

Università degli studi di Torino

SCUOLA DI SCIENZE DELLA NATURA

Corso di Laurea Magistrale in Fisica dei Sistemi Complessi



Tesi di Laurea Magistrale

Tensor Network Born Machines:

Measurement-Driven Generative Model for Quantum Data

Relatori:

Dott. Mangini Stefano
Dott. Grossi Michele
Prof. Carlini Alberto

Candidato:

Robbiano Mattia

Controrelatore:

Prof. Fariselli Piero

Anno Accademico 2024/2025

To my family

Mattia Robbiano

Abstract

Quantum generative models leverage the probabilistic structure of quantum mechanics to learn and reproduce complex data distributions beyond classical capabilities. Among these, Born machines encode probabilities via quantum amplitudes, offering efficient sampling and expressive power. However, their scalability is hindered by issues such as barren plateaus. In this thesis, we investigate Tensor Network Born Machines (TNBMs), which integrate tensor networks into the Born machine framework to overcome training obstacles while retaining expressive power. We examine their theoretical underpinnings and practical implementation for learning quantum data distributions.

Italian abstract

I modelli generativi quantistici sfruttano la struttura probabilistica della meccanica quantistica per apprendere e riprodurre distribuzioni di dati complesse, andando oltre le capacità classiche. Tra questi, le Born machines codificano le probabilità tramite ampiezze quantistiche, offrendo un sampling efficiente e una notevole espressività. Tuttavia, la loro scalabilità è ostacolata da problematiche come i barren plateaus. In questa tesi, indaghiamo i Tensor Network Born Machines (TNBMs), che integrano i tensor networks nel framework delle Born machine per superare gli ostacoli di training mantenendo il potere espressivo. Esaminiamo i loro fondamenti teorici e l'implementazione pratica per l'apprendimento di quantum data distributions.

Acknowledgements

I wish to express my deepest gratitude to all those who, directly or indirectly, made this work possible. All the lists below are in alphabetical order.

My first thanks go to my supervisors: Dr. Michele Grossi, who welcomed me into his research group at the CERN Quantum Technology Initiative and guided me from the very beginning into the world of research, which was completely new to me; and Dr. Stefano Mangini, whose expertise, constant mentorship and friendship made all of this possible.

A special thanks goes to my advisor, Professor Alberto Carlini, for his availability and the valuable advice that has guided me since even before I began this thesis journey.

I am grateful to my colleagues and friends who have stood by me over the past months at CERN: Alberto, Alice, Laura, Matteo, and Simone, for all the laughs, the experiences, the scientific thoughts, and the emotions shared.

Thanks to the friends and colleagues at the University of Torino, with whom I have shared the efforts and challenges of this journey. Among all, a special thought goes to: Bianca, Elena, Federica, Linda, Loren, Luca, Marco, Matteo, Sara, Sergio.

A special thank you to the members of the *Quantum Computing Journal Club of Torino* for all the moments and valuable discussions shared, in particular: Daniel, Enrico, Francesco, Gaetano, Niccolò.

The achievement of my academic goals in Turin would not have been possible

without the support of an exceptional group of housemates and friends. I am especially grateful to Alessandro, Erika, Fabio, Lino, Luca, Nunzio, and Oumar.

A heartfelt thank you to Abigail, for being by my side during these last challenging months, offering support and encouragement.

A special thought goes to my lifelong friends, for always supporting me with patience and encouragement—especially, this work would not have been possible without the long days spent on the *moist campus* with Francesca, Matteo, Paolo, and Pietro.

Finally, this work, like every other goal I have achieved in my life, is dedicated to my family, whose support, patience, and love are the foundation upon which everything rests.

Thank you to all who, in various ways, contributed to the completion of this work.

Contents

Acknowledgements	iv
1 Introduction	1
2 Quantum Information elements	3
2.1 Quantum States and Hilbert Spaces	3
2.1.1 The Qubit	4
2.1.2 Quantum Measurement	5
2.2 Density matrix formalism	6
2.2.1 Composite Quantum Systems	7
2.3 Quantum Gates and Operations	8
2.3.1 Single-Qubit Gates	9
2.3.2 Multi-Qubit Gates	9
2.3.3 Quantum Circuit Model	10
3 Variational quantum algorithms and machine learning elements	13
3.1 Variational Methods	13
3.1.1 Variational Principle	13
3.1.2 Variational quantum algorithms	14
3.2 Barren Plateaus and Loss Concentration	17
3.3 Optimization in Machine Learning	19
3.3.1 Loss Functions	19
3.3.2 Explicit and Implicit Models	21
3.3.3 Gradient-Based Optimization	22
3.4 Quantum Circuit Born Machines	25

CONTENTS

4 Tensor Network Elements	29
4.1 Introduction	29
4.2 Tensor Networks	31
4.3 Singular Value Decomposition	32
4.4 Matrix Product States	32
5 Tensor Network Born Machine	36
5.1 Introduction	36
5.2 Sampling from MPS	37
5.3 Architecture	39
5.3.1 Parametrized MPS	39
5.3.2 POVM Projection	40
5.3.3 Data Representation: Hyperindexed Tensor Network	42
5.4 Kullback-Leibler Divergence Calculation	43
5.5 Maximum Mean Discrepancy Calculation	44
6 Numerical Results	49
6.1 Training and Evaluation of Tensor Network Generative Models	49
6.1.1 The Bars and Stripes Dataset	49
6.1.2 Random Initialization and Initial Sampling Behavior	52
6.1.3 Training	52
6.1.4 Evaluation Metrics: Validity and Coverage	53
6.2 Loss concentration in TNBM	58
7 Conclusions	61
8 Appendix	63
8.1 Generated Samples for Different Losses and Bond Dimensions	63
Bibliography	72

Chapter 1

Introduction

Generative models have become indispensable tools for learning complex data distributions and generating new samples that resemble the original dataset. In the quantum domain, these models take on high significance as they leverage the inherently probabilistic nature of quantum systems to capture and reproduce “quantum data”. By exploiting quantum mechanics, these models can encode probability distributions that are inaccessible to classical approaches, offering efficient sampling mechanisms and paving the way for diverse applications in physics [1, 2, 3].

Two prominent quantum generative models stand out for their physical foundations: Boltzmann machines and Born machines. Boltzmann machines utilize the Boltzmann distribution from statistical mechanics to represent joint probability distributions [4], while Born machines rely on Born’s rule, encoding probabilities as squared amplitudes of wavefunctions [5, 6, 1].

Despite their promise, quantum generative models face significant challenges, particularly regarding scalability—not only from a theoretical standpoint but also in practice. Currently, these models are difficult to run because the necessary quantum hardware is not yet available: existing devices are too small, limited, and noisy. For these reasons, tensor network approaches represent a good alternative: they are well-established and high-performing techniques, originally developed to describe quantum systems. As such, they are particularly suitable for capturing, within certain limits, quantum systems behaviors and features. Issues such as barren plateaus and loss concentration —regions where gradients vanish exponentially—hinder effective training of large-scale systems [7]. To address these limitations, Tensor Network Born Machines [6, 8, 9, 10] (TNBMs) have emerged as a robust

CHAPTER 1. INTRODUCTION

alternative. By incorporating tensor networks into the Born machine framework, TNBMs mitigate barren plateau problems while maintaining the expressivity advantages of quantum-inspired models [11]. This approach represents a critical step forward in overcoming trainability barriers and enhancing the performance of quantum generative models.

In this my master thesis, we explore the theoretical foundations and practical implementations of Tensor Network Born Machines, demonstrating their ability to model complex quantum data distributions while addressing key scalability challenges.

Code for the project available at www.github.com/mattia-robbiano/tnbm.

Chapter 2

Quantum Information elements

This chapter is primarily based on the textbook *Quantum Computation and Quantum Information* by Nielsen and Chuang [12].

2.1 Quantum States and Hilbert Spaces

In quantum mechanics, the state of an isolated quantum system is completely described by a vector in a complex vector space known as a *Hilbert space*, usually denoted \mathcal{H} . A Hilbert space is a vector space equipped with an inner product that allows for the definition of length and angle, and is complete with respect to the norm induced by this inner product.

The state vectors in a Hilbert space are commonly represented using Dirac notation, also known as bra-ket notation. A “ket” vector, denoted $|\psi\rangle$, represents a quantum state and is an element of the Hilbert space \mathcal{H} . In a chosen basis, kets can be represented as column vectors. For every ket $|\psi\rangle$ in \mathcal{H} , there is a corresponding “bra” vector, denoted $\langle\psi|$, which is an element of the dual space \mathcal{H}^* . Bras can be represented as row vectors and are the conjugate transposes of their corresponding kets.

The inner product between a bra $\langle\phi|$ and a ket $|\psi\rangle$ is denoted as $\langle\phi|\psi\rangle$ and yields a complex number. This inner product quantifies the overlap between the two states. A fundamental property of quantum states is that they are normalized, meaning $\langle\psi|\psi\rangle = 1$. This ensures that the total probability of finding the system in any possible state is unity.

Every Hilbert space possesses an orthonormal basis, a set of mutually orthogonal

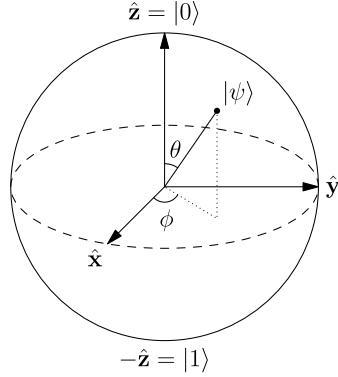


Figure 2.1: The Bloch sphere representation of a single qubit state. Each point on the surface of the unit sphere corresponds to a state $|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\phi}\sin(\theta/2)|1\rangle$, parameterized by the polar and azimuthal angles θ and ϕ . The Pauli operators X , Y , and Z generate rotations around the corresponding axes of the sphere.

and normalized vectors that span the entire space. Any quantum state $|\psi\rangle$ in \mathcal{H} can be expressed as a linear combination of these basis vectors. For example, if $\{|b_i\rangle\}$ is an orthonormal basis ($\langle b_i | b_j \rangle = \delta_{ij}$), then $|\psi\rangle = \sum_i c_i |b_i\rangle$, where $c_i = \langle b_i | \psi \rangle$ are complex coefficients.

2.1.1 The Qubit

The fundamental unit of quantum information is the *qubit*, the quantum analogue of the classical bit. A qubit is a two-level quantum system whose state resides in a two-dimensional Hilbert space. The computational basis states for a single qubit are conventionally denoted as $|0\rangle$ and $|1\rangle$. Unlike a classical bit, which can only be in a state of 0 or 1, a qubit can exist in a superposition of these states. Formally, the state of a single qubit $|\psi\rangle \in \mathbb{C}^2$ is a linear combination of its basis states: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex probability amplitudes satisfying the normalization condition $|\alpha|^2 + |\beta|^2 = 1$.

Geometrically, the state of a single qubit can be visualized on the *Bloch sphere* (see Figure 2.1). The Bloch sphere is a unit sphere in three dimensions, where each point on the surface represents a unique pure state of a single qubit. The north pole typically corresponds to $|0\rangle$ and the south pole to $|1\rangle$. The general state $|\psi\rangle$ can be represented by angles θ and ϕ :

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \quad (2.1)$$

where $0 \leq \theta \leq \pi$ and $0 \leq \phi < 2\pi$. We act on our qubit to modify it's state, with rotation operations, more on section 2.3.1.

2.1.2 Quantum Measurement

When a physical quantity (observable) of a quantum system is measured, the act of observation fundamentally alters the state of the system. Since the outcome of any measurement must yield a real number, the associated operator is required to have a purely real spectrum. At the same time, the existence of a set of definite and mutually exclusive post-measurement states requires that the operator's eigenvectors form a complete orthonormal basis for the Hilbert space. The class of operators that satisfies these two fundamental conditions is that of Hermitian operators. Therefore, in quantum mechanics, physical observables are represented by Hermitian operators acting on the Hilbert space of the system.

When an observable \hat{O} is measured on a system in state $|\psi\rangle$, the measurement yields one of the eigenvalues of \hat{O} , say λ_m . The outcome of a quantum measurement is inherently probabilistic. The probability of obtaining a specific eigenvalue λ_m is given by the *Born's rule*, $P(\lambda_m) = \langle\psi|\lambda_m\rangle\langle\lambda_m|\psi\rangle = |\langle\psi|\lambda_m\rangle|^2$, where $|\lambda_m\rangle$ is the eigenvector corresponding to the eigenvalue λ_m . After the measurement, the system's state collapses onto the eigenspace corresponding to the observed eigenvalue. Specifically, if the measurement of observable \hat{O} yields λ_m , the post-measurement state is $|\lambda_m\rangle$.

For a single qubit, a common measurement is performed in the computational basis $\{|0\rangle, |1\rangle\}$. This corresponds to measuring the observable $\hat{\sigma}_Z = |0\rangle\langle 0| - |1\rangle\langle 1|$ (Pauli-Z operator), which has eigenvalues $+1$ corresponding to eigenstate $|0\rangle$ and -1 corresponding to eigenstate $|1\rangle$. For a qubit in a superposition state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, the probability of observing the outcome $|0\rangle$ (eigenvalue $+1$) is $P(0) = |\alpha|^2$, and the probability of observing $|1\rangle$ (eigenvalue -1) is $P(1) = |\beta|^2$. After such a measurement, the qubit's state collapses to the observed basis state (e.g., if the outcome is $|0\rangle$, the qubit's state instantly becomes $|0\rangle$).

2.2 Density matrix formalism

The statevector formalism is insufficient to describe the physically relevant scenario of the statistical mixture of states. It might be the case, we know the system to be in one of a set of states $\{|\psi_i\rangle\}$ with corresponding classical probabilities $\{p_i\}$. To address these limitations, a more general statistical tool is required: the density operator.

For a system that is in one of the states $|\psi_i\rangle$ with probability p_i , where $\sum_i p_i = 1$, the density operator is the weighted average of the projectors for each state in the ensemble:

$$\hat{\rho} = \sum_i p_i |\psi_i\rangle\langle\psi_i|$$

Properties of the Density Operator Any operator $\hat{\rho}$ representing a physical state must satisfy three fundamental properties:

1. Unit Trace: The trace of the density operator is always unity.

$$\text{Tr}(\hat{\rho}) = 1$$

This is the generalization of the normalization condition $\langle\psi|\psi\rangle = 1$ for state vectors and ensures the conservation of total probability. For a mixed state, $\text{Tr}(\hat{\rho}) = \text{Tr}(\sum_i p_i |\psi_i\rangle\langle\psi_i|) = \sum_i p_i \text{Tr}(|\psi_i\rangle\langle\psi_i|) = \sum_i p_i \langle\psi_i|\psi_i\rangle = \sum_i p_i = 1$.

2. Hermiticity: The density operator is Hermitian.

$$\hat{\rho} = \hat{\rho}^\dagger$$

This ensures that the expectation values of observables, calculated via $\hat{\rho}$, are real numbers.

3. Positive Semidefiniteness: The density operator is a positive semidefinite operator, meaning its expectation value in any state $|\phi\rangle$ is non-negative.

$$\langle\phi|\hat{\rho}|\phi\rangle \geq 0 \quad \forall |\phi\rangle \in \mathcal{H}$$

This property guarantees that all calculated probabilities are non-negative.

2.2.1 Composite Quantum Systems

When multiple quantum systems are considered together, they form a *composite quantum system*. The mathematical framework for describing such systems involves the tensor product of their individual Hilbert spaces. If we have two quantum systems, A and B , with associated Hilbert spaces \mathcal{H}_A and \mathcal{H}_B respectively, the Hilbert space for the combined system, \mathcal{H}_{AB} , is given by their tensor product:

$$\mathcal{H}_{AB} = \mathcal{H}_A \otimes \mathcal{H}_B \quad (2.2)$$

If the dimension of \mathcal{H}_A is d_A and the dimension of \mathcal{H}_B is d_B , then the dimension of the composite Hilbert space \mathcal{H}_{AB} is $d_A d_B$. This principle extends to any number of subsystems; for n qubits, each with a 2-dimensional Hilbert space, the composite system resides in a 2^n -dimensional Hilbert space.

A basis for the composite system is formed by taking the tensor product of all elements from the individual subsystems' bases. For example, if $\{|a_i\rangle\}$ is an orthonormal basis for \mathcal{H}_A and $\{|b_j\rangle\}$ is an orthonormal basis for \mathcal{H}_B , then the set $\{|a_i\rangle \otimes |b_j\rangle\}$ forms an orthonormal basis for \mathcal{H}_{AB} . For a system of two qubits, the computational basis states are $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, where $|ij\rangle$ is shorthand for $|i\rangle \otimes |j\rangle$. Any state $|\Psi\rangle$ in \mathcal{H}_{AB} can be written as a linear combination of these basis states:

$$|\Psi\rangle = \sum_{i,j} c_{ij} (|a_i\rangle \otimes |b_j\rangle) \quad (2.3)$$

where c_{ij} are complex coefficients satisfying $\sum_{i,j} |c_{ij}|^2 = 1$ so the state is properly normalized.

Let's consider a composite system \mathcal{H}_{AB} in the pure state (for simplicity, but analogous concepts are valid for mixed states) $|\Psi\rangle$ this is called a *separable state* (or product state) if it can be written as the tensor product of two states, one belonging to each subsystem:

$$|\Psi\rangle = |\psi_A\rangle \otimes |\phi_B\rangle \quad (2.4)$$

where $|\psi_A\rangle \in \mathcal{H}_A$ and $|\phi_B\rangle \in \mathcal{H}_B$. In such a state, the properties of subsystem A can be described independently of subsystem B, and vice-versa.

Conversely, a pure state $|\Psi\rangle$ is defined as entangled if it is not separable. An entangled state is necessarily a linear superposition of two or more product states

that cannot be simplified into a single product term. For an entangled state, the measurement outcomes of local observables on each system are statistically correlated.

A canonical example is the Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)$. This state is manifestly a superposition of two product states, $|00\rangle$ and $|11\rangle$, and cannot be factored into a single product $|\psi_A\rangle \otimes |\phi_B\rangle$. Consequently, if one measures the $\hat{\sigma}_Z$ operator on the first qubit and finds the outcome +1 (state $|0\rangle$), a subsequent measurement on the second qubit is guaranteed to also yield +1 (state $|0\rangle$).

2.3 Quantum Gates and Operations

Computation on a quantum system is performed by applying a sequence of operations known as *quantum gates*. These gates are the basic operations employed for transforming the qubit states and are analogous to classical logic gates (like AND, OR, NOT) in digital circuits. However, unlike classical gates, quantum gates are unitary operations.

Mathematically, a quantum gate is represented by a unitary matrix U , satisfying $U^\dagger U = UU^\dagger = I$, where U^\dagger is the conjugate transpose of U , and I is the identity matrix. Being unitary, quantum gates are inherently reversible: when a generic quantum gate U is applied to a state $|\psi\rangle$, it is always possible to restore the starting quantum state by applying its inverse, which is simply its conjugate transpose, $U^{-1} = U^\dagger$. A quantum gate acting on n qubits can be expressed as a square unitary matrix of dimension $2^n \times 2^n$.

Operations on a single qubit correspond to rotations on the Bloch sphere. Important single-qubit operations include the *Pauli operators*, which are fundamental quantum gates. These are Hermitian and unitary matrices represented as:

$$\hat{\sigma}_X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \hat{\sigma}_Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \hat{\sigma}_Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad (2.5)$$

These operators induce specific rotations on the Bloch sphere around the X, Y, and Z axes, respectively. For instance, the Pauli-X gate acts as a qbit-flip, transforming $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$.

2.3.1 Single-Qubit Gates

Single-qubit gates operate on a single qubit, transforming its state within its two-dimensional Hilbert space. Some fundamental examples include the previously mentioned Pauli operators ($\hat{\sigma}_X$, $\hat{\sigma}_Y$, $\hat{\sigma}_Z$ gates). The *Hadamard gate* (H) is another crucial single-qubit gate:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad (2.6)$$

The Hadamard gate creates a superposition state from a computational basis state: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

A generic single-qubit rotation can be represented as a rotation around an arbitrary axis $\vec{n} = (n_x, n_y, n_z)$ on the Bloch sphere by an angle θ . Such a gate is denoted $R_{\vec{n}}(\theta)$ and its matrix form is:

$$R_{\vec{n}}(\theta) = \cos\left(\frac{\theta}{2}\right) I - i \sin\left(\frac{\theta}{2}\right) (n_x \hat{\sigma}_X + n_y \hat{\sigma}_Y + n_z \hat{\sigma}_Z) \quad (2.7)$$

where $\hat{\sigma}_X$, $\hat{\sigma}_Y$, $\hat{\sigma}_Z$ are the Pauli matrices. Quantum information theory states that any arbitrary single-qubit unitary transformation can be achieved by a sequence of at most three rotation gates around different axes (e.g., $R_Z(\phi)R_X(\theta)R_Z(\lambda)$ for some angles ϕ, θ, λ).

2.3.2 Multi-Qubit Gates

Operations among different qubits are essential for creating quantum phenomena like entanglement and for performing complex computations. The most fundamental multi-qubit gate, typically involving two qubits, is the *Controlled-NOT (CNOT) gate*. The CNOT gate acts on two qubits: a control qubit and a target qubit. If the control qubit is in the $|0\rangle$ state, the target qubit remains unchanged. If the control qubit is in the $|1\rangle$ state, the Pauli-X gate (NOT operation) is applied to the target qubit.

The CNOT gate can be expressed in terms of projectors onto the control qubit's computational basis states:

$$CNOT = (|0\rangle\langle 0| \otimes I) + (|1\rangle\langle 1| \otimes X) \quad (2.8)$$

where I is the identity matrix on the target qubit, and $\hat{\sigma}_X$ is the Pauli-X gate on the target qubit. The matrix representation of the CNOT gate in computational basis (assuming the control qubit is the first qubit) is:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (2.9)$$

The CNOT gate is particularly important because, together with single-qubit gates, it forms a universal set of quantum gates, meaning any arbitrary quantum computation can be approximated to any desired accuracy using only these gates. For example, a common circuit to generate a maximally entangled Bell state, such as $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, involves applying a Hadamard gate to the first qubit followed by a CNOT gate with the first qubit as control and the second as target as in figure 2.2

2.3.3 Quantum Circuit Model

A *quantum circuit* provides a conceptual and visual model for describing a quantum computation process. It outlines a sequence of operations applied to an initial quantum state to transform it into a desired output state, from which results can be extracted. The typical flow of a quantum computation using the circuit model proceeds as follows:

1. *Initialization*: The computation begins by preparing a set of qubits in a known initial state. The most common initialization is to set all qubits to the computational basis state $|0\rangle$, i.e., $|00\dots0\rangle$. Other initial states, such as all $|1\rangle$ s, may also be used depending on the specific algorithm. This initial state represents the input to the quantum computation.
2. *Quantum Gate Operations*: Once initialized, a series of quantum gates are applied to the qubits. These gates, as discussed in the preceding sections, are unitary transformations that manipulate the amplitudes of the quantum state. They can be single-qubit gates (acting on individual qubits) or multi-qubit gates (acting on two or more qubits), depending on the desired transformation and interactions between qubits. The choice and sequence of these gates are crucial as they encode the computational logic of the quantum algorithm.

Crucially, the entire sequence of quantum gates within a circuit can be represented as a single, composite unitary transformation U_{circuit} . If the circuit consists of N gates applied sequentially, G_1, G_2, \dots, G_N , where G_1 is the first gate applied and G_N is the last, the overall unitary operation is given by the product of these individual gate unitaries, applied in reverse order (from right to left, as in matrix multiplication):

$$U_{\text{circuit}} = G_N G_{N-1} \dots G_2 G_1 \quad (2.10)$$

If the initial state of the n -qubit system is $|\psi_{\text{in}}\rangle$ (e.g., $|0\rangle^{\otimes n}$), then the state of the system after all gate operations, but before measurement, is the output state $|\psi_{\text{out}}\rangle$:

$$|\psi_{\text{out}}\rangle = U_{\text{circuit}} |\psi_{\text{in}}\rangle \quad (2.11)$$

Since each individual gate G_k is unitary, their product U_{circuit} is also unitary.

3. Measurement: After the sequence of quantum gates has been applied, in a quantum circuit it is possible to perform measurements on the evolved state. This involves measuring an observable, B , at the output of the circuit. As discussed in section 2.1.2, an observable is represented by a Hermitian operator. This observable B may act on all qubits in the circuit, or just a subset of them. For instance, B might be composed of local observables acting on individual qubits (e.g., $B = \sum_i \sigma_Z^{(i)}$ for a system of n qubits).

When the measurement of observable B is performed on the output state $|\psi_{\text{out}}\rangle$, the measurement yields one of the eigenvalues of B , and the system's state collapses accordingly. A common goal in quantum computation is to determine the expectation value of the observable B , which is given by:

$$\langle B \rangle = \langle \psi_{\text{out}} | B | \psi_{\text{out}} \rangle = \langle \psi_{\text{in}} | U_{\text{circuit}}^\dagger B U_{\text{circuit}} | \psi_{\text{in}} \rangle \quad (2.12)$$

This probabilistic outcome is the result of the quantum computation. Due to the probabilistic nature of quantum measurements, quantum algorithms often require multiple runs (*or shots*) of the circuit to estimate the probabilities of different outcomes or to obtain a high-confidence result.

Quantum circuits are typically represented through diagrams, similar to classical circuit diagrams. These diagrams use horizontal lines to represent qubits evolving over time (from left to right) and specific symbols to denote quantum gates acting

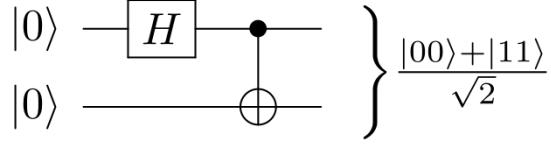


Figure 2.2: A basic quantum circuit for creating an entangled Bell state ($|\Phi^+\rangle$). A Hadamard gate (H) is applied to the first qubit, followed by a CNOT gate where the first qubit acts as the control and the second as the target.

on them. The example of the basic entangler circuit shown in Figure 2.2 illustrates this graphical representation, where the Hadamard gate and CNOT gate are applied sequentially to prepare an entangled state.

In many quantum algorithms, particularly those within the variational quantum algorithm paradigm (see next chapter), the quantum circuits are designed to be parametrized quantum circuits. This means that some of the quantum gates within the sequence are not fixed, but instead depend on a set of classical, adjustable parameters, collectively denoted by a vector $\vec{\theta} = (\theta_1, \theta_2, \dots, \theta_p)$. For instance, rotation gates like $R_X(\theta)$ or $R_Z(\phi)$ naturally incorporate such parameters. Consequently, the overall unitary transformation representing the circuit becomes a function of these parameters, $U(\vec{\theta})$

Chapter 3

Variational quantum algorithms and machine learning elements

3.1 Variational Methods

This section provides a foundational overview of variational methods and their application in quantum computing through Variational Quantum Algorithms (VQAs). VQAs, which rely on a classical optimization loop to train a parameterized quantum circuit, represent a prominent strategy for utilizing current quantum hardware. This approach is particularly well-suited to address the substantial limitations of present-day quantum devices, specifically concerning qubit count and circuit depth, that challenge the execution of purely quantum algorithms.

The review "Variational Quantum Algorithms" by Marco Cerezo et al. [13] has been an important resource for writing this summary.

3.1.1 Variational Principle

The variational principle is a fundamental concept in quantum mechanics, developed as a method for approximating the ground state energy and corresponding wavefunction of a quantum system. This approach, which minimizes the expectation value of energy with respect to parameterized trial wavefunctions, has been adapted for quantum machine learning where parameterized quantum circuits are optimized to perform computational tasks.

The core idea of the variational principle is to choose a "trial wavefunction" (also known as an *ansatz*), denoted as $|\psi(\vec{\theta})\rangle$, which depends on one or more adjustable

parameters $\vec{\theta} = (\theta_1, \theta_2, \dots, \theta_p)$. This trial wavefunction is typically designed to incorporate some physical intuition about the system's behavior. The goal is then to find the specific values of these parameters $\vec{\theta}$ for which the expectation value of the system's Hamiltonian, H , is the lowest possible. The Hamiltonian H is the operator corresponding to the total energy of the system.

According to the variational principle, for any normalizable trial wavefunction $|\psi(\vec{\theta})\rangle$, the expectation value of the Hamiltonian, $\langle H \rangle_{\vec{\theta}}$, will always be greater than or equal to the true ground state energy E_0 of the system:

$$\langle H \rangle_{\vec{\theta}} = \frac{\langle \psi(\vec{\theta}) | H | \psi(\vec{\theta}) \rangle}{\langle \psi(\vec{\theta}) | \psi(\vec{\theta}) \rangle} \geq E_0 \quad (3.1)$$

The process involves calculating this expectation value as a function of the parameters $\vec{\theta}$ and then minimizing it with respect to these parameters:

$$E_{\text{approx}} = \min_{\vec{\theta}} \left(\frac{\langle \psi(\vec{\theta}) | H | \psi(\vec{\theta}) \rangle}{\langle \psi(\vec{\theta}) | \psi(\vec{\theta}) \rangle} \right) \quad (3.2)$$

The wavefunction obtained by fixing the parameters to these optimal values ($\vec{\theta}_{\text{opt}}$) is then an approximation to the ground state wavefunction. The corresponding minimum expectation value of the energy, E_{approx} , serves as an upper bound to the true ground state energy E_0 . The closer the chosen ansatz is to the true ground state, the tighter this upper bound will be.

This principle forms the basis for a class of quantum algorithms, where the "trial wavefunction" is realized by a parametrized quantum circuit and the "optimization" is performed classically. The objective is to find the optimal parameters for the quantum circuit that minimize a specific cost function.

3.1.2 Variational quantum algorithms

These algorithms represent a hybrid approach, combining the computational power of quantum processors with the optimization capabilities of classical computers. VQAs are built upon the variational principle, extending its application from purely theoretical wavefunction approximations to practical algorithmic frameworks on quantum hardware.

In a VQA, the core idea is to encode a problem into a parametrized quantum circuit. The VQA process is typically iterative: a quantum device prepares a quantum state according to the circuit's current parameters and performs measurements.

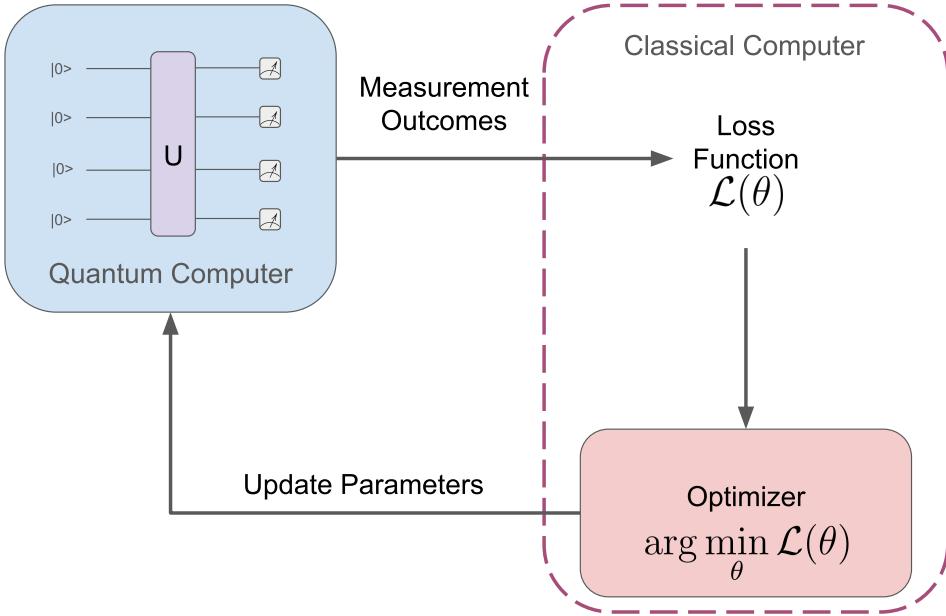


Figure 3.1: Schematic of a Variational Quantum Algorithm (VQA), showing the hybrid quantum-classical optimization loop, to go on until $\mathcal{L}(\theta)$ is lower than a chosen value or the number of maxim cycles (also said *epochs*) is met.

The results of these measurements are then fed to a classical optimizer, which updates the parameters to minimize a predefined cost function. This updated set of parameters is then sent back to the quantum device for the next iteration, forming a feedback loop that continues until a satisfactory solution is found.

Cost Function

A crucial element of any VQA is the *cost function* (or loss function), which quantifies the performance of the parametrized quantum circuit for a given set of parameters $\vec{\theta}$. Analogous to cost functions in classical machine learning, it maps the values of the trainable parameters $\vec{\theta}$ to a single real number. The objective of the classical optimizer is to minimize this cost function.

Abstractly, the cost function defines a multi-dimensional surface known as the *cost landscape*. The optimization task involves navigating this landscape to locate a (ideally global) minimum, corresponding to the optimal set of parameters for the problem at hand.

A VQA typically begins by initializing a system of qubits into a known pure

input state, $|\psi_{\text{in}}\rangle$ (e.g., $|0\rangle^{\otimes n}$). A parametrized quantum circuit, represented by a unitary operator $\hat{U}(\vec{\theta})$, is then applied to this initial state, yielding an output state:

$$|\psi_{\text{out}}(\vec{\theta})\rangle = \hat{U}(\vec{\theta})|\psi_{\text{in}}\rangle \quad (3.3)$$

The cost function $C(\vec{\theta})$ is then formulated by performing a measurement of some problem specific observable on this output state. The cost function is generally a real-valued function f of the expectation value of this observable:

$$\mathcal{L}(\vec{\theta}) = f\left(\langle\psi_{\text{out}}(\vec{\theta})|\hat{O}|\psi_{\text{out}}(\vec{\theta})\rangle\right) \quad (3.4)$$

Substituting the expression for $|\psi_{\text{out}}(\vec{\theta})\rangle$, this becomes:

$$\mathcal{L}(\vec{\theta}) = f\left(\langle\psi_{\text{in}}|\hat{U}^\dagger(\vec{\theta})\hat{O}\hat{U}(\vec{\theta})|\psi_{\text{in}}\rangle\right) \quad (3.5)$$

The specific choice of the function f , the initial state $|\psi_{\text{in}}\rangle$, and the observable \hat{O} is entirely determined by the particular problem the VQA aims to solve. Due to the probabilistic nature of quantum measurements, the expectation value $\langle\psi_{\text{out}}(\vec{\theta})|\hat{O}|\psi_{\text{out}}(\vec{\theta})\rangle$ is typically estimated by running the quantum circuit multiple times (known as "shots") and averaging the measured outcomes.

Ansatz

In this context, the ansatz refers to the specific structure and form of the parametrized quantum circuit $\hat{U}(\vec{\theta})$. It dictates how the trainable parameters $\vec{\theta}$ are incorporated into the quantum gates, and thus, how they influence the transformation of the quantum state. The choice of ansatz effectively defines the subspace of the Hilbert space that the VQA can explore during the optimization process.

Generically speaking, the form of the ansatz determines the nature and number of the parameters $\vec{\theta}$ and, consequently, how they can be trained to minimize the cost function. An ansatz is typically composed of a sequence of quantum gates, some of which are parametrized (e.g., rotation gates with angles as parameters), and others are fixed (e.g., entangling gates like CNOT). The specific structure of an ansatz is often designed based on the problem's characteristics, balancing factors such as its expressibility (the ability to generate a wide range of quantum states) and its hardware efficiency (how easily it can be implemented on a given quantum device).

Gradients

The objective of a VQA is to train the parameters $\vec{\theta}$ of the quantum circuit to solve a specific optimization problem, typically by minimizing the cost function $C(\vec{\theta})$. It is known that for many optimization tasks, using information from the cost function's gradient makes methods effective in speeding up and guaranteeing the convergence of the optimizer. One of the main advantages of many VQAs is that, as discussed below, one can evaluate the cost function gradient directly on quantum hardware.

The most common technique for evaluating these analytical gradients on a quantum computer is known as the *parameter-shift rule*. This rule provides a hardware-friendly protocol to compute the partial derivative of the cost function with respect to a specific parameter. Consider a cost function of the form $C(\vec{\theta}) = f(\langle \hat{O} \rangle_{\vec{\theta}})$, where $\langle \hat{O} \rangle_{\vec{\theta}} = \langle \psi_{\text{in}} | \hat{U}^\dagger(\vec{\theta}) \hat{O} \hat{U}(\vec{\theta}) | \psi_{\text{in}} \rangle$, and where the parameter θ_l is part of a rotation gate $e^{i\theta_l A}$ (e.g., a Pauli rotation) within the unitary $\hat{U}(\vec{\theta})$.

The parameter-shift rule states [14] that the partial derivative of the expectation value $\langle \hat{O} \rangle_{\vec{\theta}}$ with respect to θ_l can be obtained by evaluating the quantum circuit twice with slightly shifted parameter values:

$$\frac{\partial \langle \hat{O} \rangle_{\vec{\theta}}}{\partial \theta_l} = \frac{1}{2 \sin(\alpha)} \left(\langle \psi_{\text{in}} | \hat{U}^\dagger(\vec{\theta}_+) \hat{O} \hat{U}(\vec{\theta}_+) | \psi_{\text{in}} \rangle - \langle \psi_{\text{in}} | \hat{U}^\dagger(\vec{\theta}_-) \hat{O} \hat{U}(\vec{\theta}_-) | \psi_{\text{in}} \rangle \right) \quad (3.6)$$

Here, $\vec{\theta}_+ = \vec{\theta} + \alpha \vec{e}_l$ and $\vec{\theta}_- = \vec{\theta} - \alpha \vec{e}_l$, where \vec{e}_l is a vector with a 1 at the l -th position and 0s elsewhere, and α is a small real number. This method yields the analytic gradient of the parameter by virtue of the coefficient $1/\sin \alpha$. For more complex cost functions where f is not the identity, the chain rule can be applied to extend this result:

$$\frac{\partial \mathcal{L}(\vec{\theta})}{\partial \theta_l} = \frac{\partial f}{\partial \langle \hat{O} \rangle_{\vec{\theta}}} \frac{\partial \langle \hat{O} \rangle_{\vec{\theta}}}{\partial \theta_l} \quad (3.7)$$

3.2 Barren Plateaus and Loss Concentration

Barren plateaus and exponential loss concentration have emerged as fundamental obstacles to the scalability and trainability of parameterized quantum circuits, where they render optimization increasingly difficult as system size grows. These

phenomena occur when the loss landscape becomes exponentially flat, causing gradients to vanish and preventing algorithms from efficiently navigating towards optimal solutions. Tensor networks offer a classical framework for representing and manipulating quantum states and operations, providing an alternative approach as quantum-inspired machine learning without requiring quantum hardware. However, the mathematical relationships between quantum circuits and their tensor network counterparts suggest that similar optimization pathologies may persist in the classical setting [11].

A barren plateau is characterized by an exponential decay in the variance of the gradient of the loss function as the system size n increases, making optimization extremely challenging. Specifically, the variance scales as:

$$\text{Var}_{\boldsymbol{\theta}} [\partial_{\theta_k} \mathcal{L}(\boldsymbol{\theta})] \in \mathcal{O}\left(\frac{1}{\beta^n}\right), \quad \beta > 1, \quad (3.8)$$

where n is the number of qubits. As a result, the gradients become so small that they are indistinguishable from statistical noise, making it practically impossible for gradient-based optimizers to make progress.

Closely related to the vanishing of gradients is the phenomenon of exponential loss concentration. In this case, the value of the loss function itself becomes highly concentrated around a fixed value for almost all choices of parameters, rendering the optimization landscape nearly flat.

Definition 1 (Exponential concentration). *Let $X(\alpha)$ be a quantity that depends on a set of parameters α and is measured as the expectation value of an observable on a quantum computer. We say that $X(\alpha)$ is deterministically exponentially concentrated in the number of qubits n towards a fixed value μ if*

$$|X(\alpha) - \mu| \leq \beta \in \mathcal{O}(1/b^n) \quad (3.9)$$

for some $b > 1$ and all α . Alternatively, $X(\alpha)$ is probabilistically exponentially concentrated if

$$\Pr_{\alpha} [|X(\alpha) - \mu| \geq \delta] \leq \frac{\beta}{\delta^2}, \quad \beta \in \mathcal{O}(1/b^n), \quad (3.10)$$

for any $b > 1$. In other words, the probability that $X(\alpha)$ deviates from μ by more than a small amount δ becomes exponentially small as the system size increases.

The origins of the barren plateau phenomenon are closely tied to the high-dimensional nature of quantum state spaces and the expressiveness of parameterized

quantum circuits. The concept was first introduced in previous works that demonstrated that for sufficiently expressive quantum circuits, the gradients of the loss function vanish exponentially with the number of qubits [15, 16, 17, 18, 19]. Although this is a completely classical model, analogous phenomenon emerge in tensor network as well. [11, 20, 21, 22].

3.3 Optimization in Machine Learning

Machine learning models are trained to perform specific tasks, such as prediction, classification, or data generation, by learning patterns from data. This learning process is fundamentally an optimization problem, where the goal is to adjust the model’s internal parameters to minimize a predefined measure of error or maximize a desired objective.

3.3.1 Loss Functions

At the core of any machine learning task is the loss function (also known as the cost function or objective function). This function quantifies the discrepancy between the model’s predictions and the true values (in supervised learning), or more generally, measures how well the model achieves its desired objective for a given set of parameters. The ultimate goal of training is to find the set of model parameters that minimizes this loss.

The choice of loss function is critical and depends entirely on the nature of the problem being solved. Two notable examples are discussed in the following, Kullback-Leibler divergence and Maximum Mean Discrepancy, whose proprieties are an important part of this thesis work.

Maximum Likelihood Estimation

For models that explicitly define a probability distribution over the data, a principled approach to deriving a loss function is through *Maximum Likelihood Estimation* (MLE). Given a dataset $\mathcal{D} = \{x^{(i)}\}_{i=1}^N$ and a model $p_\theta(x)$ parameterized by θ , the likelihood function is defined as the joint probability of observing all data points:

$$\mathcal{L}(\theta) = \prod_{i=1}^N p_\theta(x^{(i)}) \quad (3.11)$$

The MLE objective is to find the parameters θ^* that maximize this likelihood. Equivalently, it is often more convenient to minimize the negative log-likelihood:

$$\theta^* = \arg \max_{\theta} \left(\prod_{i=1}^N p_\theta(x^{(i)}) \right) = \arg \min_{\theta} \left(- \sum_{i=1}^N \log p_\theta(x^{(i)}) \right) \quad (3.12)$$

and this is what we are gonna do in practice throughout this work.

Minimizing the negative log-likelihood (NLL) can also be interpreted as minimizing the Kullback-Leibler divergence between the empirical data distribution \hat{p}_{data} and the model distribution p_θ , $D_{\text{KL}}(\hat{p}_{\text{data}} \| p_\theta)$ (also denoted here KLD). To see this, recall the definition of KLD:

$$\begin{aligned} D_{\text{KL}}(\hat{p}_{\text{data}} \| p_\theta) &= \sum_x \hat{p}_{\text{data}}(x) \log \left(\frac{\hat{p}_{\text{data}}(x)}{p_\theta(x)} \right) = \\ &= \sum_x \hat{p}_{\text{data}}(x) \log \hat{p}_{\text{data}}(x) - \sum_x \hat{p}_{\text{data}}(x) \log p_\theta(x) \end{aligned} \quad (3.13)$$

The first term, $\sum_x \hat{p}_{\text{data}}(x) \log \hat{p}_{\text{data}}(x) = H(\hat{p}_{\text{data}})$, represents the entropy of the empirical data distribution. This term is constant with respect to the model parameters θ , as it only depends on the observed data. Therefore, minimizing $D_{\text{KL}}(\hat{p}_{\text{data}} \| p_\theta)$ is equivalent to maximizing the second term, $\sum_x \hat{p}_{\text{data}}(x) \log p_\theta(x)$. Since $\hat{p}_{\text{data}}(x) = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(x = x^{(i)})$ (where \mathbb{I} is the indicator function, which is 1 if x is the same as the i -th data point $x^{(i)}$ and 0 otherwise), this sum becomes $\frac{1}{N} \sum_{i=1}^N \log p_\theta(x^{(i)})$.

Maximum Mean Discrepancy

Maximum Mean Discrepancy (MMD) aim to make the distribution of generated samples statistically indistinguishable from the real data distribution, without requiring an explicit probability density function from the model. MMD measures the distance between the mean embeddings of two distributions in a Reproducing Kernel Hilbert Space [23, 24], effectively comparing the "statistical properties" or "feature representations" of the distributions. The choice of kernel dictates which

aspects of the distributions are compared. The expression for MMD if given by:

$$\mathcal{L}_{mmd}(\boldsymbol{\theta}) = \left\| \sum_x p_{\boldsymbol{\theta}}(x) \phi(x) - \sum_x \pi(x) \phi(x) \right\|_{\ell_2}^2 \quad (3.14)$$

where $\phi(x)$ maps x to a larger feature space and π is the probably distribution of data. Using a kernel $K(x, y) = \phi(x)^T \phi(y)$ allows us to work in a lower-dimensional space. We use the Radial basis function (RBF) kernel for this purpose, which is defined as:

$$K(x, y) = \frac{1}{c} \sum_{i=1}^c \exp\left(\frac{|x - y|^2}{2\sigma_i^2}\right) \quad (3.15)$$

Here, σ_i is the bandwidth parameter controlling the width of the Gaussian kernel. \mathcal{L} approaches to zero if and only if $p_{\boldsymbol{\theta}}$ approaches π . We can now write the loss function in terms of $K(x, y)$ as

$$\mathcal{L}_{mmd} = \mathbb{E}_{x, y \sim p_{\boldsymbol{\theta}}} [K(x, y)] - 2 \mathbb{E}_{x \sim p_{\boldsymbol{\theta}}, y \sim \pi} [K(x, y)] + \mathbb{E}_{x, y \sim \pi} [K(x, y)] \quad (3.16)$$

Since it doesn't require to reconstruct any probability distribution to be calculated, it is often referred to as "implicit" loss function, in contrast with KLD, "explicit" loss function. In the next section will be presented an overview on this topic.

3.3.2 Explicit and Implicit Models

An important yet subtle distinction in generative modeling is between explicit and implicit models [7]. Explicit generative models allow for efficient access to the model probability distribution $Q_{\boldsymbol{\theta}}(x)$. Here, "efficient" means that the probability values can be computed in polynomial time and memory with respect to the size of the data. To train explicit model the most natural choice is using a metric that compare the probability distribution of the model to a target one. One common example of what we can call *explicit loss* is the Kullback-Leibler divergence (KLD):

$$D_{KL}(P \| Q) = \sum_x P(x) \log \frac{P(x)}{Q_{\boldsymbol{\theta}}(x)}, \quad (3.17)$$

where $P(x)$ is the target distribution, and $Q_{\boldsymbol{\theta}}(x)$ is the model's distribution parameterized by $\boldsymbol{\theta}$.

In contrast, implicit generative models do not provide direct access to $Q_{\boldsymbol{\theta}}(x)$ but instead allow for efficient sampling from the model distribution. Training is done

by comparing the model's samples to the data distribution through a distance measure, losses that act such are often referred as *implicit loss*. One widely used statistical measure for training implicit models is the Maximum Mean Discrepancy, introduced in section 3.3.1

$$\begin{aligned}\mathcal{L}_{mmd}(P, Q_\theta) &= \mathbb{E}_{x, x' \sim P}[k(x, x')] - 2\mathbb{E}_{x \sim P, y \sim Q_\theta}[k(x, y)] + \mathbb{E}_{y, y' \sim Q_\theta}[k(y, y')] = \\ &= \sum_{x \in \Omega} \sum_{y \in \Omega} q_\theta(x)K(x, y)q_\theta(y) + \sum_{x \in T} \sum_{y \in T} p(x)p(y)K(x, y) - \\ &\quad - 2 \sum_{x \in \Omega} \sum_{y \in T} q_\theta(x)K(x, y)p(y)\end{aligned}\tag{3.18}$$

where p is the target distribution, q_θ is the model's distribution, $k(x, y)$ is a positive-definite kernel function. Samples are drawn from probability distributions and compared.

Generative Adversarial Networks [25, 26] and Quantum Circuit Born Machine [27] treated in section 3.4 are prominent examples of implicit models.

3.3.3 Gradient-Based Optimization

The process of finding the optimal parameters θ that minimize the chosen loss function $C(\theta)$ is performed using optimization algorithms. In this work we will focus on gradient-based methods.

Gradient Descent

The fundamental principle of gradient descent is to iteratively update the parameters in the direction that maximally decreases the loss function. This direction is given by the negative of the gradient of the loss function with respect to the parameters. Starting from an initial set of parameters θ_0 , the parameters are updated in steps:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta C(\theta_t)\tag{3.19}$$

where $\eta > 0$ is the learning rate, a hyperparameter that controls the size of each step, and t indexes the optimization iteration.

The exact computation of the full gradient $\nabla_\theta C(\theta)$ across the entire dataset at each iteration of the optimization process can be computationally prohibitive,

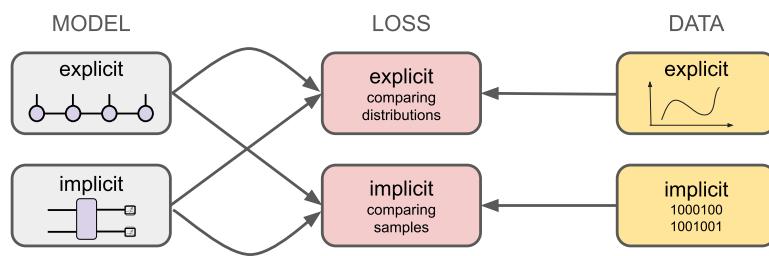


Figure 3.2: The diagram illustrates the fundamental distinction between explicit and implicit approaches in generative modeling, where explicit models (up left - represented with the example of a tensor network) enable direct computation of probability distributions and utilize loss functions that compare distributions directly, while implicit models (bottom left represented with the example of a quantum circuit) generate samples without explicit probability access and employ loss functions that compare sample sets. Cross-combinations are possible: implicit models can be used with explicit losses by sampling to approximate the distribution, while explicit models can be used with implicit losses by directly sampling from their accessible distributions.

severely hindering scalability. To mitigate this issue, several powerful alternatives have been developed, primarily relying on approximations of the gradient calculation.

One widely adopted and highly effective approach is Stochastic Gradient Descent. Rather than performing a costly evaluation over the entire dataset, SGD approximates the gradient by utilizing a randomly selected subset of data points, commonly referred to as a *mini-batch*. The parameter update rule is consequently modified to:

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} C_{\mathcal{B}}(\theta_t)$$

where $C_{\mathcal{B}}(\theta)$ denotes the loss function evaluated solely on the chosen mini-batch \mathcal{B} , and η is the learning rate. This inherent stochasticity not only drastically reduces the computational overhead per iteration but also offers auxiliary benefits. It can introduce a beneficial noise that aids in escaping shallow local minima and navigating the often complex, non-convex loss landscapes characteristic of high-dimensional parameter spaces in quantum machine learning.

Numerous sophisticated variants of SGD have been developed to improve convergence speed and stability, such as Adam, RMSProp, and momentum-based optimizers. These methods adapt the learning rate or incorporate information from past gradients to make more effective updates.

Back-Propagation and Automatic Differentiation

For models that are composed of multiple interconnected layers, such as neural networks, efficiently computing the gradients of the loss function with respect to all parameters is crucial. This is achieved through the *back-propagation algorithm*. Back-propagation is an efficient application of the chain rule of calculus to compute derivatives. It propagates the error signal backward through the network's layers, allowing the calculation of the gradient for each parameter with respect to the overall loss.

If a model acts as a composition of functions $F(x; \theta) = f_L \circ f_{L-1} \circ \dots \circ f_1(x; \theta_1, \dots, \theta_L)$, where each f_l is a differentiable layer with parameters θ_l , and the loss \mathcal{L} is a function of the model output, then the gradient with respect to θ_l can be computed recursively:

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \left(\frac{\partial \mathcal{L}}{\partial f_L} \right) \left(\frac{\partial f_L}{\partial f_{L-1}} \right) \dots \left(\frac{\partial f_{l+1}}{\partial f_l} \right) \left(\frac{\partial f_l}{\partial \theta_l} \right) \quad (3.20)$$

Modern machine learning frameworks implement back-propagation efficiently via *automatic differentiation*, enabling the training of highly complex models without manual gradient computation.

Training Procedure The typical training loop for a machine learning model, particularly for iterative gradient-based optimization, proceeds as follows:

1. *Initialization*: Model parameters θ are initialized, often randomly, though specific heuristics (e.g., Xavier or He initialization) can improve initial performance.
2. *Mini-batch Sampling*: In each iteration (or training step), a mini-batch \mathcal{B} of data is randomly sampled from the full dataset.
3. *Forward Pass*: The model processes the input data from the mini-batch to compute its outputs, and the loss function $C(\theta)$ is evaluated based on these outputs and the true targets.
4. *Backward Pass*: Gradients of the loss with respect to all trainable parameters are computed using the back-propagation algorithm.
5. *Parameter Update*: The parameters θ are updated using the chosen optimization algorithm (e.g., SGD, Adam), taking a step in the direction opposite to the gradient.
6. *Convergence Check*: The entire process (steps 2-5) repeats for a specified number of epochs (passes over the entire dataset) or until a stopping criterion is met, such as the loss converging or reaching a target performance on a validation set.

Despite potential challenges like local minima or saddle points in the loss landscape, gradient-based optimization remains the most effective and widely used approach for training complex machine learning models due to its scalability and compatibility with automatic differentiation.

3.4 Quantum Circuit Born Machines

Quantum Circuit Born Machines (QCBMs) are a class of unsupervised generative models that leverage the principles of quantum mechanics to learn and represent

probability distributions from classical datasets. These models fall under the umbrella of Variational Quantum Algorithms (VQAs), utilizing a parametrized quantum circuit to encode the candidate probability distribution in a parametric quantum states. Born machines are considered, in principle, more efficient than classical methods for represent certain classes of distributions.[27]

Given a classical dataset $D = \{x^{(i)}\}_{i=1}^N$ consisting of samples drawn from an unknown target probability distribution $\pi(x)$, a QCBM aims to learn an approximation of this distribution. This is achieved by constructing a parametrized quantum circuit $\hat{U}(\vec{\theta})$ that generates a quantum state $|\psi_{\theta}\rangle$ from an initial state $|\psi_{\text{in}}\rangle$ (typically $|0\rangle^{\otimes n}$ for an n -qubit system):

$$|\psi_{\theta}\rangle = \hat{U}(\vec{\theta}) |\psi_{\text{in}}\rangle \quad (3.21)$$

The parameters $\vec{\theta}$ are classical, real-valued variables that determine the specific unitary operations within the circuit. The probability distribution $p_{\theta}(x)$ that the QCBM implicitly learns is obtained by performing a measurement in the computational basis on the generated quantum state $|\psi_{\theta}\rangle$. According to the Born rule, the probability of observing a specific bitstring x (corresponding to the computational basis state $|x\rangle$) is given by:

$$p_{\theta}(x) = |\langle x | \psi_{\theta}\rangle|^2 \quad (3.22)$$

Here, x represents a classical bitstring, and $|x\rangle$ is the corresponding computational basis state.

The primary objective of training a QCBM is to align the model's generated probability distribution $p_{\theta}(x)$ as closely as possible with the target data distribution $\pi(x)$ (or its empirical approximation $\hat{p}_{\text{data}}(x)$ derived from the dataset D). This alignment is typically achieved by minimizing a suitable statistical distance or divergence between the two distributions, which serves as the cost function $C(\vec{\theta})$ for the VQA. As discussed in the section 3.3 common choices for this cost function include the Kullback-Leibler divergence, or Maximum Mean Discrepancy.

The optimization process for a QCBM follows the hybrid classical-quantum loop characteristic of VQAs. The quantum computer executes the parametrized circuit $\hat{U}(\vec{\theta})$ and measures the resulting probabilities $p_{\theta}(x)$ (by accumulating measurement shot statistics). These probabilities are then passed to a classical computer, which calculates the value of the cost function and its gradients with respect to $\vec{\theta}$. A classical optimizer then updates the parameters $\vec{\theta}$ to reduce the cost function, and

CHAPTER 3. VARIATIONAL QUANTUM ALGORITHMS AND MACHINE LEARNING ELEMENTS

the refined parameters are fed back to the quantum circuit for the next iteration, iteratively improving the learned distribution. An example in Fig. The optimization process for a QCBM follows the hybrid classical-quantum loop characteristic of VQAs. The quantum computer executes the parametrized circuit $\hat{U}(\vec{\theta})$ and measures the resulting probabilities $p_{\theta}(x)$ (by accumulating measurement shot statistics). These probabilities are then passed to a classical computer, which calculates the value of the cost function and its gradients with respect to $\vec{\theta}$. A classical optimizer then updates the parameters $\vec{\theta}$ to reduce the cost function, and the refined parameters are fed back to the quantum circuit for the next iteration, iteratively improving the learned distribution. An example in Fig. The optimization process for a QCBM follows the hybrid classical-quantum loop characteristic of VQAs. The quantum computer executes the parametrized circuit $\hat{U}(\vec{\theta})$ and measures the resulting probabilities $p_{\theta}(x)$ (by accumulating measurement shot statistics). These probabilities are then passed to a classical computer, which calculates the value of the cost function and its gradients with respect to $\vec{\theta}$. A classical optimizer then updates the parameters $\vec{\theta}$ to reduce the cost function, and the refined parameters are fed back to the quantum circuit for the next iteration, iteratively improving the learned distribution. An example in Fig. The optimization process for a QCBM follows the hybrid classical-quantum loop characteristic of VQAs. The quantum computer executes the parametrized circuit $\hat{U}(\vec{\theta})$ and measures the resulting probabilities $p_{\theta}(x)$ (by accumulating measurement shot statistics). These probabilities are then passed to a classical computer, which calculates the value of the cost function and its gradients with respect to $\vec{\theta}$. A classical optimizer then updates the parameters $\vec{\theta}$ to reduce the cost function, and the refined parameters are fed back to the quantum circuit for the next iteration, iteratively improving the learned distribution. An example in Fig. 3.3.

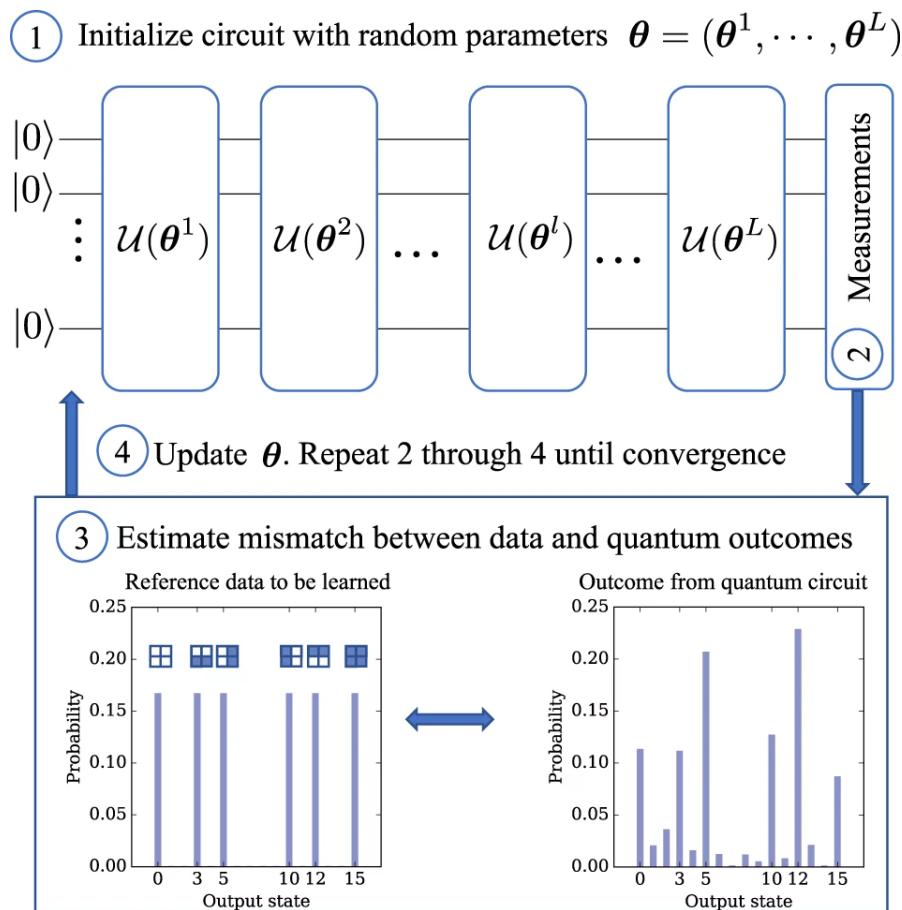


Figure 3.3: Schematics of QCBM workflow. Image taken from [28]

Chapter 4

Tensor Network Elements

4.1 Introduction

This chapter is primarily based on the textbook *Tensor Network Techniques for Quantum Computation* by Collura, Lami, Ranabhat, Santini [29], which offers a thorough treatment of Tensor Network.

Tensor networks have become a central tool in modern physics, mathematics, and data science, providing a powerful language for representing and manipulating high-dimensional data. They offer a compact and structured way to encode complex relationships, especially in quantum many-body systems, where the exponential growth of Hilbert space makes other simulation methods hard.

This introduction covers the essential concepts of tensors, tensor reshaping, index fusion and splitting, tensor networks, key decompositions, Matrix Product States *MPS*, Matrix Product Operators *MPO*.

Tensor Let V be a complex vector space of dimension d , i.e., $V \cong \mathbb{C}^d$. The dual space V^* is defined as the set of all linear maps $\phi : V \rightarrow \mathbb{C}$, and also has dimension d . Elements of V^* are called covectors. A tensor of type p, q is a multi-linear map

$$T : \underbrace{V^* \times \cdots \times V^*}_p \times \underbrace{V \times \cdots \times V}_q \rightarrow \mathbb{C}. \quad (4.1)$$

Thus, a tensor T assigns a complex number to any collection of p covectors w_1, \dots, w_p and q vectors v_1, \dots, v_q , denoted as $T_{(w_1, \dots, w_p, v_1, \dots, v_q)}$.

The number of indices possessed by a tensor, given by $p + q$, is called the rank of the tensor. Each index j_i runs over $1, \dots, d_i$, where d_i is the dimension of the

corresponding space. The total number of components is $\prod_{i=1}^k d_i$.

Reshaping Tensors can be “reshaped” by modifying their rank, which involves either fusing existing indices or splitting a single index into multiple.

Reshaping a tensor into one of lower rank, say from k to $k' < k$, is performed through *index fusion*. This operation groups a set of existing indices into a single new index. For example, given a tensor $T_{i_1, i_2, \dots, i_m, \dots}$, indices i_1, i_2, \dots, i_m can be fused into a new composite index $J = (i_1, i_2, \dots, i_m)$, resulting in a tensor of lower rank, $T_{J, \dots}$. The new composite index J effectively labels elements in the tensor product of the Hilbert (or vector) spaces corresponding to the original indices. For instance, if index i_1 belongs to space V_{i_1} and i_2 to V_{i_2} , then the fused index J effectively refers to the composite space $V_{i_1} \otimes V_{i_2}$. This process of index fusion is a unique operation: given a set of indices to be fused, there is only one way to define the new composite index and its associated Hilbert space.

Conversely, reshaping a tensor into one of higher rank, say from k to $k' > k$, involves *index splitting* (or decomposition). Here, a single index is replaced by multiple new indices. For example, an index I of a tensor T_I can be split into indices i_1 and i_2 , transforming the tensor into $T_{i_1 i_2}$. However, unlike index fusion, this operation is not unique. While the original index I labels states in a single Hilbert space V_I , this space can be viewed as the tensor product of different sets of subspaces in multiple ways, i. e., $V_I \cong V_{i_1} \otimes V_{i_2}$ for various choices of V_{i_1} and V_{i_2} . Consequently, there are many ways to define the components of the new, split indices in terms of the original single index’s components, making the decomposition non-unique without additional constraints.

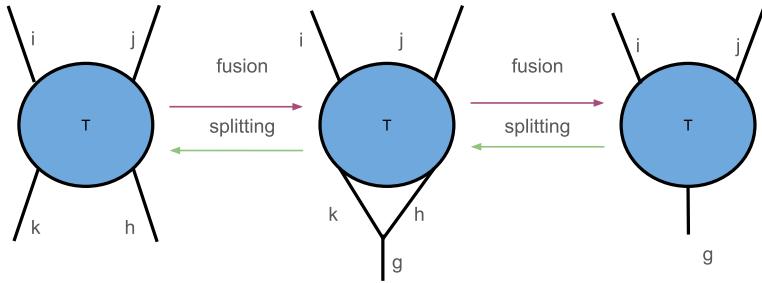


Figure 4.1: Index fusing $\text{rank} - 4 : T_{kh}^{ij} \rightarrow \text{rank} - 3 : T_g^{ij}$ and splitting $\text{rank} - 3 - > \text{rank} - 4$ vice versa.

4.2 Tensor Networks

A *tensor network* is a structured collection of tensors where a specific subset of their indices, known as *bond indices*, are contracted (summed over) according to a predefined pattern. This complex arrangement of tensors can be intuitively represented graphically, where each node symbolizes a tensor and the edges connecting them represent the contracted bond indices. Indices that are not contracted are called *open* or *free indices*, and these define the indices of the resulting tensor after all contractions are performed. An example in Fig. 4.2.

Contraction Contraction is a fundamental operation in tensor calculus that generalizes matrix multiplication to arbitrary-rank tensors. When two or more tensors share one or more indices, these can be contracted. This involves summing over all possible values that these shared indices can take, which corresponds to summing over the product of the Hilbert spaces related to those shared indices.

Mathematically, if two tensors, say A and B , have a shared index j , their contraction over j results in a new tensor C . The rank of the resulting tensor is given by the sum of the ranks of the original tensors minus two times the number of contracted indices. For example, contracting $A_{i_1, \dots, i_p, j}$ and B_{j, k_1, \dots, k_q} over the shared index j yields:

$$C_{i_1, \dots, i_p, k_1, \dots, k_q} = \sum_j A_{i_1, \dots, i_p, j} B_{j, k_1, \dots, k_q} \quad (4.2)$$

$$i \xrightarrow{M} j \xrightarrow{N} k = \sum_{j=0}^D M_{ij} N_{jkl}$$

Figure 4.2: The graphical diagrammatic notation a tensor network and the equivalent expression. i, k, l are open indices, j is a bond index with bond dimension $D + 1$. Performing the contraction simply means executing the sum operation. Image taken from reference [30]

The familiar matrix product $C_{ik} = \sum_j A_{ij}B_{jk}$ is a special case of tensor contraction where two rank-2 tensors *matrices* are contracted over one shared index, resulting in a rank-2 tensor.

4.3 Singular Value Decomposition

The Singular Value Decomposition *SVD* is a fundamental matrix factorization technique that states that *every* matrix can be uniquely expressed as a product of three distinct components. Given an $m \times n$ complex matrix M , its SVD is defined as:

$$M_{\alpha\beta} = U_{\alpha\gamma} S_{\gamma,\gamma} V_{\gamma\beta}^\dagger \quad (4.3)$$

With U is a $m \times m$ unitary matrix, whose columns form an orthonormal basis (i.e., $U^\dagger U = I_m$). This property is often referred to as U being “left-normalized.” V is an $n \times n$ unitary matrix, whose columns also form an orthonormal basis, thus called “right-normalized.” The matrix S is an $m \times n$ diagonal matrix, whose non-negative real entries on the main diagonal, $s_1 \geq s_2 \geq \dots \geq s_{\min(m,n)} \geq 0$, are called the *singular values* of M .

4.4 Matrix Product States

Matrix Product States MPS offer an efficient way to represent complex quantum many-body states, which are typically described by a high-dimensional tensor of coefficients ψ_{x_1, \dots, x_n} . The core idea behind MPS relies on the *Schmidt decomposition*,

CHAPTER 4. TENSOR NETWORK ELEMENTS

a powerful mathematical tool for understanding and simplifying quantum states, particularly those divided into two subsystems (bipartite systems).

Consider a quantum state $|\psi\rangle$ residing in the tensor product of two Hilbert spaces, $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$, with basis respectively: $\{|a_j\rangle\}$ and $\{|b_i\rangle\}$. The state can be written as:

$$|\psi\rangle = T_{ij} |a_i\rangle |b_j\rangle = \sum_{k=1}^{\chi} \Lambda_{kk} |\varphi_k^a\rangle |\varphi_k^b\rangle \quad (4.4)$$

Where in the last equivalence an SVD of the matrix T has been performed: $\Lambda = UTV^\dagger$ and new vectors $|\varphi_k^a\rangle = \sum_i U_{ik} |a_i\rangle$ and $|\varphi_k^b\rangle = \sum_j V_{kj}^* |b_j\rangle$ have been defined, called orthogonal [29] *Schmidt vectors* for subsystems A and B respectively, and Λ_{kk} are real, non-negative *Schmidt values*. The number of non-zero Schmidt values is often dubbed *Schmidt rank*.

The maximum size of the state after the Schmidt decomposition is $\chi = \min(\dim(A), \dim(B))$. This offers a systematic way to reduce computational complexity while retaining essential correlations.

The Schmidt decomposition allows for truncation or compression. If χ is too large, we can create an approximate state $|\tilde{\psi}\rangle$ by keeping only the $\tilde{\chi} < \chi$ largest Schmidt values:

$$|\tilde{\psi}\rangle = \sum_{k=1}^{\tilde{\chi}} \tilde{\Lambda}_{kk} |\varphi_k^a\rangle |\varphi_k^b\rangle \quad (4.5)$$

where $\tilde{\Lambda}_{kk} \equiv \Lambda_{kk}/(\sum_{k=1}^{\tilde{\chi}} \Lambda_{kk}^2)^{1/2}$ ensures normalization. Of course. Here is a more explicit version of the text that details the decomposition process and introduces the concept of bond dimension.

The transformation of a general quantum state into a MPS is achieved through a sequential application of the Schmidt decomposition. Let's start with the general state of an N -qubit system, described by a rank- N tensor T with 2^N coefficients:

$$|\Psi\rangle = \sum_{s_1, \dots, s_N} T_{s_1 s_2 \dots s_N} |s_1 s_2 \dots s_N\rangle \quad (4.6)$$

The process begins by performing a bipartition of the system that separates the first qubit s_1 from the remaining $N-1$ qubits (s_2, \dots, s_N) . We can reshape the coefficient tensor T into a matrix M where the rows are indexed by s_1 and the columns are indexed by the composite index $(s_2 \dots s_N)$. Applying a Schmidt decomposition to this matrix yields:

$$T_{s_1(s_2 \dots s_N)} = \sum_{\alpha_1=1}^{\chi_1} U_{s_1, \alpha_1} \Lambda_{\alpha_1 \alpha_1} V_{\alpha_1, (s_2 \dots s_N)}^\dagger \quad (4.7)$$

CHAPTER 4. TENSOR NETWORK ELEMENTS

Here, U and V^\dagger are unitary matrices, and Λ is a diagonal matrix of singular values. The first MPS tensor is defined as $A_{\alpha_1}^{[1]s_1} = U_{s_1, \alpha_1}$. The remainder of the state is absorbed into a new tensor for the rest of the chain, $T'_{\alpha_1 s_2 \dots s_N} = \Lambda_{\alpha_1 \alpha_1} V_{\alpha_1, (s_2 \dots s_N)}^\dagger$.

We then repeat this process iteratively. For the second step, we reshape the new tensor T' into a matrix with rows indexed by the composite index $(\alpha_1 s_2)$ and columns by $(s_3 \dots s_N)$, and decompose it again. This procedure is applied sequentially across all qubits.

The dimension of the new indices, such as α_1 , introduced at each step is known as the *bond dimension*, often denoted by D . For a state with limited entanglement, the singular values in Λ decay rapidly, allowing us to truncate the sum by keeping only the D largest values. This truncation is the key to the compression capabilities of MPS. The bond dimension between adjacent tensors in the MPS determines the maximum length of correlations that the model can efficiently represent. In principle, a quantum state of N qubits can be represented exactly by an MPS with bond dimension $2^{N/2}$.

By repeatedly applying this principle across the chain, we decompose the full quantum state into a one-dimensional tensor network. The final form of the MPS (see Figures 4.3) is a product of these smaller local tensors:

$$|\Psi\rangle = \sum_{s_1, \dots, s_N} (A^{[1]s_1} A^{[2]s_2} \dots A^{[N]s_N}) |s_1 s_2 \dots s_N\rangle \quad (4.8)$$

This structure naturally reduces the overall dimension of the state and forms the basis of the MPS formalism [31, 29].

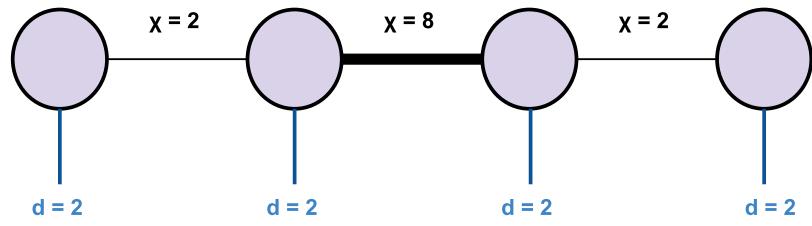


Figure 4.3: Diagram representation of Matrix Product State. The χ values represent bond dimension. Each node represent a tensor in equation (4.8).

Chapter 5

Tensor Network Born Machine

5.1 Introduction

This project focuses on constructing a classical model analogous to a quantum circuit Born machine, with a Matrix Product State serving as its foundation. The concept of using tensor networks for generative tasks have already been explored in literature [6, 8]. However, these pioneering studies did not frame their models in the context of Born machines or quantum circuits. More recent work has begun to establish this connection, highlighting the similarities between tensor network models and generative frameworks inspired by quantum mechanics [9, 10, 32].

In our approach an MPS is trained to encode a probability distribution that matches the statistics of the dataset. This method is fundamentally quantum-inspired: we are reconstructing a quantum state such that, when measured in a specific basis during sampling, gives valid samples with the same probability distribution the empirical distribution of measurement outcomes observed in the dataset.

Previous works in this area have predominantly employed the Density Matrix Renormalization Group algorithm for training the MPS. DMRG is a variational optimization algorithm that efficiently finds ground states of quantum many-body systems by iteratively updating and truncating the Hilbert space using the density matrix, retaining only the most relevant states [31]. In this context, the negative log-likelihood between the model and the data is treated analogously to a Hamiltonian, which is minimized during training [6]. We deviate from this convention by adopting a global training methodology. This approach utilizes standard gradient descent,

treating each tensor parameter as an independent variable in the optimization. Such a framework provides the flexibility to explore a wider variety of loss functions, including the Maximum Mean Discrepancy 3.3.1, as we can implement any tensor network as objects compatible with automatic differentiation within a classical machine learning environment, which in turn grants access to a broad range of established training paradigms.

Drawing inspiration from analyses of trainability in quantum generative models [7], we perform a comparative study of two different loss functions within our classical setting: the Maximum Mean Discrepancy 3.3.1 and the Kullback-Leibler divergence 3.3.1. Specifically, we examine how the variance of each loss function changes as the number of qubits increases. This allows us to investigate whether the loss concentration phenomena observed in quantum circuit Born machines also appear in our model.

Our findings reveal a partial parallel to the quantum case. The MMD loss does indeed show a rapid concentration, with its optimization landscape becoming flat at an exponential rate as the system size grows. In a significant departure from the quantum scenario, however, we find that the KLD loss does not exhibit this exponential flattening in our classical tensor network model. This result suggests a possible advantage of using explicit loss functions like the KLD for generative modeling with tensor networks.

5.2 Sampling from MPS

The first tool that we need, is an algorithm to sample efficiently from the probability distribution that the MPS represents [33]. Let's consider a matrix product state representing a tensor with N indices:

$$|\Psi\rangle = \sum_{s_1, \dots, s_N} T_{s_1 s_2 \dots s_N} |s_1 s_2 \dots s_N\rangle = \sum_{s_1, \dots, s_N} (A^{[1]s_1} A^{[2]s_2} \dots A^{[N]s_N}) |s_1 s_2 \dots s_N\rangle \quad (5.1)$$

The probability of observing the indices to have values s_1, \dots, s_N is given by

$$p(s_1, \dots, s_N) = |T^{s_1 s_2 \dots s_N}|^2 \quad (5.2)$$

with normalization

$$\sum_s p(s_1, \dots, s_N) = |T^{s_1 s_2 \dots s_N}|^2 = 1 \quad (5.3)$$

The basic principle under MPS perfect sampling algorithm [34] is the “chain rule” of probability:

$$p(s_1, s_2, s_3, \dots, s_N) = p(s_1)p(s_2 | s_1)p(s_3 | s_1s_2) \cdots p(s_N | s_1s_2s_3 \dots s_{N-1}). \quad (5.4)$$

where $p(s_1, \dots, s_N)$ indicates the probability of observing on each site $1, \dots, N$ the values s_1, \dots, s_N , and $p(s_i | s_j)$ the probability of observing s_i on site i when already s_j has been observed on site j . We can use tensor network contractions to compute each term of the product efficiently.

The probability of measuring a certain outcome for the first site is given by:

$$p(s_1) = \sum_{s_2, \dots, s_N} p(s_1, \dots, s_N) = \sum_{s_2, \dots, s_N} T^{s_1, s_2, \dots, s_N} \bar{T}^{s_1, s_2, \dots, s_N} \quad (5.5)$$

which can be computed contracting:

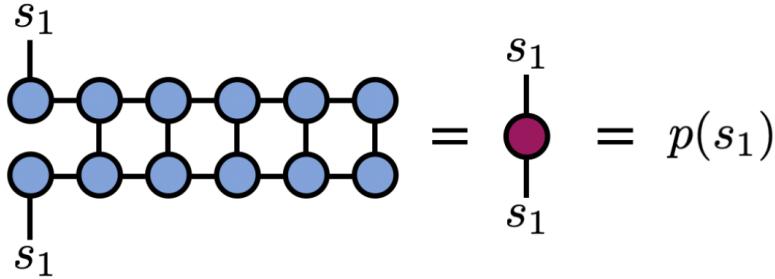


Figure 5.1: Tensor network contraction to compute $p(s_1)$ efficiently. Source [30]

we get the corresponding probabilities. For bidimensional site indices, for example, contracting the tensor network in Figure 5.1 with tensors representing $|0\rangle$ and $|1\rangle$ we get the probabilities of measuring each basis state on the first site. Given the measurement probabilities for the first site, we can sample a data \hat{s}_1 from this distribution.

We can now move on to computing $p(s_2 | s_1)$. Once extracted value on the first index \hat{s}_1 , we can contract connect to each one of the open indices the extracted tensor, cut the bond on s_2 and repeat (see Figure 5.2). One full pass of the algorithm yields a specific sample $(\hat{s}_1, \dots, \hat{s}_N)$, And one can repeat the same procedure multiple times to obtain more samples.

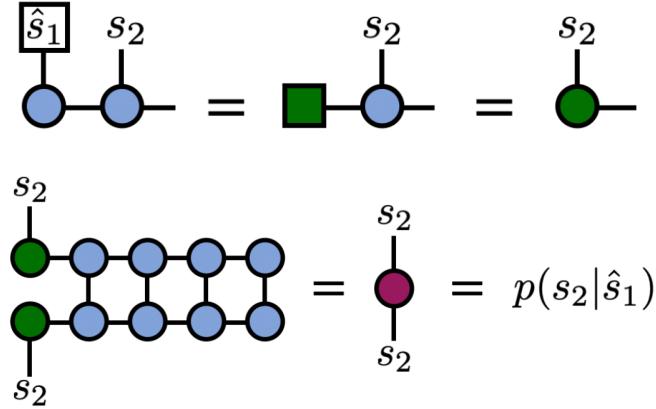


Figure 5.2: Once extracted value on the first index \hat{s}_1 , represented inside the square up left, and performed the contraction, cut the bond on s_2 , we get a structure similar to the previous step. Source [30]

5.3 Architecture

5.3.1 Parametrized MPS

Each data point to be generated by our model is represented as a bitstring, which can be interpreted as the outcome of measuring a quantum state in the computational basis (i.e., the eigen-basis of the Pauli $\hat{\sigma}_z$ operator). For a bitstring of length N , we employ a matrix product state (MPS) with N physical indices, corresponding to N nodes in the tensor network. The physical index dimension is 2, reflecting the binary nature of qubits.

As introduced in Sec. 4.4, the bond dimension between adjacent tensors determines how efficiently an MPS can represent states with long-range correlations, at the cost of increased computational resources. In this sense, it is a parameter worth optimizing. However, in our approach, it is not optimized during training; instead, it is treated as a hyperparameter that controls the expressivity and computational cost of the model.

While we focus on MPS in this work, it is important to note that our training algorithm is agnostic to the specific tensor network architecture. As already mentioned in Sec. 5.1 since the optimization acts directly on the parameters of the network, the formalism can be extended to more general tensor network structures.

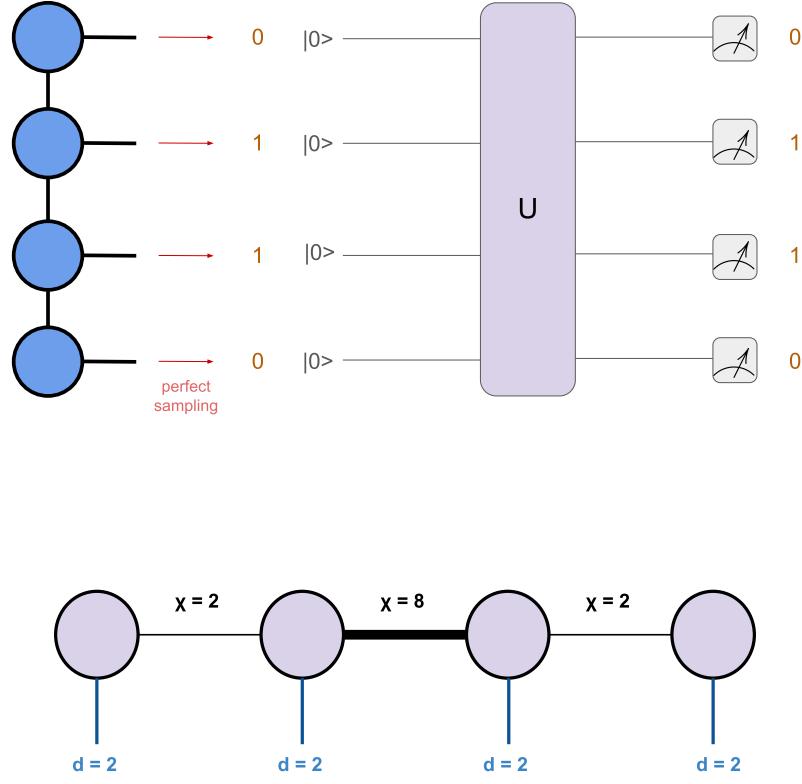


Figure 5.3: Top: Illustration of the correspondence between a quantum circuit and MPS representation. Each qubit is associated with a tensor (node) in the MPS and sampling from the MPS corresponds to projective measurement in the computational basis (i.e., $\hat{\sigma}_z$ measurement) on the quantum circuit. Bottom: Structure of the MPS used in our model, with fixed physical dimension $d = 2$ and bond dimension χ . Both are not updated in the training algorithm, and can be thought as an hyperparameter, determining the expressivity and computational cost of the model.

5.3.2 POVM Projection

The computation of MMD, as defined in Eq. (3.16), requires evaluating expectation values of the kernel function K over the model's probability distribution for both the first and second terms. In principle, this necessitates sampling the MPS

sufficiently to achieve adequate coverage of all possible bitstrings. However, for N nodes, the total number of possible bitstrings scales as 2^N , rendering this sampling approach computationally intractable for large systems. This exponential scaling issue, which is unavoidable in quantum circuits and constitutes a primary source of barren plateaus [7], can be circumvented in tensor networks due to their classical accessibility to the complete probability distribution. Rather than obtaining the probability distribution through stochastic sampling, we can directly represent it using its tensor network formulation and contract it with the kernel function, which is likewise expressed in tensor representation. This approach enables exact computation of the required expectation values without the exponential overhead associated with sampling-based methods.

For a single bitstring b , the probability of sampling this outcome is given by

$$p(b) = |\langle \psi | b \rangle|^2 = \langle \psi | b \rangle \langle b | \psi \rangle = \langle \psi | \Pi_b | \psi \rangle \quad (5.6)$$

where $\Pi_b = |b\rangle\langle b|$ is the projector onto the bitstring b .

Focusing on a single qubit, we can collect the projectors onto the states $|0\rangle$ and $|1\rangle$ into a list:

$$\Pi = [\Pi_0, \Pi_1], \quad \Pi_0 = |0\rangle\langle 0|, \quad \Pi_1 = |1\rangle\langle 1| \quad (5.7)$$

This object $\Pi = \Pi_{i,j,k}$ is a tensor of dimensions $(2, 2, 2)$, and it serves as a representation of the POVM corresponding to measurement in the computational ($\hat{\sigma}_z$) basis. The tensor is given by:

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad (5.8)$$

where the i and j indexes run on the row and columns, k distinguish the operator.

The sampling probability for a bitstring $b_1 b_2 \dots b_N$ can then be written as

$$p(b_1 b_2 \dots b_N) = \langle \psi | \Pi_{b_1} \otimes \Pi_{b_2} \otimes \dots \otimes \Pi_{b_N} | \psi \rangle \quad (5.9)$$

where each local Π_{b_i} is the $(2, 2, 2)$ tensor introduced above. Importantly, this construction yields a tensor network where the open legs correspond to the measurement outcomes b_i .

For the single-qubit case, measuring only in the $\hat{\sigma}_z$ basis, this reduces to

$$p(b) = \langle \psi | \Pi | \psi \rangle \quad (5.10)$$

where $b \in \{0, 1\}$.

This formalism provides a direct and efficient way to compute the sampling probabilities for any POVM.

5.3.3 Data Representation: Hyperindexed Tensor Network

Given a dataset of bitstrings, each bitstring can be interpreted as a computational basis state. For example, the bitstring $(1, 0)$ corresponds to the quantum state $|1\rangle \otimes |0\rangle$. In practice, each bit is represented as a one-hot vector, a common procedure in machine learning to encode categorical data

$$\begin{aligned} 1 &\rightarrow (0, 1) \\ 0 &\rightarrow (1, 0) \end{aligned}$$

Suppose we have $/T/$ data points (bitstrings), each of length N , corresponding to the number of qubits. To represent the entire dataset as a tensor network, we construct what we refer to as a *hyperindexed tensor network*. This network consists of N nodes, one for each qubit. Each node is a rank-2 tensor (i.e., a matrix) with the following structure:

- The first index, called the *hyperindex*, labels the sample from which the entry is taken. This index runs over all $|T|$ samples in the dataset.
- The second index run over the entries of the one-hot encoded single vectors.

For each qubit, the corresponding tensor encodes the observed values for that qubit across all samples, using one-hot encoding. Summing over the hyperindex is equivalent to summing over all samples in the datasets in Fig. 5.4

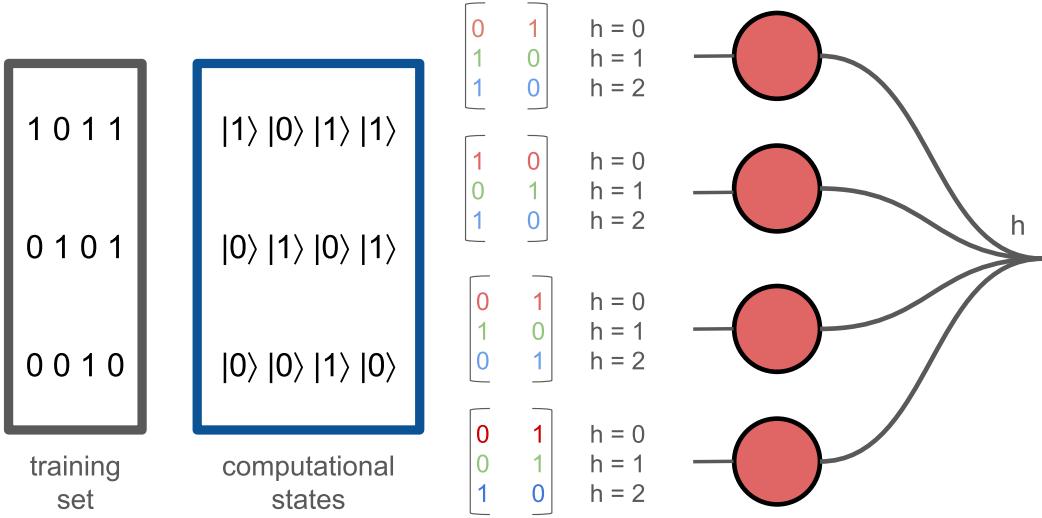


Figure 5.4: Illustration of the construction of a hyperindexed tensor network from a dataset of three bitstrings on four qubits. From left to right: the original dataset of bitstrings, their interpretation as computational basis states, the corresponding one-hot vector representations for each qubit, and the resulting hyperindexed tensor network. In the network, each node represents a qubit and is a rank-2 tensor (matrix) with one index labeling the sample (hyperindex h) and the other corresponding to the physical state of the qubit. This structure compactly encodes the entire dataset and enables efficient manipulation within the tensor network formalism. To indicate the sum on the hyperindex, we use the convention of connecting the correspondent index for each tensor.

This construction enables an efficient and compact representation of the entire dataset within a single tensor network.

5.4 Kullback-Leibler Divergence Calculation

The Kullback-Leibler divergence is a fundamental measure of the difference between two probability distributions. Given a data probability distribution $p_{\text{data}}(b)$ and a model probability distribution $q_\theta(b)$, with θ representing the parameters to be

optimized. As introduced in Sec 3.3.1 the KLD is defined as

$$D_{\text{KL}}(p_{\text{data}} \| p_{\text{model}}) = \sum_{b \in \text{dataset}} p_{\text{data}}(b) \log \frac{p_{\text{data}}(b)}{q_{\theta}(b)}. \quad (5.11)$$

Where the sum runs over all samples in the dataset. This quantity is always non-negative and is minimized when the model distribution matches the data distribution exactly.

As already shown in Sec. 3.3.1 minimizing KLD is equivalent to minimize Negative Log Likelihood:

$$\mathcal{L}_{\text{NLL}} = -\frac{1}{|T|} \sum_{b \in \text{dataset}} \log(p_{\theta}(b)). \quad (5.12)$$

The empirical data distribution $p_{\text{data}}(b)$ is approximated by the frequency with which each bitstring appears in the dataset. For the specific case of a synthetic dataset in which every data-point is unique (on which we will focus in the next chapter), this frequency is uniform across all samples.

The calculation of the quantity in equation 5.12 is performed constructing and contracting tensor networks. First, a random MPS $|\psi\rangle$ (our *generative model*) is initialized, with fixed bond dimension χ and number N of physical indexes equal to the dimension of the bitstrings to be generated as pictured in Fig. 5.3. We build the tensor network representation of the parametric probability distribution to be optimized, as:

$$q_{\theta}(b) = |\langle \psi(\theta) | b \rangle|^2 \quad (5.13)$$

represents the projector on computational basis, applied to each qubit, as shown in Eq. (5.10). The quantity $q_{\theta}(b)$, is then calculated for every example b in the training set T , contracting with the computational-MPS representations, as in 5.3.3, and taking the squared norm, which yields a real number. Since in this case is necessary to compute the logarithm of this quantity for every data-point, is not possible to leverage an hyperindex tensor network structure in an easy way, as explained in 5.4.

Operationally, this is implemented by the procedure reported in 1.

5.5 Maximum Mean Discrepancy Calculation

Maximum Mean Discrepancy (MMD) is a distance measure between two probability distributions based on the expectation values of a positive semi-definite kernel

Algorithm 1 Compute Dataset Log-Likelihood

- 1: Initialize the random MPS $|\psi\rangle = \psi_i$ with $i \in [0, 1]$, given the bond dimension and number of sites N .
- 2: From every data-point in the training set $b \in T$, a computational MPS b_j is built and stored into a list.
- 3: **while** number of epochs reached or stopping criteria met **do**
- 4: **for** each sample b_j in the training set **do**
- 5: perform the contraction operation $\psi_i b_i = c_\theta(b)$.
- 6: Compute $\log(|c_\theta(b)|^2) = \log(q_\theta(b))$ and store it.
- 7: **end for**
- 8: Average the log-probabilities over the dataset, weighted by the empirical frequencies (uniform in our case).
- 9: run gradient descent algorithm to update parameters of the MPS.
- 10: **end while**

function $K(b, b')$, as explained previously in Sec. 3.3.1. . In our context, it is used to compare the distribution $q_\theta(b)$, generated by a quantum model, with a target distribution $p(b')$ obtained from classical data, explained in section 3.3.1.

The MMD can be written in the following form:

$$\begin{aligned} \mathcal{L}_{MMD}(q_\theta, p) &= \sum_{b \in \Omega} \sum_{b' \in \Omega} q_\theta(b) K(b, b') q_\theta(b') \\ &\quad - 2 \sum_{b \in \Omega} \sum_{b' \in T} q_\theta(b) K(b, b') p(b') \\ &\quad + \sum_{b \in T} \sum_{b' \in T} p(b) p(b') K(b, b') \end{aligned}$$

where q_θ denotes the model's probability distribution, p represents the data distribution that q_θ is intended to match. In our case, we focus on a gaussian kernel function defined as Eq. (5.18) where σ is the so-called bandwidth parameter. Setting the bandwidth parameter σ correctly, is important for model convergence, as widely discussed in previous works [7].

The optimization problem Eq. (3.16) can then be expressed as:

$$\begin{aligned} \operatorname{argmin}_{\theta}(\text{MMD}(q_{\theta}, p)) &= \operatorname{argmin}_{\theta} \left[\sum_{b, b' \in \Omega} q_{\theta}(b) K(b, b') q_{\theta}(b') \right. \\ &\quad \left. - 2 \sum_{b \in \Omega} \sum_{b' \in T} q_{\theta}(b) K(b, b') p(b') \right] \end{aligned} \quad (5.14)$$

As for the KLD calculation, detailed in the previous section, the goal is to compute the quantity in equation 5.14 by constructing and contracting tensor networks.

a random MPS $|\psi\rangle$ and POVM tensor Π_{ijk} are initialized and a tensor network is built connecting a POVM tensor to each site of the MPS.

For each bitstring $b = b_1 \dots b_n$, the probability is given by

$$p(b_1 b_2 \dots b_n) = \langle \psi | \Pi_{b_1} \otimes \Pi_{b_2} \dots \otimes \Pi_{b_n} | \psi \rangle, \quad (5.15)$$

each of the local Π_b is the (2,2,2) tensors introduced above, where b_i is not an index but just a notation to identify the bit on which it is projecting. For a single-qubit space, measuring only in $\hat{\sigma}_z$ basis, this corresponds to contracting the tensor:

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad (5.16)$$

to each node of the MPS. The expression $p(b_1 b_2 \dots b_n) = \langle \psi | \Pi_{b_1} \otimes \Pi_{b_2} \dots \otimes \Pi_{b_n} | \psi \rangle$ can be built as in figure 5.5:

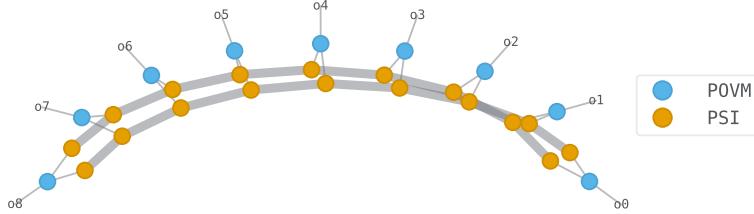


Figure 5.5: Tensor network representation of the sampling probabilities $p(b_1 b_2 \dots b_n)$. Each node corresponds to a local projector Π_{i,j,o_k} , and the open legs represent the outcomes o_k .

The kernel matrix K is defined as

$$K = \begin{pmatrix} 1 & e^{-\frac{1}{2\sigma^2}} \\ e^{-\frac{1}{2\sigma^2}} & 1 \end{pmatrix} \quad (5.17)$$

which encodes similarity between bitstrings using a Gaussian function.

$$\begin{aligned}
 K(x, y) &= e^{-\frac{|b-b'|^2}{2\sigma^2}} = e^{-\frac{\sum_i |b_i - b'_i|^2}{2\sigma^2}} = \prod_i e^{-\frac{|b_i - b'_i|^2}{2\sigma^2}} \\
 &= \binom{b_0}{b_1} \begin{pmatrix} 1 & e^{-\frac{1}{2\sigma^2}} \\ e^{-\frac{1}{2\sigma^2}} & 1 \end{pmatrix} \binom{b'_0}{b'_1}
 \end{aligned} \tag{5.18}$$

Rewriting Eq. (5.10) in tensor notation, using Einstein notation convention on indexes, \mathcal{L}_{MMD} in Eq. (5.14) becomes:

$$= \psi_b \Pi_k^{b,o} \psi^k K_o^{o'} \psi_b \Pi_k^{b,o'} \psi^k - 2 \sum_B \left[\psi^b \Pi_b^{k,o} \psi_k K_o^{o'} b^o \right] \tag{5.19}$$

The sum over B denotes averaging over the target data bitstrings y , with b^o representing the empirical frequency vector from data.

The first term, as depicted in figure 5.6, is constructed by first contracting the tensor network in figure 5.5 with a kernel tensor site by site, and then further contracting the result with that same initial tensor network.

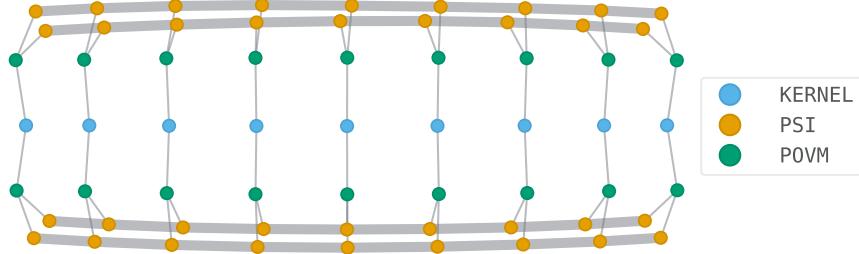


Figure 5.6: Tensor network representation of the first term in the MMD expression. The kernel matrix K acts on the output space indexed by x and y , and can be realized as a kernel MPO.

The second term needs the MPS representation of data. As detailed in section 5.3.3, an hyperindexed tensor network is built, containing the information of computational states corresponding to each data-point in the dataset 5.7.

Once the dataset is converted into tensor network formalism, we can contract the structure model-POVM-kernel tensor network with the data as in figure 6.3

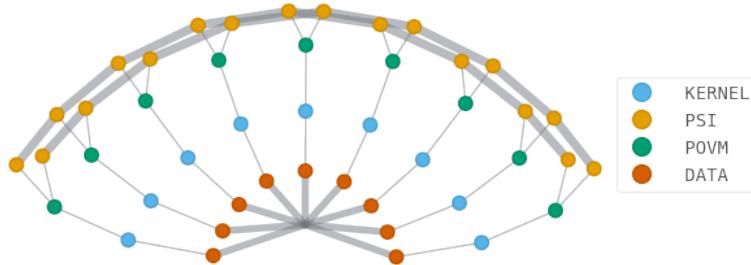


Figure 5.7: Tensor network representation of the mixed term in the MMD expression.

At each call of loss function, is performed the contraction of the model tensor network and the overlap tensor network as in 2:

Algorithm 2 Compute Dataset Maximum Mean Discrepancy

- 1: Initialize the random MPS $|\psi\rangle$ with $i \in [0, 1]$, given the bond dimension and number of sites N . Initialize POVM tensors for projecting in computational basis $\Pi_j^{i,o}$. Initialize Kernel Matrix $K_o^{o'}$.
 - 2: Create a tensor network connecting every site in the MPS with a POVM tensor $\psi_i \Pi_j^{i,o} \psi^j$.
 - 3: From the examples contained in the training set, build the hyperindexed tensor network representing all computational states as described in 5.4.
 - 4: **while** number of epochs reached or stopping criteria met **do**
 - 5: build the homogeneous term (not depending on data) 5.6 connecting the MPS-POVM tensor network with $K_o^{o'}$ and again POVM-MPS $\psi_b \Pi_k^{b,o} \psi^k K_o^{o'} \psi_b \Pi_k^{b,o'} \psi^k = h_\theta$ and perform contraction and perform the contraction.
 - 6: build the mixed term 5.7 connecting the MPS-POVM tensor network with $K_o^{o'}$ and the hyperindexed tensor network $\psi^b \Pi_b^{ko} \psi_k K_o^{o'} b^o = m$ and perform contraction.
 - 7: Compute $MMD = h - 2m + \text{constant}$.
 - 8: run gradient descent algorithm to update parameters of the MPS.
 - 9: **end while**
-

Chapter 6

Numerical Results

In this chapter, we present the numerical results of our investigation into training and evaluating the TNBM for generative modeling of discrete data. We detail the training methodology, evaluate model performance and analyze loss concentration phenomena.

6.1 Training and Evaluation of Tensor Network Generative Models

To align the parametric probability distribution q_θ with the statistical distribution of the dataset T we make use of gradient based optimization, with an ADAM full-batch gradient descent algorithm, already introduced in section 3.3. In our particular setting, we consider the tensor representation of the MPS representing our generative model $|\psi\rangle$, and assume *every single component* of this tensor to be a trainable parameter and the MPS is normalized after every parameter update. We train the model on “bars and stripes”(BAS) dataset.

In this section, details for the dataset are given and an analysis on both loss functions convergence and relative model performance is done. Furthermore the problem of loss concentration is addressed.

6.1.1 The Bars and Stripes Dataset

The *Bars and Stripes* (BAS) dataset is a canonical benchmark for generative modeling with discrete data. It consists of binary strings that encode distinct

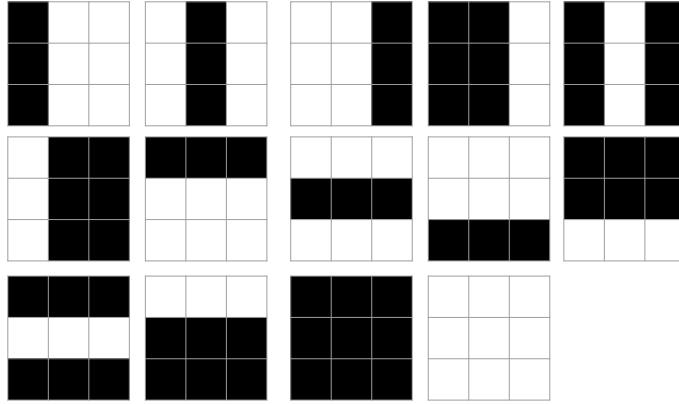


Figure 6.1: All possible patterns in Bars and Stripes 3x3 dataset. Analogue structure is generalizable to higher dimensions.

horizontal or vertical bar and stripe patterns on a two-dimensional grid 6.1. Each valid pattern corresponds to a unique bitstring, and the dataset is constructed to include all such patterns for a given grid size. The BAS dataset of dimension N consists of $2^{\sqrt{N}}$ bars and the same number of possible stripes minus 2 trivial all black and all white patterns.

$$\#\text{bas} = 2^{\sqrt{N}} + 2^{\sqrt{N}} - 2 = 2^{\sqrt{N}+1} - 2 \quad (6.1)$$

Over a total dimension of the space of 2^N possible bitstrings. Each bars and stripes example, is a black and white image, that can be represented by a matrix of 0 and 1. Each one of this matrices is flattened in a vector, in other words converting a rank-2 tensor into a rank-1 tensor. As addressed in section 4.1 index fusing is not a univocal operation. In practical terms the “path“ to follow in the image to be turned into a vector matters. The patterned used in our work is shown in Fig. 6.3, which yields bars samples to be “alternated“ paths and stripes to be “continues“ paths:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \rightarrow (1 \text{ red } 0 \text{ blue } 0 \text{ green } 1 \text{ red } 0 \text{ blue } 0 \text{ green })$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow (1 \text{ red } 1 \text{ red } 1 \text{ red } 0 \text{ blue } 0 \text{ blue } 0 \text{ green } 0 \text{ green } 0 \text{ green })$$

Figure 6.2: In flattening the bars and stripes matrices into vectors, we convert each matrix into a single row vector by concatenating its rows into a continuous sequence, where 1 will correspond to white in the image, and 0 to black.

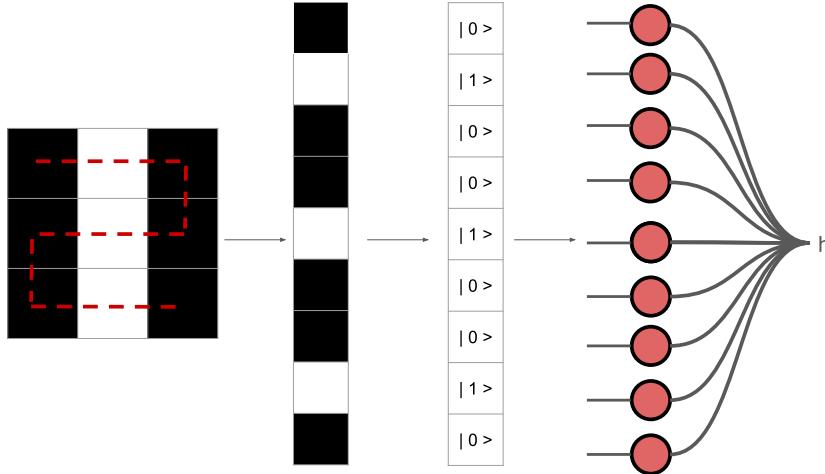


Figure 6.3: Illustration of data representation for the bars and stripes dataset. From left to right: an example image from the dataset; the same pattern flattened into a column vector; the corresponding computational basis state represented as a column of 0 and 1 values; the one-hot encoding of each bit; and the resulting hyperindexed matrix product state (MPS) with one tensor per site. Each tensor has a physical index for the one-hot encoded bit value and a hyperindex labeling the sample, as described in the main text.

As detailed in section 5.3.3 the whole training set of bitstrings is converted into an MPS in the computational basis with trivial bond dimension ($\chi = 1$).

The MPS linear architecture was selected for this project owing to its inherent

CHAPTER 6. NUMERICAL RESULTS

simplicity and its extensive recognition within the existing literature. However, different tensor network geometries are better suited to different dimensionalities.

An MPS, being a 1D construct, must funnel all the information corresponding to this growing boundary through a single connection. This creates a fundamental dimensional mismatch when applied to 2D image data, as the 1D tensor network structure cannot naturally accommodate the 2D spatial correlations present in images. Despite this clear structural incompatibility, our results demonstrate that MPS-based generative models can still achieve decent performance on 2D datasets. However, this fundamental scaling behavior explains why geometrically matched tensor networks like PEPS [35] are inherently better suited for 2D systems, as their structure is inherently designed to accommodate the area law for correlations and naturally respect the dimensionality of the data. For 3D or higher dimensional data, Tree Tensor Network architectures [8] could be implemented to better match the underlying geometric structure.

6.1.2 Random Initialization and Initial Sampling Behavior

Prior to training, the parameters of the MPS are initialized randomly, sampling each tensor element from a uniform distribution. To assess the generative behavior of the untrained model, we sample bitstrings from the randomly initialized MPS. As illustrated in Figure 6.4, the resulting samples appear completely random, lacking any identifiable structure or resemblance to the BAS patterns, the model does not possess any prior knowledge of the dataset’s structure.

6.1.3 Training

We performed the training of the model on the BAS dataset, using both Maximum Mean Discrepancy and Kullback-Leibler Divergence/Negative Log-Likelihood losses. As already mentioned in the first part of this section, we adopt a “global” training approach, in which every element of every tensor in the MPS is treated as a trainable parameter.

To investigate the effect of model expressivity, we train MPS models with varying bond dimensions $\chi = 2, 8, 16, 32$. Figures 6.5 show the evolution of the respective losses during training for each bond dimension. As expected, increasing the bond dimension enhances the expressive power of the model, enabling it to better approximate the target distribution and achieve lower loss values. Given

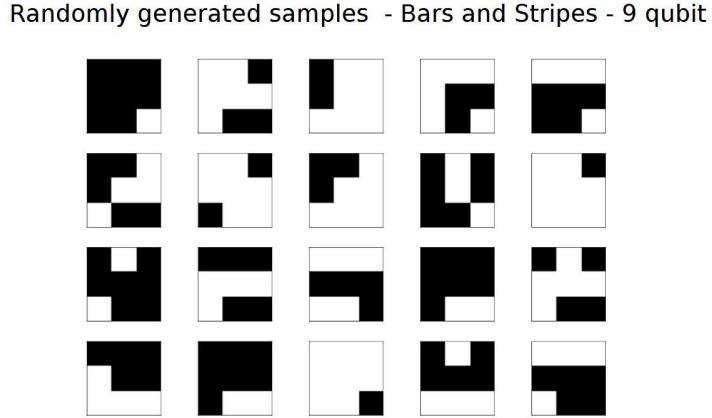


Figure 6.4: Samples generated from a randomly initialized MPS. The absence of recognizable patterns demonstrates that the model requires training to capture the structure of the 3x3 BAS dataset - that translates to MPS with 9 sites.

a bond dimension χ , an MPS can represent exactly any state of dimension $2^{\chi/2}$. We are trying to generate data in a subspace of the space of all possible bitstrings of length $N = 9$, which has dimension 2^9 . Any state of this space can be exactly represented by an MPS with bond dimension $\chi = 32$, since $2^{32/2} = 2^{16} \gg 2^9$. However, with $\chi = 16$, we get a good approximation of vectors in the subspace in which bars and stripes data lies, as $2^{16/2} = 2^8$ provides sufficient representational capacity for the structured patterns present in this dataset. With $\chi = 8, 2$ the representation capabilities are strictly limited

6.1.4 Evaluation Metrics: Validity and Coverage

To quantitatively assess the generative performance of the trained models, accuracy and coverage metrics are computed.

Accuracy: The number of generated samples that correspond to valid BAS patterns, reflecting the model’s ability to avoid “hallucinations” (i.e., generating invalid or out-of-distribution patterns).

Coverage: The number of unique valid patterns generated, indicating the diversity and completeness of the model’s output with respect to the target distribution.

CHAPTER 6. NUMERICAL RESULTS

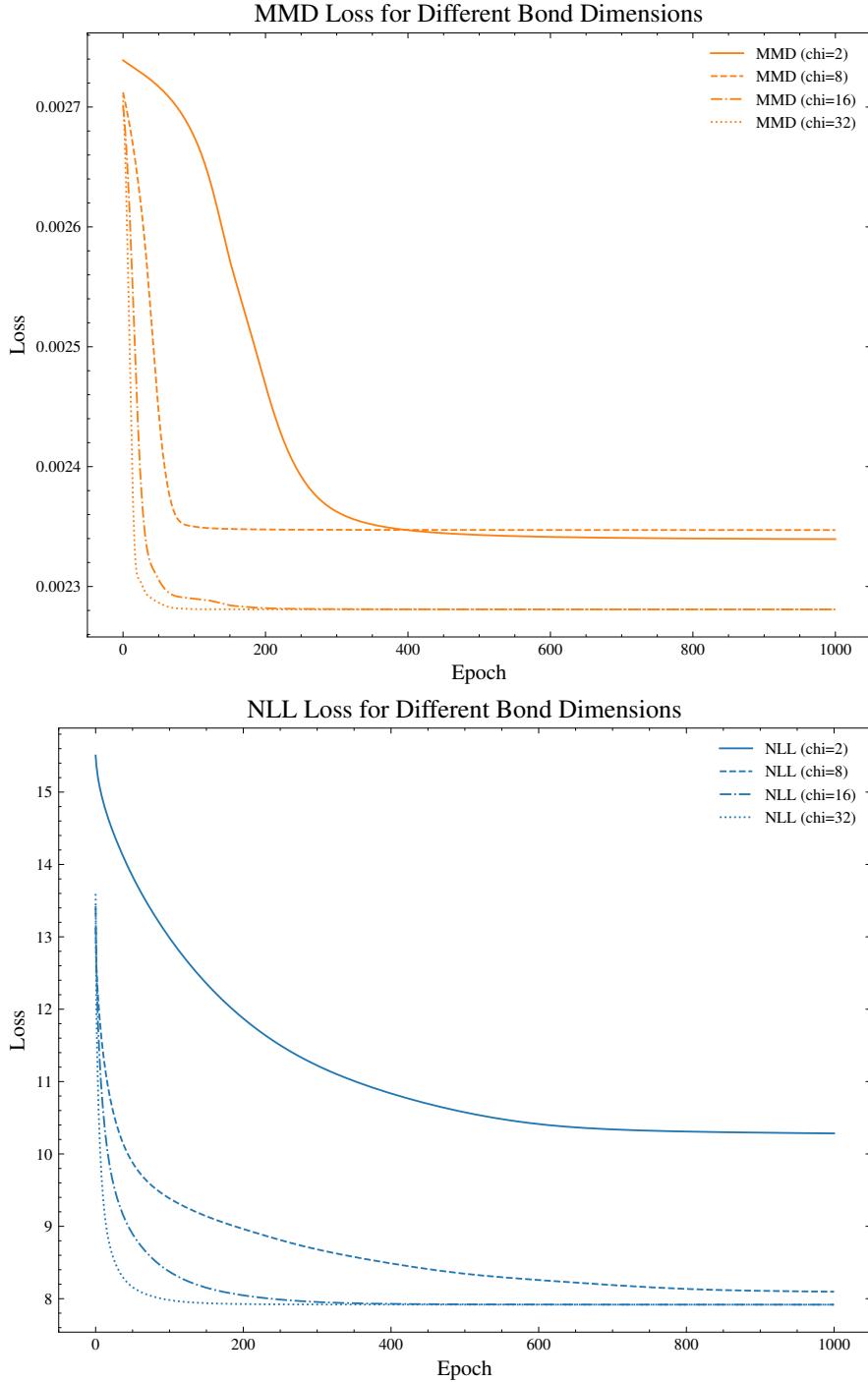


Figure 6.5: Training curves for MMD and NLL loss at different bond dimensions. The effect of increasing expressivity is evident in the improved loss values. Sigma value for MMD kernel function $\sigma = 0.55$. All of 14 3x3 BAS samples were used.

CHAPTER 6. NUMERICAL RESULTS

Figures 6.6 present the results for both metrics, evaluated on samples generated from models trained with MMD and NLL losses at each bond dimension. The analysis reveals some specific trends:

models trained with Maximum Mean Discrepancy consistently avoid hallucinations across all bond dimensions, producing only valid BAS patterns. However, at low bond dimensions ($\chi = 2$ and 8), the model generates only a single unique pattern, indicating that it has learned to reproduce just one mode of the distribution. This is reflected in the coverage metric, which remains at one for these cases. As the bond dimension increases, coverage improves, but even at $\chi = 32$, the model captures only 11 out of 14 possible BAS patterns, suggesting a trade-off between validity and diversity.

models trained with Negative Log Likelihood exhibit a different behavior. At low bond dimensions ($\chi = 2$ and 8), the model frequently hallucinates, generating invalid patterns not present in the BAS dataset. This is reflected in a lower validity score. However, as the bond dimension increases, the model's expressivity improves, hallucinations disappear, and both validity and coverage approach their maximum values. At $\chi = 32$, the NLL-trained model is able to generate all valid patterns without hallucinations, demonstrating the expected benefit of increased expressivity.

A detailed collection of generated samples for each combination of loss function and bond dimension is provided in Appendix 7.1. Inspection of these samples confirms the quantitative findings: for MMD with $\chi = 2$ and 8 , the model deterministically generates a single valid pattern, indicating that it has collapsed onto a single mode of the distribution. This behavior is a consequence of the limited expressivity at low bond dimension. In contrast, NLL-trained models at low bond dimension produce a mixture of valid and invalid patterns, reflecting a different failure mode characterized by hallucinations.

CHAPTER 6. NUMERICAL RESULTS

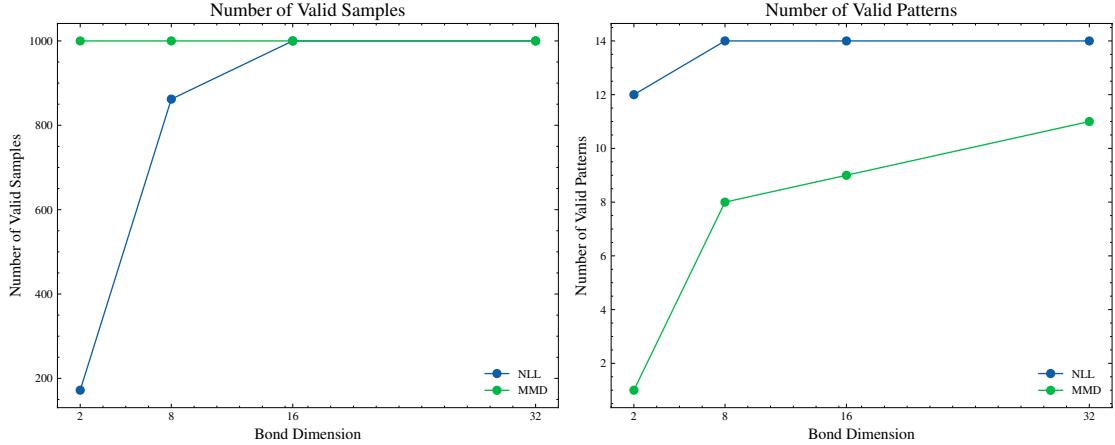


Figure 6.6: Left: number of valid samples generated as a function of bond dimension, for MMD and NLL losses. MMD avoids hallucinations at all bond dimensions, while NLL requires higher expressivity to achieve full validity.
Right: number of unique valid patterns generated (coverage) as a function of bond dimension. MMD is limited at low bond dimension, while NLL achieves full coverage at high expressivity.

Bodyness of Observables

To address the phenomenon of loss concentration in MMD loss function presented in the previous section, the concept of *bodyness* of an operator is needed, and thus introduced in this section.

The *bodyness* of an observable is a quantitative measure of how many qubits a given operator acts upon non-trivially within a quantum system. More precisely, an observable is said to be k -bodied if it can be written as a sum of terms, each of which acts non-trivially on at most k qubits, while acting as the identity on the remaining qubits. For example, a single-qubit Pauli operator is 1-bodied, a two-qubit interaction term (such as $Z_i Z_j$) is 2-bodied, and a global observable that acts on all n qubits is n -bodied.

Low-bodied observables (small k) are typically associated with local measurements and are less susceptible to the exponential concentration effects that lead to barren plateaus [7]. In contrast, high-bodied (global) observables can induce barren plateaus even in shallow circuits, as their expectation values and gradients are more likely to concentrate exponentially around their mean.

CHAPTER 6. NUMERICAL RESULTS

Let's consider the MMD loss function defined as:

$$\begin{aligned}\mathcal{L}_{\text{MMD}}(\theta) &= \mathbb{E}_{x,y \sim p_\theta} [K(x,y)] - 2 \mathbb{E}_{x \sim p_\theta, y \sim \pi} [K(x,y)] + \mathbb{E}_{x,y \sim \pi} [K(x,y)] = \\ &= \sum_{x,y \in \mathcal{X}} q_\theta(x) q_\theta(y) K(x,y) - 2 \sum_{x,y \in \mathcal{X}} q_\theta(x) p(y) K(x,y) \\ &\quad + \sum_{x,y \in \mathcal{X}} p(x) p(y) K(x,y)\end{aligned}\tag{6.2}$$

where

$$K_\sigma(x,y) = \exp\left(-\frac{\|x-y\|_2^2}{2\sigma}\right) = \prod_{i=1}^n \exp\left(-\frac{(x_i-y_i)^2}{2\sigma}\right)\tag{6.3}$$

We can rewrite each term in the MMD loss as an expectation value of an observable:

$$\mathcal{M}(\rho, \rho') = \text{Tr} \left[\hat{O}_{\text{MMD}}^{(\sigma)} (\rho \otimes \rho') \right]\tag{6.4}$$

where

$$\hat{O}_{\text{MMD}}^{(\sigma)} := \sum_{x,y} K_\sigma(x,y) |x\rangle\langle x| \otimes |y\rangle\langle y|\tag{6.5}$$

We can also rewrite the observable as the sum of Pauli strings, each one containing

$$\hat{O}_{\text{MMD}}^{(\sigma)} = \sum_{l=0}^n \binom{n}{l} (1-p_\sigma)^{n-l} p_\sigma^l \hat{D}_{2l}\tag{6.6}$$

where

$$\hat{D}_{2l} = \frac{1}{\binom{n}{l}} \sum_{\substack{A \subseteq \mathcal{N} \\ |A|=l}} \bigotimes_{i \in A} (Z_i \otimes Z_{n+i})\tag{6.7}$$

where the summation runs over all possible subsets $A \subseteq \mathcal{N} = \{1, 2, \dots, n\}$ of size l , and

$$p_\sigma = \frac{1 - \exp(-\frac{1}{2\sigma})}{2}.\tag{6.8}$$

The operator in equation 6.7 is a normalized sum of *all possible* $\hat{\sigma}_z$ strings of length $2l$, which yields $2l$ – body interaction, and thus an operator of bodyness $2l$. The operator in 6.6 is written as a weighted sum of operators of bodyness $2l$, in which

the prefactors decrees with l . The first point is the series for which we can consider the weight of the operator approximately *zero*, determines the bodyness of \hat{O}_{MMD} .

6.2 Loss concentration in TNBM

As introduced in Sec 3.2, variational quantum circuits and quantum inspired models suffer from Barren-Plateau and Loss concentration phenomenons, that limit the trainability of the model.

To investigate the presence of barren plateaus in our setting, we analyze the variance of the loss function at random initialization as a function of the number of physical indices (qubits), for both MMD and NLL losses and across different bond dimensions. Specifically, we compute the normalized variance of the loss over multiple random initializations, and plot its logarithm as a function of system size. The results, shown in Fig. 6.7, reveal a striking difference between the two loss functions:

Maximum Mean Discrepancy the variance decreases exponentially with system size, as evidenced by the linear decay in the log plot. This is a clear signature of a barren plateau, indicating that the gradients vanish rapidly as the system grows, and optimization becomes increasingly difficult.

Negative Log-Likelihood the variance remains approximately constant across system sizes, suggesting that the barren plateau phenomenon is absent or significantly mitigated for this loss function.

Explicit loss As described in Section 3.3.2, explicit models such as TNBM provide direct access to the full probability distribution, whereas implicit models like QCBM only yield samples. A key result of [7] is that a specific form of barren plateaus in QCBMs can arise due to a mismatch between implicit models and explicit loss functions.

This issue emerges when estimating explicit losses (e.g., the Kullback-Leibler divergence) on implicit models. Since the model's probability distribution is only accessible via sampling, and the training bitstrings represent an exponentially small fraction of the full bitstring space, the probability of observing any given training

CHAPTER 6. NUMERICAL RESULTS

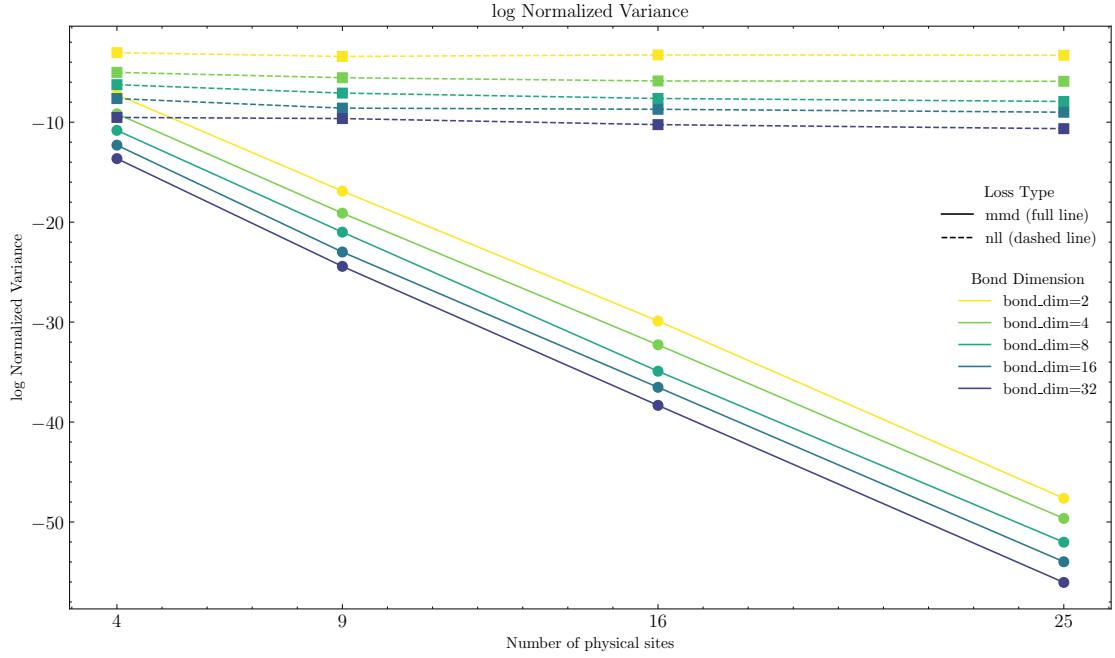


Figure 6.7: Logarithm of normalized variance of the loss function at random initialization, as a function of the number of physical indices, for different bond dimensions. MMD exhibits exponential decay (barren plateau), while NLL remains nearly constant.

bitstring is exponentially suppressed. Consequently, estimates of the loss function become statistically unreliable.

In *Proposition 1*, the authors show that if both the target distribution $p(x)$ and the model distribution $q_\theta(x)$ are known exactly, the variance of the exact KLD need not vanish exponentially. This holds true provided that the target distribution is estimated using a sufficient number of samples from the model. However, while quantum generative models can represent distributions over exponentially large sample spaces (i.e., 2^n bitstrings for n qubits), the number of samples that can be realistically drawn scales at most polynomially with n . This creates a fundamental mismatch between the exponential support of the model and the polynomially limited training data.

Since this issue is rooted in a sampling bottleneck, it has been argued that quantum-inspired models with direct access to probabilities may be trainable using explicit losses like the KLD. Our numerical experiments with TNBM trained using KLD support this hypothesis (see Fig. 6.7).

CHAPTER 6. NUMERICAL RESULTS

Implicit loss As discussed in Section 6.1.4, the MMD loss with a Gaussian kernel can be interpreted as the expectation value of an observable whose *bodyness* depends on the kernel bandwidth σ . When σ is constant with respect to system size ($\sigma \in O(1)$), the corresponding observable results composed mostly of global terms, that have been proven to bring to exponential concentration effects [7]. In this regime, the variance scales as:

$$\text{Var}_\theta[L_{\text{MMD}}^{(\sigma)}(\theta)] \in O(1/b^n) \quad (6.9)$$

with $b > 1$, bringing a bodyness-driven flavor of barren plateau for implicit models.

When training our TNBM model with MMD loss using a fixed bandwidth of $\sigma = 0.55$, we observed signs of loss concentration consistent with these theoretical predictions.

Chapter 7

Conclusions

In this thesis, we have investigated the capabilities of tensor network-based generative models, developing and analyzing a Tensor Network Born Machine. Our work show that tensor networks offer a powerful, interpretable, quantum-inspired framework for dequantizing quantum algorithms, currently limited by various practical constraints of NISQ era.

Our approach differs from previous work [6, 8, 10, 9] both in training methodology and implementation. On the methodological side, we employ a global training scheme where all parameters are optimized simultaneously, enabling the use of gradient descent methods and providing a straightforward, well-established platform for experimenting with various loss functions. On the implementation side, we developed the framework using modern and optimized tools, leveraging advanced techniques for efficient auto-differentiation and tensor contractions to ensure both flexibility and computational efficiency [36, 37].

Building on the theoretical framework proposed in previous work [7], we investigated the performance and trainability with explicit and implicit loss functions, considering varying bond dimensions and numbers of physical indices. The MPS encoding gives direct access to the full probability distribution, which naturally avoids the loss concentration problems that occur in QCBMs due to the mismatch between explicit loss functions and implicit model representations. Our numerical experiments provide empirical validation of theoretical predictions regarding the trainability of quantum-inspired generative models in Sec. 6.2.

We have successfully trained TNBM models across various bond dimensions on the bars and stripes dataset, demonstrating the model’s capability to learn structured patterns for both explicit and implicit loss formulations. The results

CHAPTER 7. CONCLUSIONS

obtained are consistent with theoretical expectations regarding the relationship between bond dimension and model expressivity.

Future directions DMRG-like training approaches, where one tensor is optimized at a time, represent a potentially more efficient alternative to our global optimization scheme for larger networks. This strategy reduces memory usage and enables dynamic bond dimension adjustment during training.

Our current model is designed to work with data from computational-basis measurements. A valuable extension would involve adapting it to handle informationally richer datasets, such as those obtained from measurements performed across multiple bases. A straightforward way to implement this would be to modify the final tensor layer of the network, which currently projects onto the computational basis, so it utilizes tensors corresponding to a specific POVM. As a result, the model would no longer be restricted to generating binary bitstrings but would instead produce data reflecting the outcomes of these generalized measurements.

Mapping discrete MPS outputs to continuous data represents another interesting direction, where discrete coefficients can be interpreted as parameters of a continuous physically motivated function, such as a Fourier expansion [9] extending the applicability of TNBM beyond discrete domains.

Chapter 8

Appendix

8.1 Generated Samples for Different Losses and Bond Dimensions

This appendix presents the generated samples produced by the trained Matrix Product State (MPS) models for the *bars and stripes* dataset, across all combinations of loss function (Maximum Mean Discrepancy, MMD; and Negative Log-Likelihood, NLL) and bond dimension ($\chi = 2, 8, 16, 32$). For each configuration, we display representative samples drawn from the model after 1000 training iterations on a 9-site system.

These plots provide qualitative insight into the generative behavior of the models, complementing the quantitative metrics discussed in the main text. In particular, they illustrate the effects of model expressivity and loss function choice on the diversity and validity of generated patterns.

Figures 8.1–8.4 show the samples generated by MPS models trained with the MMD loss at increasing bond dimensions.

CHAPTER 8. APPENDIX

Samples - mmd - Bars and Stripes - 9 sites - Bond dim 2- Iterations 1000

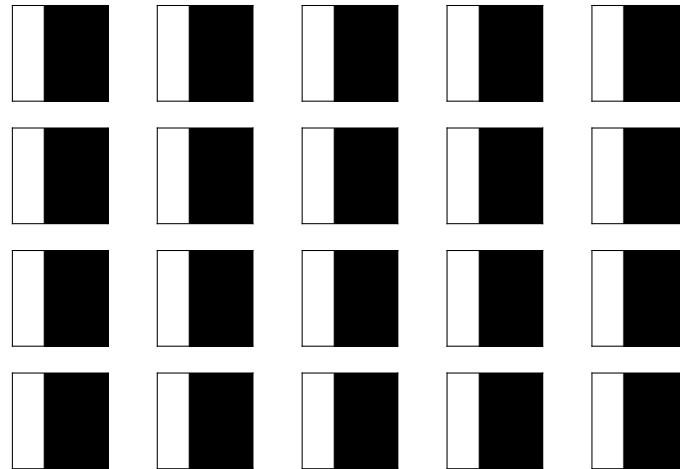


Figure 8.1: Samples generated with MMD loss, bond dimension $\chi = 2$. The model produces a single valid pattern, indicating mode collapse due to limited expressivity.

Samples - mmd - Bars and Stripes - 9 sites - Bond dim 8- Iterations 1000

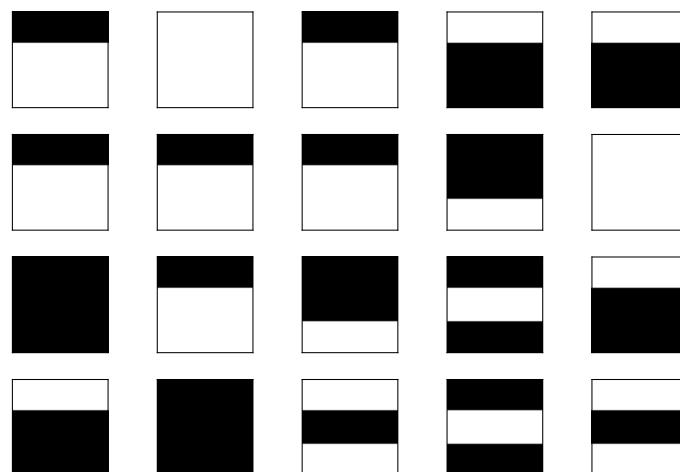


Figure 8.2: Samples generated with MMD loss, bond dimension $\chi = 8$. The model still produces only few valid pattern, showing that expressivity remains insufficient to capture the full dataset.

CHAPTER 8. APPENDIX

Samples - mmd - Bars and Stripes - 9 sites - Bond dim 16- Iterations 1000

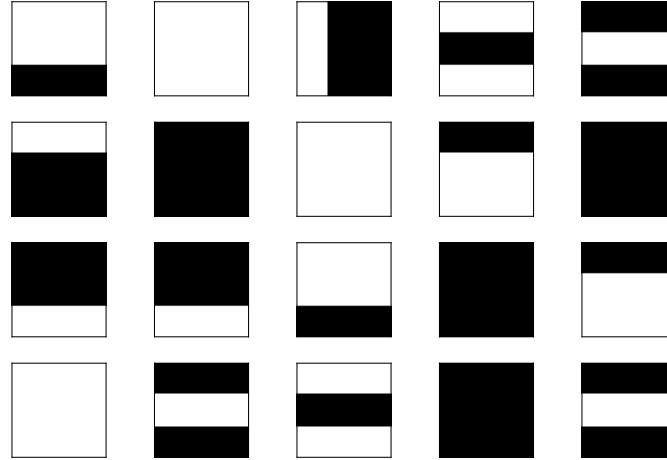


Figure 8.3: Samples generated with MMD loss, bond dimension $\chi = 16$. The model begins to generate a greater variety of valid patterns, though not all possible modes are covered.

Samples - mmd - Bars and Stripes - 9 sites - Bond dim 32- Iterations 1000

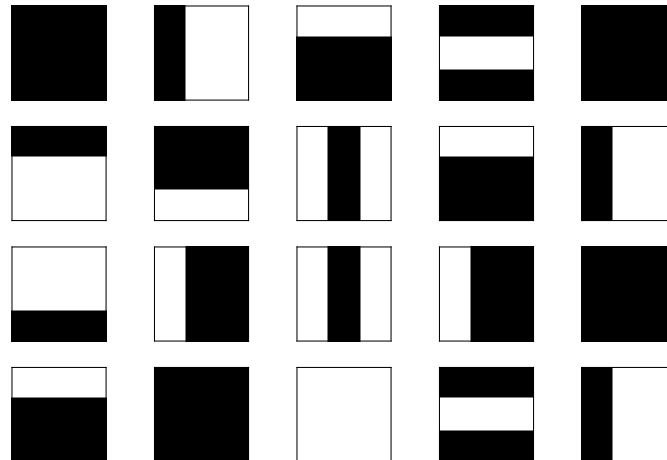


Figure 8.4: Samples generated with MMD loss, bond dimension $\chi = 32$. The diversity of generated patterns increases, but some valid patterns remain missing, as discussed in the main text.

Samples - nll - Bars and Stripes - 9 sites - Bond dim 2- Iterations 1000

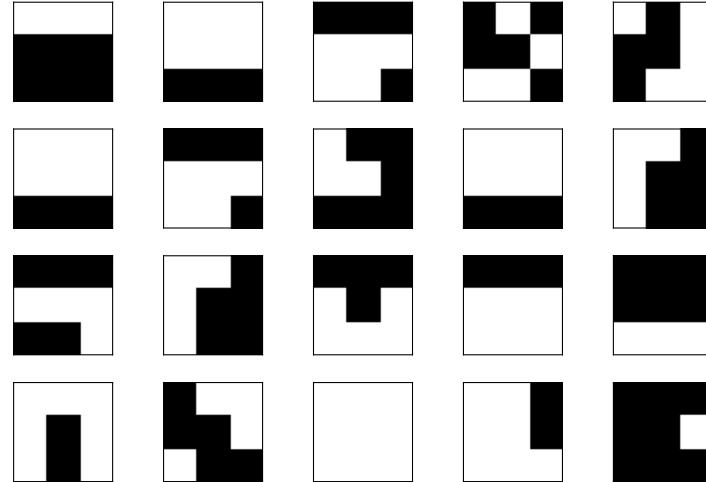


Figure 8.5: Samples generated with NLL loss, bond dimension $\chi = 2$. The model produces a mix of valid and invalid patterns, indicating hallucinations due to insufficient expressivity.

Samples - nll - Bars and Stripes - 9 sites - Bond dim 8- Iterations 1000

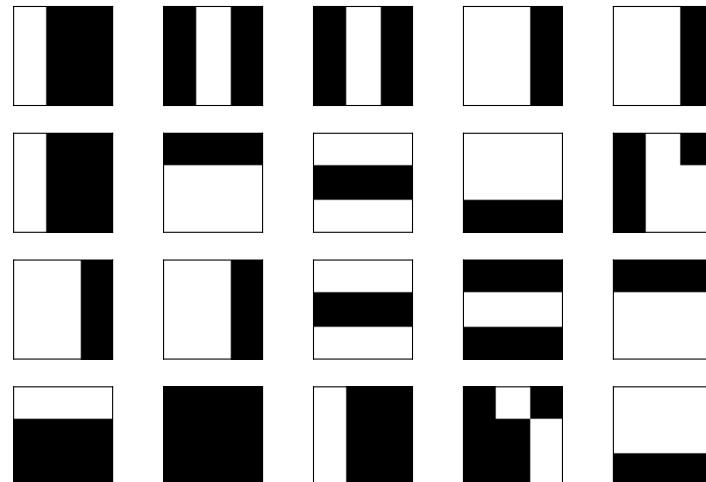


Figure 8.6: Samples generated with NLL loss, bond dimension $\chi = 8$. Hallucinations persist, but the number of unique valid patterns increases.

CHAPTER 8. APPENDIX

Samples - nll - Bars and Stripes - 9 sites - Bond dim 16- Iterations 1000

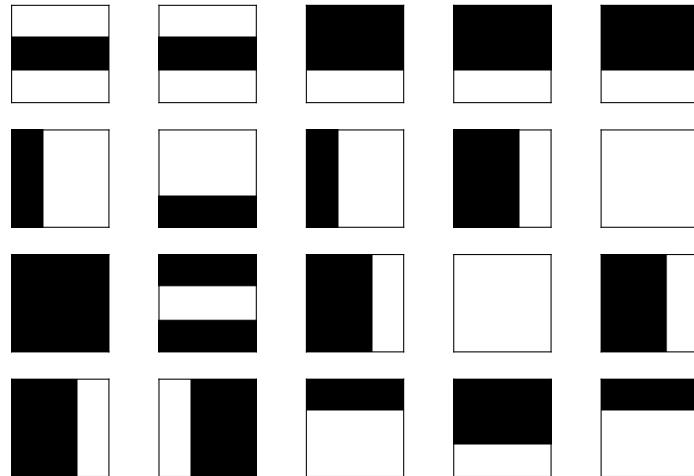


Figure 8.7: Samples generated with NLL loss, bond dimension $\chi = 16$. The model generates a larger set of valid patterns, with hallucinations becoming rare.

Samples - nll - Bars and Stripes - 9 sites - Bond dim 32- Iterations 1000

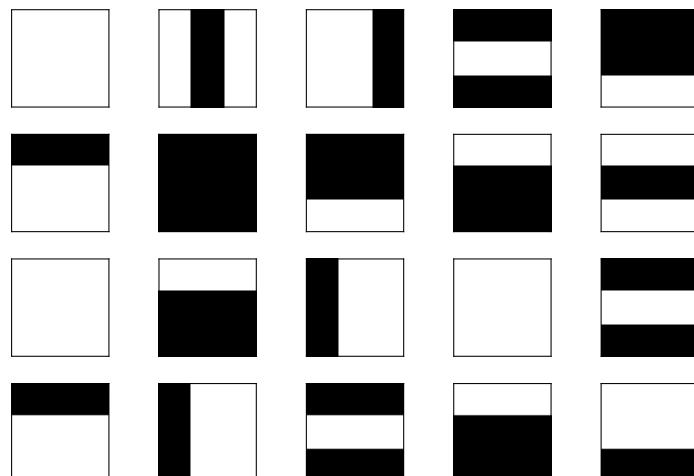


Figure 8.8: Samples generated with NLL loss, bond dimension $\chi = 32$. The model successfully generates all valid patterns without hallucinations, demonstrating full coverage and high expressivity.

List of Figures

LIST OF FIGURES

4.1	Index fusing $rank - 4 : T_{kh}^{ij} \rightarrow rank - 3 : T_g^{ij}$ and splitting $rank - 3 - \rightarrow rank - 4$ vice versa.	31
4.2	The graphical diagrammatic notation a tensor network and the equivalent expression. i, k, l are open indices, j is a bond index with bond dimension $D + 1$. Performing the contraction simply means executing the sum operation. Image taken from reference [30] . . .	32
4.3	Diagram representation of Matrix Product State. The χ values represent bond dimension. Each node represent a tensor in equation (4.8).	35
5.1	Tensor network contraction to compute $p(s_1)$ efficiently. Source [30]	38
5.2	Once extracted value on the first index \hat{s}_1 , represented inside the square up left, and performed the contraction, cut the bond on s_2 , we get a structure similar to the previous step. Source [30]	39
5.3	Top: Illustration of the correspondence between a quantum circuit and MPS representation. Each qubit is associated with a tensor (node) in the MPS and sampling from the MPS corresponds to projective measurement in the computational basis (i.e., $\hat{\sigma}_z$ measurement) on the quantum circuit. Bottom: Structure of the MPS used in our model, with fixed physical dimension $d = 2$ and bond dimension χ . Both are not updated in the training algorithm, and can be thought as an hyperparameter, determining the expressivity and computational cost of the model.	40
5.4	Illustration of the construction of a hyperindexed tensor network from a dataset of three bitstrings on four qubits. From left to right: the original dataset of bitstrings, their interpretation as computational basis states, the corresponding one-hot vector representations for each qubit, and the resulting hyperindexed tensor network. In the network, each node represents a qubit and is a rank-2 tensor (matrix) with one index labeling the sample (hyperindex h) and the other corresponding to the physical state of the qubit. This structure compactly encodes the entire dataset and enables efficient manipulation within the tensor network formalism. To indicate the sum on the hyperindex, we use the convention of connecting the correspondent index for each tensor.	43

LIST OF FIGURES

5.5	Tensor network representation of the sampling probabilities $p(b_1 b_2 \dots b_n)$. Each node corresponds to a local projector Π_{i,j,o_k} , and the open legs represent the outcomes o_k .	46
5.6	Tensor network representation of the first term in the MMD expression. The kernel matrix K acts on the output space indexed by x and y , and can be realized as a kernel MPO.	47
5.7	Tensor network representation of the mixed term in the MMD expression.	48
6.1	All possible patterns in Bars and Stripes 3x3 dataset. Analogue structure is generalizable to higher dimensions.	50
6.2	In flattening the bars and stripes matrices into vectors, we convert each matrix into a single row vector by concatenating its rows into a continuous sequence, where 1 will correspond to white in the image, and 0 to black.	51
6.3	Illustration of data representation for the bars and stripes dataset. From left to right: an example image from the dataset; the same pattern flattened into a column vector; the corresponding computational basis state represented as a column of 0 and 1 values; the one-hot encoding of each bit; and the resulting hyperindexed matrix product state (MPS) with one tensor per site. Each tensor has a physical index for the one-hot encoded bit value and a hyperindex labeling the sample, as described in the main text.	51
6.4	Samples generated from a randomly initialized MPS. The absence of recognizable patterns demonstrates that the model requires training to capture the structure of the 3x3 BAS dataset - that translates to MPS with 9 sites.	53
6.5	Training curves for MMD and NLL loss at different bond dimensions. The effect of increasing expressivity is evident in the improved loss values. Sigma value for MMD kernel function $\sigma = 0.55$. All of 14 3x3 BAS samples were used.	54

LIST OF FIGURES

6.6	Left: number of valid samples generated as a function of bond dimension, for MMD and NLL losses. MMD avoids hallucinations at all bond dimensions, while NLL requires higher expressivity to achieve full validity. Right: number of unique valid patterns generated (coverage) as a function of bond dimension. MMD is limited at low bond dimension, while NLL achieves full coverage at high expressivity.	56
6.7	Logarithm of normalized variance of the loss function at random initialization, as a function of the number of physical indices, for different bond dimensions. MMD exhibits exponential decay (barren plateau), while NLL remains nearly constant.	59
8.1	Samples generated with MMD loss, bond dimension $\chi = 2$. The model produces a single valid pattern, indicating mode collapse due to limited expressivity.	64
8.2	Samples generated with MMD loss, bond dimension $\chi = 8$. The model still produces only few valid pattern, showing that expressivity remains insufficient to capture the full dataset.	64
8.3	Samples generated with MMD loss, bond dimension $\chi = 16$. The model begins to generate a greater variety of valid patterns, though not all possible modes are covered.	65
8.4	Samples generated with MMD loss, bond dimension $\chi = 32$. The diversity of generated patterns increases, but some valid patterns remain missing, as discussed in the main text.	65
8.5	Samples generated with NLL loss, bond dimension $\chi = 2$. The model produces a mix of valid and invalid patterns, indicating hallucinations due to insufficient expressivity.	66
8.6	Samples generated with NLL loss, bond dimension $\chi = 8$. Hallucinations persist, but the number of unique valid patterns increases. .	66
8.7	Samples generated with NLL loss, bond dimension $\chi = 16$. The model generates a larger set of valid patterns, with hallucinations becoming rare.	67
8.8	Samples generated with NLL loss, bond dimension $\chi = 32$. The model successfully generates all valid patterns without hallucinations, demonstrating full coverage and high expressivity.	67

Bibliography

- [1] A. Perdomo-Ortiz, M. Benedetti, J. Realpe-Gómez, and R. Biswas, “Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers,” *Quantum Sci. Technol.*, vol. 3, p. 030502, June 2018. Publisher: IOP Publishing.
- [2] R. Sweke, J.-P. Seifert, D. Hangleiter, and J. Eisert, “On the Quantum versus Classical Learnability of Discrete Distributions,” *Quantum*, vol. 5, p. 417, Mar. 2021. arXiv:2007.14451 [quant-ph].
- [3] X. Gao, E. R. Anschuetz, S.-T. Wang, J. I. Cirac, and M. D. Lukin, “Enhancing Generative Models via Quantum Correlations,” *Phys. Rev. X*, vol. 12, p. 021037, May 2022. Publisher: American Physical Society.
- [4] G. Hinton and T. Sejnowski, “Learning and relearning in Boltzmann machines,” *Parallel Distributed Processing*, vol. 1, Jan. 1986.
- [5] J.-G. Liu and L. Wang, “Differentiable learning of quantum circuit Born machines,” *Phys. Rev. A*, vol. 98, p. 062324, Dec. 2018. Publisher: American Physical Society.
- [6] Z.-Y. Han, J. Wang, H. Fan, L. Wang, and P. Zhang, “Unsupervised Generative Modeling Using Matrix Product States,” *Phys. Rev. X*, vol. 8, p. 031012, July 2018. Publisher: American Physical Society.
- [7] M. S. Rudolph, S. Lerch, S. Thanasilp, O. Kiss, O. Shaya, S. Vallecorsa, M. Grossi, and Z. Holmes, “Trainability barriers and opportunities in quantum generative modeling,” *npj Quantum Inf.*, vol. 10, p. 116, Nov. 2024.
- [8] S. Cheng, L. Wang, T. Xiang, and P. Zhang, “Tree tensor networks for generative modeling,” *Phys. Rev. B*, vol. 99, p. 155131, Apr. 2019.

BIBLIOGRAPHY

- [9] A. Meiburg, J. Chen, J. Miller, R. Tihon, G. Rabusseau, and A. Perdomo-Ortiz, “Generative Learning of Continuous Data by Tensor Networks,” July 2024. arXiv:2310.20498 [cs].
- [10] M. Ben-Dov and J. Chen, “Regularized second-order optimization of tensor-network Born machines,” Jan. 2025. arXiv:2501.18691 [cs].
- [11] E. C. Martín, K. Plekhanov, and M. Lubasch, “Barren plateaus in quantum tensor network optimization,” *Quantum*, vol. 7, p. 974, Apr. 2023. arXiv:2209.00292 [quant-ph].
- [12] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.
- [13] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles, “Variational quantum algorithms,” *Nature Reviews Physics*, vol. 3, pp. 625–644, Sep 2021.
- [14] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, “Evaluating analytic gradients on quantum hardware,” *Physical Review A*, vol. 99, Mar. 2019.
- [15] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, “Barren plateaus in quantum neural network training landscapes,” Mar. 2018.
- [16] M. Larocca, P. Czarnik, K. Sharma, G. Muraleedharan, P. J. Coles, and M. Cerezo, “Diagnosing Barren Plateaus with Tools from Quantum Optimal Control,” *Quantum*, vol. 6, p. 824, Sept. 2022. Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften.
- [17] Z. Holmes, K. Sharma, M. Cerezo, and P. J. Coles, “Connecting Ansatz Expressibility to Gradient Magnitudes and Barren Plateaus,” *PRX Quantum*, vol. 3, p. 010313, Jan. 2022. Publisher: American Physical Society.
- [18] C. Ortiz Marrero, M. Kieferová, and N. Wiebe, “Entanglement-Induced Barren Plateaus,” *PRX Quantum*, vol. 2, p. 040316, Oct. 2021. Publisher: American Physical Society.
- [19] S. Wang, E. Fontana, M. Cerezo, K. Sharma, A. Sone, L. Cincio, and P. J. Coles, “Noise-induced barren plateaus in variational quantum algorithms,”

BIBLIOGRAPHY

- Nature Communications*, vol. 12, p. 6961, Nov. 2021. Publisher: Nature Publishing Group.
- [20] Z. Liu, L.-W. Yu, L.-M. Duan, and D.-L. Deng, “Presence and absence of barren plateaus in tensor-network based machine learning,” *Physical Review Letters*, vol. 129, Dec. 2022.
 - [21] F. Barratt, J. Dborin, and L. Wright, “Improvements to gradient descent methods for quantum tensor network machine learning,” 2022.
 - [22] Z. Liu, Q. Ye, L.-W. Yu, L. M. Duan, and D.-L. Deng, “Theory on variational high-dimensional tensor networks,” 2023.
 - [23] M. A. Alvarez, L. Rosasco, and N. D. Lawrence, “Kernels for vector-valued functions: a review,” 2012.
 - [24] J. H. Manton and P.-O. Amblard, “A primer on reproducing kernel hilbert spaces,” 2015.
 - [25] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” 2014.
 - [26] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, “A review on generative adversarial networks: Algorithms, theory, and applications,” 2020.
 - [27] B. Coyle, D. Mills, V. Danos, and E. Kashefi, “The Born supremacy: quantum advantage and training of an Ising Born machine,” *npj Quantum Inf*, vol. 6, pp. 1–11, July 2020. Publisher: Nature Publishing Group.
 - [28] M. Benedetti, D. Garcia-Pintos, O. Perdomo, V. Leyton-Ortega, Y. Nam, and A. Perdomo-Ortiz, “A generative modeling approach for benchmarking and training shallow quantum circuits,” *npj Quantum Information*, vol. 5, May 2019.
 - [29] M. Collura, G. Lami, N. Ranabhat, and A. Santini, *Tensor Network Techniques for Quantum Computation*. SISSA Medialab s.r.l., Dec. 2024.
 - [30] wwwtensornetwork.org, “Sampling from mps,” 2023.
 - [31] U. Schollwöck, “The density-matrix renormalization group in the age of matrix product states,” *Annals of Physics*, vol. 326, pp. 96–192, Jan. 2011.

BIBLIOGRAPHY

- [32] I. Glasser, R. Sweke, N. Pancotti, J. Eisert, and I. Cirac, “Expressive power of tensor-network factorizations for probabilistic modeling,” in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.
- [33] E. M. Stoudenmire and S. R. White, “Minimally entangled typical thermal state algorithms,” *New Journal of Physics*, vol. 12, p. 055026, May 2010.
- [34] A. J. Ferris and G. Vidal, “Perfect sampling with unitary tensor networks,” *Phys. Rev. B*, vol. 85, p. 165146, Apr 2012.
- [35] F. Verstraete and J. I. Cirac, “Renormalization algorithms for quantum-many body systems in two and higher dimensions,” 2004.
- [36] J. Gray, “quimb: a python library for quantum information and many-body calculations,” *Journal of Open Source Software*, vol. 3, no. 29, p. 819, 2018.
- [37] J. Gray and S. Kourtis, “Hyper-optimized tensor network contraction,” *Quantum*, vol. 5, p. 410, Mar. 2021.