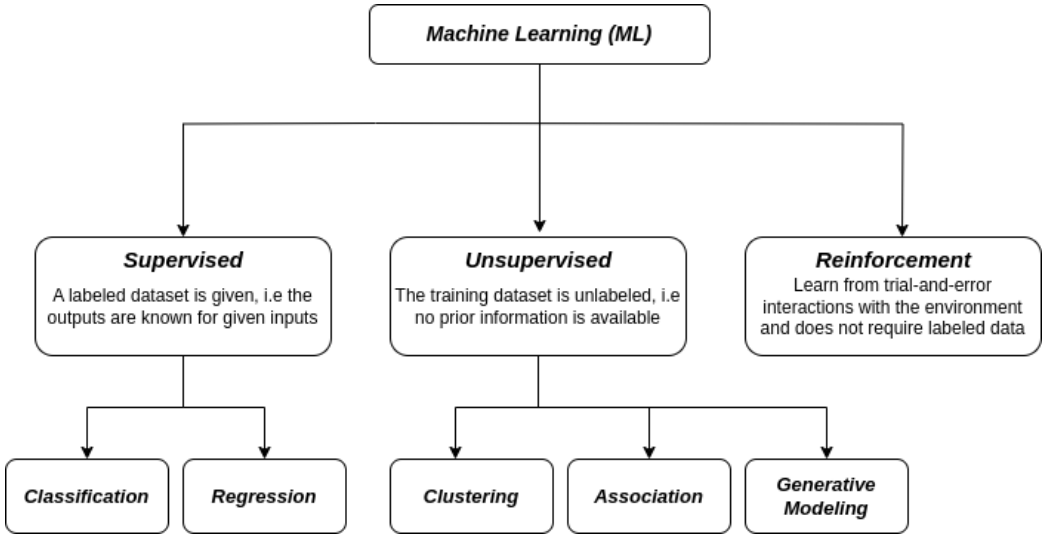


Figure 4: ML Systems Categorization.

For a detailed presentation on the concepts of neural networks and training, the reader is refered to Appendix A.

3.2 Generative Modeling

Generative modeling is a form of unsupervised learning where the training dataset is unlabeled. This means that there is no prior information on what the output should be with respect to the input data. The goal of the model is to identify the relationships and interdependencies in intricate datasets and then produce similar data by drawing from the learned distribution. That is, it models an unknown probability distribution and generates synthetic data. Models of this kind have found several applications including computer vision, speech synthesis, inference of missing text, noise removal from images, chemical design and much more.

Assume that we have an unknown target distribution p and the distribution learned by the model $q_{\vec{\theta}}$. The objective is to minimize the *divergence* $\mathcal{D}$ of these distributions, i.e

$$\vec{\theta}^* = \arg\min_{\vec{\theta}} \mathcal{D}(p, q_{\vec{\theta}}) \tag{49}$$

where $\vec{\theta}^*$ are the parameters such that the distribution represented by the model approximately conforms to the targeted distribution as indicated by the training data in terms of divergence in the statistical manifold. As the

actual distribution is not known a priori, it is estimated using a dataset $\{\vec{v}_i\}_{i=1}^{N}$ that is available to us and follows the distribution p.

In information theory and statistics, divergence $\mathcal{D}$ between two distributions is a kind of statistical distance that indicates how 'close' these distributions are. Consider the probability distributions p and q on a sample space $\mathcal{X}$. Of prime importance is the so-called *Kullback–Leibler* divergence or else, the *relative entropy* defined as

$$\mathcal{D}_{KL}(p,q) = \sum_{x\in\mathcal{X}} p(x)\log\left(\frac{p(x)}{q(x)}\right) = -\sum_{x\in\mathcal{X}} p(x)\log\left(\frac{q(x)}{p(x)}\right) \tag{50}$$

The relative entropy is always a non-negative value and it equals zero only when p and q are identical. Despite this fact, it cannot be considered as a true metric of distance between distributions because it fails to exhibit symmetry and does not adhere to the triangle inequality. Nevertheless, it is commonly viewed as a measure of "distance" between distributions for practical purposes, as in the case of training the generative models we discuss in this section.

Other loss functions used in generative modeling tasks include the *Wasserstein distance*, *Perceptual loss* and *Reconstruction loss*.

3.3 Boltzmann Machines

A Boltzmann machine is a unique category of neural network that serves as essential component in deep learning structures. Even today, they hold significant relevance in the field of both practical and theoretical machine learning. The idea for this type of network goes back to 1982 from J.J. Hopfield's work, where he presents a completely connected network comprising interdependent, deterministic units and possess the capability to store and recall binary patterns [10]. A Boltzmann machine [11] is a modified version of the Hopfield network consisting of stochastic units. Every unit modifies its state with time, relying on a probabilistic approach based on the state of neighboring units. This proposal addresses various problems of Hopfield networks that are presented in his work.

The general structure of a Boltzmann machine consists of the so-called *visible units* that play the role of the input layer and the *hidden units*, serving as underlying variables that shape a conditional and concealed representation of the data.

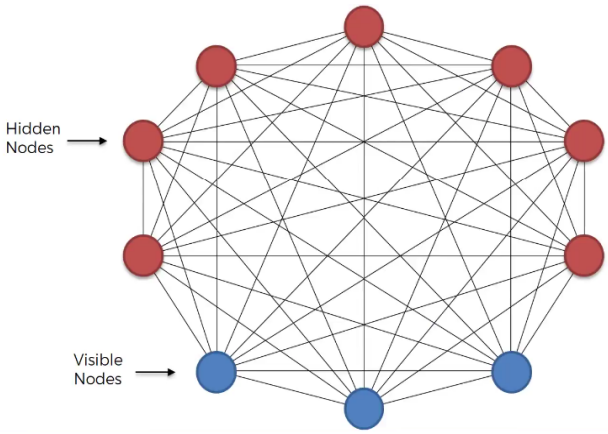


Figure 5: A Boltzmann machine of 3 visible units and 7 hidden units. Source: Google.

3.3.1 Restricted Boltzmann Machine

Interestingly, Boltzmann machines can be trained for generative modeling tasks by using an alternative structure called *restricted Boltzmann machine* [12, 13, 14]. This version tackles with the problem of learning the parameters of the model, which is computationally intensive due to full connectivity of the network. A restricted Boltzmann machine, on the other hand, retains a similar structure with the difference that the units in the same layer are not interconnected, resulting in a bipartite graph as presented below:

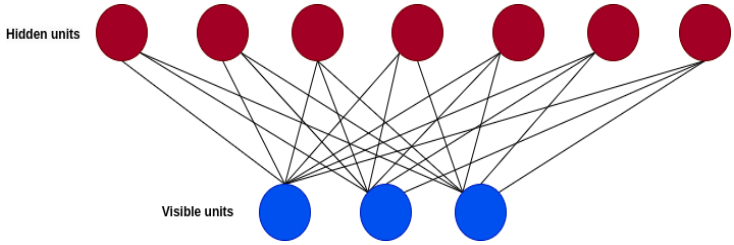


Figure 6: A restricted Boltzmann machine of 3 visible units and 7 hidden units.

A restricted Boltzmann machine (RBM) is a generative stochastic artificial neural network that can be used for unsupervised learning tasks, such as dimensionality reduction, feature learning, collaborative filtering, data denoising and topic modeling. RBMs are particularly useful for modeling complex distributions of high-dimensional data, such as images, speech sig-

nals, genomic data and have been successfully applied in a wide range of fields, including computer vision, speech recognition, natural language processing and bioinformatics. RBMs are also used as building blocks for more complex deep learning models, such as deep belief networks and deep autoencoders, which are used for tasks like image and text generation and anomaly detection.

Restricted Boltzmann machine units are binary stochastic units that exhibit probabilistic behavior, taking on a value of either 0 or 1. The joint state of the visible layer VL is a bitstring $\mathbf{v} \in \{0,1\}^V$, where V is the number of units. Same goes for the hidden layer HL as $\mathbf{h} \in \{0,1\}^H$. In a restricted Boltzmann machine, every unit possesses a bias and every connection has a weight. Functioning as a generative model, the machine characterizes a probability distribution. The model determines the joint distribution of each conceivable pair of visible and hidden vectors as

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z}\, e^{-E(\mathbf{v},\mathbf{h})} \tag{51}$$

Also, the marginal probability distribution of the visible layer can be evaluated by summing over all possible vectors of the hidden layer as

$$p(\mathbf{v}) = \sum_{\mathbf{h}} \frac{1}{Z}\, e^{-E(\mathbf{v},\mathbf{h})} \tag{52}$$

and that of the hidden layer by doing the opposite as

$$p(\mathbf{h}) = \sum_{\mathbf{v}} \frac{1}{Z}\, e^{-E(\mathbf{v},\mathbf{h})} \tag{53}$$

The energy value $E(\mathbf{v}, \mathbf{h})$ of a given joint configuration $(\mathbf{v}, \mathbf{h})$ in a restricted Boltzmann machine depends on the biases and pairwise interactions of the units as

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in VL} a_i v_i - \sum_{j \in HL} b_j h_j - \sum_{i,j} w_{ij} v_i h_j \tag{54}$$

which can be written in a more concrete way by defining the VL bias vector $\mathbf{a}$, the HL bias vector $\mathbf{b}$ and the weight matrix $\mathbf{W}$ between the layers, so

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{a}^T\mathbf{v} - \mathbf{b}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h} \tag{55}$$

In equations above, Z refers to the so-called *partition function* which is calculated by summing over all possible pairs of visible and hidden vectors as

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{56}$$

which can be interpreted as a normalizing constant to ensure that we have a valid probability distribution, i.e probabilities sum to 1. However, it is clear that the computational cost in order to calculate Z is of exponential time and feasible for problems with low dimensionality. This translates to the evaluation of the joint distribution as well.

A common technique to overcome this problem is by using an algorithm called *Gibbs Sampling*. Gibbs sampling is a type of Markov Chain Monte Carlo (MCMC) algorithm used to generate samples from a probability distribution [15]. The method is commonly used in machine learning and statistical inference. A starting value is chosen for each variable in the model, and then, in each iteration, one variable is updated by sampling from its conditional distribution given the values of the other variables. The process repeats, and after a sufficiently large number of iterations, the samples converge to the true underlying distribution. One of the advantages of Gibbs sampling is that it can be used to sample from complex distributions where direct sampling is not possible. However, convergence to the true distribution can be slow and the method may require a large number of iterations to produce accurate results.

By utilizing Bayes theorem and some simple mathematical operations, we can derive

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}) \tag{57}$$

and

$$p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h}) \tag{58}$$

hence, it implies conditional independence of visible units conditioned on all hidden units and vice-versa, as it is a product of probabilities.

At last, we can express (57), (58) using the sigmoid function σ as

$$p(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_i w_{ij} v_i\right) \tag{59}$$

and

$$p(v_i = 1|\mathbf{h}) = \sigma\left(a_i + \sum_j w_{ij} h_j\right) \tag{60}$$

respectively.

3.3.2 Learning with a restricted Boltzmann machine

In order to train a restricted Boltzmann machine, we recall what we said in the beginning of this section on generative modeling and Appendix A.

Consider a training dataset with N samples $\mathcal{S} = \{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, ..., \mathbf{v}^{(N)}\}$ following the distribution $p_{data}(\mathbf{v})$, where $\mathbf{v} \in \mathcal{S}$. We denote the model distribution as $q_\theta(\mathbf{v})$.

We are interested in minimizing the relative entropy between the model and the data distribution

$$\mathcal{D}_{KL}(p_{data}, q_\theta) = \sum_{\mathbf{v}\in\mathcal{S}} p_{data}(\mathbf{v}) \log\left(\frac{p_{data}(\mathbf{v})}{q_\theta(\mathbf{v})}\right) \tag{61}$$

as

$$\arg\min_\theta \mathcal{D}(p_{data}, q_\theta) \tag{62}$$

If we work this out, we can see that minimizing the relative entropy is equal to maximizing the *log-likelihood*

$$\arg\min_\theta \mathcal{D}(p_{data}, q_\theta) = \arg\min_\theta \sum_{\mathbf{v}\in\mathcal{S}} (p_{data}(\mathbf{v}) \log p_{data}(\mathbf{v}) - p_{data}(\mathbf{v}) \log q_\theta(\mathbf{v})) \tag{63}$$

$$= \arg\max_\theta \sum_{\mathbf{v}\in\mathcal{S}} p_{data}(\mathbf{v}) \log q_\theta(\mathbf{v}) \tag{64}$$

Gradient-based optimization methods are typically employed to maximize the log-likelihood. The log-likelihood for a particular vector $\mathbf{v}$ and parameter θ can be obtained by using (52) as

$$\ln q_\theta(\mathbf{v}) = \ln\left(\sum_{\mathbf{h}} \frac{1}{Z}\, e^{-E(\mathbf{v},\mathbf{h})}\right) \tag{65}$$

$$= \ln \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} - \ln \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{66}$$

The gradient with respect to the parameter θ is

$$\nabla_\theta \ln q_\theta(\mathbf{v}) = -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \nabla_\theta E(\mathbf{v}, \mathbf{h}) + \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v}, \mathbf{h}) \nabla_\theta E(\mathbf{v}, \mathbf{h}) \tag{67}$$

which comes by differentiation of (65). For the given dataset of N samples, the log-likelihood gradient is

$$\frac{1}{N} \sum_{l=1}^{N} \nabla_\theta \ln q_\theta(\mathbf{v}^{(l)}) = \mathbb{E}_q[\nabla_\theta E(\mathbf{v}, \mathbf{h})] - \mathbb{E}_{p_{data}}[\nabla_\theta E(\mathbf{v}, \mathbf{h})] \tag{68}$$

where $\mathbb{E}$ denotes the expectation value with respect to the corresponding distribution.

For the restricted Boltzmann machine, the parameter θ corresponds to all weights and biases of the model. From the last equation, we derive the following

$$\Delta w_{ij} = \mathbb{E}_{p_{data}}[v_i h_j] - \mathbb{E}_q[v_i h_j] \tag{69}$$
$$\Delta a_i = \mathbb{E}_{p_{data}}[v_i] - \mathbb{E}_q[v_i] \tag{70}$$
$$\Delta b_j = \mathbb{E}_{p_{data}}[h_j] - \mathbb{E}_q[h_j] \tag{71}$$

hence, we update the parameters as

$$w_{ij}^* = w_{ij} + \eta \Delta w_{ij} \tag{72}$$
$$a_i^* = a_i + \eta \Delta a_i \tag{73}$$
$$b_j^* = b_j + \eta \Delta b_j \tag{74}$$

where η is the learning rate.

The expectation value of the probabilities of the hidden layer with respect to the training data $\mathbb{E}_{p_{data}}$ can be easily obtained and it's called the *positive phase*. The second term $\mathbb{E}_q$ is called the *negative phase* and calculates the joint probability of the visible and the hidden layer. As we said previously, the evaluation of this expression is exponential. It depends on the size of the smallest layer because the joint probability $p(\mathbf{v}, \mathbf{h})$ can be expressed with respect to both conditional probabilities $p(\mathbf{h}|\mathbf{v})$ or $p(\mathbf{v}|\mathbf{h})$.

A Markov chain Monte Carlo (MCMC) algorithm for sampling is once again used to obtain the negative phase of our model. The samples are collected from the Markov chain at the point of reaching the steady-state distribution. However, if the chain is allowed to converge at every iteration, it would result in a high computational cost. To circumvent this issue, we employ a technique called *Contrastive Divergence (CD)* [14], which helps us avoid the computational overhead. The intuition behind CD is that the positive phase computes the activations that the RBM expects to see when the input is a training sample, while the negative phase computes the activations that the RBM generates when the input is generated from the RBM itself. By minimizing the difference between these activations, CD encourages the RBM to generate samples that are similar to the training samples.

To reduce the computational burden, one can initialize the Markov chain with dataset samples, thereby bringing it closer to the target distribution. Additionally, instead of computing the expectation over the entire converged distribution, we can obtain a single sample $\mathbf{v}^k$ by running the Markov chain for k steps. This enables us to bypass most of the computational expense involved in obtaining samples from the fully converged distribution.

Hinton has explained that 'contrastive divergence' can be viewed as the disparity between two Kullback-Leibler divergences. He also argues that we

should not focus on minimizing the relative entropy between the original data distribution and the fully converged distribution of the Markov chain, but we can minimize the following:

$$\mathcal{D}_{KL}(p_{data}(\mathbf{v}), q_\theta(\mathbf{v})) = \mathcal{D}_{KL}(p_{data}(\mathbf{v}), q_\theta(\mathbf{v})) - \mathcal{D}_{KL}(p_k(\mathbf{v}), q_\theta(\mathbf{v})) \tag{75}$$

where $p_k(\mathbf{v})$ is the distribution after k steps of the Markov chain. For the majority of problems, a single iteration of Gibbs sampling is adequate in practice.

3.3.3 Data Denoising and Reconstruction

In this subsection, we will focus on the tasks of data denoising and reconstruction. RBMs can be adjusted to remove different types of noise that may be present on the input samples. The MNIST dataset of handwritten digits will be used due to simple statistical patterns of the data. RBMs may not be effective on very complex statistics because of their limited architecture in terms of the number of layers and units in each layer.

Training restricted Boltzmann machines turns out to be a very challenging procedure. To train the model, we follow Hinton's practical guide to training restricted Boltzmann machines [16] which outlines some important aspects to consider. In [7], there is a notebook where I train a RBM on Fashion-MNIST to reconstruct images. The purpose of this is to show that it is more challenging to achieve a lower reconstruction error on datasets with complex statistics. Also, there are some tests with different hyperparameters in order to present the effects that Hinton states in his guide regarding mometum and learning rate decrease.

The MNIST dataset consists of 60,000 training samples and 10,000 validation samples. Each sample is an image of 28×28 pixels with values in [0,255]. As the RBM requires binary input, we do some simple pre-processing and normalize the data to binary, i.e [0,1], with a threshold of 127. Below, we can see some samples from MNIST before normalization:

Figure 7: MNIST handwritten digits.

First, we train the model on the MNIST dataset such that it learns the underlying distribution of the data. By using the CD algorithm to maximize the log-likelihood and minimize the reconstruction error with just one step of Gibbs sampling, we get sufficient quality reconstructions of handwritten digits without overfitting to the training data. The reconstruction error on the training set is really close to than of the validation set, meaning that the model generalizes well to unseen data.

The pictures below show original handwritten digits and their reconstructions from the RBM. The model was trained for 20 epochs, a batch size of 10 and a learning rate of 0.0125. The choice of the learning rate came after several tests which can be found in the notebook where I used the Fashion-MNIST dataset for experimentation purposes. The RBM consists of 784 visible units (equal to the number of pixels in an image) and 400 hidden units in order to retain a balance between training time, reconstruction quality and denoising performance. Also, just one step of Gibbs sampling was sufficient for achieving adequate results.

Figure 8: Original handwritten digits from the validation set.

Figure 9: The reconstructed digits from the validation set.

Since the RBM has learned the underlying statistics of the data we are interested in, we can use this for the task of data denoising.

Figure 10: 2% bit-flipping noise.

Figure 11: Denoised images from RBM.

For higher-levels of noise, as well as other types (e.g Gaussian noise), one may use more complex architectures and methods, such as deep-belief networks (stacked RBMs that form a larger network), convolutional networks or autoencoders which are suitable and shown to be effective for tasks involving image processing. Various learning rate and sampling schedules, along with hyperparameter optimization and fine-tuning are some aspects that may be further explored.

3.4 Generative Adversarial Networks

A *Generative Adversarial Network (GAN)* is a more advanced technique for generative modeling tasks and has been a central area of research since its conception in 2014 by Ian J. Goodfellow et.al [17]. It consists of two ANNs that compete with each other in a zero-sum game. These two sub-networks are commonly seen as the *generator* and the *discriminator* respectively.

The following picture represents the general structure of a GAN:

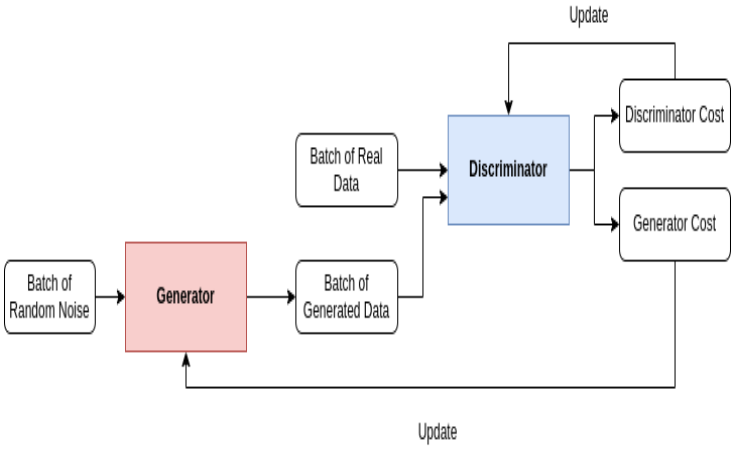


Figure 12: GAN Training Pipeline.

Some of the applications of GANs include image and video synthesis, data augmentation, style transfer, text generation, anomaly detection and drug discovery.

3.4.1 Overview

The generator's task is to generate conceivable data. The discriminator is mainly a classifier and it is trained using the generated instances as negative examples and the real data as positive examples. By differentiating between the generator's synthetic data and real data, the discriminator penalizes the generator for creating unrealistic outputs. At the start of training, the generator's output is evidently artificial and the discriminator learns to identify it as such. With time, the generator's performance improves and it generates more realistic data, leading to the discriminator making more classification errors and ultimately decreasing its accuracy. If the generator is trained effectively, the discriminator's ability to distinguish between real and fake data diminishes, causing it to misclassify fake data as real and lower its accuracy.

As we can see, the generator tries to maximize the error of the discriminator, whereas the latter wants to minimize the generator's error by providing information feedback to update its parameters.

Neural networks typically require input data to operate. However, when a network generates novel instances as output, we must determine an appropriate form of input to supply to the network. At a fundamental level, a GAN utilizes random noise as its input. Subsequently, the generator processes this noise into a relevant output, allowing the GAN to produce a diverse range of data instances by drawing from distinct locations in the desired distribution. Empirical studies indicate that the nature of the noise distribution is not significantly relevant. Hence, we can use a simple distribution, such as a uniform distribution, for noise generation. Furthermore, the noise space is typically of smaller dimension than the output space for ease of use.

Training a GAN is a complex process because its training algorithm must address two complications. The first is that we have to train two different networks. Thus, there is a need of scheduling two procedures. This leads to the second difficulty, the identification of the convergence of the GAN training as a whole. A method of alternating training is adopted during the design of a GAN. First, the discriminator trains for one or more iterations. Then, the generator trains for one or more iterations as well. These two steps are repeated to train the two respective networks. During the discriminator training phase, we keep the generator constant, meaning that we do not proceed to the update of its parameters. Respectively, we keep the discriminator constant during the generator's training.

In the original GAN formulation [17], the discriminator tries to predict the probability of a sample being real or fake. This is often done by using a sigmoid activation function in the discriminator output layer. In that case, a natural loss function is the *binary cross-entropy* and the optimization task can be formulated as

$$\min_G \max_D \mathop{\mathbb{E}}_{x \sim P_r} [\log (D(x))] + \mathop{\mathbb{E}}_{\tilde{x} \sim P_g} [\log (1 - D(\tilde{x}))]$$

where G is the generator, D the discriminator, x is a sample from the distribution of real data P_r and $\tilde{x}$ is a generated sample from the modeled distribution P_g.

In the best case scenario where the generator performs exceptionally well, the discriminator would have a 50% accuracy, indicating that it is no better than random guessing at classification. The evolution of the generator and discriminator relationship presents a challenge for the convergence of the GAN. As training progresses, the feedback from the discriminator to the generator becomes less relevant and if training persists beyond this point, the

generator might receive spurious feedback, ultimately resulting in a decline in quality.

3.4.2 Design Considerations: Simple Application

Designing a GAN is a very complex procedure and there are some concerns that must be taken into consideration for each task at hand:

- Types of generator and discriminator networks (multi-layer, convolutional, recurrent etc).
- Types of cost functions for the respective networks.
- The problem of *vanishing/exploding gradients*, where the loss function saturates and there cannot be meaningful updates to the generator network (due to the discriminator network being more powerful, which results to high accuracy in distinguishing between real and fake data).
- The problem of *mode collapse*, where the generator fails to generalize and produces only some subsets (modes) of the data.

Nevertheless, GANs have been successfully employed for image generation, data augmentation, fraud detection, style transfer and text inference among others. The following picture depicts the output of a GAN trained on MNIST with TensorFlow [8] for 50 epochs, a batch size of 64, learning rate of 0.0001 and parameter $\beta = 0.5$ using Adam optimizer and binary cross-entropy loss for both networks [7].

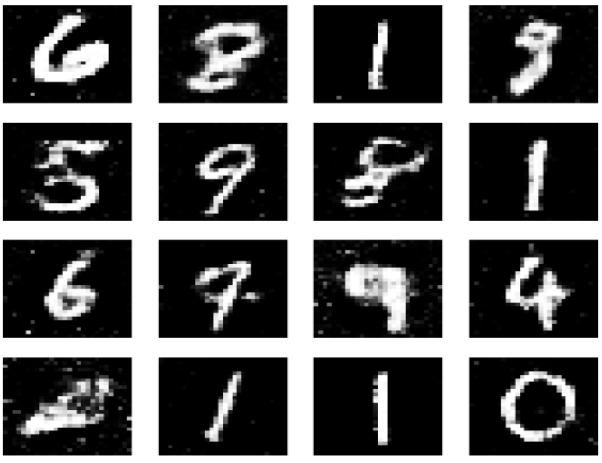


Figure 13: Notice that some outputs do not resemble a digit at all and others are noisy, so adjustments are needed for better performance.

4 Quantum Machine Learning

4.1 A Gentle Introduction

Quantum Machine Learning (QML) refers to the exploitation of quantum algorithms as a part of a larger and complex implementation. Quantum algorithms have the potential to outperform classical ones for specific problems resulting to the so-called *quantum speedup*. This notion has to do with the number of *queries* and *gates* needed by some algorithm, as well as a *scaling advantage* between a wide variety of problems. The latter, however, would require the implementation of a quantum computer in order to get a comparison between the classical and quantum realms, so it is a *benchmarking problem*. Moreover, the best performance for some classical algorithms is not always known for certain tasks. Consider the *integer factorization* problem. Recalling some fundamentals of complexity theory, this problem belongs to the $\mathcal{NP}$ class of exponential complexity, but there is no proof that it can be solved in polynomial time or if it is $\mathcal{NP}$-complete. In 1994, Peter Shor developed a quantum algorithm that can solve the problem in polynomial time [18]. But with the current available technology, it is far from being realized as it requires a *fault-tolerant* quantum computer that can scale up to a few thousands qubits and is not sensitive to noise, in contrast to the state of the art quantum processors that exist today with a few tens to hundreds of qubits and a high sensitivity to noise and errors.

Quantum speedups are idealized models for quantifying resources, as they are still characterized by measures of classic complexity theory. Research is being conducted on how to map this idealization to reality. Since fault-tolerant quantum computing is a long way off from this, study is focused on near-term devices that can perform quantum computations. Currently, the community is interested in *noisy-intermediate scale quantum (NISQ)* devices. These refer to quantum computers with limited qubits and error-prone operations, yet are capable of performing specific quantum tasks beyond classical computing capabilities.

Nevertheless, quantum algorithms that exhibit speedup have already been proposed. For example, common linear algebra subroutines such as Fourier transforms, finding eigenvectors and eigenvalues and solving systems of linear equations fall into the category of *quantum-based linear algebra subroutines (qBLAS)* [19] based on the fact that quantum mechanics, by nature, is all about statistics and linear algebra. Surprisingly, qBLAS translate into quantum speedups for a broad collection of machine learning and data analysis algorithms such as principal component analysis (PCA) [20], support vector machines (SVMs) [21], gradient descent and more (see Table 1 below).

Method	Speedup
Bayesian Inference [32]	$O(\sqrt{N})$
Least-squares fitting [33]	$O(\sqrt{N})$
Quantum PCA [34]	$O(\log N)$
Quantum SVM [35]	$O(\log N)$

Table 1: Speedups of QML algorithms compared to the classical counterparts.

QML is a general approach and techniques such as supervised, unsupervised or reinforcement learning may be adopted. However, there is a distinction between data and algorithms used. In fact, based on this difference we can design hybrid classical-quantum systems where some parts of a specific problem involve classical computations and others are processed using current state-of-the-art NISQ devices. A gentle categorization of based on data and algorithms is shown in the following figure.

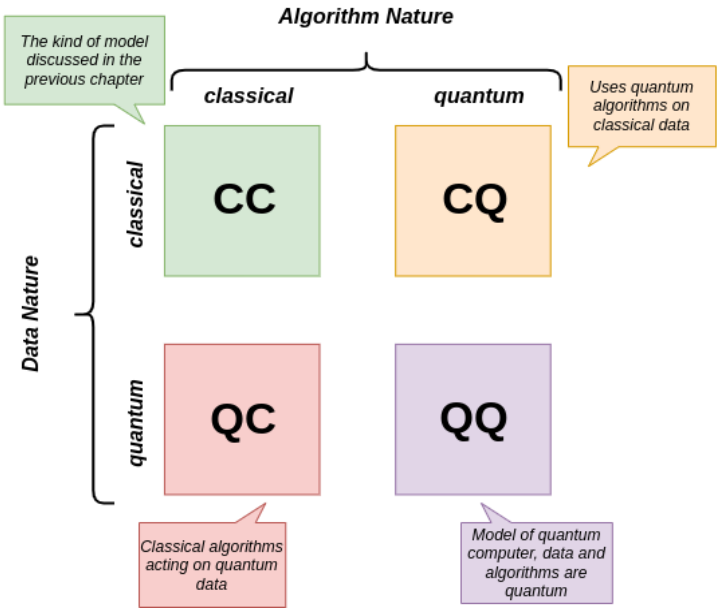


Figure 14: Hybrid systems categorization based on data and algorithms.

4.1.1 Fault-tolerant Quantum Computing

The idea of fault-tolerant computing goes back to the 1950s, when J. von Neumann introduced the *threshold theorem* [27] stating that an ideal circuit of $p(n)$ gates can be simulated to precision ε by a gate-faulty circuit

of $\mathcal{O}(p(n)\log^c(p(n)/\varepsilon)$ gates (c is constant), whose components have an error rate p below some constant threshold p_c and under fair assumptions regarding hardware noise. About half a century later, Peter Shor stated that for any quantum computation with t gates, a polynomial size quantum circuit that tolerates $\mathcal{O}(1/\log^c t)$ amounts of inaccuracy and decoherence per gate can be implemented by showing that operations can be performed on quantum data encoded by quantum error-correcting codes without decoding this data [28]. In this way, the *quantum fault-tolerance theorem* was introduced.

There are several concerns around fault-tolerant quantum computers. One of the main difficulties in implementing such hardware is *decoherence*, which tends to destroy the superposition of states resulting in information loss and infeasibility of long computations. Moreover, the accumulation of the inaccuracies of quantum state transformations during computation make the system unreliable. To gain better intuition, we present a fundamental subroutine of QML algorithms and quantum computation field in general named after its originators *Harrow, Hassidim and Lloyd* as the *HHL* algorithm [29].

HHL attempts to solve systems of linear equations with a quantum computer, specifically $A\vec{x} = \vec{b}$, where $\vec{x}, \vec{b} \in \mathbb{C}^N$ and $A \in \mathbb{C}^{N\times N}$, by constructing a quantum state proportional to $A^{-1}|b\rangle$, where A^{-1} is the inverse matrix of A. In the case where A is not square or has zero eigenvalues, HHL can be generalized to find the state $|x\rangle$ that minimizes $|A|x\rangle - |b\rangle|$ [30]. The best classical algorithm computes $\vec{x}$ in $\mathcal{O}(N\log N)$, whereas HHL promises $\mathcal{O}((\log N)^2)$ quantum steps to output $|x\rangle$. However, there are serious limitations that are related to fault-tolerant quantum computations regarding encoding and decoding of data.

The Input problem: The first issue is that vector $\vec{b}$ must be encoded to the quantum state $|b\rangle$ of $\log N$ qubits and be prepared on a quantum device. This requires the use of a *quantum random access memory (qRAM)* [31] that takes $\mathcal{O}(N)$ operations divided by $\mathcal{O}(\log N)$ steps that can be performed in parallel in order to map data vector $\vec{b}$ to the state $|b\rangle$. Nevertheless, designing a qRAM induces a high cost for big data which makes it infeasible at the time of speaking, but without it, there is a risk of losing the quantum speedup. Encoding should be done in polynomial time.

Speaking about encoding, we present three general techniques that can be adopted, since this is not a standard process and relies on the nature of the problem:

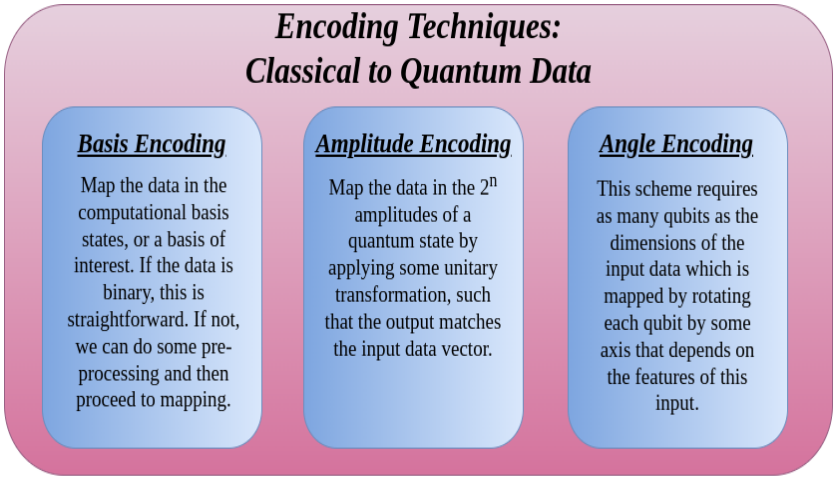


Figure 15: Encoding schemes.

The Output problem: The second matter in question is reading the solution from state $|x\rangle$. Decoding takes $\mathcal{O}(N)$ repetitions to reconstruct the components of $\vec{x}$ and get the full answer from $|x\rangle$. The latter contains the values of $\vec{x}$ encoded in its $\log N$ amplitudes. Again, this imposes a risk on speedup, but this issue is related to the probabilistic nature of measurements and quantum mechanics which is inevitable.

The Simulation problem: At last, we are concerned with the matrix of coefficients A. The condition number connected with $A\vec{x} = \vec{b}$ establishes a limit on the level of imprecision in the approximation of the solution vector $|x\rangle$. This must be low, so that A is well-conditioned because it affects the number of times that the state preparation circuit needs to be applied to succeed. Also, unitary operations of the form e^{-iAt} should be efficiently simulated by a quantum computer for a wide range of t values. The simulation should be polynomial in time in order to retain speedup.

To sum up, it is evident that the HHL algorithm needs a higly fault-tolerant device with immense cost, so it is far from being implemented with towering accuracy. However, just like the quantum Fourier transform algorithm, we are not interested on the process itself. If special constraints are taken care of, HHL yields a significant subroutine for achieving speedups compared to some classical machine learning algorithms. Some remarkable mentions include *Bayesian Inference* [32], *quantum PCA* [34] and *quantum SVMs* [35]. HHL also offers a blueprint for encoding and preparing state $|x\rangle$, implementing unitary transformations and measuring the results. Most notably, it demonstrates how to calculate the cost of these operations and assess their efficiency compared to classical algorithms. The following figure

depicts the problems that the reseachers are concerned with when designing a quantum computer.

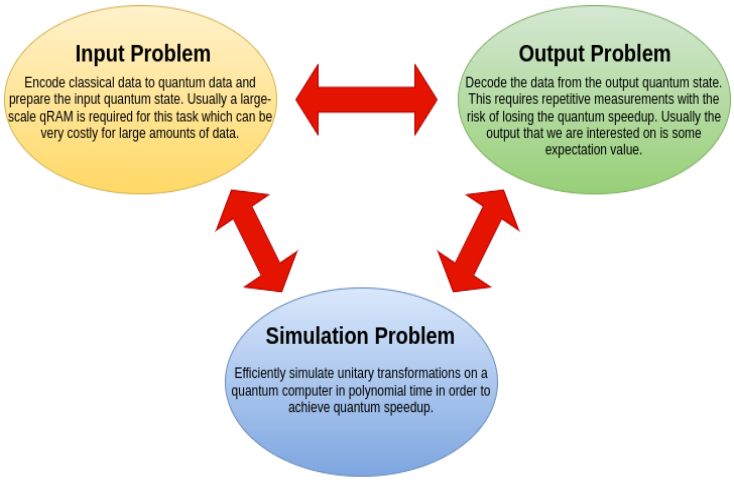


Figure 16: Problems in the design of a fault-tolerant quantum computer.

4.1.2 Noisy-Intermediate Scale Quantum (NISQ) Era

Error-free quantum computers capable of handling millions of qubits are a prerequisite for many proposed quantum algorithms, a requirement that stands in stark contrast to the current quantum processors available, which only provide a few unreliable qubits.

NISQ refers to the current state of quantum computing, which involves quantum processors that have a limited number of qubits (usually fewer than 100) and suffer from significant quantum noise and errors [3]. These quantum processors are considered to be at an intermediate stage between classical computing and full-scale quantum computing. Despite the limitations of NISQ processors, researchers and businesses are still actively exploring ways to use them for practical applications. The goal is to develop algorithms and techniques that can work with the noise and limited number of qubits in NISQ processors to solve problems that are currently beyond the reach of classical computers. Some areas of research in NISQ include quantum chemistry, machine learning and optimization.

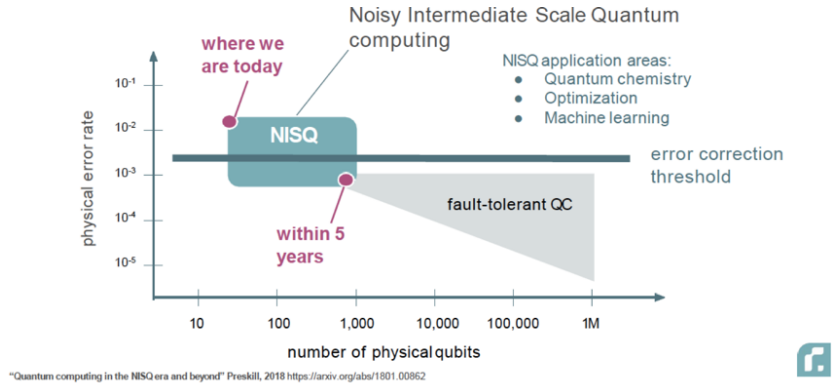


Figure 17: Physical error rate vs Number of physical qubits. Source: [3] Figure is from 2018. The current state-of-the-art quantum computer in terms of the number of qubits is IBM's Osprey superconducting quantum computer, which has 433 qubits and was announced in November 2022. It has a physical error-rate of approximately 0.8% for two-qubit gate operations and 3.7% for readout errors.

The dominant blueprint for obtaining quantum advantage in the NISQ era is the so-called *variational models* [36]. It is a type of machine learning model that are based on the principle of *variational inference*. Variational inference is a mathematical framework for approximating the posterior distribution of a probabilistic model, given some observed data. In variational models, the idea is to model the posterior distribution as a simpler distribution, typically a parametric distribution such as a Gaussian and to find the parameters of this distribution that best approximate the true posterior distribution. The best approximation is typically found by minimizing the difference between the true posterior distribution and the approximated one, as measured by a certain objective function called the *variational lower bound*.

Variational models have become popular in machine learning because they offer several advantages over other methods for approximating posterior distributions, such as Markov Chain Monte Carlo (MCMC) methods [15]. For example, variational models are computationally efficient and can be trained on large datasets, making them well-suited for real-world applications. They also have the ability to learn complex distributions, which can be useful in a variety of settings such as Bayesian deep learning and generative models. Moreover, they have been applied to a wide range of machine learning tasks, including density estimation, latent variable models and reinforcement learning. Despite their success, however, they are not without limitations and ongoing research is focused on finding ways to improve the accuracy and scalability of these models.

The following figure represents the variational model approach as a part of a larger general hybrid system implementation:

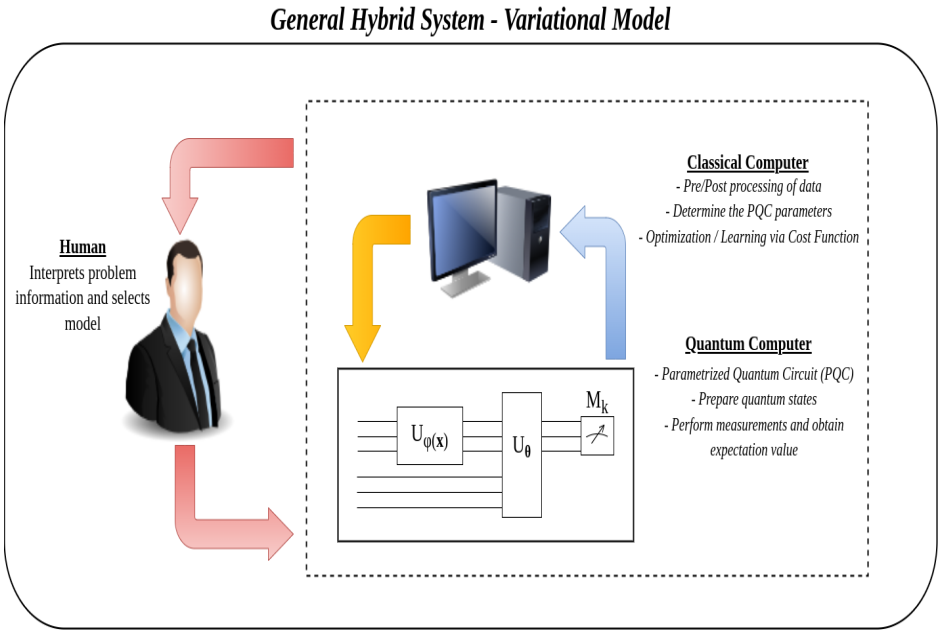


Figure 18: Variational Model Approach.

In summary, the way variational models work is fairly straightforward. The data is mapped into quantum states using a parameterized quantum circuit $U(x;\theta)$ [37]. Then, an output measurement is taken to produce an expectation value. The quality of the prediction is assessed using a cost function, with optimal parameters found through minimizing the cost via classical optimization techniques. This classical optimization step is handled by a classical computer in order to minimize the burden on quantum devices. The flexibility of variational models makes them ideal for dealing with the limitations of current quantum computers.

4.2 Quantum Generative Modeling

Quantum generative modeling is a cutting-edge field that harnesses the power of quantum computing to generate data distributions and has the potential to impact a wide range of industries and scientific domains. Its development is closely tied to advancements in quantum hardware and hybrid classical-quantum algorithms. Quantum generative models leverage quantum properties, such as superposition and entanglement, to perform certain computations faster and more efficiently. The key point is to develop algorithms

that can learn to produce samples from joint probability distributions, which is an important task in a wide variety of fields. Some notable examples include *quantum autoencoders* [38], *quantum Boltzmann machines (QBMs)* [39] *quantum circuit Born machines (QCBMs)* [40, 41, 42] and *quantum GANs* [63, 64].

As the scope of this thesis is to present some generative modeling applications, we will focus on QCBMs and QGANs, where the latter is an extended idea of an application we will present on financial time series in the next chapter.

4.2.1 Parameterized Quantum Circuits

The quantum equivalent of a neural network is a parameterized quantum circuit (PQC) [37]. As the name suggests, it consists of unitary gates that are parameterized with a set of parameters. Basically, a PQC is decomposed in a series of adjustable gates such as single-qubit rotation gates or entangling gates. For instance, we can have parameterized rotations along the X, Y and Z axes respectively with the operators

$$R_X(\theta) = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix}, \quad R_Y(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$$

$$R_Z(\theta) = \begin{pmatrix} e^{-i(\theta/2)} & 0 \\ 0 & e^{i(\theta/2)} \end{pmatrix}$$

or coupling gates between two qubits to introduce entanglement, such as the Mølmer-Sørensen gate,

$$\begin{aligned} XX(\theta_{ij}) &= \exp\left(-\frac{i}{2}\theta_{ij} X_i \otimes X_j\right) \\ &= \begin{pmatrix} \cos\left(\frac{\theta_{ij}}{2}\right) & 0 & 0 & -i\sin\left(\frac{\theta_{ij}}{2}\right) \\ 0 & \cos\left(\frac{\theta_{ij}}{2}\right) & -i\sin\left(\frac{\theta_{ij}}{2}\right) & 0 \\ 0 & -i\sin\left(\frac{\theta_{ij}}{2}\right) & \cos\left(\frac{\theta_{ij}}{2}\right) & 0 \\ -i\sin\left(\frac{\theta_{ij}}{2}\right) & 0 & 0 & \cos\left(\frac{\theta_{ij}}{2}\right) \end{pmatrix} \end{aligned}$$

where X is the Pauli gate and θ_{ij} is a parameter.

Of course, we can use different parameters for each rotation and coupling gate. The general machine learning model of classical pre/post-processing and a PQC is shown below:

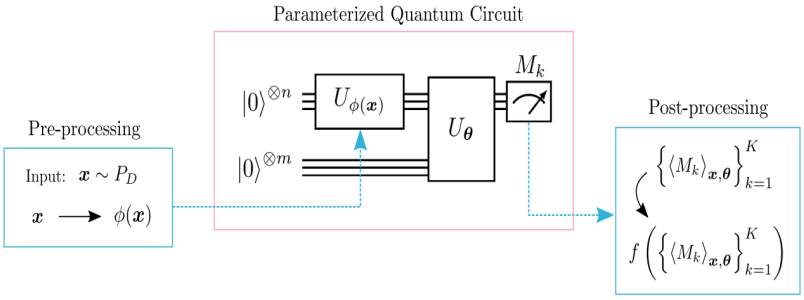


Figure 19: Source: [37]

A data vector is sampled from the dataset distribution, $\mathbf{x} \sim P_D$. During pre-processing, map it to the vector $\phi(\mathbf{x})$ that parameterizes the encoder circuit $U_{\phi(\mathbf{x})}$. A variational circuit U_θ, parameterized by a vector θ, acts on the state prepared by the encoder circuit and possibly on an additional register of ancilla qubits, producing the state $U_\theta U_{\phi(\mathbf{x})} |0\rangle$. A set of observable quantities $\{\langle M_k \rangle_{\mathbf{x},\theta}\}_{k=1}^{K}$, is estimated from the measurements. These estimates are then mapped to the output space through classical post-processing function f. For a supervised model, this output is the forecast associated to input $\mathbf{x}$. Generative models can be expressed in this framework with small adaptations. The output may be obtained indirectly via expectation values, or directly using projective measurements such as the Born machine model we discuss next. For further reading on PQCs, the reader is referd to [37].

4.2.2 Quantum Circuit Born Machines

A *Quantum Circuit Born Machine (QCBM)* is a *parameterized quantum circuit (PQC)* model for unsupervised generative modeling tasks that was first proposed in 2018 by researchers at MIT and have since garnered significant interest in the quantum computing community. It a type of what we call a *quantum neural network (QNN)*. The samples x are generated directly via projective measurements on the qubits using the Born rule

$$x \sim P_\theta(x) = |\langle x | \psi(\theta) \rangle|^2 \tag{76}$$

where they follow a probability distribution $P_\theta(x)$ with parameters θ. The quantum system wave-function $\psi(\theta)$ is prepared by applying a parameterized unitary as:

$$|\psi(\theta)\rangle = U(\theta) |0\rangle \tag{77}$$

The architecture of the PQC $U(\theta)$ is based on current NISQ capabilities at the time of speaking.

The ansatz for the QCBM is comprised of layers of parameterized gates, where two types of layers are commonly used. The first one contains *arbitrary single-qubit rotations*, while the second type corresponds to the so-called *entangling layer* which contains coupling gates between two qubits. The circuit is designed by stacking these two types of layers consecutively. A high-level overview of a QCBM is shown below:

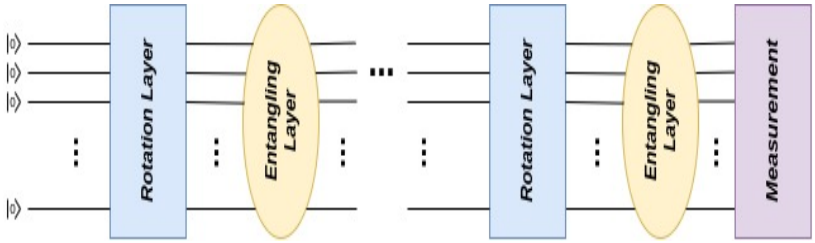


Figure 20: QCBM Ansatz.

Arbitrary single-qubit rotation layers

Generally, the rotation layer consists of 3 parameterized rotation gates that correspond to rotations on the Bloch sphere, applied on each qubit. These unitary gates are the usual R_X, R_Y, R_Z rotations around each axis respectively. Let us define the unitary

$$U^{(l)}(i) = R_z^{(l,i)}(\alpha_i) \cdot R_x^{(l,i)}(\beta_i) \cdot R_z^{(l,i)}(\gamma_i) \tag{78}$$

where l is the layer index and $\alpha_i, \beta_i, \gamma_i$ are the respective parameters of R_Z, R_X, R_Z, each acting on qubit i.

For N qubits, an arbitrary rotation layer requires $3N$ parameters. Since the circuit is initialized to the state $|0 \dots 0\rangle$, we can omit the first rotation $R_Z^{(l,i)}(\alpha_i)$ around the Z-axis since only a phase is added and the result is the same state. This reduces the number of trainable parameters in the first layer to $2N$. Moreover, if an odd number of layers is used, the last layer of the circuit corresponds to a rotation layer and we can omit the last rotation $R_Z^{(l,i)}(\gamma_i)$, as the phase is not measurable by the Born rule.

Reducing the number of parameters is crucial, as shallow quantum circuits are more preferable for NISQ devices. Deeper circuits significantly affect the fidelity of quantum states due to decoherence. Also, more parameters result in more complex optimization techniques. As QCBMs utilize the Born rule for generating samples directly, we can leverage this and reduce the number of parameters of the model by choosing a specific sequence of rotations.

Entangling layers

The entangling layers contains coupling gates between two qubits and the architecture depends on the experimental platform at hand. In previous implementations on trapped-ion quantum computers, the entangling layers consisted of Mølmer-Sørensen gates. Also, one may use not parameterized gates for coupling, such as CZ.

As for the complexity of this layer, the number of entangling gates differ depending on the connectivity we wish to implement. We will study 5 possible topologies, the *chain*, *star*, *ring*, *all-to-all* and *grid*. For 4 qubits, these are illustrated below:

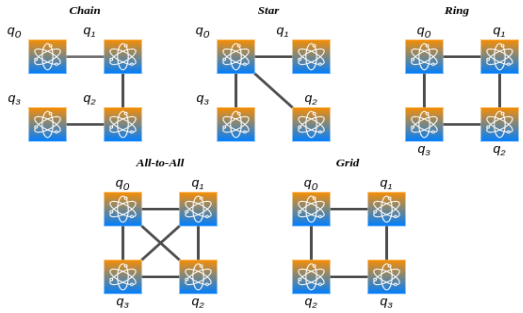


Figure 21: Topologies of the entangling layers.

Training a QCBM

To approximate an unknown target distribution $P_{\mathcal{D}}$, a QCBM is trained using a *hybrid quantum-classical variational model*. The training is performed on a dataset $\mathcal{D}$ consisting of independent and identically distributed (i.i.d) samples from $P_{\mathcal{D}}$. The model generates samples that are similar to the target distribution by adjusting the circuit parameters through classical optimization, which uses the output of a quantum computer as input.

Classical optimization may involve a gradient-based or a gradient-free method. Using a gradient-free method, such as particle swarm optimization (PSO) [65], is more preferable as there is no need to devise a differentiable cost function and thus, gradients computations are circumvented. In [41], the authors devise an efficient gradient-based learning algorithm for the QCBM by minimizing the kerneled maximum mean discrepancy loss and show the complexity of this process, as well as the advantages over gradient-free optimization schemes.

For our purpose, we opt to go with PSO and use the *clipped negative log-likelihood* as the cost function:

$$C_{nll} = -\sum_{x\in\{0,1\}^N} P_{\mathcal{D}}(x) \cdot \log\left\{\max\left(\epsilon, P_\theta(x)\right)\right\} \tag{79}$$

where ϵ is a quantity close to zero in order to avoid singularities that occur when the estimated probability of some data is zero and thus, the cost is infinite due to the logarithmic function. Typical values are in the range of $\epsilon = 0,001$ or $\epsilon = 0,0001$. In [42], the authors present the results of training a QCBM with various topologies for the entangling layers, using the clipped negative log-likelihood cost function and two different gradient-free optimization methods, PSO and Bayesian optimization [66]. They use a simple dataset called Bars-and-Stripes (BAS), which can be easily encoded using the basis encoding scheme.

The Bars and Stripes (BAS) dataset is a simple synthetic dataset used in machine learning and pattern recognition tasks. It consists of images of bars and stripes, where bars are horizontal and stripes are vertical. The images are represented as binary matrices where 1 represents a filled pixel and 0 represents an empty pixel. The images that belong to the dataset comprise a subset of pictures with $n \times m$ pixels. From simple combinatorics, it is trivial to derive that for $n \times m$ pixels, there are $BAS(n, m) = 2^n + 2^m - 2$ images in the subset. The simplest case is the $BAS(2, 2)$. The figure below presents the data that is in the BAS set along with the target distribution and those that do not correspond to BAS samples.

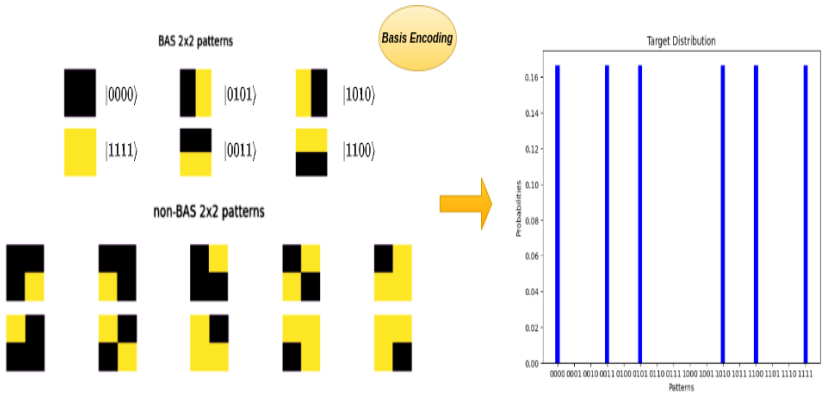


Figure 22: Each BAS sample can be interpreted as a N-dimensional vector $\mathbf{x}$, where $N = n \cdot m$. Since the patterns are black and white, we have binary vectors $\mathbf{x} \in \{0, 1\}^N$. Thus, we can have one-to-one mapping with the computational basis of N-qubit quantum systems, i.e $\mathbf{x} \to |x_1 x_2 \dots x_N\rangle$. For simplicity, we assume that the samples are i.i.d.

Using the *Qiskit* library for quantum programming purposes [67], we present the QCBM circuits with 2 and 4 layers for the topologies we mentioned previously:

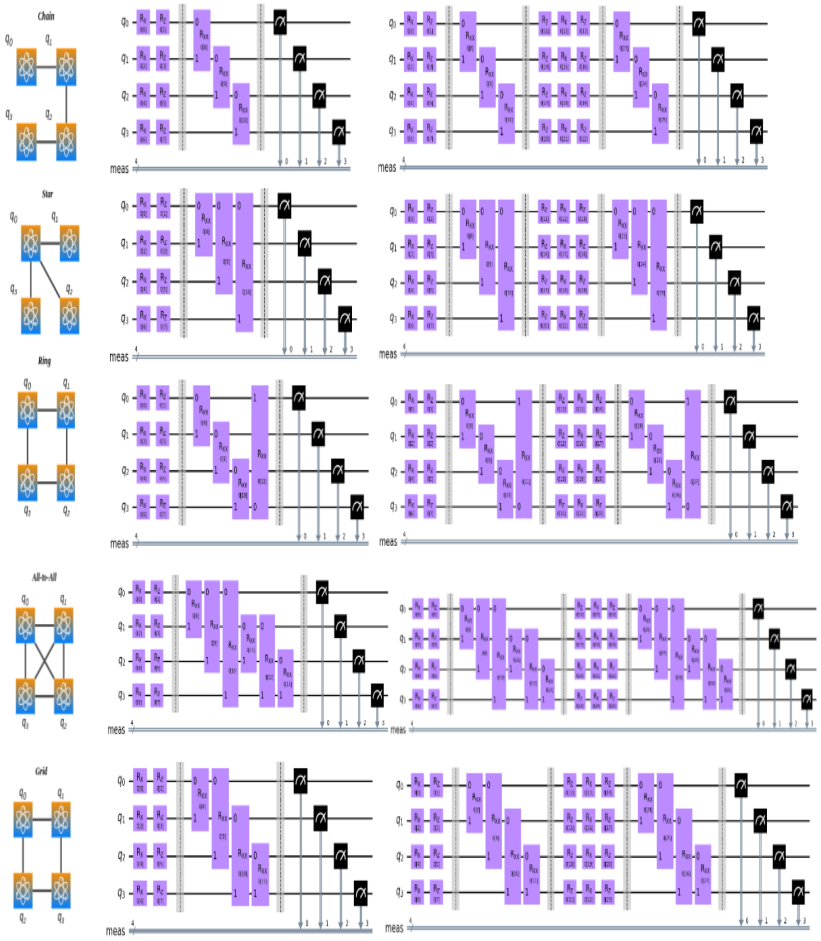


Figure 23: QCBM circuits of 2 (left) and 4 (right) layers.

We tested only 2 and 4 layers, as deeper circuits may enhance expressivity but can also lead to overfitting and create a model that is difficult to train. For the fully-connected topology, a single rotation layer and an entangling layer are sufficient to capture the dataset patterns and correlations for this task. For the other topologies, an additional pair of layers, one rotational and one entangling is needed to achieve better accuracy, but, at the presence of high noise. We can confirm these facts by looking at the results below:

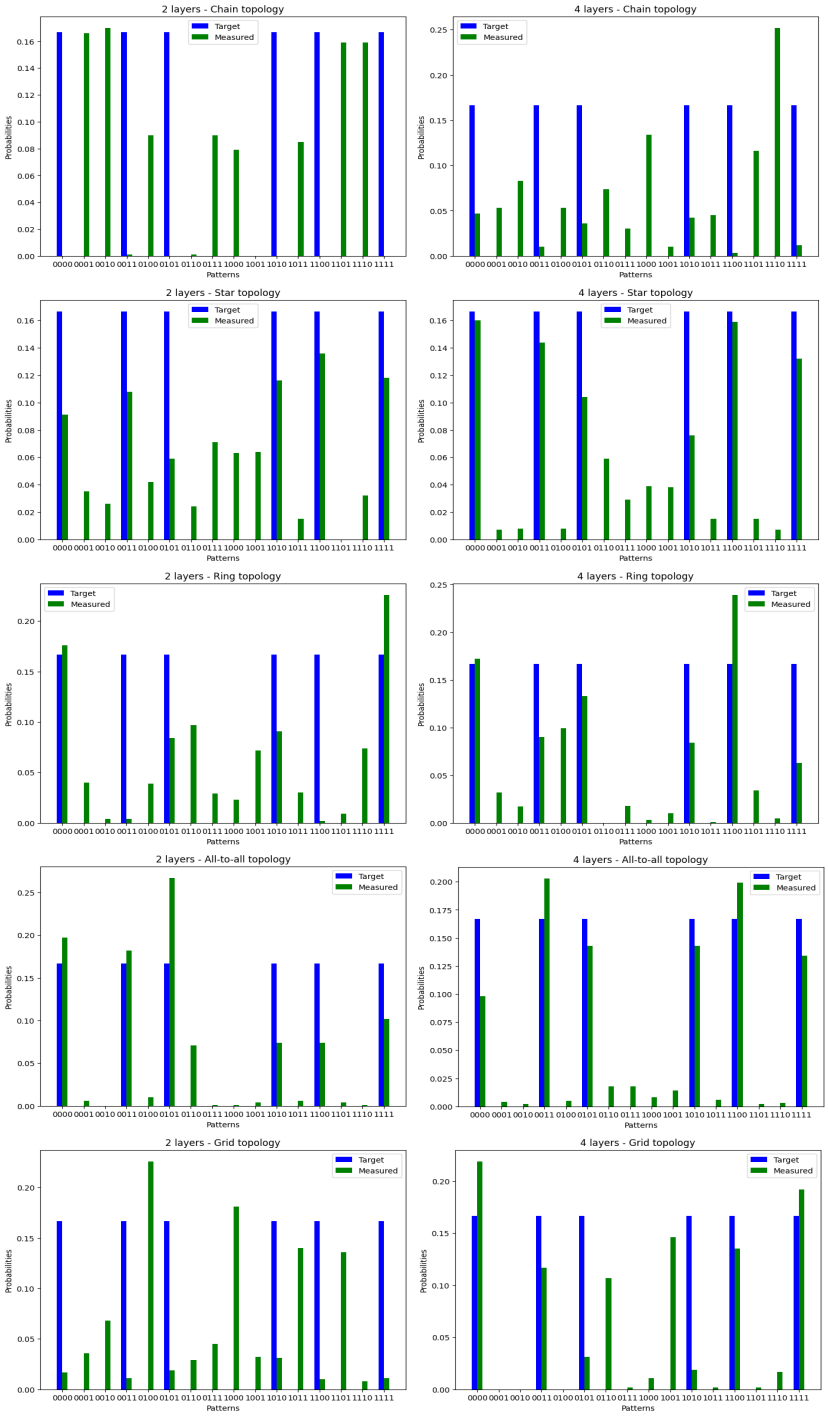


Figure 24: Training results after 100 iterations and 1000 circuits shots. We point out the all-to-all and grid connectivities to show the best performance. We used the same PSO initial parameters as in [42].

We are limiting our focus to shallow circuits because longer circuits are more suspectible to noise, especially for NISQ devices. If the shallow circuit yields poor performance, it can be safely assumed that the longer circuit will also yield poor performance. The number of layers is closely related to the entangling capability and expressibility of a circuit [43].

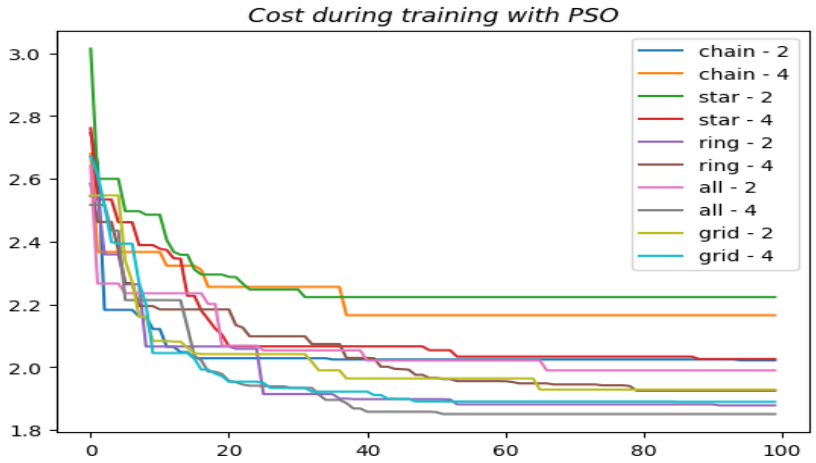


Figure 25: QCBM Training Cost History.

4.2.3 Quantum Generative Adversarial Networks

A Quantum Generative Adversarial Network (QGAN) is another type of a quantum neural network (QNN). It retains the structure of the classical GAN, but the generator and/or the discriminator is replaced by a *parameterized quantum circuit (PQC).* In this work, we focus on the hybrid approach, where the generator is a quantum circuit and the discriminator is a classical neural network.

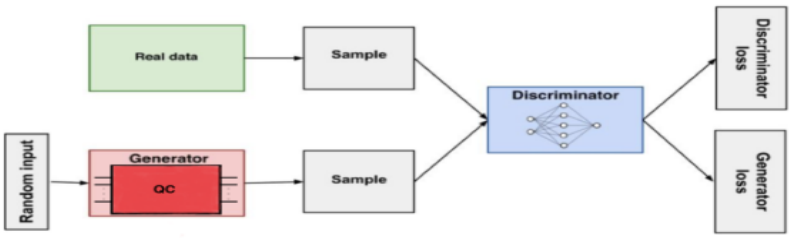


Figure 26: A quantum GAN with a quantum circuit as the generator and a classical discriminator network.

5 Quantum Modeling & Finance

Economists study the functioning of economies and develop theories to explain and predict economic phenomena. Understanding the economy is important because it can affect the well-being of individuals and societies as a whole, including their standard of living, income, employment opportunities and access to goods and services. One of these theories is a mathematical framework which we refer to as *financial time series*.

5.1 Financial Time Series

A time series is generally a set of points indicating some quantity over discrete steps in time. This may include temperatures, data produced by sensors or some other value. The S&P 500 index daily closing price values over a period of time is such an example. When we are dealing with finance, we call this model a financial time series.

As the mechanism that generates financial data is unknown, generative modeling for finance is an extremely active area of research and has found various applications involving *predictions, assets* and *risk management, portfolio optimization* and studying the overall health of an economy.

5.1.1 The S&P 500 Index

The S&P 500 index is a market capitalization-weighted index of 500 large-cap stocks traded on the two largest U.S. stock exchanges: the New York Stock Exchange (NYSE) and the Nasdaq Stock Market. These stocks are selected based on certain criteria such as market capitalization, liquidity and financial viability and are representative of the U.S. economy. It is widely regarded as one of the best measures of the performance of the US equities market and it is often used as a benchmark for the overall performance of the US stock market. The S&P 500 index is frequently used as a gauge of the health of the US economy and is tracked by investors, analysts and financial media around the world.

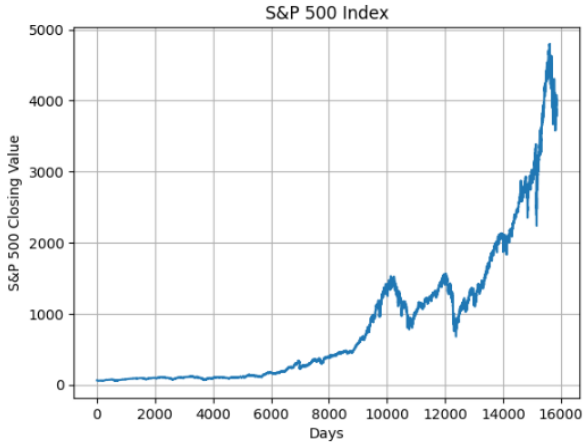


Figure 27: The S&P 500 index closing prices from 1/1/2000 since 31/12/2022.

The S&P 500 index is clearly in an uptrend over the years. This is logical, as it kind of measures the overall performance between 500 most successful companies in the U.S.

We are going to make use of this index in order to study some aspects of financial time series, as well as develop a generative adversarial network that generates synthetic data that resembles the S&P 500 index. Then, we will extend this idea to the quantum computing field, where we will implement a hybrid classical-quantum variational model with a parametrized quantum circuit (PQC) as the generator of a higher-level framework known as quantum GAN (QGAN).

5.1.2 Raw price fluctuations vs Logarithmic Returns

The upward trend is a common property in many financial time series, especially in indexes such as the S&P 500, which means that the value of the index may increase over time even if there are fluctuations in the short term. Therefore, studying the index's price alone may not be sufficient to compare the index's performance over different periods of time. Instead, analysts use the *returns* instead of prices for such purposes. The *logarithmic returns* of a stock or an index is defined by the equation

$$r_t = \ln(p_t) - \ln(p_{t-1}) = \ln\left(\frac{p_t}{p_{t-1}}\right) \tag{80}$$

where r_t are the logarithmic returns at time t and p_t is the stock or index price at time t. Also, we assume that the prices are equally spaced by some constant Δt (in our case, this is one day). For further details, refer to [44].

The logarithmic returns have some special properties that we wish to exploit, which are not present if we used the direct returns instead, given by $p_t - p_{t-1}$. Let us visualize the direct returns and the logarithmic ones to get an idea:

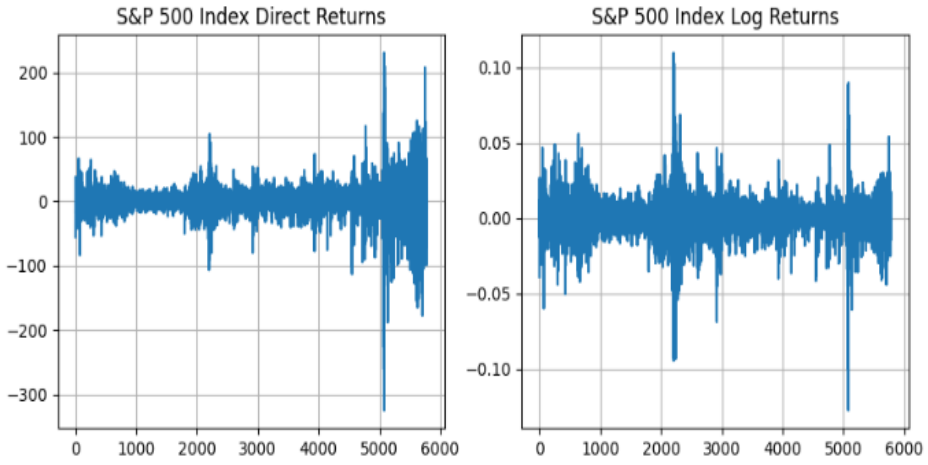


Figure 28: Direct Returns (Left), Log-Returns (Right).

Scaling over time

The first thing to notice is how the returns scale over time. The log-returns stay within an approximate range, whereas the bounds of the direct returns range are altered. Raw price fluctuations can vary widely depending on the value of the index, which can make it difficult to compare fluctuations at different points in time. However, log-returns account for the percentage change in price, which helps to normalize the data and makes it easier to compare fluctuations across different periods. This is because of the compression that the logarithmic function induces on the values. Additionally, log-returns are additive, which means that the return over a longer period can be calculated by summing the returns over shorter periods, which is not the case with raw price fluctuations. This allows for simple arithmetic operations, such as averaging, which can be useful for various applications.

Returns Rate

Also, the *returns rate* R_t at time t is closely related to log-returns as

$$R_t = \frac{p_t - p_{t-1}}{p_{t-1}} \approx \ln\left(\frac{p_t}{p_{t-1}}\right) \tag{81}$$

The returns rate is simply the percentage change in the value of an investment over a specific period of time. Log-returns are calculated using

the natural logarithm of the ratio between the final and initial values of an investment. Since the logarithmic function is continuous and differentiable, log-returns can be interpreted as the continuously compounded returns rate. This means that a log-return of 0.01 (or 1%) can be interpreted as a continuously compounded return rate of 1%.

Volatility Smile

Log-returns are also related to *stock volatility*, which measures how much the price of a stock fluctuates over time. The relationship between log-returns and stock volatility is known as the *volatility smile*. It is a common feature of financial markets where implied volatility, which is derived from options prices, varies with respect to the strike price and time to expiration. In general, the volatility smile implies that log-returns are not normally distributed, but instead have fatter tails than a normal distribution would suggest. We will come to this right after.

5.1.3 Stylized Empirical Facts

Over the years, analysts have extracted several statistical properties from financial time series that are present in almost every market, which are known as *stylized facts*. This is essential, as there is no standard procedure of how the values of such a series are produced. As a result, predicting future stock prices is out of the question. As a consequence of many independent empirical studies on the statistical properties of financial markets, there is a wide collection of stylized facts [45]. We will be concerned with some of them.

Log-Returns Density is not Gaussian

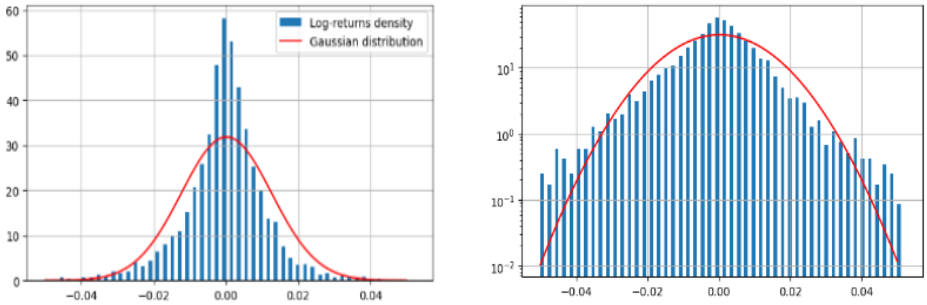


Figure 29: Non-Gaussianity of Logarithmic Returns.

On the left, we have the histogram of the log-returns along with a Gaussian distribution with the same parameters and on the right, we plot the same results in logarithmic scale to inspect in further detail. The log-returns

density peak is almost doubled with respect to the Gaussian distribution. Also, by inspecting the right figure, we see that the tails of the log-returns do not decay as fast as the Gaussian. This property is known as *fat tails*.

It is worth noting that these two properties are not so well clarified when we increase the time scale from days to weeks or months. This is so-called *aggregational normality*.

Absence of Linear Autocorrelation

The previous observation comes from a statistical perspective and is independent of time. However, there are some stylized facts that deal with properties related to time. One such property is the absence of linear autocorrelations. We can inspect this by calculating the *autocorrelation function (ACF)* of the log-returns of the S&P 500 for a range of different lags

$$\rho(\tau) = corr(r_t, r_{t+\tau}) = \frac{cov(r_t, r_{t+\tau})}{\sigma_{r_t}\sigma_{r_{t+\tau}}} \tag{82}$$

where r_t, $r_{t+\tau}$ are the returns, σ_{r_t}, $\sigma_{r_{t+\tau}}$ are the standard deviations and $cov(r_t, r_{t+\tau})$ is the covariance of the returns between times t and $t+\tau$, where τ is the lag. Since we have daily timeframe, $\tau \in \mathbb{N}^*$.

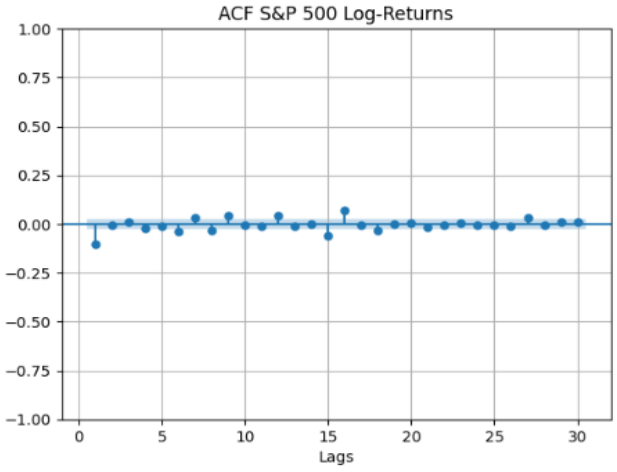


Figure 30: ACF of S&P 500 index logarithmic returns.

As we can see, the values are very close to zero, so there is no sufficient autocorrelation. In an efficient market, stock prices are believed to reflect all known information about the market, which means that any new information that becomes available will be quickly incorporated into the stock price. As a result, there should be no predictable pattern of price movements over time and the autocorrelation of the stock returns should be close

to zero. This is because, if investors are rational and acting on all available information, the current stock price should reflect the expected future value of the stock and any deviation from that expected value would be quickly corrected as new information becomes available. Therefore, the absence of autocorrelation is actually to be expected in an efficient market, since the current price already incorporates all available information and there should be no predictable pattern of price movements over time. Also, as lags are getting higher, autocorrelation decreases.

Volatility Clustering

Volatility clustering is a phenomenon in financial markets where periods of high volatility tend to be followed by other periods of high volatility and periods of low volatility tend to be followed by other periods of low volatility. This means that volatility tends to cluster together in time, rather than being randomly distributed.

We can get an intuitive understanding of volatility clustering by looking at the log-returns. However, we wish to quantify this phenomenon. Specifically, we can do this by looking at the autocorrelation function ACF of the *absolute log-returns* $|r_t|$ and $|r_{t+\tau}|$ over a range of lags τ defined by

$$\rho_{abs}(\tau) = corr(|r_t|, |r_{t+\tau}|) \tag{83}$$

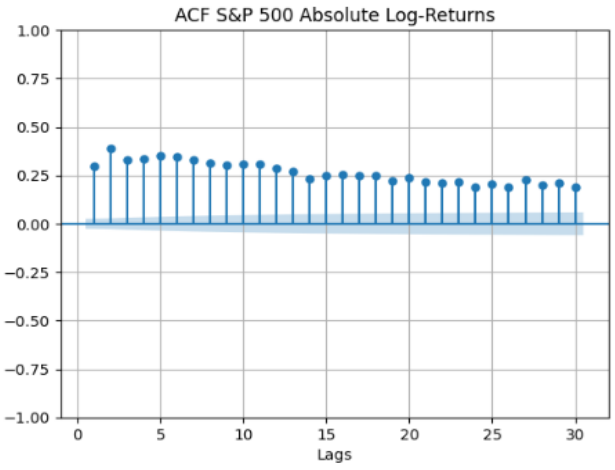


Figure 31: ACF of S&P 500 index absolute logarithmic returns.

The presence of significant autocorrelation for absolute log-returns is an indication that there are patterns in the data that are related to the magnitude of price changes and that these patterns can be characterized by volatility clustering. Also, autocorrelation decreases as the lag is increased. This is

expected in efficient markets. This plot provides insights into the persistence of volatility over time in the financial time series.

The Leverage Effect

But, how the returns are correlated to volatility ? The *leverage effect* is a stylized fact that deals with this and is commonly present in financial time series. It is given by

$$L(\tau) = corr\left(r_{t+\tau}^2, r_t\right) \tag{84}$$

where $r_{t+\tau}^2$ is another widely used metric for volatility.

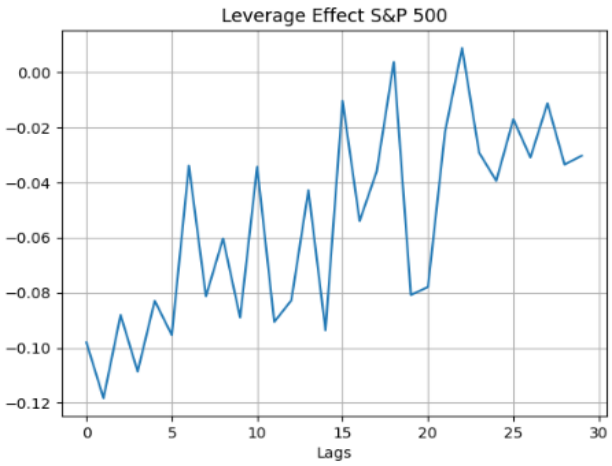


Figure 32: Leverage Effect of S&P 500 index logarithmic returns.

As the lag increases, the leverage effect goes to 0, starting from a lower negative value. The returns are negatively correlated to price volatility. Specifically, it means that as the price of an asset falls, the volatility of that asset tends to increase and vice versa. This implies that when the market is experiencing negative returns, the volatility of that market tends to increase. The reason for the leverage effect is due to the presence of financial leverage, which refers to the use of borrowed funds to invest in assets. When investors use leverage to buy an asset, they increase the risk of their investment since they have to pay back the borrowed funds regardless of the performance of the asset. If the asset performs poorly, then the investor may be forced to sell at a loss to pay back the borrowed funds, which can further amplify the downward price movement and increase the volatility of the asset.

5.2 Utilization of GANs in Finance

5.2.1 Domain Applications & Financial Data Generation

The utilization of GANs in the realm of finance is a burgeoning area of interest. In [46], the authors introduce the concept of incorporating adversarial learning into stock price prediction models. In their framework, the price prediction model is optimized to minimize the discriminator's loss, thus ensuring more realistic price movement predictions. Another interesting work lies in [47], where GANs are employed to refine systematic trading strategies and discover effective combinations of trading approaches. In the domain of fraud detection, GANs are exploited in [48] and [49].

Generating realistic financial data can be challenging due to the complex and dynamic nature of financial markets. The generated samples may not fully capture the intricacies of real-world financial data and caution should be exercised when using generated data for decision-making or risk management purposes. It's important to acknowledge the limitations of generative modeling in financial time series.

Producing financial time series data that exhibit stylized facts is a demanding and interesting task. The adoption of GANs has brought about a significant transformation in the realm of generative modeling, particularly in handling intricate data types like images, videos and more recently, financial time series data. GAN-based modeling stands out for its ability to capture the intricate characteristics found in financial time series, requiring minimal assumptions about the underlying dataset [50].

Since the area of research in terms of various model implementations is vast and rapidly growing, we opt to follow a similar implementation as in [51]. *Wasserstein GANs with gradient penalty* [52] have shown to avoid the problems of *mode collapse* and *vanishing/exploding gradients* by adding a regularization term in the loss function of the critic network, the gradient penalty. Schwander conducted a thorough analysis on generating synthetic logarithmic returns by utilizing this model. In his work, he compares two GAN models, one with a *multi-layer feedforward* architecture for the critic network and one with *1D convolutional layers* (in WGANs, the discriminator is referred to as the *critic* network). In [53], Takahashi et al. show that using convolutional layers improve the quality of the generated data in terms of stylized facts and Schwander confirms this in [51]. Also, we will not be concerned with the generator architecture, as the scope of this thesis is to show general applications involving generative modeling and how we can leverage quantum computing in this area. The generator network will be replaced by a *parameterized quantum circuit (PQC)*. Schwander integrates a

quantum generator in the WGAN-GP framework [51]. We will investigate and extend this research when we will talk about quantum GANs.

5.2.2 Wasserstein GANs

WGANs use a different type of loss function called the *Wasserstein distance*, which measures the distance between two probability distributions. In a WGAN, the generator produces synthetic financial time series data and the discriminator outputs a scalar value representing the probability that a given sequence of financial time series data is real or synthetic. The WGAN training process involves minimizing the Wasserstein distance between the probability distributions of the real and synthetic financial time series data. This is a variant of the GAN framework.

The Wasserstein distance, also known as *Earth Mover's Distance (EMD)* is a quantitative metric that allows the comparison of two distributions that two different datasets follow, without assuming any underlying distribution. Assume that the reference distribution is denoted by P and the model distribution by Q. As we are interested with one-dimensional data, we use the so-called Wasserstein-1 distance defined by

$$W_1(P,Q) = \int_{-\infty}^{\infty} |F_P(x) - F_Q(x)|\, dx \tag{85}$$

where $F_P(x)$ and $F_Q(x)$ are the cumulative distribution functions of P and Q respectively and x denotes the samples. This needs a numerical method in order to be computed. It can be found in the Python *SciPy* library [54], which is widely used for scientific and technical computing.

The key difference between WGAN and other GAN variants is the use of a *critic* instead of a discriminator. The critic is trained to output a real number rather than a probability and is optimized to minimize the Wasserstein distance between the real and generated distributions. The WGAN with gradient penalty (WGAN-GP) is an improved version of the original WGAN that introduces a gradient penalty term to the critic's loss function. In the WGAN-GP, the critic is penalized when its gradient norm deviates from 1, which is the desired value for a Lipschitz continuous function. This is achieved by computing the gradient penalty as the squared difference between the norm of the critic's gradient and 1 and adding it to the original Wasserstein distance objective. If we assume that the output of the critic is denoted by $C(x)$, where x is a sample, the critic's loss function becomes

$$W_1'(P,Q) = \int_{-\infty}^{\infty} |F_P(x) - F_Q(x)|\, dx + \left(\|\nabla_x C\| - 1\right)^2 \tag{86}$$

This encourages the critic to have a gradient norm close to 1 everywhere, which in turn enforces the Lipschitz continuity constraint and leads to more stable training and improved sample quality. Also, it helps avoid the problem of vanishing gradients and mode collapse. The magnitude of the gradient penalty is controlled by a hyperparameter called the penalty coefficient, which determines the strength of the penalty relative to the original Wasserstein distance objective.

5.2.3 Model Performance

When dealing with GANs for financial time series, one should carefully consider what metrics will define the model performance. The generated synthetic data must be close to the original data and should be able to replicate the stylized facts up to some extent.

In order to monitor the performance in terms of the stylized facts during training, we will use the *Root Mean Square Error (RMSE)* metric as in [51]. For the autocorrelation of the log-returns, we can define the following metric

$$\text{RMSE}(\rho^{S\&P}(\tau), \rho^{\theta}(\tau)) = \left(\frac{1}{\tau_{max}} \sum_{\tau=1}^{\tau_{max}} \left(\rho^{S\&P}(\tau) - \rho^{\theta}(\tau) \right)^2 \right)^{\frac{1}{2}} \tag{87}$$

where $\rho^{S\&P}(\tau)$ is the ACF of the log-returns of the S&P 500, $\rho^{\theta,i}(\tau)$ is the ACF of the log-returns of the generated data of the model with parameters θ and the others are implied.

With the same approach, we can define the RMSE for the absolute log-returns (volatility clustering) as

$$\text{RMSE}(\rho_{abs}^{S\&P}(\tau), \rho_{abs}^{\theta}(\tau)) = \left(\frac{1}{\tau_{max}} \sum_{\tau=1}^{\tau_{max}} \left(\rho_{abs}^{S\&P}(\tau) - \rho_{abs}^{\theta}(\tau) \right)^2 \right)^{\frac{1}{2}} \tag{88}$$

where $\rho_{abs}^{S\&P}(\tau)$ is the ACF of the absolute log-returns of the original data and $\rho_{abs}^{\theta}(\tau)$ is the ACF of the absolute log-returns of the generated data from the model with parameters θ.

Last, we define the RMSE for the leverage effect as

$$\text{RMSE}(L^{S\&P}(\tau), L^{\theta}(\tau)) = \left(\frac{1}{\tau_{max}} \sum_{\tau=1}^{\tau_{max}} \left(L^{S\&P}(\tau) - L^{\theta}(\tau) \right)^2 \right)^{\frac{1}{2}} \tag{89}$$

where the variables are implied.

In addition, we can monitor the EMD by numerically evaluating the Wasserstein distance using SciPy. We also use a visualization technique called the *Q-Q plot* which gives the ability of better inspection of the specific areas of the distribution where the two distributions differ [55]. A Q-Q plot (quantile-quantile plot) is a graphical tool used in statistics to assess whether a dataset follows a particular theoretical distribution, such as a normal distribution. It is primarily used to compare the distribution of the observed data to the expected distribution, which could be a theoretical distribution or the distribution of another dataset.

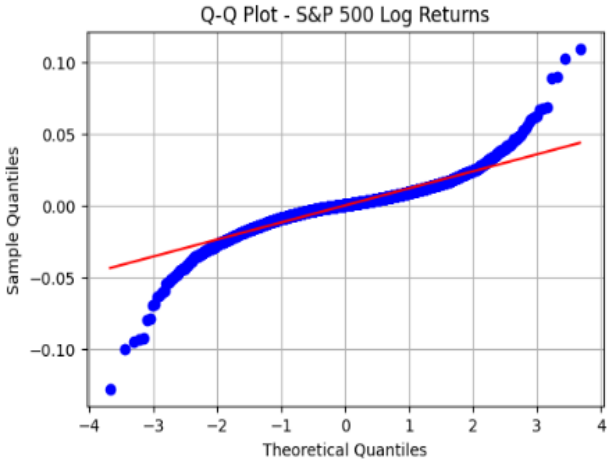


Figure 33: Q-Q plot of the original S&P 500 log-returns. The points form an S-shaped curve, which indicates a distribution with more extreme values than the theoretical distribution.

When training neural networks, it is a common practice to evaluate the result over multiple samples. So, we should bear in mind to calculate the costs above over multiple generated time series. Also, we should reverse the pre-processing of data as the generator learns to generate samples based on the pre-processed dataset that is fed into the critic.

5.2.4 Data Pre-Processing & Implementation

Financial modeling requires careful data pre-processing in order to deal with training stability and overall performance. Apart from the standard normalization step (scale the data to have zero mean and unit variance), models that deal with financial data include some additional pre-processing techniques.

Given the limited number of samples in the tails, learning a heavy-tailed distribution can be difficult. To address this challenge, we can employ a method to decrease the influence of these tails. One approach involves applying an inverse Lambert-W transform to the normalized log returns [56]. By applying this transformation, we can convert the heavy-tailed data into a distribution that closely resembles a Gaussian distribution. This can be advantageous for further analysis or modeling tasks that assume Gaussianity.

Given Lambert's W function which is the inverse of $z = ue^u$ with $z : \mathbb{R} \to \mathbb{R}$, we can define the following transform to our normalized heavy-tailed data set V as

$$W_\delta(v) = sgn(v)\sqrt{\frac{W(\delta v^2)}{\delta}} \tag{90}$$

where $v \in V$, $\delta \geq 0$ a tunable parameter and $sgn(v)$ the sign of v and W the Lambert function.

The Gaussianized data can be transformed back to its original state using the equation

$$v = W_\delta(v) \exp\left(\frac{\delta}{2} W_\delta^2(v)\right) \tag{91}$$

In practice, equation the Lambert transform may behave poorly and give unreasonable values, so we can perform clipping to avoid this issue.

As financial data is usually sequential, we can also employ the *rolling window* technique in order to introduce a larger dataset of smaller length time series, specified by the window length. Rolling window also involves a stride, which determines the slide step of the window. Frequently, the step size is smaller than the window's size, causing the samples to partially overlap and become correlated. While this correlation among training samples is not ideal, having a larger pool of training samples can ultimately enhance the model's performance.

The following figure depicts the pre-processing pipeline and shows how the inverse Lambert transform helps to Gaussianize the input data:

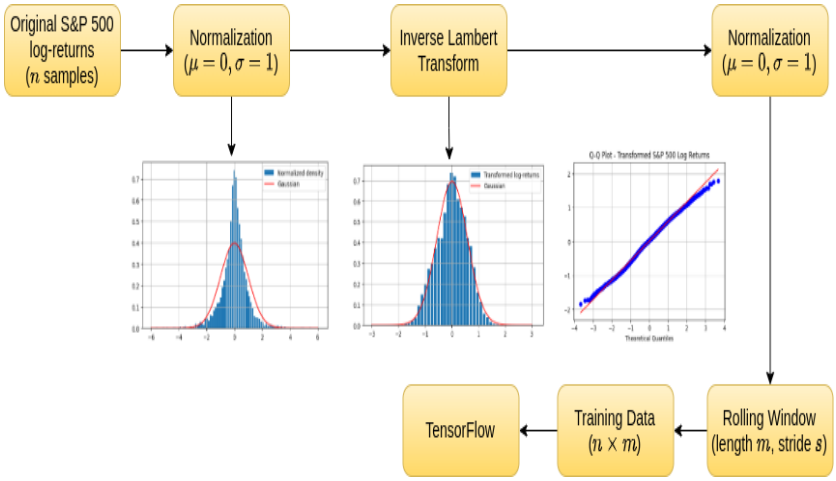


Figure 34: Pre-processing pipeline of the S&P 500 log-returns data.

For the implementation, we will use TensorFlow library provided by Google [8]. The algorithm is taken from [52] and is modified with different hyperparameters as shown below. The code can be found in [57].

Algorithm 1 WGAN with gradient penalty. We use values of $\lambda = 10$, $n_{critic} = 5$, $\eta = 0.0001$, $m = 32$ and default values for β_1, β_2 parameters of the Adam optimizer

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters η, β_1, β_2.

Require: Initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, \ldots, n_{critic}$ **do**
    for $i = 1, \ldots, m$ **do**
      Sample real data $\mathbf{x} \sim \mathbb{P}_r$, latent variable $\mathbf{z} \sim p(\mathbf{z})$, $\epsilon \sim U[0, 1]$.
      $\tilde{\mathbf{x}} \leftarrow G_\theta(\mathbf{z})$
      $\hat{\mathbf{x}} \leftarrow \epsilon \mathbf{x} + (1 - \epsilon)\tilde{\mathbf{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\mathbf{x}}) - D_w(\mathbf{x}) + \lambda \left(\|\nabla_{\hat{\mathbf{x}}} D_w(\hat{\mathbf{x}})\|_2 - 1\right)^2$
    end for
    $w \leftarrow Adam\left(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \eta, \beta_1, \beta_2\right)$
  end for
  Sample a batch of latent variables $\left\{\mathbf{z}^{(i)}\right\}_{i=1}^{m} \sim p(\mathbf{z})$.
  $\theta \leftarrow Adam\left(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\mathbf{z})), \theta, \eta, \beta_1, \beta_2\right)$
end while

At last, we present the training pipeline in a flowchart for better understanding of the process:

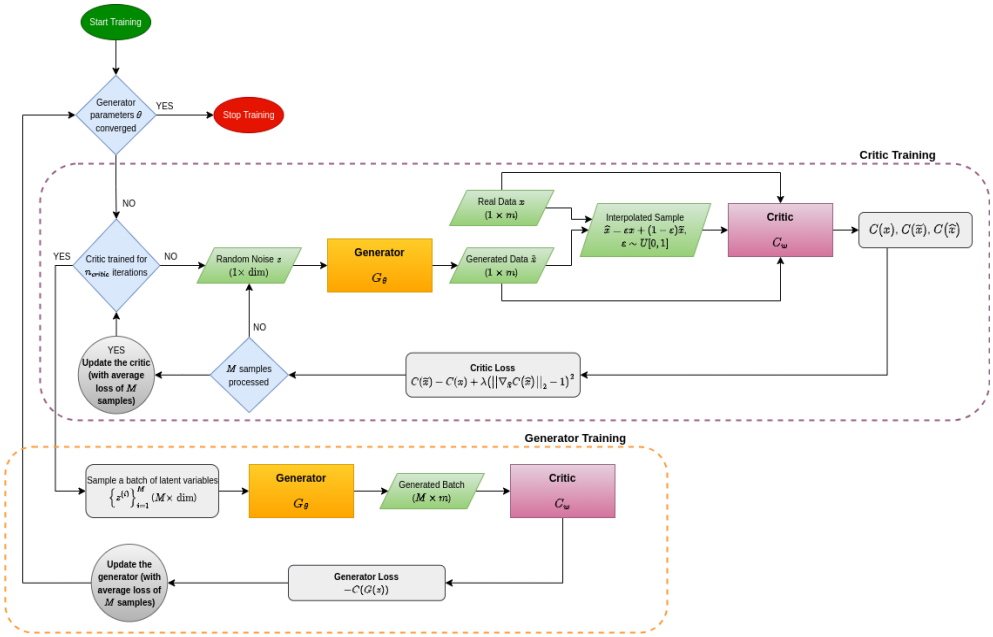


Figure 35: Training pipeline of the WGAN-GP model. The model is trained for 3000 epochs. We used a window size of 30 and stride 1 for the rolling window and the generator's input is a normally distributed noise vector of length 8. The results differ from [51] as we implemented the code from scratch and used different hyperparameters and smaller length training data.

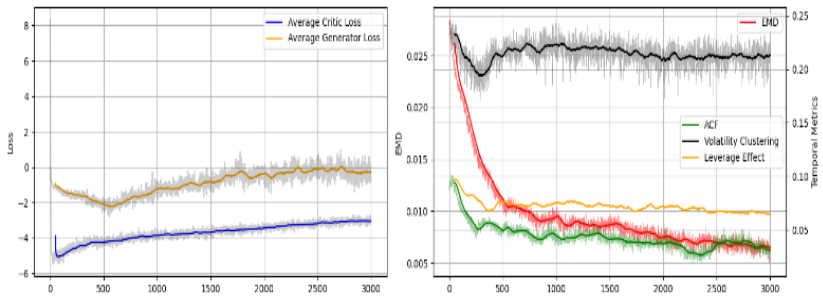


Figure 36: Training history of critic and generator losses (left) along with the RMSEs and EMD (right). We should note that the generator loss lacks significance as it exclusively evaluates generated samples, failing to consider the combination of real and generated samples required to approximate the EMD.

Let us inspect the generated data properties:

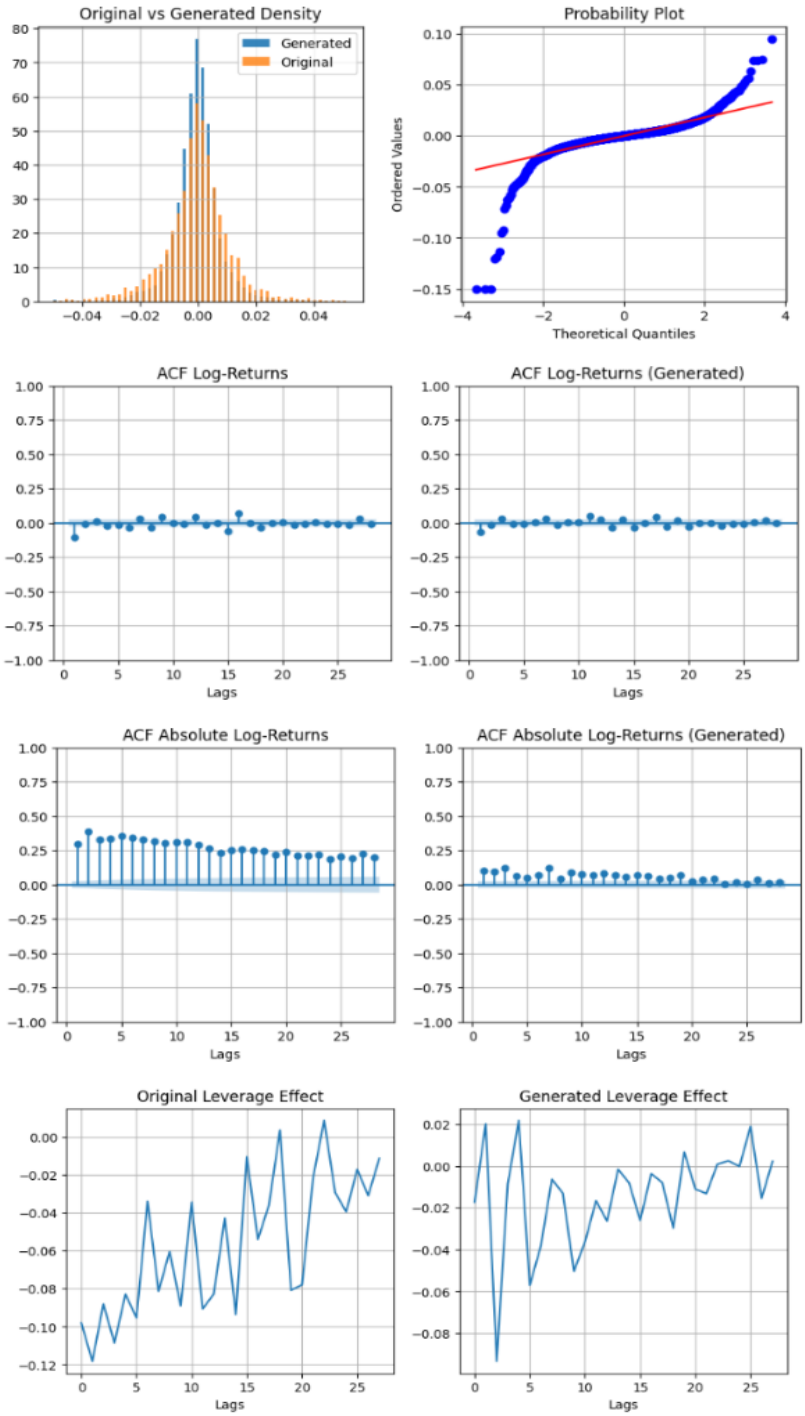


Figure 37: Original vs Generated data properties.

The generated log-returns are shown below:

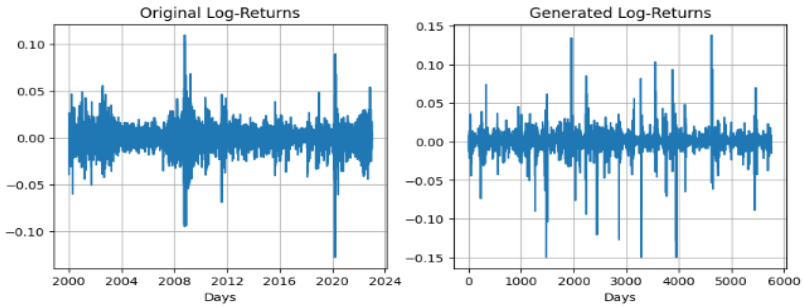


Figure 38: Original vs Generated Log-Returns. The reason for the occurence of several spikes may be due to the poor performance of the Lambert transform when reversing the pre-processing and introduces extreme values which are clipped with a predefined threshold. This threshold is a hyperparameter that may be further optimized.

From the performance metrics of the stylized facts, we can see a steady decline towards zero, apart from volatility clustering. This property is challenging to model due to the non-stationary, multimodal and long-range dependent nature of financial time series data, as well as its sensitivity to regime shifts, data noise and limited historical information. To model volatility clustering effectively, researchers often turn to specialized techniques such as GARCH (Generalized Autoregressive Conditional Heteroskedasticity) models, stochastic volatility models, or other generative architectures that can handle these challenges.

Inspecting the results of the WGAN model, we can clearly see that it has the ability to generate financial data with stylized facts, but up to some extent. Of course, we cannot assume at all that the model is primed for such a task, as there are other empirical facts that we did not consider [45]. The generated distribution is relatively close to the original, even in the tails as it can be seen from the Q-Q plot. The absence of autocorrelation is there and there seems to be some volatility clustering and leverage effect, but they are significantly different from the S&P 500 benchmark.

In general, our goal was to present such an application in the context of generative modeling and not to implement a model that can perfectly capture real-world financial data properties. The performance may be increased further by exploring other architectures for the critic and generator networks, as well as other available optimization techniques. Also, the hyperparameters can be tuned for this purpose, but the possible combinations require time and computational resources, a task that is out of the scope of this thesis.

5.3 Quantum GANs in Finance

In the ever-evolving landscape of modern finance, technological innovations continue to play a pivotal role in reshaping the industry. Among the most promising advancements is the integration of quantum computing with financial processes. Quantum generative adversarial networks (QGANs) stand at the forefront of this intersection, offering a unique and powerful toolset to address complex financial challenges. QGANs, an extension of classical GANs, harness this quantum supremacy to generate and manipulate data with unparalleled efficiency and accuracy. This transformative technology holds immense potential for applications in finance, addressing critical issues and unlocking new opportunities across the sector.

Quantum generative modeling is making significant strides in revolutionizing the financial sector with diverse applications such as portfolio optimization, risk assessment, options pricing, fraud detection, algorithmic trading and quantitative research. The reader is refered to [58, 59, 60] for a general presentation of various applications and methods of quantum computing for finance.

The current state of the art quantum models for financial time series include QGANs for generating synthetic data that exhibit stylized facts [61] and QCBMs that learn the correlations between currency pairs [62].

In [51], it is also shown that the QGAN outperforms a QCBM in learning the correlations between currency pairs. After conducting extensive parameter sweeps in terms of the number of layers and qubits for the PQC generator, the QGAN is able to produce time series data that exhibit stylized facts up to some extent. The quantum generator utilzes non-parameterized entangling layers with CZ gates and two different topologies are considered. Motivated by this, we are going to further extend the research and consider more topologies, as well as entangling layers with trainable parameters.

5.3.1 Design Considerations

One may wonder why we did not choose to use a QCBM as a quantum generator. The goal is to generate a time series of continuous values and of specific length. However, a QCBM generates samples directly via projective measurements and is more suitable when dealing with binary data. If we consider using a QCBM, we have to deal with complex data encoding and decoding, which will result to reduced resolution and precision, as well as a large number of qubits.

But, can't we encode a range of continuous values in a basis state ? We could proceed to do so by using bins, but in this case, the number of qubits

determines the precision of the data values. Moreover, there is the need of performing multiple measurements to generate a time series, as the length is determined by the number of circuit runs.

Could we encode a time series as a whole on a single basis state in order to produce a sample with a single measurement ? This would be really challenging and one should explore other encoding techniques such as amplitude encoding, feature maps or quantum encoding circuits. The decoding would make the whole process even more complex. All of these may be researched in a future work, as for now it is out of the scope of this thesis.

5.3.2 Quantum Wasserstein GAN

As in [51], we will follow another approach used in quantum machine learning where the samples are generated indirectly via expectation values of tensor products of Pauli operators called Pauli strings. A Pauli string is nothing but an element of the set $\{I, X, Y, Z\}^{\otimes N}$. The set of all possible Pauli strings is a subset of all N-qubit Hermitian operators. The number of Pauli strings will determine the length of the output series. Expectation values are in the range [-1,1]. Recall the third postulate of quantum mechanics that states that the only measurable quantities in practice are the eigenvalues of observables. Following the spectral decomposition theorem, the expectation value of some unitary operator is strongly connected to the eigenvalues of the same observable. So, we can leverage this to encode data in these values!

For the types of layers used, as well as the number of qubits and the overall PQC implementation, we follow a similar approach as in [51]. We use an encoding layer in the circuit input consisting only of R_x rotation gates whose parameters are uniformly distributed random values. A crucial difference is that we do not apply trainable scaling parameters to the final expectation values. We chose to scale the data in the range [-1,1] after the inverse Lambert transform during pre-processing, without the last normalization step. The training dataset is generated by a rolling window of length 20 and stride 5. Also, we use the Adam optimizer with default learning rate and beta values and we consider the same set of Pauli strings, as with just a single qubit at our disposal, it becomes apparent that we can, at most, derive two uncorrelated expectation values for our Pauli matrices. This limitation necessitates careful consideration when selecting our set of Pauli strings. Furthermore, given that we do not follow a classical post-processing step, such as a classical dense layer, it's crucial to recognize that each Pauli string corresponds to a single logarithmic return. Therefore, it may be prudent to explore a collection of closely related Pauli strings to generate our log returns. However, conducting experiments with a broader spectrum of Pauli strings could yield a deeper

understanding of their overall impact on the outcomes.

Using a similar PQC consisting of 10 qubits and a circuit depth of 4 layers, we will attempt to generate time series samples of length 20. We use the observable set $\{X_1, Z_1, X_2, Z_2, \ldots, X_9, Z_9, X_{10}, Z_{10}\}$. Also, we will attempt to introduce more parameters in the model by using Mølmer-Sørensen gates in the entangling layers.

5.3.3 Non-Parameterized Entanglement

For the sake of brevity, we present an instance of the quantum generator of 4 qubits. The encoding layer is omitted for simplicity, as it is placed only in the input and has no trainable parameters.

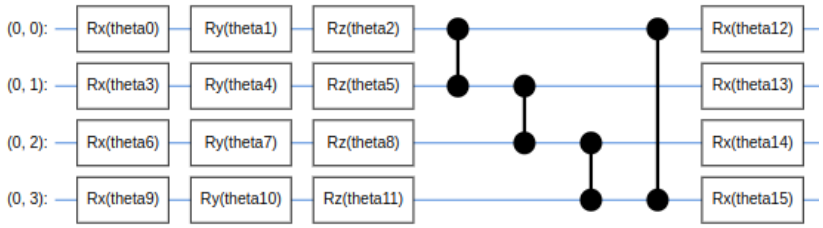


Figure 39: The rotation and entangling layers along with a data re-uploading layer is shown. The entangling layer uses CZ gates with ring topology.

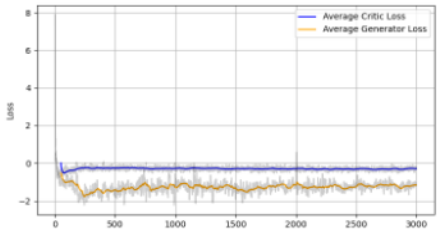
The figure above presents a hidden layer in the PQC. With N qubits, there are $3N$ rotation gates, ring topology requires N coupling gates and at last, we have N rotation gates for data-reuploading. A hidden layer thus, requires $4N$ single-qubit gates and N entangling gates, resulting to a total of $5N$ gates and $4N$ parameters, since CZ is not a parameterized gate. If we assume that there are L hidden layers, we have $5NL$ gates and $4NL$ parameters.

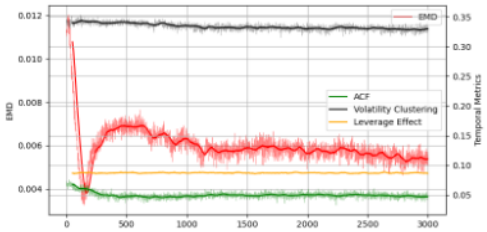
For the full generator circuit, we must consider the input and output layers as well. The output layer of the PQC includes only rotations, as we observed that it yields better results than an entangling layer. This is maybe due to the correlations (entanglement) between log-returns, which we wish to avoid for our task. The spatial complexity of the quantum generator is thus,

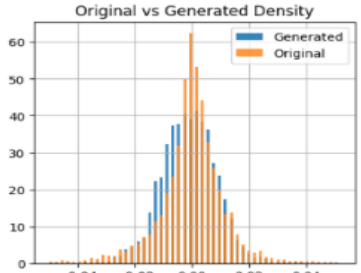
$$\underbrace{N}_{input} + \underbrace{5NL}_{hidden} + \underbrace{3N}_{output} = N(5L + 4) \tag{92}$$

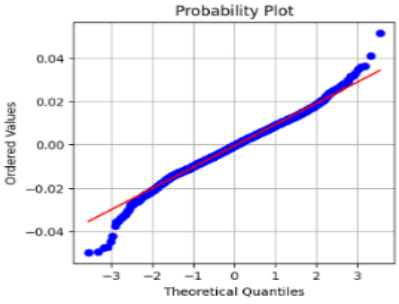
gates and the total number of parameters is $N(4L + 3)$.

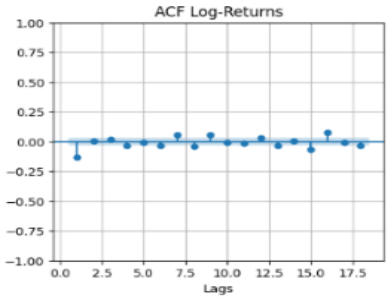
In the next page, we can see the training history along with the generated data properties versus the original.

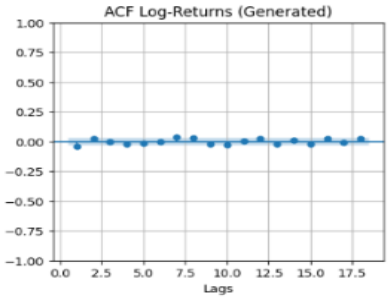

Average Critic Loss
Average Generator Loss
Loss

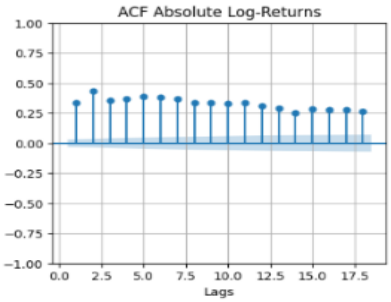

EMD
ACF
Volatility Clustering
Leverage Effect
EMD
Temporal Metrics

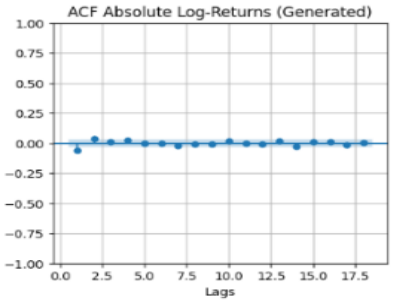

Original vs Generated Density
Generated
Original

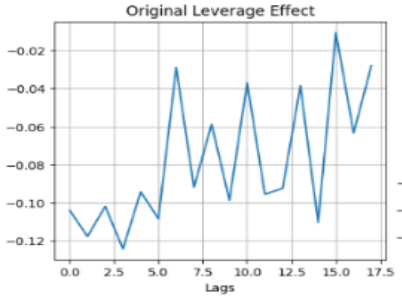

Probability Plot
Ordered Values
Theoretical Quantiles

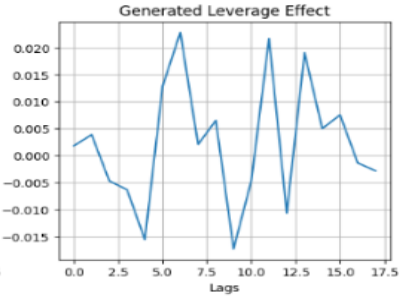

ACF Log-Returns
Lags


ACF Log-Returns (Generated)
Lags


ACF Absolute Log-Returns
Lags


ACF Absolute Log-Returns (Generated)
Lags


Original Leverage Effect
Lags


Generated Leverage Effect
Lags

The generated log-returns are shown below:

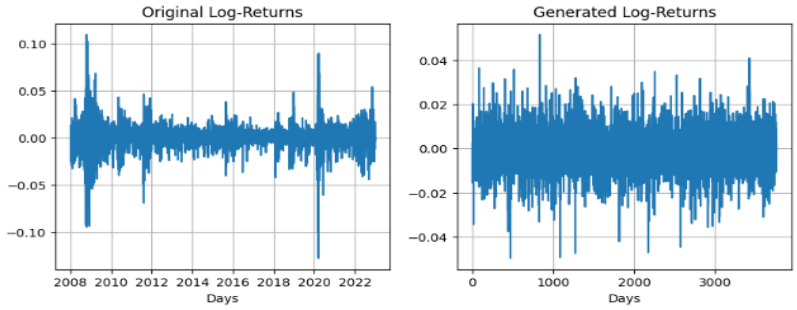


Figure 40: Original vs Generated Log-Returns. The reason for the high density of the generated log-returns may lie to the pre-processing where we scale the training data in the range $[-1, 1]$. This induces limitations to the generated data that the network learns.

We should note that the generator loss lacks significance as it exclusively evaluates generated samples, failing to consider the combination of real and generated samples required to approximate the EMD. Comparing to the classical model though, the generator loss lies below the critic loss.

From simple inspection, all properties deviate from the original ones. However, we should note that, due to limited computational and time resources, we used a learning rate of 0.001, a batch size of 10 and the critic is trained for 2 rounds in each iteration, compared to the classical hyperparameters of 0.0001, 32 and 5 respectively. Moreover, the S&P 500 training data is taken from 2008 to 2022, reducing the available number of samples. The training time for the QWGAN is x4 slower than the classical model.

Let us now assess the performance of a similar circuit by using another topology for the entangling layer, which arranges the gates in a star-like structure.

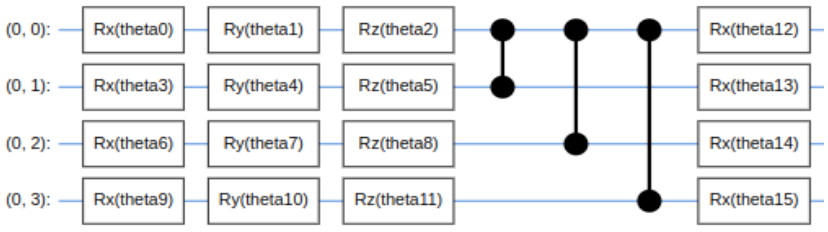


Figure 41: The rotation and entangling layers along with a data re-uploading layer is shown. The entangling layer uses CZ gates with star topology.

With this topology, the first qubit is connected to all the others to create entanglement, so for N qubits, we need $N-1$ CZ gates in a hidden layer. Here, the spatial compexity of the generator with L layers is

$$\underbrace{N}_{input} + \underbrace{(5N-1)L}_{hidden} + \underbrace{3N}_{output} = N(5L+4) - L \tag{93}$$

gates and the total number of parameters is $N(4L+3)$. In this case, we have a shallower circuit with the same number of parameters.

Using the same training procedure and hyperparameters, we can inspect the cost history of the model below:

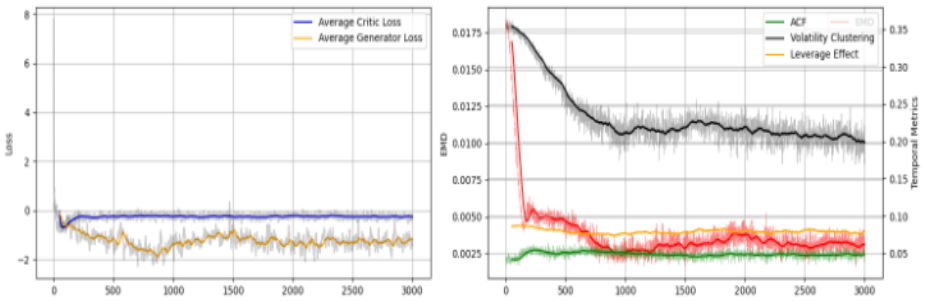


Figure 42: Training history of the QWGAN with star topology for the entangling layers with CZ gates.

Comparing to the model with ring topology, it is obvious that the RMSEs show better behavior, especially the EMD, which decays even more. A crucial and interesting observation is the performance of volatility clustering metric. While the circuit with ring entanglement does not show any improvement over the absolute log-returns RMSE, it seems that the star-like approach has helped the model to get better in capturing volatility clusters. Let us present the generated data and its properties for further assessment.

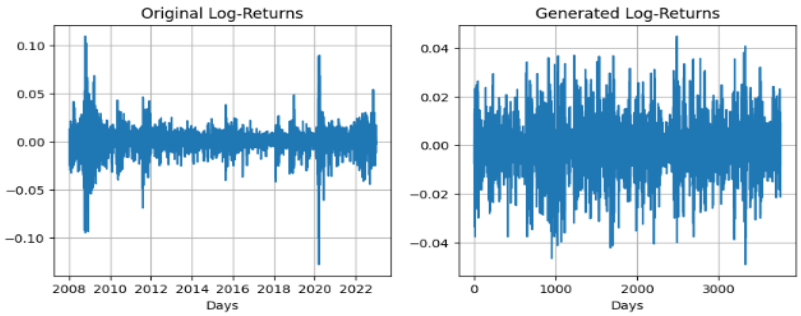


Figure 43: Original vs Generated Log-Returns. The reason for the high density was described previously.

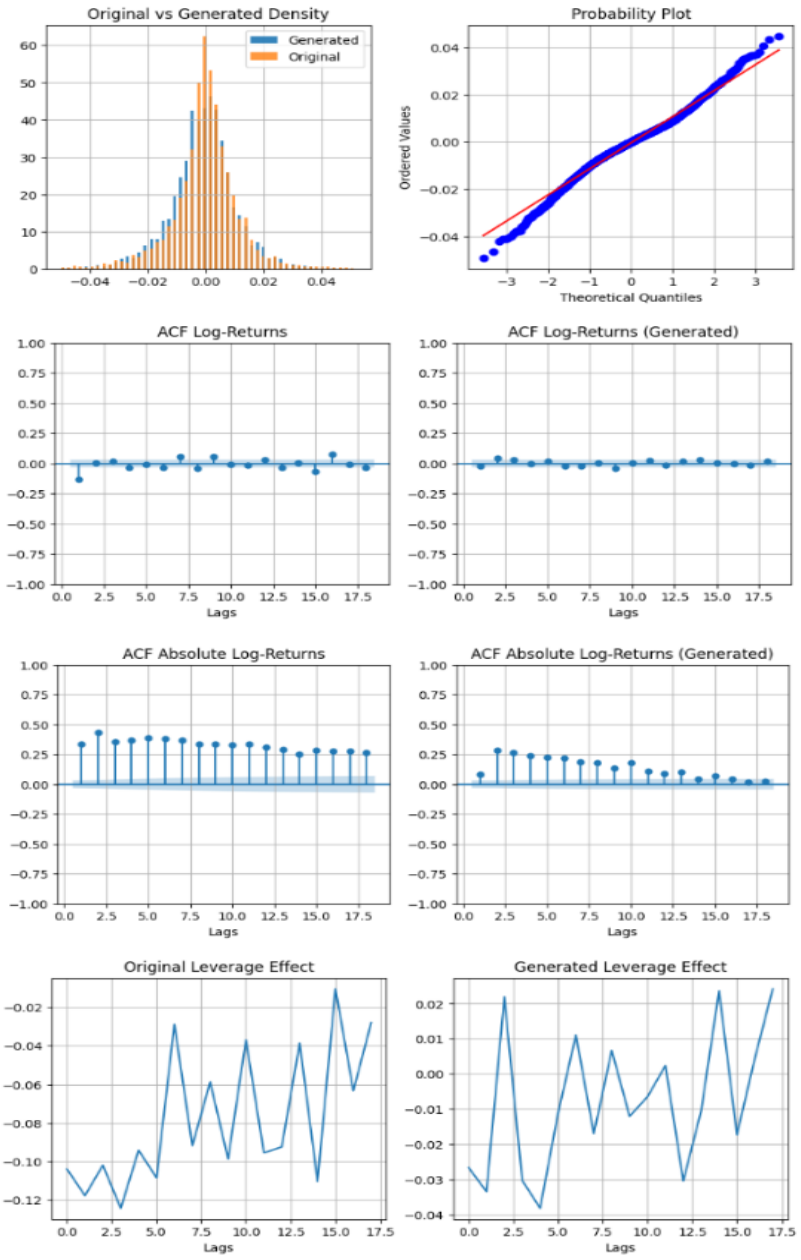


Figure 44: Original vs Generated properties.

At first sight, it is obvious that the star connectivity outperforms the ring topology in all stylized facts. The generated distribution is much closer to the original and there seems to be some volatility clustering, although it decays at higher delays. Notable is the fact that we used a smaller-depth circuit that seems to have better performance.

As we have seen in the previous chapter where we trained QCBMs with various topologies, the all-to-all connectivity had shown the best performance. Let us arrange the coupling gates in such a topology.

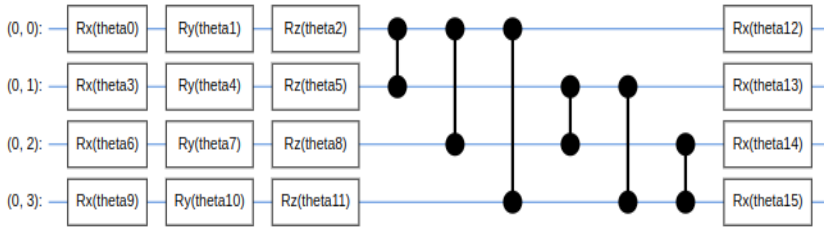


Figure 45: The rotation and entangling layers along with a data re-uploading layer is shown. The entangling layer uses CZ gates with all-to-all topology.

With this topology, each qubit is connected to all the others, so there are more gates involved. For N qubits, we need $\sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}$ coupling gates, so a generator with L hidden layers requires

$$\underbrace{N}_{input} + \underbrace{(\frac{N(N-1)}{2} + 4N)L}_{hidden} + \underbrace{3N}_{output} = \frac{LN^2 + 7LN + 8N}{2} \tag{94}$$

gates and the total number of parameters is $N(4L + 3)$. In this case, we have a circuit with a quadratic complexity and with the same number of parameters.

Using the same training procedure and hyperparameters, we can inspect the cost history of the model below:

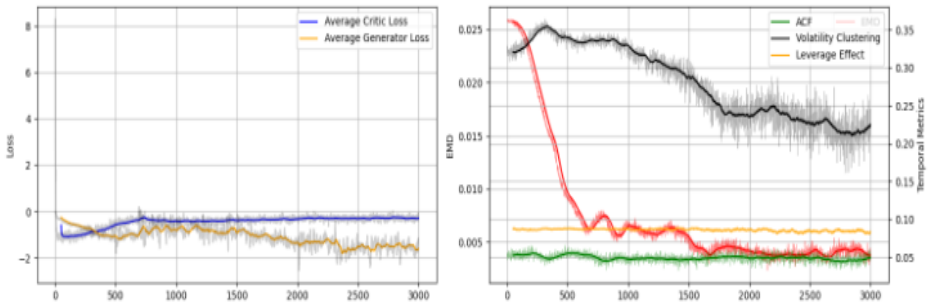


Figure 46: Training history of the QWGAN with all-to-all topology for the entangling layers with CZ gates.

This model also shows better performance in capturing volatility clustering and the EMD is slightly lower than both previous implementations. However, it requires more time to learn the underlying data properties.

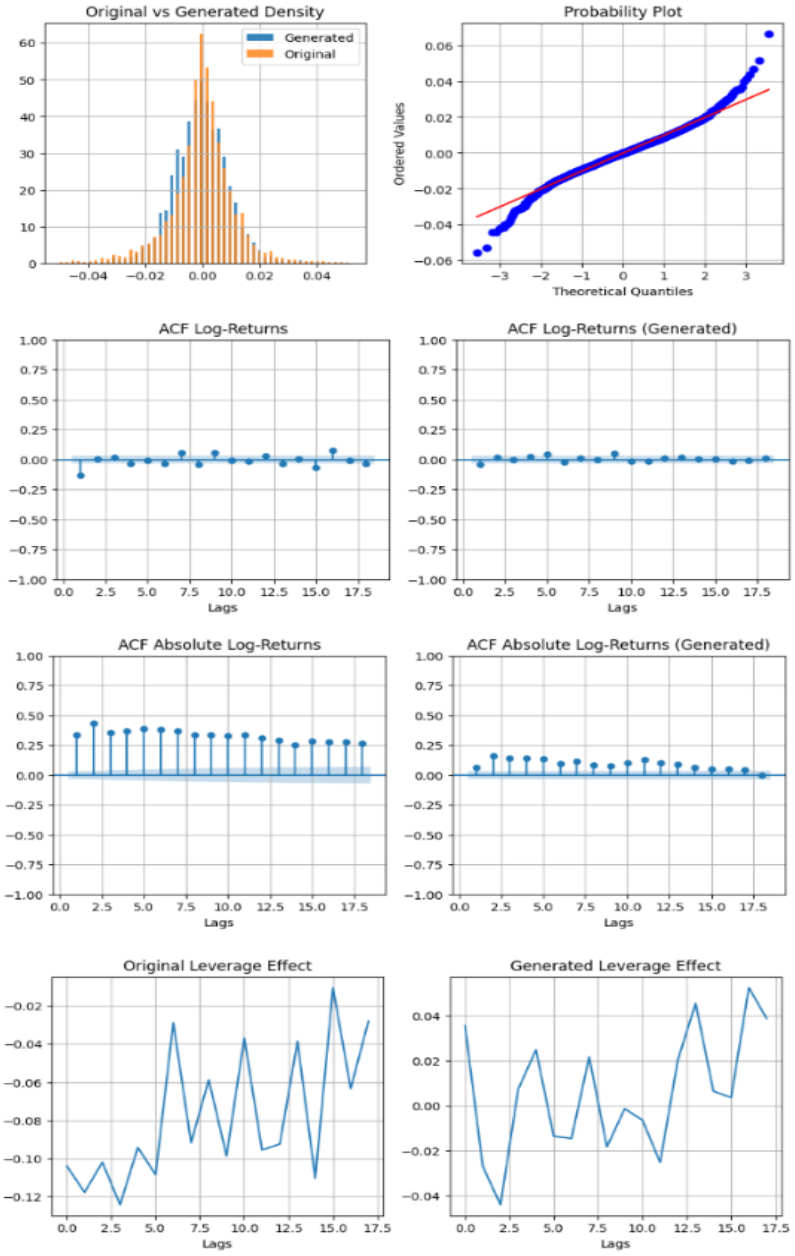


Figure 47: Original vs Generated properties.

All-to-all connectivity clearly outperforms the ring topology but not the star-like structure. Increasing the depth of the entangling layer does not seem to make things better. As we can see, the topology of the entangling layer plays a crucial role in model performance. Moreover, the star topology seems to outperform both previous ones and has the least number of gates.

The generated log-returns are shown below:

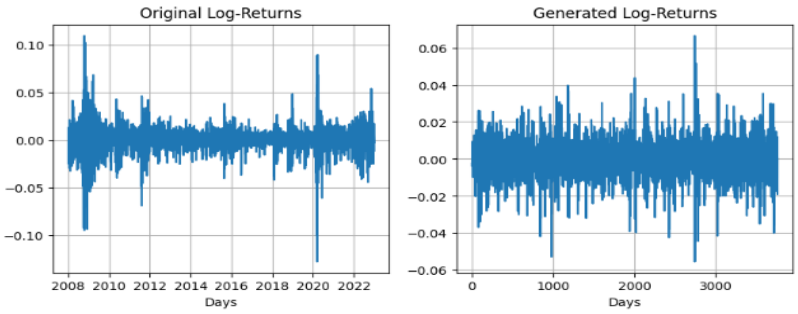


Figure 48: Original vs Generated Log-Returns.

5.3.4 Parameterized Entanglement

In this approach, we attempt to add some extra parameters to the model by using Mølmer-Sørensen (MS) gates in the entangling layers. This is the same gate we used in the previous chapter for the quantum circuit Born machine and is defined as

$$XX(\theta_{ij}) = \exp\left(-\frac{i}{2}\theta_{ij} X_i \otimes X_j\right)$$
$$= \begin{pmatrix} \cos\left(\frac{\theta_{ij}}{2}\right) & 0 & 0 & -i\sin\left(\frac{\theta_{ij}}{2}\right) \\ 0 & \cos\left(\frac{\theta_{ij}}{2}\right) & -i\sin\left(\frac{\theta_{ij}}{2}\right) & 0 \\ 0 & -i\sin\left(\frac{\theta_{ij}}{2}\right) & \cos\left(\frac{\theta_{ij}}{2}\right) & 0 \\ -i\sin\left(\frac{\theta_{ij}}{2}\right) & 0 & 0 & \cos\left(\frac{\theta_{ij}}{2}\right) \end{pmatrix}$$

where X is the Pauli gate and θ_{ij} is a parameter. In this way, we introduce trainable parameters to the entangling layer that the model can optimize throughout the learning process.

Apart from the training time, the QWGAN model settings remain the same. For simplicity, we do not show circuit instances here, as they are exactly the same in terms of connectivities and they only differ in the type of gates used in the entangling layers. What we need to consider though, is the increased number of parameters of the model. Recall that non-parameterized entanglement layers of N qubits using the CZ gate in ring topology required $5N$ gates with $4N$ parameters. Since MS gates come with a new parameter each, we have an additional N parameters and the spatial complexity of the

quantum circuit is

$$\underbrace{N}_{input} + \underbrace{5NL}_{hidden} + \underbrace{3N}_{output} = N(5L+4) \tag{95}$$

gates and the total number of parameters is $N(5L+3)$, where we exclude the encoding layer that has no trainable parameters.

The performance metrics are shown below:

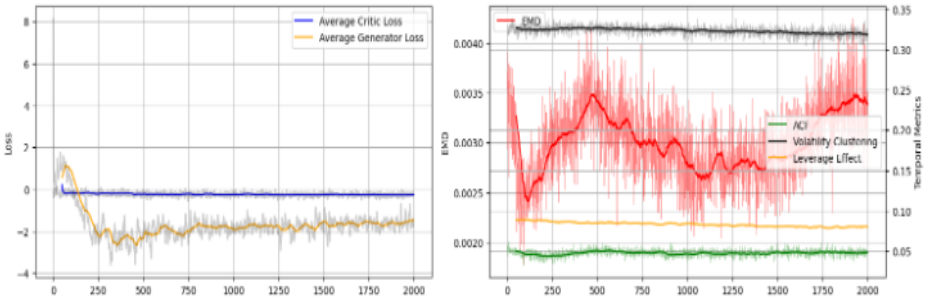


Figure 49: Training history of the QWGAN with ring topology for the entangling layers with MS gates.

The generated log-returns:

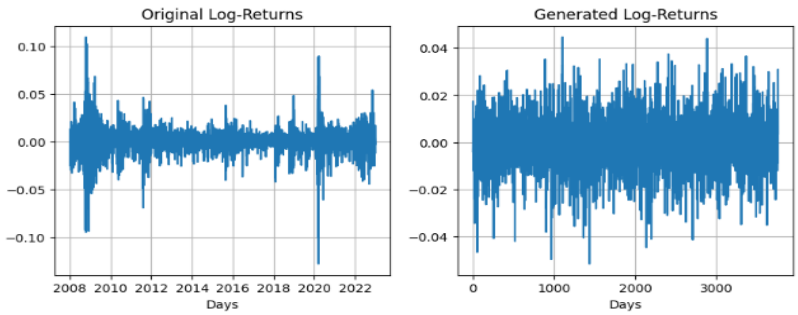


Figure 50: Original vs Generated Log-Returns.

About the performance metrics, we see a very different behavior for the EMD. Although it seems to increase with high fluctuations, the values in which it does so are the ones that the model without coupling parameters seems to have converged during the whole training session. This means that the QWGAN with parameterized couplings learns the data distribution faster than the circuits with CZ gates. We should also mention that this model was trained for 1000 epochs less.

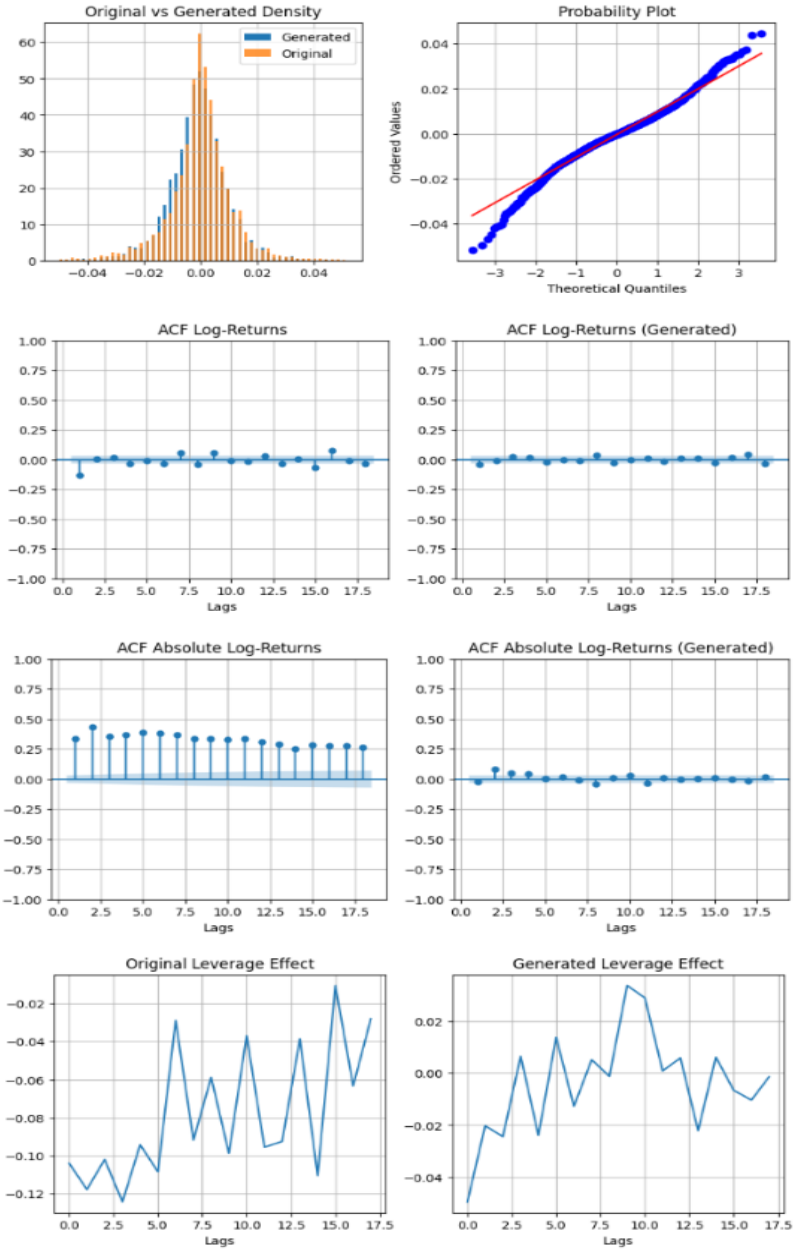


Figure 51: Original vs Generated properties.

With less training rounds, it is obvious that the generated distribution matches the original way better with parameterized couplings. Moreover, the leverage effect seems to have lower values and a smoother increase to zero with more detail. Nevertheless, the temporal properties deviate significantly from the S&P 500 benchmarks. With more computational resources available

and careful hyperparameter tuning, the performance is expected to increase, much more than the performance of the model with non-parameterized entangling layers.

Closing this part, let us investigate the effect of star connectivity on parameterized entanglement. As we have seen previously, this model requires

$$\underbrace{N}_{input} + \underbrace{(5N-1)L}_{hidden} + \underbrace{3N}_{output} = N(5L+4) - L \tag{96}$$

gates and the total number of parameters is $N(4L+3)$. In this case, we use MS gates, so the model parameters increases to $N(5L+3) - L$.

The performance metrics are shown below:

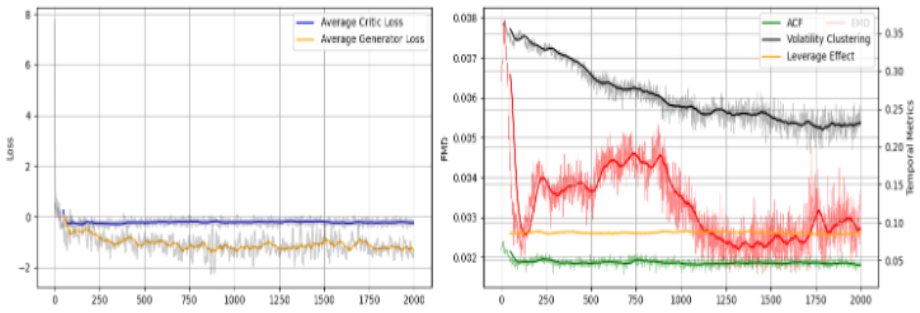


Figure 52: Training history of the QWGAN with star topology for the entangling layers with MS gates.

The generated log-returns:

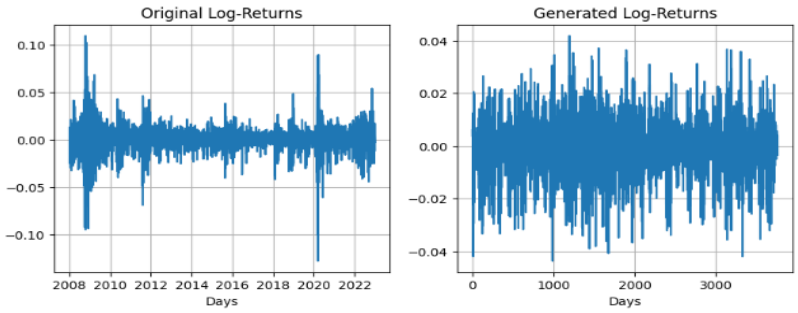


Figure 53: Original vs Generated Log-Returns.

Once again, we can see from the performance metrics that the model with parameterized star connectivity outperforms all previous models, since it learns the underlying data properties much faster.

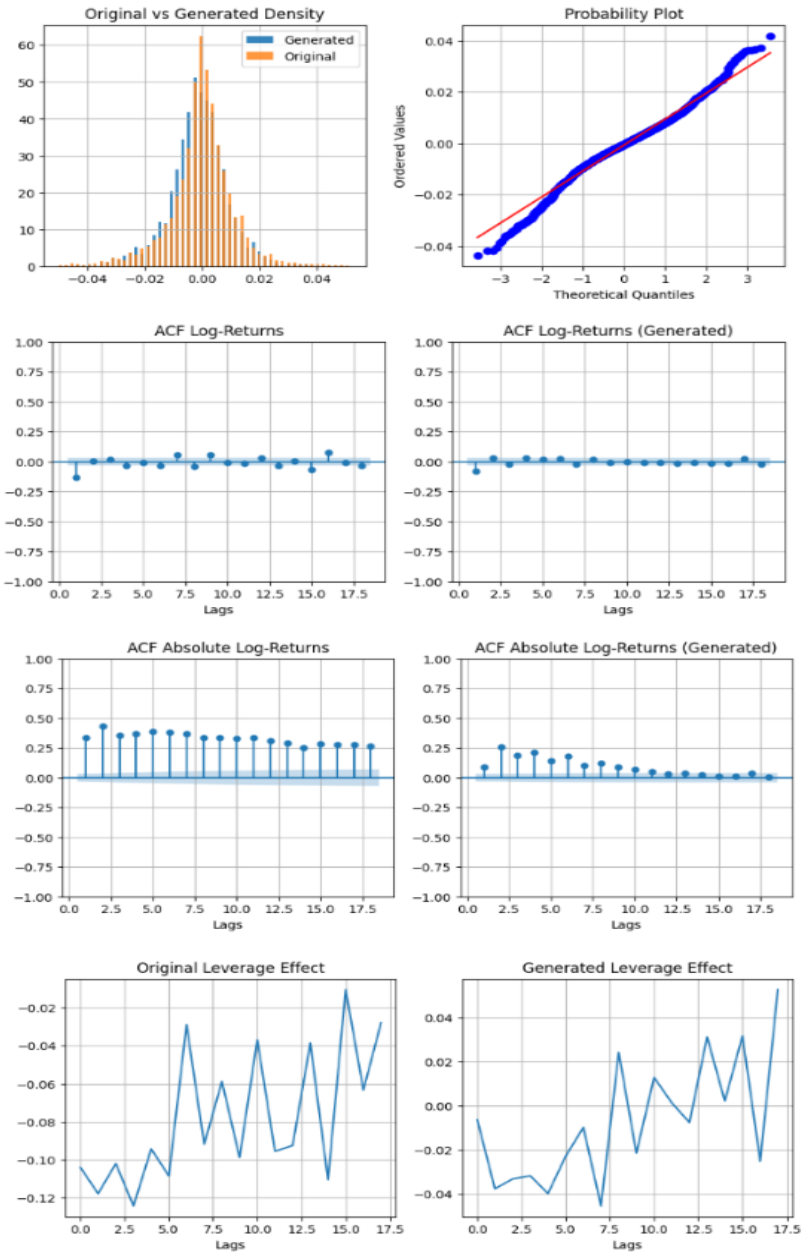


Figure 54: Original vs Generated properties.

At this point, we should note that we chose not to go with the all-to-all connectivity with parameterized couplings, mainly due to limited computational resources, since the increased number of parameters and gates severely deteriorates execution time and memory usage. The code for all models may be found in [57].

5.4 On Model Complexity

In this last subsection, we are going to discuss and try to compare the generative models we implemented for generating financial time series data, as well as highlight the complexity of such a task. We begin with a comparison of the classical WGAN with the QWGAN in terms of model architectures.

5.4.1 Classical GAN

The table below shows the amount of training data used and how it was partitioned for the learning process using the rolling window with length and stride as hyperparameters.

Data	Span	Observations	Length	Stride	Training Set
S&P 500	2000-2022	5787	30	1	(5758,30)

Table 2: Dataset Information for WGAN.

Using Tensorflow, we can inspect the total number of parameters of the model which we showcase below for reference:

Critic Parameters	Generator Parameters	Total
369,665	8,450	378,115

Table 3: Number of trainable parameters of the WGAN.

In Appendix B, we present the architecture of the networks as they are displayed by TensorFlow. In addition to those parameters, the model requires careful tuning of quite a few hyperparameters. This makes it obvious that there are endless possibilities in fine-tuning a WGAN with gradient penalty. The hyperparameters that we used were set based on various works that we have already referenced. Our goal was not to design a model for an industrial case, but to build a solid structure that can be scaled to the quantum computing field.

5.4.2 Quantum GAN

Since simulating quantum circuits requires lots of computational resources and access to quantum computers on the cloud is costly, we limited ourselves to a smaller time span and used a smaller training set for the Quantum Wasserstein GAN (QWGAN).

Adam learning rate η
Adam β_1,β_2
Batch size m
of Epochs
of critic rounds n_{critic}
Gradient penalty λ
LeakyReLU slope α
Dropout rate
Clipping threshold for Lambert
Input noise dimensions

Table 4: Hyperparameters of the WGAN.

Data	Span	Observations	Length	Stride	Training Set
S&P 500	2008-2022	3777	20	5	(752,20)

Table 5: Dataset Information for QWGAN.

As we have seen, the quantum generator architecture depends on the topology of the entangling layers, whether parameterized or non-parameterized gates are used. Moreover, when parameterized gates are used for entanglement, the number of parameters of the circuit also depends on the layer architecture. Let us summarize the complexity we derived in the tables below. Assume that we have N qubits and L layers for the PQC, which are hyperparameters.

	Ring	Star	All-to-all
# of gates	$N(5L+4)$	$N(5L+4)-L$	$\frac{LN^2+7LN+8N}{2}$
# of parameters	$N(4L+3)$	$N(4L+3)$	$N(4L+3)$

Table 6: Quantum Generator Architecture with CZ gates.

	Ring	Star	All-to-all
# of gates	$N(5L+4)$	$N(5L+4)-L$	-
# of parameters	$N(5L+3)$	$N(5L+3)-L$	-

Table 7: Quantum Generator Architecture with MS gates.

6 Conclusion & Outlook

Considering all the facts, we built upon the classical WGAN-GP framework and incorporated a parameterized quantum circuit based on quantum machine learning techniques in order to deal with financial data following the work in [51]. As one will notice, the process of doing so is a very complex task and achieving decent results requires patience, persistence and a vast amount of research along with expertise.

6.1 Discussion

By looking back at the results, we can conclude that the QWGAN with parameterized entangling layers is more promising for the task of generating financial time series data. Moreover, the model where the entangling layers utilize star connectivity is the least complex and has the ability to capture temporal effects more efficiently. We have shown that the topology of the entangling layers can impact model performance, since ring topology does not seem to capture volatility clustering at all, whereas the star topology is capable to do so.

One important thing to mention is that the quantum generator has way less parameters compared to the classical, but it seems it is still possible to generate financial time series data with temporal properties at the cost of a long training time. This is logical, as the quantum circuit is sampled 1000 times in each generation to get the average expectation values. With more available resources and utilization of NVIDIA CUDA Toolkit [68] capabilities for GPU acceleration with TensorFlow and TensorFlow Quantum, the training time may be reduced dramatically and proper hyperparameter tuning is expected to give more decent results.

In our case, we used 10 qubits and 4 layers, so if we consider the star topology circuit with parameterized entangling layers, we can easily derive that the quantum generator has 226 parameters in contrast to the 8,450 parameters of the classical generator network! This is a huge decrease in the number of trainable parameters of the model! But, is this a fair comparison? In [62], the authors fixed the models to have the same numbers of trainable parameters as a benchmark of fairness. This is achieved by adjusting the number of hidden units in the restricted Boltzmann machine architecture. Of course, adjusting the quantum circuit structure to match the number of parameters of the classical model is out of the question. Optimizing the balance between the number of qubits and circuit depth becomes a critical consideration in NISQ-era, acknowledging the limitations of available hardware. The preference for shallow circuits often involves a trade-off between

the number of qubits used and the complexity of the computation. With fewer layers, the quantum algorithm might demand more qubits to achieve similar computational outcomes as deeper and more complex circuits. In order to match the number of parameters of our quantum circuit, we would need to reduce the number of layers and neurons in the classical generator of the WGAN. This will result in a significant decrease in the expressibility of the generator, following a decrease in overall model performance. Moreover, for a clear and straightforward comparison, we should also use a fixed training dataset for both approaches. As we saw, we trained the quantum model on a smaller dataset and different preprocessing steps, as well as a different set of hyperparameters mainly due to computational burden.

6.2 Future Outlook

For a future work following ours and [51], one may also consider different critic architectures that can help the quantum circuit during the learning process, or even use different optimization algorithms for both networks. It has been shown that temporal convolutional neural networks (TCNs) are primed for such a task [61], so one may investigate an approach by replacing the critic network. Moreover, alternative data pre-processing pipelines may be followed, such as introducing training parameters to the output expectation values of the quantum circuit and let the model learn to scale the output and match the original data [51]. Instead, our approach limits the scaling of the output expectations, so it is a part that surely has better potential.

Another interesting approach regarding synthetic data generation would be for prediction purposes. In [69], the authors propose a classical GAN architecture using the S&P 500 daily closing values and try to predict the daily closing price. To achieve this, they use a long short-term memory (LSTM) as the generator network and a multi-layer perceptron (MLP) discriminator and their results show promising performance compared with other models. This exact application may be explored using the WGAN-GP model we discussed in this thesis, as it is not included in their comparison. Then, the QWGAN may be exploited for this purpose. Since they try to generate a single value, the QWGAN may be adjusted such that the quantum generator outputs a single expectation by using a set of a single Pauli string. In this case, the critic architecture is also reduced, along with the required computational resources. Furthermore, one could test their approach without altering the optimization process by replacing just the LSTM generator with a parameterized quantum circuit featuring diverse architectures, just like we did with the WGAN.

The ultimate objective of this work is to showcase how we can approach generative modeling tasks with the rapidly evolving field of quantum technology. By harnessing the fundamental laws of quantum mechanics and the properties of such systems, we are entering an era where it is possible to reform existing state-of-the-art systems regarding real-world applications, or even replace them with pure quantum mechanical counterparts. Alas, realization of a fault-tolerant quantum computer is not possible in the near future. This should not limit us to explore innovative ideas, as hybrid quantum-classical computing applications are very promising and the industry is already interested on the subject, especially for real-world tasks involving generative modeling.

"Nature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly, it's a wonderful problem because it doesn't look so easy."

Richard P. Feynman

"Generative models are a family of machine learning models that can generate data with high fidelity, both in terms of visual appearance and, in many cases, in terms of statistical quality."

Ian J. Goodfellow

A Artificial Neural Networks

A.1 Basic Structure

As the name suggests, ANNs form a simplified model of the human brain. Generally, we can think of an ANN as a *nonlinear function* which transforms input data to output, based on the training on many data samples and parameters. In 1943, Warren McCulloch and Walter Pitts suggested the computational model of a neuron. For the sake of completeness, know that the human brain consists of approximately 100 billion neurons (brain cells) and 100 trillion synapses (permit interactions between neurons).

An ANN is a collection of connected processing units (neurons) which consist a network. Each of these units is able to execute a simple and specific mathematical operation.

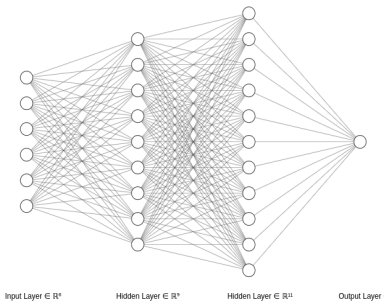


Figure 55: A dense ANN with two hidden layers.

Every neural network must obviously have an *input layer* as well as an *output layer*. The in-between are called the *hidden layers*, but the simplest form of a network is just a single neuron (or perceptron) with one or more inputs, some processing and a single output.

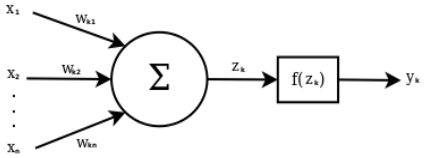


Figure 56: Basic Neuron Model (Perceptron).

The inputs are denoted as $x_j \in \mathbb{R}$ where $j = 1, ..., n$. The output of the neuron is a *nonlinear function* of weighted sums of inputs:

$$z_k = \sum_j w_{kj} x_j + b_k \tag{97}$$

where $w_{kj} \in \mathbb{R}$ are the *weights* and $b_k \in \mathbb{R}$ is some *offset* or *bias*. Then, (1) is the input of some nonlinear function which can be the same for all neurons in a network. Two examples of commonly used functions include the *rectified linear unit (RELU)*, where $y_k = f(z_k) = z_k$ and a constant derivative equal to one for $z_k \geq 0$ and zero elsewhere. Also, the *sigmoid function*

$$f(z_k) = \frac{1}{1 + e^{-z_k}} \tag{98}$$

provides a smooth variation in the output within the interval $[0, 1]$, which is useful when we want to model uncertainty. The derivative is equal to $(1 - y_k)y_k$. One may also consider other activation functions as well, as long as changes in the inputs do not cause big changes in the output.

Let us consider a simple network with no hidden layers and introduce some indices that help avoid consufion when analyzing the underlying processes, as well as when scaling the network with more layers.

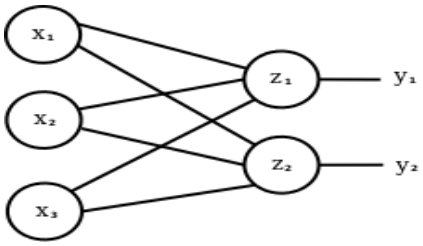


Figure 57: Simple ANN without hidden layers.

We will use the index j for the input neurons and k for the output neurons. So, from (1):

$$z_k = \sum_{j=1}^{n} w_{kj} x_j + b_k \tag{99}$$

for $k = 1, 2$, hence

$$y_k = f(z_k) \tag{100}$$

By simple inspection on the indices and some linear algebra, we can write (3) in a more compact and computationally effective form as

$$\vec{\mathbf{z}} = \mathbf{W} \cdot \vec{\mathbf{x}} + \vec{\mathbf{b}} \tag{101}$$

where $\vec{\mathbf{z}} = \begin{bmatrix} z_1 & z_2 \end{bmatrix}^T$ is the output k-dimensional vector, $\mathbf{W}$ is the weight matrix of $k \times j$ dimensions acting with the dot product on the input j-dimensional vector $\vec{\mathbf{x}} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$ and $\vec{\mathbf{b}}$ is the bias vector of k dimensions. The number of rows of the weight matrix is equal to the number of output neurons and the number of columns is equal to the number of input neurons corresponding to that matrix. At last, we have a k-dimensional vector $\vec{\mathbf{y}}$ consisting of the nonlinear outputs of some activation function. It is also clear that there is a *feedforward process*, where the outputs of some layer are passed as inputs to the next layer.

A.2 The Universal Approximation Theorem

The *Universal Approximation Theorem* states that any arbitrary smooth function with vector input and vector output can be approximated as well as desired by an ANN with at least one hidden layer, as long as we allow for sufficiently many neurons [9].

All of the aforementioned may seem a little bit abstract in terms of the values of weights and biases. One must adjust these parameters in order to get the desired output value. This value should be of high accuracy and we can achieve this through *training* of the network at hand.

Training refers to the adjustment of those parameters based on some training data samples. This is a form of supervised learning. Assume that a neural network is denoted by F_w (contains weights and biases), its input vector by x^{in} and its output vector by y^{out}, such that

$$y^{out} = F_w(x^{in}) \tag{102}$$

Also, let the desired or target function be F. A training dataset consists of inputs x^{in} and respective outputs y^{target}, such that $y^{target} = F(x^{in})$. Obviously, we would like the network output to approximate with high accuracy the target output, hence

$$y^{out} \approx y^{target} = F(x^{in}) \tag{103}$$

We need a *cost function* that can measure the deviation between y^{out} and y^{target} with respect to the parameters. One may use various functions based on the underlying task, but one of the most common cost functions is the *least-squares* defined as

$$C(w) = \frac{1}{2} \left\langle \|F_w(x^{in}) - F(x^{in})\|^2 \right\rangle \tag{104}$$

where the vector *norm* is used and the *average over all samples* is taken.

There is process called *batch training*, where the respective model is trained on many samples in parallel. In this case, we would need a matrix of samples with dimensions $N_{samples} \times j$. That is the reason we use the average in expression (8). The 1/2 factor is used to ease derivative evaluation, but with modern computers this is not a matter to be concerned of.

A.3 Gradient Descent and Backpropagation

The purpose of training is to find the "best" weights and biases by minimizing the cost function with respect to its parameters. We can achieve this by using the *gradient descent method*. The gradient of the cost function is evaluated and then, the parameters should follow the path of the steepest descent, as the method's name suggests. Remember that the gradient is a vector pointing in the direction of the steepest ascent, so we need to find the negative gradient for minimization. One may also reverse the problem based on the underlying task and try to maximize the cost by using the so-called *gradient ascent*. We will stick to the minimization of the cost, meaning that if $C(w) \to 0$, the model tends toward better accuracy.

The problem is that evaluating $C(w)$ would mean averaging over all training samples. However, when we deal with lots of data we tend to average only a few samples and get an approximate cost. In each step, different samples are taken. This is called *stochastic gradient descent (SGD)*.

For sufficiently small steps, the sum over many steps approximates the true gradient. The following figure helps visualizing the difference between SGD and the true gradient:

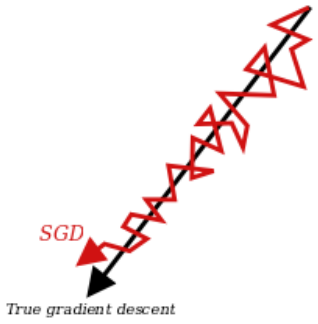


Figure 58: SGD vs True Gradient Descent.

After evaluating the gradient of the cost function, we proceed to the update of the parameters as

$$w_* \;\to\; w_* - \eta \frac{\partial C(w)}{\partial w_*} \tag{105}$$

where η is the so-called *learning rate*, a small constant to ensure convergence to a local minimum. The learning rate can be also interpreted as a step size that shows how fast the gradient is moving and thus, how fast the learning process is. We assume that w_* is some weight somewhere in the network including the bias, as the introduction of this offset is the same as if we considered an extra input $x_0 = 1$ with weight $w_{k0} = b_k$.

For deep networks in general, the question is how do we calculate the gradient with respect to some inner weight? If a network consists of one million weights, we need to evaluate it one million times! Fortunately, the *chain rule* for derivatives is the holy grail here!

Let us consider a simple network with two inputs and one output as shown below:

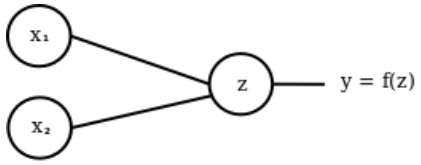


Figure 59: A simple ANN with two inputs and one output.

From (3), we have

$$z = w_1 x_1 + w_2 x_2 + b \tag{106}$$

and the cost function reads

$$C(w) = \frac{1}{2} \left\langle (f(z) - F(x_1, x_2))^2 \right\rangle \tag{107}$$

where F is the target function. Using the last two expressions, the gradient reads

$$\nabla C(w) = \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \\ \frac{\partial C}{\partial w_2} \end{bmatrix} = \begin{bmatrix} \left\langle (f(z) - F) f'(z) \frac{\partial z}{\partial w_1} \right\rangle \\ \\ \left\langle (f(z) - F) f'(z) \frac{\partial z}{\partial w_2} \right\rangle \end{bmatrix} = \begin{bmatrix} \langle (f(z) - F) f'(z) x_1 \rangle \\ \\ \langle (f(z) - F) f'(z) x_2 \rangle \end{bmatrix}$$

and thus, we have all the values to evaluate it. We proceed with an update for both weights as in (9) and feed another input data until the cost function reaches a local minimum.

Next, we consider the general case of an ANN with n hidden layers:

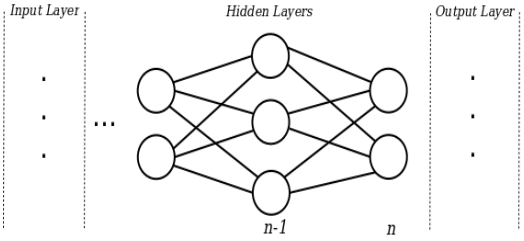


Figure 60: General ANN with n hidden layers.

In order to work with the full network when it comes for the gradient layer-by-layer, we adopt the well-known *backpropagation algorithm*. The name will be clear later on. First, we need to keep track of the indices very carefully:

- x^{in}, the ANN input
- y_j^n, the value of neuron j in layer n
- z_j^n, the input value for $y_j^n = f(z_j^n)$
- $w_{jk}^{n,n-1}$, the weight from the k-th neuron in layer $n-1$ feeding the j-th neuron of layer n

The cost for one particular input is $C(w) = \langle C(w, x^{in}) \rangle$. The derivative with respect to some weight w_* somewhere in the network is

$$\frac{\partial C(w, x^{in})}{\partial w_*} = \sum_j (y_j^n - F_j(x^{in})) \frac{\partial y_j^n}{\partial w_*} \tag{108}$$

$$= \sum_j (y_j^n - F_j(x^{in})) f'(z_j^n) \frac{\partial z_j^n}{\partial w_*} \tag{109}$$

where F_j is the target function. Remember that $y_j^n = f(z_j^n)$, so if we apply the chain rule repeatedly for the term $\frac{\partial z_j^n}{\partial w_*}$, we get

$$\frac{\partial z_j^n}{\partial w_*} = \sum_k \frac{\partial z_j^n}{\partial y_k^{n-1}} \frac{\partial y_k^{n-1}}{\partial w_*} = \sum_k w_{jk}^{n,n-1} f'(z_k^{n-1}) \frac{\partial z_k^{n-1}}{\partial w_*} \tag{110}$$

This last expression runs over all the weights corresponding to the j-th neuron of layer n. Plugging this into (13), we must do the same for all neurons of layer n. Here comes an important insight to compute this efficiently. We can construct a matrix $\mathbf{M}$ whose element with index (j, k) is

$$M_{jk}^{n,n-1} = w_{jk}^{n,n-1} f'(z_k^{n-1}) \tag{111}$$

.

In this way, each pair of layers $(l, l-1)$ backwards through the network contributes with multiplication of a matrix $\mathbf{M}$ with dimensions $(l \times l-1)$. Instead of having to go through all the weights of some pair of layers, we can use repeated matrix multiplication starting from the last hidden layer up to the input as

$$\frac{\partial z_j^n}{\partial w_*} = \sum_{k,l,\ldots,u,v} M_{jk}^{n,n-1} M_{kl}^{n-1,n-2} \ldots M_{uv}^{2,1} \frac{\partial z_v^1}{\partial w_*} \tag{112}$$

We see that in order to compute the derivative of the cost function, we need values from layer n, as well as values from the previous layer $n-1$. To continue with the evaluation of (14), we need the values of layer $n-1$ and layer $n-2$. In this way, we propagate the results starting from the n-th hidden layer backwards onto the first hidden layer. That is the backpropagation algorithm, which we declare below:

Algorithm 2 Backpropagation

Input: $y_j^n, F_j(x^{in}), f'(z_j^n)$ ▷ n is the output layer
Output: $\frac{\partial C(w,x^{in})}{\partial w_*}$ ▷ gradient vector with respect to all weights

$\Delta_j \leftarrow (y_j^n - F_j(x^{in}))f'(z_j^n)$ ▷ initialize vector from output layer
$l \leftarrow n$
repeat ▷ for each pair of layers
  for all $j \in l$ **do** ▷ neurons in l-th layer
    for all $k \in (l-1)$ **do** ▷ neurons in $(l-1)$-th layer
      $\mathbf{M}_{jk}^{n,n-1} \leftarrow w_{jk}^{n,n-1} f'(z_k^{n-1})$ ▷ construct layer matrix
    end for
  end for

  $\Delta_k^{new} \leftarrow \sum_j \Delta_j \mathbf{M}_{jk}^{n,n-1}$ ▷ multiply vector by matrix
  $dC \leftarrow$ store cost derivatives for all weights and biases in layer l
  $l \leftarrow l-1$
until first hidden layer and input layer pair

Once we get the gradient with respect to all the parameters of the network, we proceed to the update. Then, the same procedure is followed with the updated parameters on some other subset of the training dataset. After all, everything reduces to matrix and vector multiplications which can be easily achieved through the efficiency of modern computer algebra. This is the

power of the backpropagation algorithm, which combines basic aspects of calculus and linear algebra. Notice that we do not have to evaluate with respect to all possible weights w_* of the network. We just start from some weight in the output layer and via the chain rule, all weights contribute to the gradient vector, as it was intended to. This is the basis for the training methods that we refer to as **gradient-based**. Gradient-based optimization methods utilize gradient information to iteratively update model parameters in order to minimize or maximize a cost function. These methods are effective when the cost function is differentiable and gradients can be efficiently calculated. The most common include *Stochastic Gradient Descent (SGD)*, *Mini-Batch Gradient Descent* and *Adaptive Moment Estimation (Adam).* The other class of training methods are called **gradient-free**. Gradient-free optimization methods do not require gradient information and are suitable for optimizing non-differentiable or noisy functions. Common techniques include *Random Search*, *Genetic Algorithms*, *Particle Swarm Optimization (PSO)* [65] and *Bayesian Optimization* [66].

A.4 On Hyperparameters, Datasets and Training

The art of machine learning lies in the *fine-tuning* of the parameters and the structure of the respective model. In general, there is no standard procedure that one should follow in order to adjust the settings of a model, as it all comes down to the background and the experience of the designer. Before heading towards implementation though, we have to deal with the *data* that the network will be trained on, especially in the case of supervised learning. The stage of *data pre-processing* has to do with collection, feature extraction and various techniques involving statistical analysis and pattern recognition, such as missing data issues, noisy samples, outliers removal and normalization. The following diagram portrays the various concerns that arise when designing a machine learning prototype:

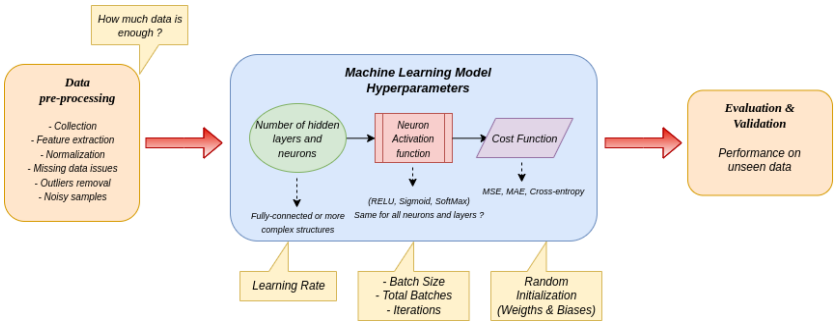


Figure 61: Learning is a stochastic and very complex nonlinear process.

Basic machine learning model implementations with *Python*, as well as a summary of how parameter selection can influence the training process can be found in my personal *github* repository [7].

B WGAN Architectures

B.1 Critic Network

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 30, 64)	704
leaky_re_lu (LeakyReLU)	(None, 30, 64)	0
conv1d_1 (Conv1D)	(None, 30, 128)	82048
leaky_re_lu_1 (LeakyReLU)	(None, 30, 128)	0
conv1d_2 (Conv1D)	(None, 30, 128)	163968
leaky_re_lu_2 (LeakyReLU)	(None, 30, 128)	0
flatten (Flatten)	(None, 3840)	0
dense (Dense)	(None, 32)	122912
leaky_re_lu_3 (LeakyReLU)	(None, 32)	0
dropout (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 1)	33

Total params: 369,665
Trainable params: 369,665
Non-trainable params: 0

Figure 62: Critic network with window length 30.

B.2 Generator Network

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 30)	270
leaky_re_lu_4 (LeakyReLU)	(None, 30)	0
dense_3 (Dense)	(None, 50)	1550
leaky_re_lu_5 (LeakyReLU)	(None, 50)	0
dense_4 (Dense)	(None, 50)	2550
leaky_re_lu_6 (LeakyReLU)	(None, 50)	0
dropout_1 (Dropout)	(None, 50)	0
dense_5 (Dense)	(None, 50)	2550
leaky_re_lu_7 (LeakyReLU)	(None, 50)	0
dropout_2 (Dropout)	(None, 50)	0
dense_6 (Dense)	(None, 30)	1530

Total params: 8,450
Trainable params: 8,450
Non-trainable params: 0

Figure 63: Generator network with window length 30.

References

[1] R. P. Feynman. "Simulating Physics with Computers", International Journal of Theoretical Physics, VoL 21, Nos. 6/7, 1982

[2] T. D. Ladd, et al. "Quantum computers", Nature 2010 464:7285 464, 45 (2010).

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

[4] M.A. Nielsen and I. L. Chuang. "Quantum Computation and Quantum Information", 2000

[5] `https://github.com/dkomni/quantum-algorithms`

[6] A. M. Turing, On Computable Numbers, with an Application to the ENTSCHEIDUNGSPROBLEM, 1936

[7] `https://github.com/dkomni/machine-learning-overview`

[8] `https://www.tensorflow.org/`

[9] Hornik, Kurt; Stinchcombe, Maxwell; White, Halbert (1989). "Multilayer Feedforward Networks are Universal Approximators". Neural Networks. Vol. 2. Pergamon Press. pp. 359–366.

[10] John J Hopfield. "Neural networks and physical systems with emergent collective computational abilities." In: Proceedings of the national academy of sciences 79.8 (1982), pp. 2554–2558.

[11] David H Ackley, Geoffrey E Hinton and Terrence J Sejnowski. "A learning algorithm for Boltzmann machines". In: Cognitive science 9.1 (1985), pp. 147– 169.

[12] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Tech. rep. Colorado Univ at Boulder Dept of Computer Science, 1986.

[13] Y Freund and D Haussler. "Unsupervised learning of distributions on binary vectors using two layer networks (Technical Report UCSC-CRL-94-25)". In: University of California, Santa Cruz (1994).

[14] Geoffrey E Hinton. "Training products of experts by minimizing contrastive divergence". In: Neural computation 14.8 (2002), pp. 1771–1800.

[15] Joshua S. Speagle. "A Conceptual Introduction to Markov Chain Monte Carlo Methods", https://arxiv.org/abs/1909.12313, 2019

[16] Geoffrey E Hinton. "A practical guide to training restricted Boltzmann machines". In: Neural networks: Tricks of the trade. Springer, 2012, pp. 599–619.

[17] Ian J Goodfellow et.al. "Generative Adversarial Networks", https://arxiv.org/abs/1406.2661, 2014

[18] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", arXiv:quant-ph/9508027, 1994

[19] J. Biamonte, et al. "Quantum Machine Learning", arXiv:1611.09347v2 [quant-ph], 2018

[20] Jonathon Shlens. "A Tutorial on Principal Component Analysis", https://arxiv.org/abs/1404.1100, 2014

[21] M.A. Hearst et al. "Support vector machines", https://doi.org/10.1109/5254.708428, 1998

[22] Aram W. Harrow, Ashley Montanaro. Quantum Computational Supremacy. arXiv:1809.07442v1 [quant-ph]

[23] Lund, A.P., Bremner, M.J. & Ralph, T.C. Quantum sampling problems, BosonSampling and quantum supremacy. npj Quantum Inf 3, 15 (2017). https://doi.org/10.1038/s41534-017-0018-2

[24] Wiebe, N., Kapoor, A. & Svore, K. M. Quantum deep learning. Preprint at https://arxiv.org/abs/1412.3489, 2014.

[25] Adachi, S. H. & Henderson, M. P. Application of quantum annealing to training of deep neural networks. Preprint at https://arxiv.org/abs/arXiv:1510.06356, 2015.

[26] Amin, M. H., Andriyash, E., Rolfe, J., Kulchytskyy, B. & Melko, R. Quantum Boltzmann machine. Preprint at https://arxiv.org/abs/arXiv:1601.02036, 2016.

[27] Von Neumann, J. (1956) "Probabilistic Logics and Synthesis of Reliable Organisms from Unreliable Components". In: Shannon, C.E. and McCarthy, J., Eds., Automata Studies, in Annals of Mathematical Studies, No. 34, Princeton University Press, Princeton, 43-98.

[28] Peter W. Shor. "Fault-Tolerant Quantum Computation". http://arXiv.org/abs/quant-ph/9605011v2, 1997

[29] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum algorithm for linear systems of equations". In: Physical review letters 103.15 (2009), p. 150502.

[30] Wiebe, N., Braun, D. & Lloyd, S. "Quantum algorithm for data fitting". Phys. Rev. Lett. 109, 050505 (2012). DOI 10.1103/PhysRevLett.109.050505.

[31] Vittorio Giovannetti, Seth Lloyd, Lorenzo Maccone. "Quantum random access memory". http://arXiv.org/abs/0708.1879v2, 2008

[32] Guang Hao Low, Theodore J. Yoder, Isaac L. Chuang. "Quantum Inference on Bayesian Networks". arXiv:1402.7359v1 [quant-ph], 2014

[33] Wiebe et. al. "Quantum algorithm for data fitting". Phys. Rev. Lett. 109, 050505 (2012). DOI 10.1103/PhysRevLett.109.050505.

[34] Seth Lloyd, Masoud Mohseni, Patrick Rebentrost. "Quantum principal component analysis". http://arxiv.org/abs/1307.0401v2, 2013

[35] Patrick Rebentrost, Masoud Mohseni and Seth Lloyd. "Quantum support vector machine for big data classification". http://arxiv.org/abs/1307.0471v3, 2014

[36] Stefano Mangini. "Variational quantum algorithms for machine learning: theory and applications", https://arxiv.org/abs/2306.09984, 2023

[37] Marcello Benedetti, et al. "Parameterized quantum circuits as machine learning models". 2019 Quantum Sci. Technol. 4 043001

[38] J. Romero, J. P. Olson, A. Aspuru-Guzik. "Quantum autoencoders for efficient compression of quantum data", https://arxiv.org/abs/1612.02806, 2016

[39] Mohammad H. Amin, et al. "Quantum Boltzmann Machine", arXiv:1601.02036v1 [quant-ph], 2016

[40] M. Benedetti, et al. "A generative modeling approach for benchmarking and training shallow quantum circuits", arXiv:1801.07686v4 [quant-ph], 2019

[41] Jin-Guo Liu and Lei Wang. "Differentiable Learning of Quantum Circuit Born Machine", arXiv:1804.04168v1 [quant-ph], 2018

[42] D. Zhu, et al. "Training of Quantum Circuits on a Hybrid Quantum Computer", arXiv:1812.08862v2 [quant-ph], 2019

[43] S. Sim, et al. "Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms" Advanced Quantum Technologies 2, 1900070 (2019).

[44] A. Chakraborti et.al. "Econophysics: An Introduction", Wiley-vch, 2011

[45] R. Cont. "Empirical properties of asset returns: Stylized facts and statistical issues", Centre de Mathematiques Appliqu ´ ees, Ecole Polytechnique, F-91128 ´ Palaiseau, France, 2001

[46] X. Zhou, Z. Pan, G. Hu, S. Tang, C. Zhao. "Stock market prediction on high-frequency data using generative adversarial nets", Mathematical Problems in Engineering 2018 (2018) 4907423.

[47] A. Koshiyama, N. Firoozye, P. Treleaven. "Generative adversarial networks for financial trading strategies fine-tuning and combination", arXiv preprint arXiv:1901.01751 (2019).

[48] U. Fiore, A. D. Santis, F. Perla, P. Zanetti, F. Palmieri. "Using generative adversarial networks for improving classification effectiveness in credit card fraud detection", Information Sciences 479 (2019) 448–455.

[49] Y.-J. Zheng, X.-H. Zhou, W.-G. Sheng, Y. Xue, S.-Y. Chen. "Generative adversarial network based telecom fraud detection at the receiving bank", Neural Networks 102 (2018) 78–86.

[50] F Eckerli, J Osterrieder. "Generative Adversarial Networks in finance: an overview", `https://arxiv.org/abs/2106.06364`, 2021

[51] Schwander, E. (2022). "Quantum generative modelling for financial time series".

[52] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. "Improved Training of Wasserstein GANs", Advances in Neural Informa- tion Processing Systems , 5769 (2017).

[53] S. Takahashi, Y. Chen, and K. Tanaka-Ishii. "Modeling financial time-series with generative adversarial networks", Physica A: Statistical Mechanics and its Applications 527, 121261 (2019).

[54] P. Virtanen et al. "SciPy 1.0: fundamental algorithms for scientific comput- ing in Python", Nature Methods 17, 261 (2020).

[55] NIST/SEMATECH, e-Handbook of Statistical Methods, 2013.

[56] G. M. Goerg. "The Lambert Way to Gaussianize heavy tailed data with the inverse of Tukey's h transformation as a special case", (2013).

[57] `https://github.com/dkomni/financial-modeling-gans`

[58] R. Orús, et al. "Quantum computing for finance: Overview and prospects", `https://doi.org/10.1016/j.revip.2019.100028`, 2019

[59] M. Pistoia, et al. "Quantum Machine Learning for Finance", arXiv:2109.04298v1 [quant-ph], 2021

[60] D. A. Herman, et al. "A Survey of Quantum Computing for Finance", arXiv:2201.02773v4 [quant-ph], 2022

[61] M. Wiese, et al. "Quant GANs: Deep Generation of Financial Time Series", arXiv:1907.06673v2 [q-fin.MF], 2019

[62] B. Coyle, et al. "Quantum versus classical generative modelling in finance", `https://doi.org/10.1088/2058-9565/abd3db`, 2020

[63] S. Lloyd, C. Weedbrook. "Quantum generative adversarial learning", arXiv:1804.09139v1 [quant-ph], 2018

[64] C. Zoufal, A. Lucchi, S. Woerner. "Quantum Generative Adversarial Networks for Learning and Loading Random Distributions", arXiv:1904.00043v2 [quant-ph], 2019

[65] J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of ICNN'95 - International Conference on Neural Networks, Perth, WA, Australia, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

[66] J. Snoek, H. Larochelle, R. P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms", `https://arxiv.org/abs/1206.2944`, 2012

[67] https://qiskit.org/

[68] https://developer.nvidia.com/cuda-toolkit

[69] K. Zhang, et.al. "Stock Market Prediction Based on Generative Adversarial Network", Procedia Computer Science, Volume 147, 2019, Pages 400-406, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2019.01.256.