Quantum State Tomography with a fully Spiking Variational Autoencoder

On Neuromorphic Hardware

Computer and Embedded Systems Engineering: Master Thesis

Jim van Leeuwen

Delft University of Technology

TU Delft

Quantum State Tomography with a fully Spiking Variational Autoencoder

On Neuromorphic Hardware

by

Jim van Leeuwen

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Tuesday February 18, 2024 at 09:45 AM.

Student number:	5409837	
Project duration:	April 22, 2024 – February 18, 2025	
Thesis committee:	Assoc. Prof. R. Ishihara,	TU Delft, supervisor
	Asst. Prof. R. Bishnoi,	TU Delft, Core Member 2
	Asst. Prof. H. Abunahla,	TU Delft, Core Member 3
	E. Hua,	TU Delft, Advisor

Cover:	Microsoft Bing Image Creator
Prompt:	A quantum computer connected to a neuromorphic computing platform that runs a neural network to perform quantum state tomography

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.
All source code of this thesis is available at `https://github.com/Jameylion/QSVAE/`.

TU Delft

Preface

I would like to thank the research group for their guidance throughout this thesis project. In particular, I am grateful to **Assoc. Prof. R. Ishihara** for his support and valuable insights. I also appreciate **Asst. Prof. R. Bishnoi** and **Asst. Prof. H. Abunahla** for their feedback and encouragement. A special thanks to **Erbing Hua** for helping to add depth to the research and to **Elias Arnold** for assisting with technical challenges related to BrainScaleS-2. I also appreciate **Stefanie Czischek** for sharing her code and providing valuable input on my research ideas. Finally, I would like to thank my parents and my girlfriend for their support throughout this journey.

Jim van Leeuwen
Delft, February 2025

Summary

As quantum systems increase in complexity, accurately reconstructing quantum states becomes a fundamental challenge. Quantum state tomography (QST) provides a framework for reconstructing quantum states from experimental measurements. However, the computational resources required for QST scale exponentially with the number of qubits as the Hilbert space dimension grows as $N = 2^n$, where n is the number of qubits and N the resulting probability amplitudes. This exponential growth renders traditional approaches computationally prohibitive for systems beyond a few qubits. Recent advancements in machine learning have introduced neural network-based approaches to QST, such as variational autoencoders (VAEs) and restricted Boltzmann machines (RBM). These methods leverage the representational power of neural networks to approximate high-dimensional quantum states efficiently. However, the energy-intensive nature of artificial neural networks (ANNs) poses scalability concerns, particularly as quantum systems grow larger. Recent advancements in machine learning have introduced neural network-based approaches to QST, such as VAE and RBM. These methods leverage the representational power of neural networks to approximate high-dimensional quantum states efficiently. Neuromorphic computing offers a biologically inspired paradigm for addressing the challenges of QST. This event-driven architecture allows for asynchronous computation with significantly lower energy consumption compared to traditional neural networks. The RBM trained on the BrainScales-2 (BS2) Neuromorphic platform exhibited low fidelity because of the limited neuron availability and the 6-bit weights on the BS2. The novelty of this thesis is the idea that a variational autoencoder can be split on the level encoder and decoder and let the encoder outside of the BS2 use a bigger model. This results in the question of how a fully spiking variational autoencoder (FSVAE) can be effectively implemented on neuromorphic hardware to achieve high fidelity and scalability for quantum state tomography. Firstly, a quantum state is prepared using unitary operators, and S (also known as Shots) times are measured to create a dataset to interconnect the Qubit information with the FSVAE. This dataset consists of a set of one hot encoded vectors that represent the quantum state and are scaled by 4N. Lastly, the FSVAE consists of an encoder that is built on the CPU and a decoder that is built on the BS2. The results came from implementing these methods by training the FSVAE (encoder-decoder) on a CPU-CPU configuration and a CPU-BS2 configuration. Two experiments were conducted to compare the performance of BS2 in the configuration mentioned earlier. The CPU-CPU configuration could be trained for 3 to 7, and the CPU-BS2 configuration could be trained for 2 to 5 qubits in the Greenberger–Horne–Zeilinger (GHZ) state. The two configurations are compared, and the MSE loss did not converge on the CPU-BS2 configuration as low as the CPU-CPU configuration. The discrepancy results from BS2 having 6-bit weights and the CPU 32-bit weights. Despite this difference, the fidelity of reconstructing by the FSVAE of 4 qubits is improved by around 20% compared to the RBM architecture.

Contents

Preface i

Summary ii

Nomenclature v

1 Introduction 1
- 1.1 Challenges in Quantum State Tomography . . . 1
- 1.2 Neuromorphic Computing as a Solution . . . 1
- 1.3 Research Focus and Objectives . . . 2
- 1.4 Structure of the Thesis . . . 2

2 Background 3
- 2.1 Qubits . . . 3
- 2.2 Quantum State Tomography . . . 4
  - 2.2.1 Quantum State Tomography Methods . . . 5
- 2.3 Artificial Neural Networks . . . 8
  - 2.3.1 (Multi-Layer) Perceptron . . . 8
  - 2.3.2 Forward-Pass and Backpropegation . . . 8
  - 2.3.3 Loss Functions in ANN . . . 9
  - 2.3.4 Recurrent Networks (RNN) . . . 10
  - 2.3.5 Convolutional Neural Networks (CNNs) . . . 10
  - 2.3.6 Spiking Neural networks . . . 12
  - 2.3.7 Hybrid Spiking Neural Network on neuromorphic platform . . . 15
  - 2.3.8 BrainScaleS-2 . . . 15
  - 2.3.9 Spiking RBM . . . 18
  - 2.3.10 Spiking Variational auto-encoder (SNNs) . . . 18

3 Proposed Method 19
- 3.1 Creating the dataset . . . 19
  - 3.1.1 SIC POVM . . . 19
  - 3.1.2 Two-Qubit POVM Example . . . 20
  - 3.1.3 Input Vector Creation and One-Hot Encoding . . . 21
  - 3.1.4 encoder . . . 22
  - 3.1.5 Latent Space Construction . . . 23
  - 3.1.6 Loss Function . . . 24
  - 3.1.7 Fidelity Calculation in the Quantum State Tomography Model . . . 25
  - 3.1.8 Density Matrix Reconstruction . . . 26

4 Results 28
- 4.1 Spiking poisson on CPU . . . 29
- 4.2 Spiking poisson on BS2 . . . 30
- 4.3 Comparison between CPU-CPU and CPU-BS2 Implementation . . . 31
- 4.4 Comparison with State of the Art . . . 32

5 Conclusion 33
- 5.1 Summary of Contributions . . . 33
- 5.2 Findings and Impact . . . 33
- 5.3 Future Work . . . 33

References 34

A Calculations **37**
A.1 Calculations Measurement operators 37

B Other results **39**
B.1 Initial experiment 39
B.2 Results of the initial SVAE 41
B.3 Training details per qubit 42
B.4 BS2 optimisations 44
B.4.1 Result of training 3 qubits 44
B.4.2 Result of 3 qubits with increased hidden size 45
B.4.3 Result of 3 qubits with increased latent size 46

Nomenclature

Abbreviations

Abbreviation	Definition
BS2	BrainScaleS-2 neuromorphic hardware
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
CPU	Central Processing Unit
GHZ	Greenberger–Horne–Zeilinger state
IC	Informationally Complete
KL	Kullback-Leibler divergence
LIF	Leaky Integrate-and-Fire neuron
MLE	Maximum-Likelihood Estimation
MPS	Matrix Product State
MMD	Maximum Mean Discrepancy
QST	Quantum State Tomography
RBM	Restricted Boltzmann Machine
SIC	Symmetric Informationally Complete
POVM	Positive Operator-Valued Measurement
SNN	Spiking Neural Network
SVAE	Spiking Variational Autoencoder
VAE	Variational Autoencoder

Symbols

Symbol	Definition	Unit
Latin Symbols		
F	Fidelity between two quantum states	—
$P(a)$	Probability of measurement outcome a	—
$M(a)$	POVM element for measurement outcome a	—
$V(t)$	Membrane potential at time t	[V]
$I(t)$	Input current at time t	[A]
C_m	Membrane capacitance	[F]
z	Latent space variable	—
T	Total time steps in SNN simulation	—
Greek Symbols		
α, β	Probability amplitudes of qubit states	—
$\sigma_x, \sigma_y, \sigma_z$	Pauli matrices	—
Θ	Heaviside step function	—
$\|\psi\rangle$	Quantum state vector	—
ρ	Density matrix of a quantum state	—
τ_m, τ_s	Membrane and synaptic time constants	[s]

1 Introduction

Qubits, the computational units of a quantum computer, determine not only quantum memory but also its computational power. The number of qubits in quantum processors is growing, and companies like Google and IBM are racing to have the most qubits. This growth heralds a new era of computational power, where quantum computers will tackle problems that classical machines can never hope to solve in a reasonable time. Unlike classical bits, qubits leverage the principles of superposition and entanglement, enabling exponentially larger state spaces and computational power [1, 2].

As quantum systems increase in complexity, accurately reconstructing quantum states becomes a fundamental challenge. Quantum state tomography (QST) provides a framework for reconstructing quantum states from experimental measurements [2, 3]. However, the computational resources required for QST scale exponentially with the number of qubits, as the Hilbert space dimension grows as $N = 2^n$, where n is the number of qubits. This exponential growth renders traditional approaches computationally prohibitive for systems beyond a few qubits [4, 5].

1.1. Challenges in Quantum State Tomography

Traditional methods such as Maximum Likelihood Estimation (MLE) and Bayesian Estimation have been successful for small systems, ensuring physical validity and incorporating prior knowledge into quantum state reconstructions [3, 4]. However, these methods face several key challenges:

- Scalability: Classical methods struggle to handle the high-dimensional parameter space of large quantum systems efficiently [2, 6].
- Noise Sensitivity: Noise in quantum devices often degrades the performance of traditional algorithms, particularly in experimental settings [7, 8].
- Energy and Computational Cost: The resource demands of classical algorithms make them impractical for noisy intermediate-scale quantum (NISQ) devices [9].

Recent advancements in machine learning have introduced neural network-based approaches to QST, such as variational autoencoders (VAEs) [10] and restricted Boltzmann machines (RBM) [11, 12]. These methods leverage the representational power of neural networks to approximate high-dimensional quantum states efficiently. However, the energy-intensive nature of artificial neural networks (ANNs) poses scalability concerns, particularly as quantum systems grow larger [13].

1.2. Neuromorphic Computing as a Solution

Neuromorphic computing offers a new computation paradigm inspired by biological systems and is well-suited to address the challenges posed by QST. Spiking neural networks (SNNs), implemented on neuromorphic hardware such as BrainScaleS-2 (BS2), mimic the event-driven dynamics of biological neurons [14]. This event-driven architecture allows for asynchronous computation with significantly lower energy consumption compared to traditional neural networks [15, 16].

The hybrid analog-digital design of BS2 provides a unique platform for efficiently processing spiking neuron computations, which align naturally with the probabilistic nature of quantum mechanics [7, 12]. Recent work demonstrates that spiking neuromorphic systems can represent quantum states with high fidelity, offering a promising alternative to classical and ANN-based methods for QST [13].

1.3. Research Focus and Objectives

This thesis explores the integration of spiking neural networks with the BS2 neuromorphic platform to develop a scalable, energy-efficient solution for quantum state tomography. The primary contribution is a hybrid fully spiking variational autoencoder (FSVAE), where:

- The encoder, implemented on a classical CPU, processes quantum measurement data into a latent spiking representation.
- The decoder, implemented on BS2, reconstructs quantum states from the latent space using spiking neuron dynamics.

The research addresses the following question:

> *How can a spiking variational autoencoder be effectively implemented on neuromorphic hardware to achieve high fidelity and scalability for quantum state tomography?*

To address this, the following objectives are pursued:

- Develop a hybrid FSVAE framework that combines spiking neural networks with the BS2 platform [17, 18].
- Achieve fidelity exceeding 80% for reconstructing 3- and 4-qubit quantum states, surpassing benchmarks of RBM-based QST on BS2 [7].
- Optimize loss functions, latent space representations, and BS2 calibration to enhance robustness and scalability [10, 13].

1.4. Structure of the Thesis

The thesis is structured as follows:

- Chapter 2: A detailed background and review of related work on spiking neural networks, variational autoencoders, and the BrainScaleS-2 platform.
- Chapter 3: Description of the proposed FSVAE architecture and the methods for dataset preparation, training, and evaluation.
- Chapter 4: Experimental results demonstrating the performance of the FSVAE on CPU and BS2 platforms.
- Chapter 5: Summary of findings, implications, and directions for future research.

By integrating quantum computing, neuromorphic hardware, and spiking neural networks, this thesis aims to advance the field of quantum state tomography with a scalable and energy-efficient solution.

.

2

Background

This project is a ligature of multiple scientific principles that need some explanation in order to understand why these concepts here meet. The main objective is to improve Quantum State Tomography (QST) by leveraging the properties of Spiking Neural Networks (SNN) deployed on Neuromorphic hardware and compare findings with existing results. This chapter explains the different scientific concepts needed to set up the experiment later discussed in chapter 3.

2.1. Qubits

Qubits and quantum gates are the core computational blocks of a quantum computer and quantum logic gates. Qubits always exist in a state of superposition of outcomes before being measured with a particular quantum gate. The Bloch sphere in Fig. 2.1 is a visualization of the Hilbert space that describes a quantum state with a size of N = 2. The Bloch sphere can represent any state of a qubit $|\psi\rangle$ with form:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$ and where α, β are the probability amplitudes of the qubit being in the state $|0\rangle$ or $|1\rangle$, respectively.

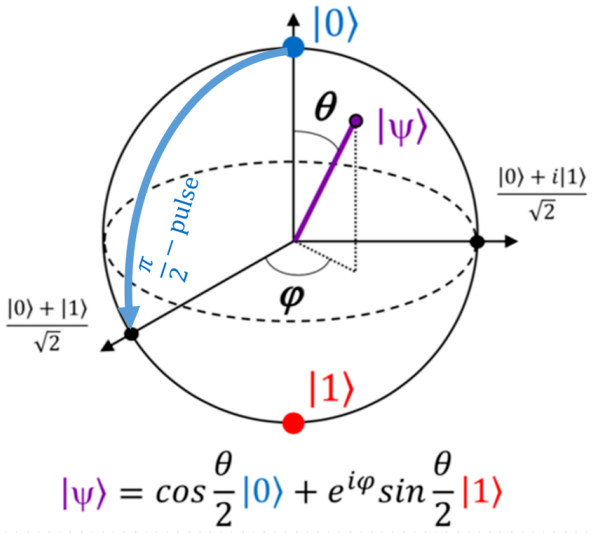


Figure 2.1: Bloch sphere

Quantum logic gates can be used to manipulate qubits in order to build quantum circuits. The expectation value is calculated in order to project a measurement on the Pauli matrices. The expectation value

of an operator M is defined as:

$$\langle M \rangle = \langle \psi | M | \psi \rangle$$

Where $M \in \mathbb{C}^{N*N}$ and $\psi \in \mathbb{C}^N$. Here are some of the gates that are used to create quantum states later:

- CNOT Gate (Controlled NOT Gate): Acts on two qubits, flips the target qubit if the control qubit is $|1\rangle$.
  $$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$
- Rotation Gates: The angle parameter (θ, ϕ, γ) defines the degree of rotation.
  $$R_x(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix},$$
  $$R_y(\phi) = \begin{bmatrix} \cos\frac{\phi}{2} & -\sin\frac{\phi}{2} \\ \sin\frac{\phi}{2} & \cos\frac{\phi}{2} \end{bmatrix},$$
  $$R_z(\gamma) = \begin{bmatrix} e^{-i\gamma/2} & 0 \\ 0 & e^{i\gamma/2} \end{bmatrix}$$
- Pauli-X Gate (X): $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
  Swaps $|0\rangle$ and $|1\rangle$.
- Pauli-Y Gate (Y): $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
  Swaps $|0\rangle$ and $|1\rangle$ with a phase factor.
- Pauli-Z Gate (Z): $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
  Leaves $|0\rangle$ unchanged and flips the sign of $|1\rangle$.

2.2. Quantum State Tomography

Quantum State Tomography (QST) is the process of reconstructing a quantum state by measuring the same quantum state repeatedly until the reconstructed state is found. The QST experiment is repeated until a marginal distribution is found that consists of a list of all possible outcomes of the experiment and the number of times the experiment has resulted in that outcome. We take for example the bell state $|\phi^+\rangle$, which is one of the maximally entangled states for two qubits. The $|\phi^+\rangle$ state is defined as:

$$\left|\phi^+\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{2.1}$$

The Born rule tells us that the probabilities of the possible outcomes $|00\rangle$ or $|11\rangle$ in the computational basis of $|\phi^+\rangle$, are the probability amplitudes squared of the normalised coefficients belonging to those particular outcomes [19].In this case the probability of finding the bell state in $|00\rangle$ or $|11\rangle$ is $|\frac{1}{\sqrt{2}}|^2$ = $\frac{1}{2}$. To find this probability experimentally you would have to do repeated measurements until you find a marginal distribution. Consider this analogy: take flipping a coin that has probability 50/50 landing head/tail. Flipping the coin a thousand times will give you an approximation of the probability distribution P(x=t) = P(x=h) = 0.5 and the same can be said about the quantum state.

This example shows that it is very easy to see the probabilities of a quantum system that consists of two qubits unfortunately that changes when generalising this example to N qubits. In Fig 2.2A is an ideal 2 qubit bell state seen with 4 amplitudes that show probabilities of that all the combinations of qubit states. Although the 4 qubit GHZ state in Fig. 2.2C is also relatively easy to decipher what the original state is from the picture, this will be much harder when there are more qubits. The noise seen in Fig. 2.2B and D makes it even harder for classic computational processes to like maximum-likelihood estimation, Bayesian estimation and Matrix Product State (MPS) [3, 4, 20].

This reconstructed state is an approximation of the original state and the quality of the process is defined by the fidelity. The fidelity of a quantum state is how similar one quantum state is to another quantum state. The parameters and amount of measurements increases exponentially as the amount of qubits (N) in a system increases linearly [1]. The exponential growth is a problem when we want to use a classical computer to characterise these quantum states of interest.

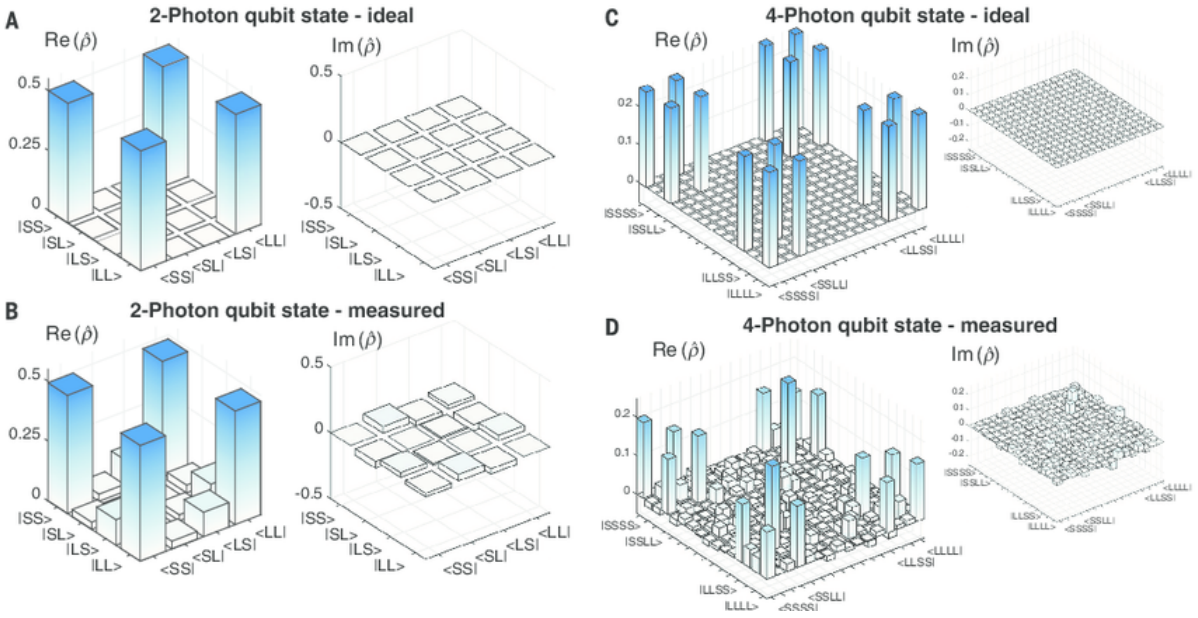


Figure 2.2: Density matrix 2 and 4 qubit GHZ state [21]

2.2.1. Quantum State Tomography Methods

Quantum state tomography (QST) is essential to building reliable quantum computers and is used to determine the full quantum state of a system. However, as the Hilbert space grows, exact QST becomes infeasible due to the exponentially increasing number of measurements and computations required. Classical methods like maximum-likelihood estimation, Bayesian estimation, and Matrix Product State (MPS) have been employed to tackle this challenge, though each faces limitations as system size grows.

Maximum-Likelihood Estimation (MLE)

Maximum-Likelihood Estimation (MLE) is a widely used method for quantum state tomography that reconstructs the quantum state by optimizing a likelihood function based on the measurement data [3]. This method ensures the reconstructed density matrix is both physical (positive semi-definite and normalized) and consistent with experimental results. While MLE is effective for small systems, its computational cost grows exponentially with the number of qubits, limiting its scalability to larger quantum systems [2].

Measurement Data and Born's Rule In QST, the quantum system is probed using a series of measurements corresponding to a Positive Operator-Valued Measure (POVM) $\{M_a\}$, where each measurement outcome a is associated with a positive semidefinite operator M_a. The probability of observing the outcome a for a state ρ is given by Born's rule [3]:

$$P(a|\rho) = \mathrm{Tr}(\rho M_a). \tag{2.2}$$

Here, Tr denotes the matrix trace. The measurement outcomes $\{a_i\}$, collected over many trials, form the dataset used to reconstruct ρ.

Likelihood Function The likelihood function indicates the probability of observing the entire dataset based on specific parameters for a quantum state ρ. For N measurement outcomes $\{a_1, a_2, \ldots, a_N\}$, the likelihood function can be written as:

$$L(\rho) = \prod_{i=1}^{N} P(a_i|\rho). \tag{2.3}$$

Taking the logarithm, the log-likelihood becomes:

$$\mathcal{L}(\rho) = \sum_{i=1}^{N} \ln P(a_i|\rho). \tag{2.4}$$

Optimization The goal of MLE is to find the density matrix ρ that maximizes the likelihood function $\mathcal{L}(\rho)$, subject to the physical constraints:

1. $\rho \succeq 0$ (positive semidefiniteness),
2. $\mathrm{Tr}(\rho) = 1$ (trace normalization).

These constraints ensure that ρ is a valid quantum state. The optimization problem can be stated as:

$$\rho_{\mathsf{MLE}} = \arg\max_{\rho} \mathcal{L}(\rho). \tag{2.5}$$

Iterative Procedure Due to the complexity of the constraints, the optimization is typically performed using iterative algorithms, such as:

- Expectation-Maximization (EM) algorithms, which iteratively refine ρ to increase the likelihood [4].
- Gradient-based methods, which compute updates for ρ while ensuring the constraints are satisfied.

Advantages of MLE MLE offers several advantages:

- It produces a physically valid density matrix even in the presence of noisy measurement data [4].
- It incorporates the statistical properties of the measurement outcomes to provide an optimal estimate of ρ.
- The reconstruction scales efficiently for small quantum systems, though it becomes computationally demanding for larger systems [2].

Bayesian Estimation

Bayesian estimation provides an alternative approach by incorporating prior knowledge of the quantum state into the estimation process [4]. By updating prior beliefs based on experimental data, Bayesian methods provide a probabilistic framework for quantum state reconstruction. This approach is particularly useful in noisy environments, as it can mitigate uncertainty and error. However, similar to MLE, Bayesian estimation is computationally expensive for large systems due to the need for high-dimensional integration [9].

Bayesian Framework In Bayesian estimation, the goal is to compute the posterior probability distribution of the density matrix ρ given the measurement data $\mathcal{D}$. Using Bayes' theorem, the posterior distribution is expressed as:

$$P(\rho|\mathcal{D}) = \frac{P(\mathcal{D}|\rho)P(\rho)}{P(\mathcal{D})}, \tag{2.6}$$

where:

- $P(\rho|\mathcal{D})$: The posterior distribution, representing the probability of ρ given the data $\mathcal{D}$.
- $P(\mathcal{D}|\rho)$: The likelihood function, capturing the probability of observing the data $\mathcal{D}$ for a given state ρ.
- $P(\rho)$: The prior distribution, encoding prior knowledge or assumptions about ρ.
- $P(\mathcal{D})$: The marginal likelihood or evidence, ensuring normalization of the posterior.

Likelihood Function The likelihood function $P(\mathcal{D}|\rho)$ is derived from the measurement outcomes and Born's rule:

$$P(\mathcal{D}|\rho) = \prod_{i=1}^{N} P(a_i|\rho), \tag{2.7}$$

where $P(a_i|\rho) = \mathrm{Tr}(\rho M_{a_i})$ is the probability of observing outcome a_i under the density matrix ρ and the corresponding POVM element M_{a_i} [2].

Advantages of Bayesian Estimation Bayesian estimation provides several benefits:

- It quantifies the uncertainty in the reconstructed state, providing credibility intervals for ρ [4].
- Incorporating prior information allows Bayesian methods to handle noisy or incomplete data more effectively.
- Priors can be tailored to specific problems or experimental setups, enabling domain-specific inference [9].

Matrix Product State (MPS)
Matrix Product State (MPS) methods provide a structured and compact way to represent quantum states in systems with limited entanglement. MPS methods efficiently approximate states by factorizing the quantum wave function into a chain of tensors [2]. This representation is especially beneficial for systems that comply with area laws of entanglement or that display one-dimensional structures.

Advantages of MPS in QST MPS offers several advantages:

- It significantly reduces computational complexity for systems with low entanglement, scaling linearly with system size.
- Adjustable bond dimensions allow a trade-off between accuracy and computational efficiency [2].
- It aligns well with tensor network techniques, enabling efficient simulation of many-body quantum systems.

Limitations of MPS Despite its strengths, MPS has limitations:

- Highly entangled states require large bond dimensions, reducing computational efficiency.
- MPS methods are best suited for 1D systems, with extensions like PEPS required for higher dimensions [2].

2.3. Artificial Neural Networks

Artificial Neural networks (ANN) are quite known amongst people since the rise of chat-gpt and other deep generative models. At the hart of these models lies a networks of nodes called perceptrons. The perceptron was invented by Warren McCulloch and Walter Pitts in 1943 [22]. This invention was based on the biological neuron and was build in the Mark 1 Perceptron machine build in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt. The Mark 1 had 400 photocells as input nodes, 512 perceptrons in the hidden layer, 8 perceptrons as output layer, weighed 5 tons and was the size of a room(!). Frank Rosenblatt described the perceptron as a system which operates according to probabilistic principles instead of deterministic principles and gains it reliability from the properties of statistical measurements from large populations of elements. This description can be interpreted as the foundation on which modern machine learning is based off [23].

2.3.1. (Multi-Layer) Perceptron

Mathematically, the perceptron can be seen as a linear function passed to an activation function. The linear function takes some vector x as input and multiplies x with a weight w and sum all the outcomes, accumulate the bias b:

$$\hat{y} = f\left(\sum_{i=1}^{n} w_i x_i + b\right) = f(\mathbf{x}^\top \mathbf{w} + b) \tag{2.8}$$

where $w \in \mathcal{R}^n$, $x \in \mathcal{R}^n$, $b \in \mathcal{R}$ and n is the number of inputs in the perceptron as seen in Fig 2.3. The activation function $f(x)$introduces a non linearity to solve problems that are not linear classifiable per nature.

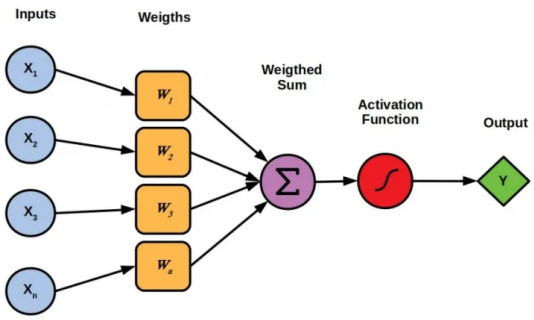


Figure 2.3: Perceptron [24]

Mltiple perceptrons are connected in parallel, what result in a layer, and after stacking the layers, the multi layered perceptron (MLP) arises that is also known as a fully connected (FC) NN or model. The function of the MLP network is defined by passing function in Eq. 2.8 in a chain so it becomes:

$$\hat{y} = f(x) = f^{(3)}(f^{(2)}(f^{(1)})) \tag{2.9}$$

Where $f^{(1)}$ is called the first layer or input layer, $f^{(2)}$ is called the second layer or hidden layer and $f^{(3)}$ is called the third layer or output layer. These layers can be added or remove according to the desires of the developer to create the network that fits the nature of the data.

2.3.2. Forward-Pass and Backpropegation

Passing a value x to the first layer and through the chain of functions in Eq. (2.9) is called a forward pass. The result from the forward pass is compared with the label of that value during training and the difference is called the loss. For example, x is a picture of a cat and the label y is the string ”cat”. If the forward pass of the model predicts the word ”dog” $(\hat{y})$ the loss would be bigger then 0 because the prediction ”dog” is different then the label ”cat”. The loss is then used to change all the weights in

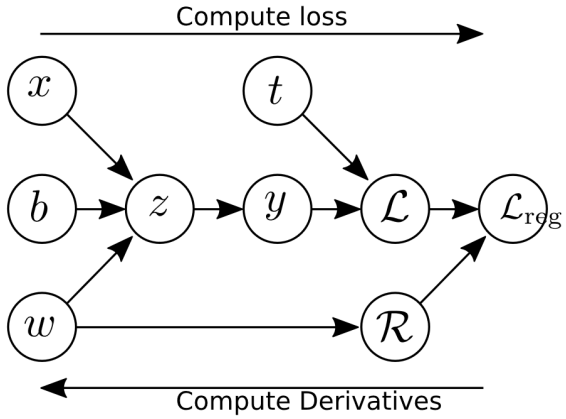


Figure 2.4: Enter Caption

the system for some fraction to make the next prediction to have a lower loss. The process is called backpropegation and computes essentially the chain rule of derivatives of the loss [25]:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}. \tag{2.10}$$

If we take Eq. 2.8 and we take some model:

$$\mathcal{L} = \frac{1}{2} \left(\sigma(wx + b) - t\right)^2 \tag{2.11}$$

where $\mathcal{L} = \frac{1}{2}(y - t)^2$, $z = wx + b$, $y = \sigma(z)$ for some activation function σ. The function $f(x)$ is differentiated to w and b using the formula from 2.10 and use Leipniz notation, results in:

$$\frac{d\mathcal{L}}{dw} = \frac{d\mathcal{L}}{dy}\frac{dy}{dz}\frac{dz}{dw} \tag{2.12}$$

$$\frac{d\mathcal{L}}{db} = \frac{d\mathcal{L}}{dy}\frac{dy}{dz}\frac{dz}{db} \tag{2.13}$$

The methods of passing values through the network and backpropagation of the loss give the network its learning power that is leveraged to learn quantum states.

2.3.3. Loss Functions in ANN

During training, the goal is to minimize the loss, which ensures that the model learns to perform better on its task. Loss functions typically quantify differences between predicted and actual values or distributions, providing feedback that guides the optimization process through backpropagation.

Types of Loss Functions

Different types of loss functions are tailored for specific tasks and applications. Some of the most commonly used loss functions include:

Mean Squared Error (MSE) The Mean Squared Error (MSE) is widely used for regression tasks. It measures the average squared difference between predicted values $\hat{y}_i$ and actual values y_i, ensuring that larger errors are penalized more heavily:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

This loss function is crucial for minimizing reconstruction errors in models such as autoencoders, where the goal is to approximate input data.

Kullback-Leibler (KL) Divergence KL divergence is a loss function used to compare two probability distributions, P and Q. It measures how one probability distribution diverges from a reference distribution:

$$D_{\mathsf{KL}}(P\|Q) = \sum_i P(i) \ln\left(\frac{P(i)}{Q(i)}\right)$$

KL divergence is particularly important in generative models like VAEs, where the objective is to ensure that the learned latent space distribution aligns with a prior distribution, such as a Gaussian.

Maximum Mean Discrepancy (MMD) The Maximum Mean Discrepancy (MMD) measures the difference between two distributions based on their samples. Using a kernel function, such as the Radial Basis Function (RBF), MMD computes:

$$\mathsf{MMD}^2(P,Q) = \mathbb{E}_{x,x'\sim P}[k(x,x')] + \mathbb{E}_{y,y'\sim Q}[k(y,y')] - 2\mathbb{E}_{x\sim P, y\sim Q}[k(x,y)]$$

This loss function is especially useful in ensuring that two distributions are similar.

2.3.4. Recurrent Networks (RNN)

RNNs have a wide range of applications. In the context of this thesis, RNNs are utilized for quantum state tomography, a process that involves reconstructing the properties of a quantum state based on sequential measurement data. A scalable recurrent machine learning procedure exists that identifies the ground states of Hamiltonians relevant to condensed matter, cold atomic systems, and quantum simulators. This method leverages Positive-Operator Valued Measures (POVMs) to describe the system, enabling the extraction of critical quantum properties such as fidelity and Kullback-Leibler (KL) divergence to benchmark the model's performance [26]. Unlike traditional methods such as Restricted Boltzmann Machines (RBMs), which often require significant computational resources, this RNN approach offers a more efficient training process, with a linear scaling in the number of sample measurements relative to the system size. This efficiency makes RNNs particularly appealing for practical implementations of QST in complex quantum systems. Furthermore, its adaptability across different quantum platforms, including simulators and experimental setups, demonstrates its versatility and robustness in quantum applications.

Others approach the many-body quantum state computing problem by utilizing attention-based models, which have been shown to outperform traditional methods such as the maximum-likelihood estimation (MLE) process used in IBM's quantum computers [8]. These attention mechanisms allow the model to focus on key features of the quantum system, improving its ability to reconstruct quantum states with high fidelity. The attention-based framework excels in capturing long-range correlations within many-body systems, making it particularly useful for understanding complex quantum interactions. Additionally, the efficiency of these models in handling large datasets suggests that they could play a pivotal role in advancing scalable quantum state tomography, especially as the number of qubits in experimental systems continues to grow.

2.3.5. Convolutional Neural Networks (CNNs)

Convolutional Neural Network (CNN) methods have also been employed to reproduce quantum states with higher fidelity compared to traditional Stokes reconstruction techniques [11]. By leveraging the hierarchical structure of CNNs, these methods can capture intricate patterns in measurement data that are often missed by classical techniques. This allows for a more accurate reconstruction of quantum states, even in the presence of noise or incomplete data. Furthermore, CNN-based approaches are inherently parallelizable, making them suitable for deployment on high-performance computing platforms. The adaptability of CNNs to various types of quantum systems, ranging from photonic qubits to superconducting circuits, underscores their potential as a versatile tool in quantum state estimation.

The problem with VAE

Later in chapter 3 we will create a dataset what we now will call $X = \{x^{(i)}\}_{i=1}^N$ consisting of N i.i.d samples of discrete variable x that result from the quantum measurement dataset and an unobserved random variable z. The process consists of two steps: (1) a value $z^{(i)}$ is generated from some prior distribution $p_{\theta^*}(z)$. The prior distribution is a distribution that is familiar like the Gaussian distribution;

(2) a value $x^{(i)}$ is generated from some conditional distribution $p_{\theta^*}(x|z)$ called the likelihood distribution. Like the authors of [27], we introduce a recognition model $q_\phi(z|x)$ to approximate the intractable probability distribution of a quantum state or true posterior $p_\theta(z|x)$. The recognition model $q_\phi(z|x)$ encodes the datapoint x into a distribution over the possible values of the code z from which the datapoint could have been generated. Likewise, given a code z, $p_\theta(z|x)$ decodes a distribution over the possible values of x. Later in chapter 3 the encoder and decoder will be the parts of the VAE that are made up from two neural networks.

Loss function

The loss function of a VAE according to [27] is defined as:

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = -D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \parallel p_\theta(\mathbf{z})\right) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right] \tag{2.14}$$

Where the first RHS equation is the Kullback-Leibler (KL) divergence that compares two distributions and calculate how close they are to each other. The network learns the probability distribution of the quantum state using the KL divergence of the model:

$$D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \parallel p_\theta(\mathbf{z})\right) = \sum_x q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \ln\left(\frac{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}{p_\theta(\mathbf{z})}\right) \tag{2.15}$$

where $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ is the target distribution and $p_\theta(\mathbf{z})$ is the model distribution from the network. The second RHS term of Eq. 2.14 penalises the difference in the outcome in the approximated posterior and the true posterior. In our case that would be the generated quantum state probabilities and the true quantum state probabilities.

Reparametrisation trick

As explained, the VAE exists of an encoder with parameters ϕ and a decoder with parameters θ. This model uses probability distributions in the loss function instead of continuous variables like a classic MLP. The problem with this method is that backpropegation is now not possible due to the non tractable differentiation of the expected value of the second RHS term in Eq. 2.14. A random variable is generated from the datapoint x ($z \sim q_\phi(z|x)$) and can be transformed using a differentiable transformation $g_\phi(\epsilon, x)$ with a noise variable ϵ taken from the distribution $p(\epsilon)$. Now the random variable z is made deterministic and mostly in the form of a vector called the latent vector or latent space. The decision to sample z from the Gaussian distribution: $z \sim q_\phi(z|x) = \mathcal{N}(\mu.\sigma^2)$ also means that we take the noise variable ϵ from this distribution and that means the we define $z = \mu + \sigma\epsilon$. To decision to sample z from the Gaussian distribution is dependant of the nature of the distribution of $q_\phi(z|x)$. The choice of distribution is important regarding the inherent discreet nature of SNNs.

2.3.6. Spiking Neural networks

Neuromorphic computing

Neuromorphic computing mimics the functionality of the human brain to learn and process information. The brain can learn very fast and use multiple orders of magnitude lower energy then a von Neumann CPU architecture would. [28, 29]. To keep using deep learning models in the future, a shift in the way we use computers to learn and perform inference must change as the International Energy Agency (IEA) claims that energy consumption of AI will use up to 6% of worldwide energy consumption [30].

Neurons

Neurons are nerve cells that use electrical and chemical signals to carry information around in the brain. The neuron consist of a body, an axon and dendrites and synapses. Synapses are placed at the end of an axon and use neurotransmitters to send information to the next neuron. The sensitivity of the synapses can change and this can be seen as the weights of the neuron. The electrical properties of the neuron's cell membrane can be modelled to fit electrical components and in this way its possible to translate a biological neuron to a neuron that can be build using existing CMOS technology. The membrane consists of a lipid bilayer and insulates the inside of the cell from the outside what results in the cell having a capacitance of around $1\mu F/cm^2$. Neurons uses this capacitance to create a gradient in voltage by pumping three sodium ions out of the cell and two potassium ions into the cell. This sodium-potassium pumps are chained along the membrane and a voltage potential across the cell membrane is propagated by the pumps pumping in a sequential fashion. In in Fig. 2.5 that the resting potential of the neuron's membrane is around 70mV and as the sodium potassium pump is stimulated to pump, a spike in membrane potential is seen and this stimulates the next pump etc.

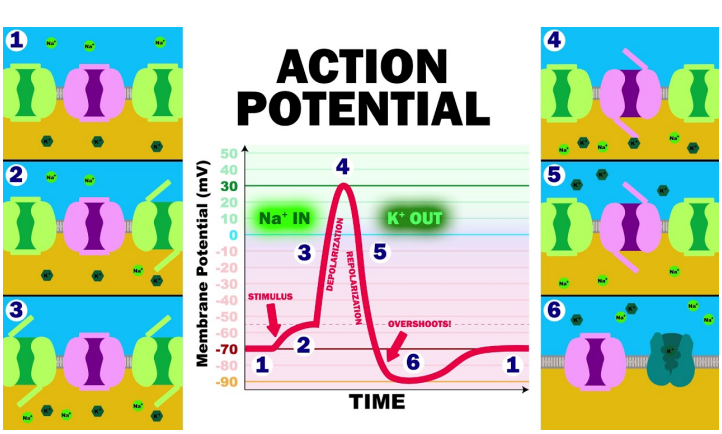


Figure 2.5: Membrane potential spike caused by sodium potassium pump [31]

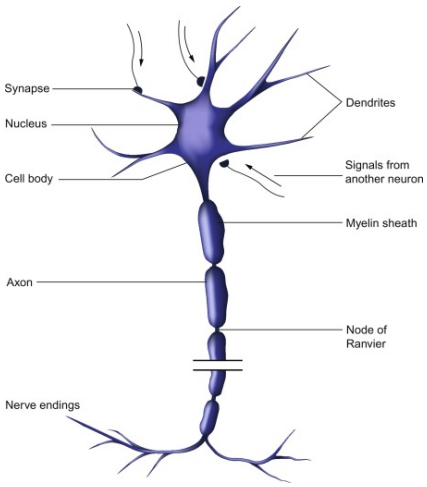


Figure 2.6: Neuron anatomy [32]

Electrical Neurons

We discussed in the previous section that the invention of the perceptron was modelled after the biological neuron. This mathematical model follows a continuous linear function that is nicely differentiable using the chain rule of calculus. The Spiking Neural Network is also modelled after the biological neuron but now the perceptron we now call a neuron and instead of continuous functions we now make use of discrete spike trains and membrane potentials. A famous model described in 1952 first by Alan Hodgkin and Andrew Huxley is the Hodgkin-Huxley (HH) model [33]. The neuron was modelled as an electrical circuit that represents the cell membrane as seen in Fig. 2.7. The HH model is seen as the most biological accurate model and is very computational expensive to use in the case of SNN [34].

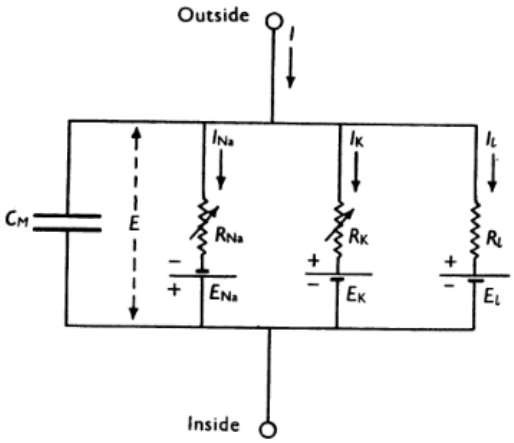


Figure 2.7: Electrical circuit representing the membrane [33]

Another popular model is the Leaky Integrate-and-Fire (LIF) neuron that is also modelled with an electrical circuit as seen in Fig. 2.8 and is the one we will use because its for now a good balance between biological realism and computational complexity.

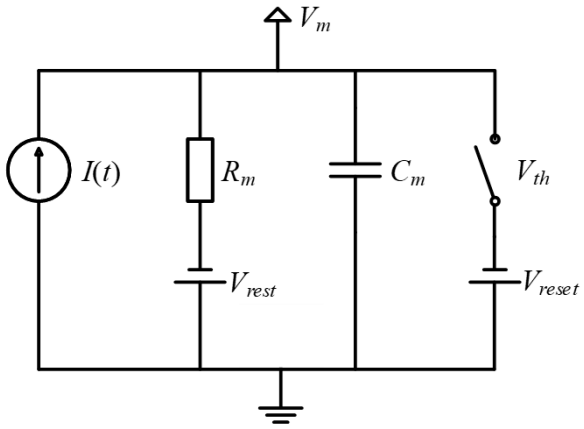


Figure 2.8: Equivalent circuit of LIF neuron model [35]

We can use the Kirchoff Current Law (KCL) on the circuit in Fig. 2.8 to arrive to the mathematical description of the membrane potential V_m of the LIF neuron [36]:

$$I(t) - \frac{V_m(t)}{R_m} = C_m \cdot \dot{V}_m(t) \tag{2.16}$$

with $I(t)$ being the input current, R_m the membrane resistance and C_m the membrane capacitance. A spike in the form of a delta function $\delta(t)$ is propagated when a threshold V_{th} is breached and after the membrane potential is reset to V_{reset}. After solving the differential equation of $V_m(t)$ we get:

$$\dot{V}_m(t) = \frac{I(t)}{C_m} - \frac{V_m(t)}{R_m C_m} \tag{2.17}$$

To use this equation in a SNN we need to discretise the equation by formulating a recursion as follows:

$$V_{t+1} = V_t + \left(C_m^{-1} \cdot x_t - V_t \cdot R_m^{-1} C_m^{-1}\right) \cdot \Delta t \tag{2.18}$$

With $I(t)$ being x_t and two extra equations are included to describe the moment that the threshold is exceeded and the resetting after:

$$\tilde{V}_{t+1} = V_t + \left(C_m^{-1} \cdot x_t - V_t \cdot R_m^{-1} C_m^{-1}\right) \cdot \Delta t \tag{2.19}$$

$$y_{t+1} = \Theta(\tilde{V}_{t+1} - V_{\text{thresh}}), \quad \text{spiking} \tag{2.20}$$

$$V_{t+1} = V_m \cdot \Theta(-\tilde{V}_{t+1} + V_{\text{thresh}}), \quad \text{resetting} \tag{2.21}$$

Where $\Theta(x)$ is the Heaviside step function. To help summarize the update rule of the LIF unit, some parameters are renamed:

$$C_m^{-1} \Delta t = w_{\text{input}} \tag{2.22}$$

$$R_m^{-1} C_m^{-1} \cdot \Delta t = w_{\text{leak}} \tag{2.23}$$

The final update rule is then:

$$V_t = w_{\text{input}} \cdot x_t + (1 - w_{\text{leak}}) \cdot V_{t-1} \cdot \Theta(V_{\text{thresh}} - V_{t-1}), \tag{2.24}$$

$$y_t = \Theta(V_t - V_{\text{thresh}}). \tag{2.25}$$

In Fig. 2.9 is the forward pass visualised. The input spikes $(x(t)$ are multiplied with the weights w_{input} and added to the membrane potential V_m. If the membrane potential exceeds the threshold V_t, the output y is 1 else y is 0. If the uutput is zero the membrane potential decays with the leak weight.

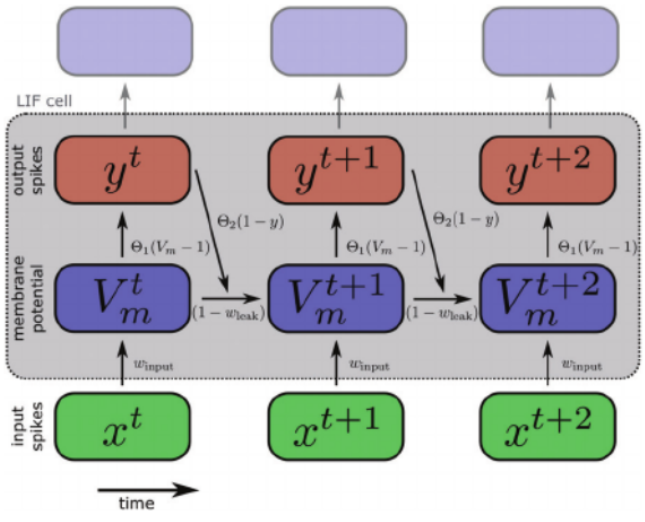


Figure 2.9: Forward pass SNN [37]

Backpropagation in SNN's starts just like the ANN with the differentiation of the loss with respect to the weights seen in Eq. 2.13:

$$\frac{dL}{dw} = \frac{\partial L}{\partial y_{\text{out}}} \cdot \frac{\partial y_{\text{out}}}{\partial y} \cdot \frac{\partial y}{\partial x} \cdot \frac{dx}{dw} \tag{2.26}$$

The term $\frac{\partial y}{\partial x}$ is the derivative of the LIF output with regard to the input. This derivative is zero when there is no spike at that time stamp because the Heaviside function Θ is zero everywhere except at 0. This results in the weights being 0 most of the time and so training is not possible with this configuration. To combat the zeroing of the weights without modifying the forward pass, redefining the gradient is a solution called the surrogate gradient approach. We have seen from Eq. 2.18 that a recurrency in the LIF node is introduced and thus the new gradient has to be implemented for each time step.

$$\frac{\partial y_t}{\partial x_t} = \frac{\partial y_t}{\partial V_t}\frac{\partial V_t}{\partial x_t} = \Theta_1'(V_t - V_{\text{thresh}}) \cdot w_{\text{input}} \tag{2.27}$$

$$\frac{\partial y_t}{\partial x_{t-1}} = \frac{\partial y_t}{\partial V_t}\frac{\partial V_t}{\partial V_{t-1}}\frac{\partial V_{t-1}}{\partial x_{t-1}} \tag{2.28}$$

$$= \Theta_1'(V_t - V_{\text{thresh}}) \cdot (1 - w_{\text{leak}}) \cdot [\Theta_2(V_{\text{thresh}} - V_{t-1}) + V_{t-1} \cdot \Theta_2'(V_{\text{thresh}} - V_{t-1})] \cdot w_{\text{input}} \tag{2.29}$$

$$\frac{\partial y_t}{\partial x_{t-n}} = \Theta_1'(V_t - V_{\text{thresh}}) \cdot w_{\text{input}}(1 - w_{\text{leak}})^n \prod_{i=1}^{n} \Theta_2(V_{\text{thresh}} - V_{t-i}). \tag{2.30}$$

The gradient of Θ_1 ìs defined as 1 and the gradient of Θ_2 as 0. This is needed because setting Θ_1 to 0 sets the weights to 0 [37].

2.3.7. Hybrid Spiking Neural Network on neuromorphic platform

Variational Autoencoders have proven effective in learning the probability distributions of hard quantum states, offering an alternative to traditional reconstruction techniques [6, 10]. For instance, in one study, a hybrid Spiking Neural Network (SNN) was implemented partially on Intel's Loihi Neuromorphic chip and partially on a conventional computer, showcasing the potential of neuromorphic hardware for quantum applications that require low power [38]. Although this hybrid approach faced limitations due to the low (6-bit) precision of synaptic weight values, it demonstrated the feasibility of using neuromorphic hardware for quantum tasks. These developments suggest that combining advanced machine learning techniques with emerging hardware technologies could unlock new possibilities for efficient quantum state tomography, enabling researchers to tackle increasingly complex quantum systems.

2.3.8. BrainScaleS-2

The University of Heidelberg, Germany was able to give access to the BrainScaleS-2 (BS2) accelerated Neuromorphic system via the Ebrains platform. This opens the possibility to train, test and validate the VAE on actual neuromorphic hardware that is build to simulate the way the brain processes information and calculations. The BS2 is a network of neurons connected via plastic synapses that work partly on a analog basis so neuron simulation directly correspond to the modelled biological processes and partly on a digital controllable, configurable substrate based on well-understood CMOS technology seen in 2.10A. The overall design goal of the BS2 was to enable large-scale accelerated emulation of spiking neural networks. The intel Loihi is based on purely digital design but is not able support plasticity programs with complex control and data dependencies. The BS2 system is able to perform massive-parallel data acquisition of analog system observables like membrane voltage traces and is able to evaluate efficiently programmable plasticity rules which makes the BS2 system unique. One distinguishing feature is that the synaptic crossbar can process weighted spikes so for a SNN the analog vector-matrix multiplication can be used for training and inference. [39].

System Architecture

The system architecture seen in Fig. 2.10C consist of four synaptic crossbar arrays each with 256 rows and 128 columns making the system capable of simulating 512 neurons that associated to the columns (128 * 4). The digital processors on the top and the bottom use single instruction multiple data (SIMD) vector extensions to read and write the digital state of the synaptic crossbar row wise in parallel and readout analog traces via a 512 channel column analog to digital converter (CADC). In Fig. 2.10D is seen how the crossbar facilitates the recurrent connections of Eq. 2.18 needed to realise the electrical model of the LIF neuron.

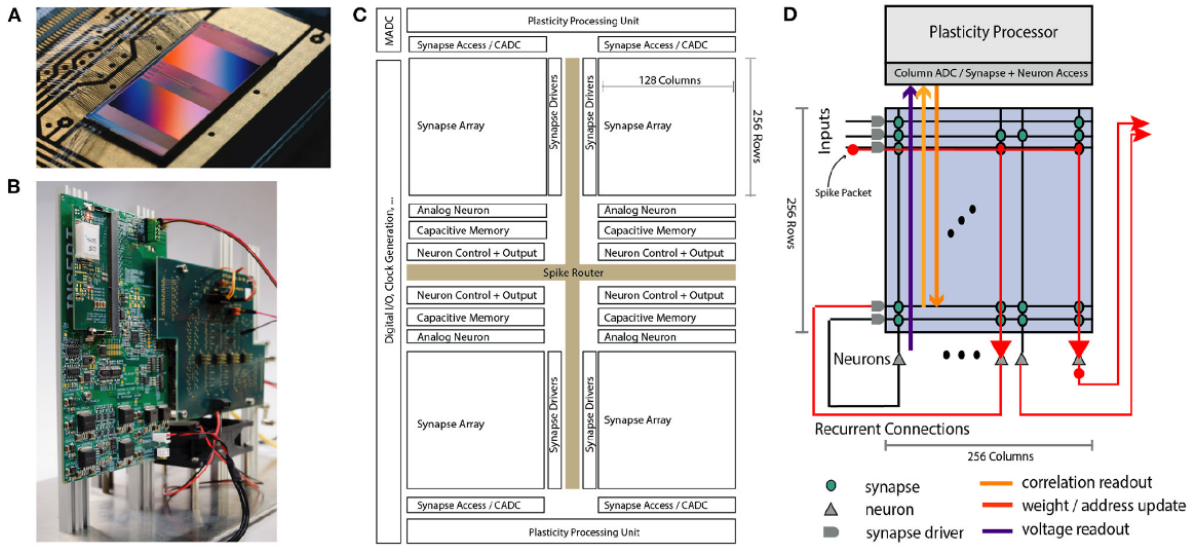


Figure 2.10: Overview of the BrainScaleS-2 system [39]

(AdEx) LIF neuron on BrainScaleS-2

A spike from a neuron is a sudden change in potential over the membrane and it is modelled a a dirac function as we saw in the previous chapters. The researchers and engineers behind the BS2 had some difficulties to overcome when realising the behaviour of neurons in the CMOS technology. Firstly, a biological threshold in neurons is not existing as a comparable integer value in terms of voltage but rely on some non-linear term as well that cannot be captured by the LIF model that is linear. Secondly, some neurons can exhibit non constant frequency oscillatory behaviour that drives the generation of spikes and the LIF model does not incude this. The BS2 uses the adaptive exponential integrate-and-fire model that covers both of these issues. The equation they use to model the is a rewritten form of Eq.2.16:

$$C\frac{dV(t)}{dt} = -g_{\text{leak}}(V(t) - V_{\text{leak}}) + I(t), \tag{2.31}$$

where:

- $V(t)$ is the membrane potential,
- g_{leak} is the leak conductance,
- V_{leak} is the resting potential,
- $I(t)$ is the input current.

Whenever $V(t)$ crosses a threshold ϑ, a spike is emitted, and the membrane potential resets.

The spiking behavior is non-differentiable due to the Heaviside step function:

$$S(t) = \Theta(V(t) - \vartheta). \tag{2.32}$$

To enable backpropagation, a smooth approximation is used for the derivative of $S(t)$:

$$\frac{\partial S(t)}{\partial V(t)} \approx g(V) = \frac{\partial}{\partial V}\left[\frac{1}{1 + e^{-\beta(V-\vartheta)}}\right], \tag{2.33}$$

where:

- β controls the steepness of the surrogate gradient,
- ϑ is the firing threshold.

To account for device-specific variations, recorded values from the BrainScaleS-2 hardware are integrated into the computation graph. The recorded membrane potential $V_{\text{measured}}[t]$ is combined with the modeled potential $\tilde{V}[t]$ using an auxiliary function:

$$f(x, \tilde{x}) \equiv x, \quad \text{with} \quad \frac{\partial f}{\partial x} = 0, \ \frac{\partial f}{\partial \tilde{x}} = 1. \tag{2.34}$$

This allows the gradients to flow through the modeled variables $\tilde{V}[t]$ while using $V_{\text{measured}}[t]$ for accurate evaluation.

The forward pass computes the neuron dynamics:

$$\tilde{V}[t+1] = f(V_{\text{measured}}[t+1], \tilde{V}[t]e^{-\Delta t/\tau_m} + \tilde{I}[t]), \tag{2.35}$$

$$\tilde{I}[t+1] = \tilde{I}[t]e^{-\Delta t/\tau_s} + \sum_j W_j S_j[t], \tag{2.36}$$

where:

- τ_m and τ_s are the time constants for membrane potential and synaptic current,
- W_j is the weight of the synapse from presynaptic neuron j,
- $S_j[t]$ is the spike emitted by neuron j at time t.

The backward pass calculates gradients using surrogate derivatives:

$$\frac{\partial S[t]}{\partial V[t]} \approx g(V). \tag{2.37}$$

The framework supports various loss functions, including:

- Spike-based loss:
$$L_{\text{spike}} = \sum_t \text{MSE}(S_{\text{target}}[t], S[t]),$$
where $S_{\text{target}}[t]$ is the desired spike train.
- Regularization for sparse spiking:
$$L_{\text{activity}} = \rho_b \cdot \frac{1}{N} \sum_{i=1}^{N} \left(\sum_t S_i[t] \right)^2.$$
- Voltage clipping to avoid saturation:
$$L_{\text{voltage}} = \rho_v \cdot \sum_t (V[t] - V_{\text{threshold}})^2.$$

The training dynamically adjusts weights and compensates for hardware mismatches:

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_t \frac{\partial \mathcal{L}}{\partial S[t]} \frac{\partial S[t]}{\partial \tilde{V}[t]} \frac{\partial \tilde{V}[t]}{\partial W}. \tag{2.38}$$

Updated weights are written back to the hardware, completing the in-the-loop (ITL) training process.

2.3.9. Spiking RBM

A restricted Boltzmann machine (RBM) has been used to successfully reconstruct quantum states on a neuromorphic hardware chip named BrainScaleS-2 (BS2) from Heidelberg University [7]. The RBM is composed of a hidden layer and a visible layer as seen in Fig. 2.11a that are initialized with some weights that are adjusted according to the Kullbach-Leibner divergence. The input of the network is the noise of the chip generated by neurons that produce Poisson distributed noise, and the output is the value of the spiking neuron at the visible layer at time T as seen in Fig. 2.11c. The output of the neurons is decoded to decide the POVM representation as shown in Fig. 2.11d, and reading the output neurons each time step produces Monte Carlo-Markov Chain (MCMC) like sampling of the output neurons and this results in a POVM probability matrix that can be used to calculate the density matrix shown in 2.11e. A fidelity of > 0.99 of 2 qubits spins was achieved and for 3 and 4 qubits the fidelity was around 0.85 and 0.7. This is due to the hardware limitations of the BS2 that is limited to 512 neurons.

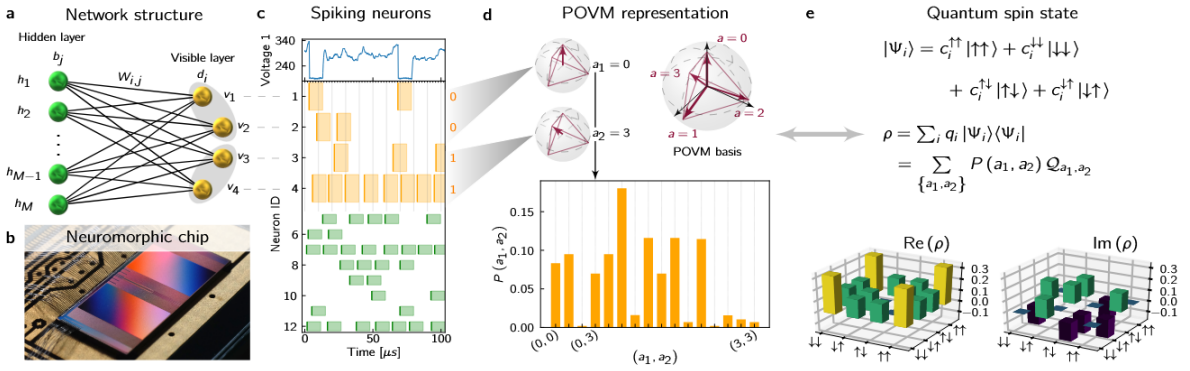


Figure 2.11: Reconstructing a quantum state from spiking RBM from [7].

2.3.10. Spiking Variational auto-encoder (SNNs)

Spiking Variational Autoencoders (SVAEs) and Efficient Spiking Variational Autoencoders (ESVAEs) extend this concept by integrating spiking neural network architectures to mimic the temporal dynamics of biological neurons [18]. These spiking models achieve comparable performance to their continuous counterparts while providing a more biologically plausible framework for quantum state reconstruction. Moreover, incorporating spike-timing-dependent sampling processes, such as Poisson distributions, aligns well with the nature of quantum measurements, enhancing the interpretability of the results [40]. The computational efficiency of these models, combined with their ability to generalize across different quantum platforms, highlights their significance in addressing the challenges of large-scale quantum state tomography.

3

Proposed Method

The methods used in this thesis are explained in this chapter. Firstly, a quantum state is prepared using unitary operators and measured S (Shots) times to create a dataset to interconnect the Qubit information with the Spiking Variational Autoencoder (SVAE). This dataset will be processed further according to machine learning standards. Lastly, the SVAE consists of an encoder that is build on the CPU an a decoder that is build on the BrainScales-2 (BS2).

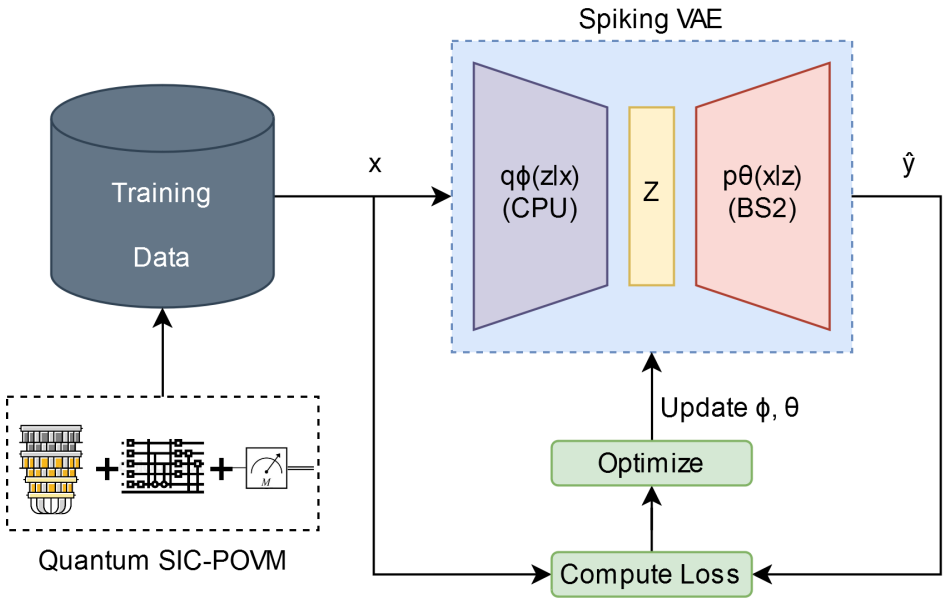


Figure 3.1: Architecture of the Variational Autoencoder

3.1. Creating the dataset

The dataset is build by saving quantum measurements and encode them in vectors so they are compatible with common used machine learning dataloaders. The qubits are measured using SICPOVM and then one hot encoded to form the database.

3.1.1. SIC POVM

The preparation of quantum states to create the GHZ includes the Positive Operatation Value Measures (POVM) rotations to get an Informational Complete (IC) dataset. Each Qubit is measured by projecting the state on one of the 4 basis that form a Tetrahedron in the bloch sphere as depicted in Fig 3.2.

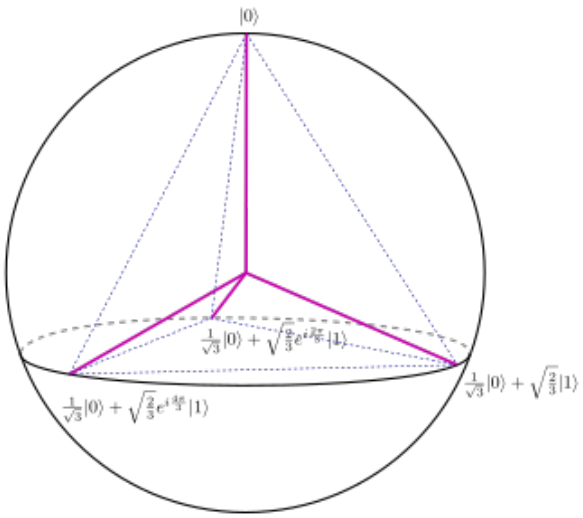


Figure 3.2: Basis of the SIC-POVM in the Bloch sphere[41]

The four SIC-POVM states for a single qubit are:

$$|\psi_0\rangle = |0\rangle \tag{3.1}$$

$$|\psi_1\rangle = \frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}|1\rangle \tag{3.2}$$

$$|\psi_2\rangle = \frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}e^{i\frac{2\pi}{3}}|1\rangle \tag{3.3}$$

$$|\psi_3\rangle = \frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}e^{-i\frac{2\pi}{3}}|1\rangle \tag{3.4}$$

The qubits can be measured in the computational basis by using unitary operations that rotate the qubits with the same angle as the 4 POVM states. A quantum state can be defined in angle as:

$$|\psi\rangle = \cos(2\theta)|0\rangle + e^{i\phi}\sin(2\theta)|1\rangle \tag{3.5}$$

Where $\alpha = \cos(2\theta)$, $\beta = e^{i\phi}\sin(2\theta)$ and in this case is the azimuthal angle ϕ already separated in Eq. 3.1.1 and 3.1.1. The arrccosine can be applied on the probability amplitude of ket 0 to find θ:

$$\theta = 2 * \arccos(\alpha) \tag{3.6}$$

3.1.2. Two-Qubit POVM Example

The two qubit POVM is compromised from single qubit POVMs, what is also true for n qubits systems.

Single-Qubit POVMs

The single-qubit POVM elements $M^{(a)}$ are calculated form the vectors in 3.1.1. The full calculations can be found in A.1:

$$M^{(0)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad M^{(1)} = \frac{1}{3}\begin{bmatrix} 1 & \sqrt{2} \\ \sqrt{2} & 2 \end{bmatrix},$$

$$M^{(2)} = \frac{1}{3}\begin{bmatrix} 1 & -\frac{\sqrt{2}}{2} - i\frac{\sqrt{6}}{2} \\ -\frac{\sqrt{2}}{2} + i\frac{\sqrt{6}}{2} & 2 \end{bmatrix}, \quad M^{(3)} = \frac{1}{3}\begin{bmatrix} 1 & -\frac{\sqrt{2}}{2} + i\frac{\sqrt{6}}{2} \\ -\frac{\sqrt{2}}{2} - i\frac{\sqrt{6}}{2} & 2 \end{bmatrix}.$$

Two-Qubit POVMs

The two-qubit POVM elements are tensor products:

$$M^{(a_1,a_2)} = M^{(a_1)} \otimes M^{(a_2)}.$$

For example:

$$M^{(0,0)} = M^{(0)} \otimes M^{(0)} = \frac{1}{16}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad M^{(0,1)} = M^{(0)} \otimes M^{(1)} = \frac{1}{16}\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Visualization of All POVM Elements

Combine all $M^{(a_1,a_2)}$ into a 16×16 matrix, where each block corresponds to $M^{(a_1)} \otimes M^{(a_2)}$.

$$\mathbf{POVM} = \begin{bmatrix} M^{(0,0)} & M^{(0,1)} & M^{(0,2)} & M^{(0,3)} \\ M^{(1,0)} & M^{(1,1)} & M^{(1,2)} & M^{(1,3)} \\ M^{(2,0)} & M^{(2,1)} & M^{(2,2)} & M^{(2,3)} \\ M^{(3,0)} & M^{(3,1)} & M^{(3,2)} & M^{(3,3)} \end{bmatrix}.$$

Each $M^{(a_1,a_2)}$ block is a 4×4 matrix. Highlighted here:

$$\mathbf{POVM} = \left[\begin{array}{c|c|c|c} M^{(0,0)} & M^{(0,1)} & M^{(0,2)} & M^{(0,3)} \\ \hline M^{(1,0)} & M^{(1,1)} & M^{(1,2)} & M^{(1,3)} \\ \hline M^{(2,0)} & M^{(2,1)} & M^{(2,2)} & M^{(2,3)} \\ \hline M^{(3,0)} & M^{(3,1)} & M^{(3,2)} & M^{(3,3)} \end{array}\right].$$

3.1.3. Input Vector Creation and One-Hot Encoding

The input vectors for the Spiking Variational Autoencoder (SVAE) are generated using a one-hot encoding scheme. This encoding supports the model's spiking neural network structure by generating binary inputs compatible with spiking dynamics.

Measurement Representation

The SVAE processes quantum measurement data from a Positive-Operator Valued Measure (POVM) dataset. Each measurement result corresponds to a specific quantum state outcome, which is represented as a categorical label.

One-Hot Encoding Process

The one-hot encoding process converts categorical labels into binary vectors:

1. Suppose there are C possible measurement outcomes.
2. Each outcome is represented as an integer label $l \in \{0, 1, \ldots, C-1\}$.
3. The one-hot encoding creates a binary vector $\mathbf{x} \in \mathbb{R}^C$ such that:

$$x_i = \begin{cases} 1, & \text{if } i = l \\ 0, & \text{otherwise} \end{cases}$$

Example: One-Hot Encoding for a 4-State System

Consider a system with $C = 4$ possible outcomes labeled $\{0, 1, 2, 3\}$. The corresponding one-hot encoded vectors are:

$$\begin{aligned}
\text{Label } l = 0 &\implies \mathbf{x} = [1, 0, 0, 0] \\
\text{Label } l = 1 &\implies \mathbf{x} = [0, 1, 0, 0] \\
\text{Label } l = 2 &\implies \mathbf{x} = [0, 0, 1, 0] \\
\text{Label } l = 3 &\implies \mathbf{x} = [0, 0, 0, 1]
\end{aligned}$$

Dataset Creation

The dataset used for training the SVAE is constructed as follows:

- Each quantum measurement generates a label based on its outcome.
- These labels are converted into one-hot encoded vectors.
- The resulting binary vectors form the input dataset for the SVAE.

Importance for Spiking Neural Networks

The one-hot encoded vectors are particularly well-suited for spiking neural networks because:

- Each vector contains binary values (0 or 1), matching the spiking representation.
- The encoding ensures a clear, non-overlapping representation of quantum states.

Training Considerations

During training:

- The input one-hot encoded vectors are fed into the encoder as binary spike trains.
- The reconstruction target for the decoder is the same one-hot encoded vector, ensuring that the model learns to map from the encoded latent space back to the original measurement representation.

3.1.4. encoder

The encoder of the Spiking Variational Autoencoder (SVAE) is designed using a fully connected spiking neural network (SNN) with two layers of Leaky Integrate-and-Fire (LIF) neurons. The layers are from the snntorch library that is build on top of the pytorch library [42]. It transforms the input data into a spiking representation, suitable for further latent space processing. The encoder part of the model is executed on a CPU.

Initialization

The encoder consists of two fully connected layers and corresponding LIF neurons:

- **Fully Connected Layers**:
  - fc_1: Projects the input from the input size d_{input} to the hidden size d_{hidden}.
  - fc_2: Projects from the hidden size d_{hidden} to the output size d_{output}.
- **LIF Neurons**:
  - LIF neurons are used after each fully connected layer. They model spiking behavior with a membrane potential governed by a decay factor β.

Forward Pass

The forward pass of the encoder processes the input over multiple time steps T. The following steps are performed at each time step t:

1. Initialize the membrane potentials mem_1 and mem_2 to zero.
2. For each time step t:
   (a) Compute the current cur_1 from the first fully connected layer:
   $$cur_1(t) = fc_1(x)$$
   (b) Pass the current through the first LIF neuron:
   $$spk_1(t), mem_1(t) = \mathsf{LIF}_1(cur_1(t), mem_1(t-1))$$
   (c) Compute the next current using spk_1:
   $$cur_2(t) = fc_2(spk_1(t))$$
   (d) Pass this current through the second LIF neuron:
   $$spk_2(t), mem_2(t) = \mathsf{LIF}_2(cur_2(t), mem_2(t-1))$$
3. Record the spikes $spk_2(t)$ and membrane potentials $mem_2(t)$ for each time step.

Memory Optimization

To reduce memory usage, intermediate variables are deleted after each computation step, and memory is cleared using garbage collection.

Output

The encoder outputs two tensors:

- Spike train tensor spk_2: a sequence of binary spikes over time.
- Membrane potential tensor mem_2: the recorded potentials at each time step.

3.1.5. Latent Space Construction

The latent space in the spiking variational autoencoder (SVAE) is designed using spiking neuron models, enabling a biologically plausible representation of encoded information. This section describes how the latent space is constructed, including the reparameterization process and spiking representation. The latent space in the SVAE is constructed using spiking neuron dynamics, where the encoder's spiking output determines the latent representation through firing rate estimation. Reparameterization using a Poisson sampling process enables spike-based encoding, allowing the decoder to reconstruct the input while ensuring efficient and biologically plausible representation. [18]

Firing Rate Estimation

After the encoder processes the input, the spiking output from the encoder is used to estimate the firing rate:

$$r_p = \frac{1}{T} \sum_{t=1}^{T} spk_2(t)$$

where:

- r_p is the firing rate vector representing the latent space.
- $spk_2(t)$ is the spike output from the encoder at time t.
- T is the total number of time steps.

Reparameterization Process

The latent representation is obtained using a reparameterizable sampling process. Two methods are considered based on the type of sampling:

Gaussian Reparameterization

For standard VAEs, the mean μ and variance σ^2 are computed using:

$$z = \mu + \epsilon \cdot \sigma$$

where:

- z is the sampled latent vector.
- $\epsilon \sim \mathcal{N}(0, 1)$ is a random noise vector.
- μ and σ are the outputs from the encoder.

Poisson Spiking Reparameterization

In the spiking VAE, the latent space follows a Poisson spiking process:

$$z = H(r_p - u)$$

where:

- r_p is the estimated firing rate from the encoder.
- $u \sim \mathcal{U}(0, 1)$ is a uniformly distributed random variable.
- $H(\cdot)$ is the Heaviside step function.

This sampling mechanism generates a binary spike train, enabling compatibility with spiking neural network architectures.

Latent Variable Sampling

The latent vector z sampled using the spiking process is fed into the decoder to reconstruct the input. This latent representation preserves important features through biologically inspired mechanisms.

Latent Space Regularization

The SVAE uses a Maximum Mean Discrepancy (MMD) loss to regularize the latent space, that will be explained in the next section as it uses the same function as for the loss.

3.1.6. Loss Function

The loss function of the Spiking Variational Autoencoder (SVAE) combines two essential components: Maximum Mean Discrepancy (MMD) loss and Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) loss. These components ensure that the latent space is well-regularized and that the model reconstructs the input effectively [17, 18, 40].

Maximum Mean Discrepancy (MMD) Loss

The MMD loss measures the distance between the distributions of the latent representations from the encoder (posterior) and the prior distribution. It uses a kernel-based distance measure, typically a Radial Basis Function (RBF) kernel, and is defined as follows:

$$\text{MMD}^2(p(r_p), q(r_q)) = \mathbb{E}_{r_p, r'_p}[k(r_p, r'_p)] + \mathbb{E}_{r_q, r'_q}[k(r_q, r'_q)] - 2\mathbb{E}_{r_p, r_q}[k(r_p, r_q)]$$

Where:

- $p(r_p)$: The posterior distribution of the firing rates from the encoder.
- $q(r_q)$: The prior distribution of the firing rates.
- $k(r_i, r_j)$: The kernel function, usually the RBF kernel:

$$k(r_i, r_j) = \exp\left(-\frac{\|r_i - r_j\|^2}{2\sigma^2}\right)$$

The MMD loss encourages the latent space to match the prior distribution, ensuring meaningful latent representations.

Reconstruction Loss: MSE or RMSE

The reconstruction loss ensures that the model's output closely resembles the input. It measures the reconstruction error using either Mean Squared Error (MSE) or Root Mean Squared Error (RMSE), depending on the application.

Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\|x_i - \hat{x}_i\|^2$$

Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\|x_i - \hat{x}_i\|^2}$$

Where:

- x_i: The original input sample.
- $\hat{x}_i$: The reconstructed output from the decoder.
- n: The number of samples.

Combined Loss Function

The final loss function is a weighted combination of the reconstruction and MMD losses:

$$\mathcal{L} = \lambda_{\text{MMD}} \cdot \text{MMD}^2(p(r_p), q(r_q)) + \lambda_{\text{rec}} \cdot \text{MSE}(x, \hat{x})$$

Where:

- λ_{MMD}: Weight for the MMD loss.
- λ_{rec}: Weight for the reconstruction loss.

This combined loss function ensures that the latent space maintains a structured distribution while minimizing reconstruction errors. It balances both terms to achieve optimal performance in generating accurate reconstructions and meaningful latent representations.

3.1.7. Fidelity Calculation in the Quantum State Tomography Model

The fidelity calculation in the quantum state tomography model measures how accurately the reconstructed quantum state matches the true quantum state. This comparison is crucial for evaluating the performance of the spiking variational autoencoder (SVAE). Fidelity provides a value between 0 and 1, where 1 indicates perfect reconstruction.

Definition of Fidelity

The fidelity $F(\rho, \sigma)$ between two quantum states ρ and σ is defined as:

$$F(\rho, \sigma) = \left(\text{Tr}\left[\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right]\right)^2$$

Where:

- ρ: The reconstructed quantum state represented as a density matrix.
- σ: The true quantum state represented as a density matrix.

For two pure states $|\psi\rangle$ and $|\phi\rangle$, the fidelity simplifies to:

$$F(|\psi\rangle, |\phi\rangle) = |\langle\psi|\phi\rangle|^2$$

Probability-Based Fidelity

The model also uses probability-based fidelity when working with the predicted and true measurement probabilities obtained from the POVM measurements. The fidelity is computed as the sum of the geometric mean of corresponding probabilities:

$$F(P_{\text{true}}, P_{\text{rec}}) = \sum_i \sqrt{P_{\text{true},i} P_{\text{rec},i}}$$

Where:

- P_{true}: True probabilities obtained from the dataset.
- P_{rec}: Reconstructed probabilities computed from the model output.

Density Matrix Fidelity

The model also reconstructs the quantum state using the estimated and true density matrices ρ_{rec} and ρ_{true}. The process includes the following steps:

1. Density Matrix Square Root:
$$\sqrt{\rho} = U\sqrt{D}U^\dagger$$
Where D represents the eigenvalues from the spectral decomposition.
2. Product Matrix Calculation:
$$A = \sqrt{\rho_{\text{true}}} \cdot \rho_{\text{rec}} \cdot \sqrt{\rho_{\text{true}}}$$
3. Fidelity Computation:
$$F(\rho_{\text{true}}, \rho_{\text{rec}}) = \left(\text{Tr}\left[\sqrt{A}\right]\right)^2$$

Model Evaluation with Fidelity

During training and validation, the computed fidelity is logged and printed after each iteration. A fidelity score close to 1 indicates high similarity between the reconstructed and true quantum states, reflecting successful training of the SVAE. This metric provides direct insight into the model's performance in learning quantum states from its training dataset.

3.1.8. Density Matrix Reconstruction

The quantum state ρ is reconstructed using the measurement probabilities obtained from the informationally complete (IC) POVM. The process ensures that the quantum state can be fully recovered based on the measured probability distribution. Below, we describe the mathematical details of the reconstruction process as implemented in the code.

Measurement Probabilities and Born's Rule

The probability of observing a specific measurement outcome a is given by Born's rule:

$$P(a) = \text{Tr}[M^{(a)}\rho]$$

Where:

- $M^{(a)}$: The POVM element corresponding to the measurement outcome a.
- ρ: The quantum state's density matrix.

For a system with N qubits, the POVM element $M^{(a)}$ is expressed as:

$$M^{(a)} = M^{(a_1)} \otimes M^{(a_2)} \otimes \cdots \otimes M^{(a_N)}$$

Where a_i indicates the measurement outcome for the i-th qubit.

Overlap Matrix and Inversion

To reconstruct ρ, the overlap matrix $T_{a,a'}$ is used:

$$T_{a,a'} = \mathrm{Tr}[M^{(a)} M^{(a')}]$$

This matrix encodes the relationships between the POVM elements and is invertible when the POVM is informationally complete. The reconstruction equation for ρ is then:

$$\rho = \sum_{a,a'} P(a) T^{-1}_{a,a'} M^{(a')}$$

Where:

- $P(a)$: The measured probabilities for the outcome a.
- $T^{-1}_{a,a'}$: The inverse of the overlap matrix.
- $M^{(a')}$: The POVM elements.

Reconstruction Implementation in Code

The implementation reconstructs the density matrix ρ_{rec} from the measured probabilities P_{rec}:

$$\rho_{\text{rec}} = \sum_i P_{\text{rec}}(i) M^{(i)}$$

Similarly, the true density matrix ρ_{true} is reconstructed using the true probabilities P_{true}:

$$\rho_{\text{true}} = \sum_i P_{\text{true}}(i) M^{(i)}$$

The code calculates the tensor product of single-qubit POVM matrices to construct the multi-qubit POVM matrices $M^{(a)}$, which are precomputed and stored for efficiency. The reconstructed matrices are normalized to ensure:

$$\mathrm{Tr}(\rho) = 1$$

Integration in the Model

The reconstruction process integrates the measured probabilities into density matrices. This is performed during both training and validation to compare the predicted quantum state ρ_{rec} with the ground truth ρ_{true}. The comparison is used to compute fidelity, which evaluates the accuracy of the reconstruction. By employing this method, the model leverages IC POVMs to reconstruct the quantum state effectively, ensuring that the training process aligns with the underlying physical quantum system.

4

Results

In the methods chapter is described how a SVAE can be set up to be trained in order to learn a quantum state. In this chapter the results are described that came from implementing these methods by training the SVAE on:

1. a CPU with the snnTorch python library that is used to train both the encoder and decoder implemented on a Intel Xeon E5-2620 v3 and a Nvidia GeForce RTX 2080 Ti [42].
2. a hybrid setup that trains the encoder on a CPU but the decoder on the BrainScales-2 (BS2) mixed signal neuromorphic chip using the hxtorch library framework [43].

Both the snnTorch library and hxtorch library use the pytorch framework so integration was possible and the codes could be held almost identical for both training rounds. The table Tab. 4.1 summarizes the parameters that remain constant across all tests, along with their descriptions.

To compare the 2 different training algoritms, the parameters of the encoders and VAE structure are equal but the parameters of the 2 decoders are different because the BS2 has more flexible parameter settings due to its analog plastic neurons that are measured with ADC's so threshold voltages, calibration settings, and time step measurement setting can be adjusted.

Parameter	**Value / Description**
Shots	$100,000$
Beta	0.819 (Hyperparameter for regularization)
Number of Steps	100 (Training steps per epoch)
Number of Epochs	5 (Total epochs for training)
Learning Rate	1×10^{-3} (Initial)
Number of Workers	4 (For data loading)
Shuffle	`False` (Data shuffling disabled)
Input Size	$4 \cdot n$ (Proportional to the number of qubits, n)
Hidden Size	$20 \cdot n$ (Proportional to the number of qubits, n)
Output (Latent) Size	$2 \cdot 2^n$ (Proportional to the number of qubits, n)
Alpha	1 (Scaling factor for loss terms)
Model Recovery	`False` (No recovery of previous models)

Table 4.1: Common parameters used for all tests.

4.1. Spiking poisson on CPU

This section presents the training and validation process for quantum state tomography using a variational autoencoder (VAE) with a spiking Poisson-based encoding scheme implemented on a CPU. Table 4.2 summarizes the key parameters, including the number of qubits, batch size for training, and the number of validation samples. The model is trained over five epochs with a total of 5×10^5 samples per epoch, resulting in 50×10^5 training samples for qubit configurations ranging from 3 to 7 qubits. The corresponding training fidelity results are depicted in Figs. B.3a to B.7a, while the fidelity validation trends are illustrated in Fig. 4.1.

Qubits	Batch Size for Training	Validation Samples
3	100	20,000
4	300	100,000
5	600	$4^5 \cdot 500 = 512,000$
6	600	$4^6 \cdot 500 = 2,048,000$
7	600	$4^7 \cdot 500 = 8,192,000$
8	1000	$4^8 \cdot 500 = 32,768,000$

Table 4.2: Parameters for training and validation across different numbers of qubits.

The fidelity of the qubits demonstrates convergence after approximately 50×10^5 training samples. However, a notable delay is observed in the training of 5-qubit states compared to other qubit configurations, as shown in Fig. 4.1. This discrepancy can be attributed to the gradient descent optimization process struggling to initially identify the optimal gradient direction. Consequently, fidelity improvement for the 5-qubit case exhibits a slower onset but eventually aligns with the fidelity achieved by other qubit configurations.

To further analyze the fidelity progression, Fig. 4.1 presents the validation fidelity trends for 3 to 7 qubits, aligned by the total number of samples processed. The results indicate that despite the initial training delay for 5-qubit states, the model ultimately reaches comparable fidelity levels after sufficient training iterations.

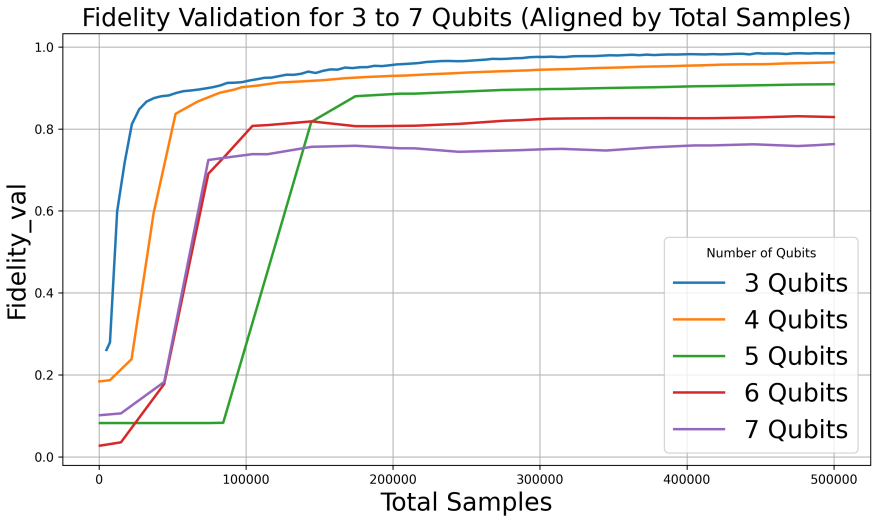


Figure 4.1: 3 to 7 qubits evaluated fidelity

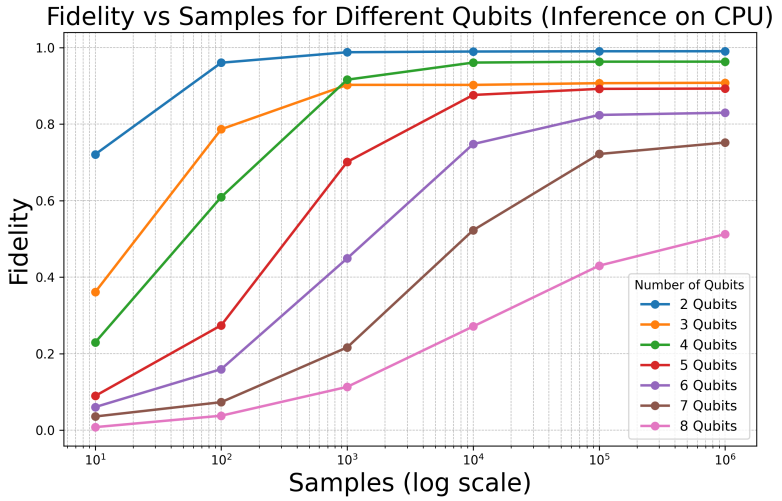


Figure 4.2: Fidelity of inference with increasing sample sizes.

4.2. Spiking poisson on BS2

In this section the model was trained using a hybrid setup where the encoder remains the same but the decoder is moved to the BS2 chip. Additional parameters are needed because of the analog synapses of the BS2 that where not present in the simulated synapses on the CPU. In table 4.3 are the metrics stated that are held the same during training and validation.

Parameter	**Value**
Time Step (`DT`)	2.0×10^{-6} s
Membrane Time Constant (τ_{mem})	1.0×10^{-6} s
Synaptic Time Constant (τ_{syn})	6.0×10^{-6} s
Scaling Factor (α)	300.0
Hidden Trace Shift (`trace_shift_hidden`)	$.0e-06/DT$
Output Trace Shift (`trace_shift_out`)	$.0e-06/DT$
Hidden Weight Initialization Range (`weight_init_hidden`)	$(0.2, 0.6)$
Output Weight Initialization Range (`weight_init_output`)	$(0.4, 0.6)$
Weight Scaling Factor (`weight_scale`)	100.0
Trace Scaling Factor (`trace_scale`)	0.0147
Input Repetitions (`input_repetitions`)	1

Table 4.3: Metrics and Parameters of the SNN Model.

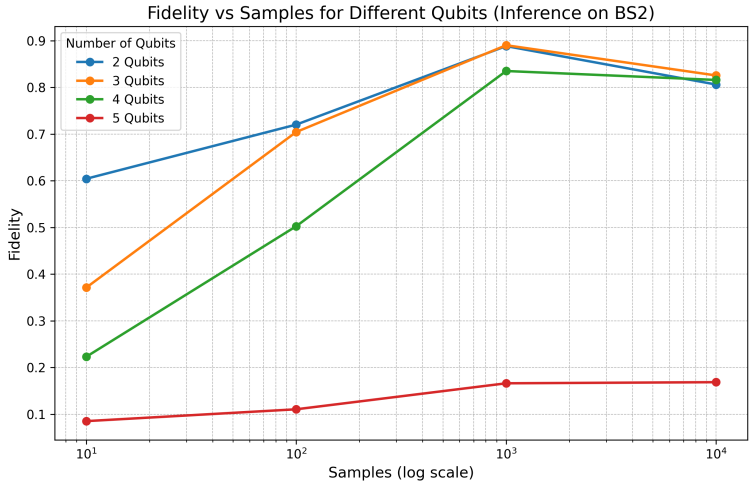


Figure 4.3: Fidelity of validation on BS2

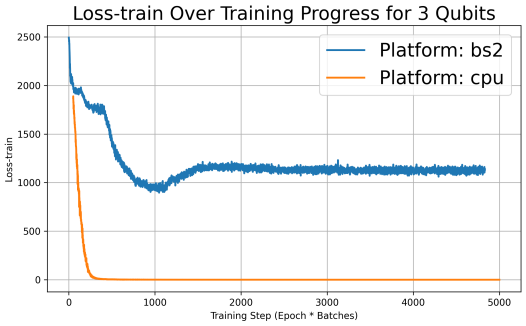
4.3. Comparison between CPU-CPU and CPU-BS2 Implementation

This section presents a comparative analysis of the training performance of a fully spiking variational autoencoder (FSVAE) for quantum state tomography on two distinct computational platforms: a CPU-only implementation and a hybrid CPU-BrainScaleS-2 (BS2) neuromorphic hardware implementation. Fig. 4.4 illustrates the key performance metrics observed during training.

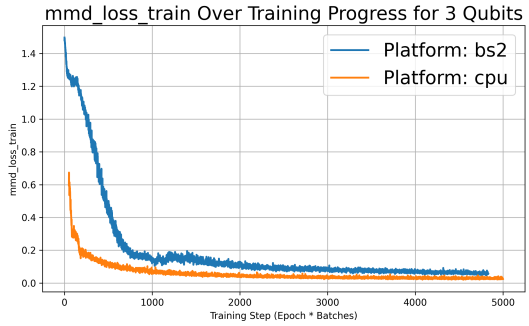
As depicted in Fig. 4.4a, the fidelity achieved during training is systematically lower for the CPU-BS2 implementation in comparison to the CPU-only implementation. This discrepancy suggests that the neuromorphic hardware introduces additional constraints that impact the overall reconstruction accuracy. The total training loss, shown in Fig. 4.4b, further substantiates this observation, as the loss reduction is significantly more pronounced in the CPU-only implementation. Conversely, the CPU-BS2 implementation exhibits a more gradual convergence, indicating potential limitations in its optimization dynamics.

To gain further insights into this behaviour, we analyse the individual contributions of the Maximum Mean Discrepancy (MMD) loss and the Mean Squared Error (MSE) loss, presented in Figs. 4.4c and 4.4d, respectively. Notably, while the MMD loss follows a similar downward trajectory for both implementations, a substantial deviation is observed in the MSE loss. Specifically, the CPU-BS2 implementation exhibits a persistently higher MSE loss throughout training, which accounts for its overall inferior performance in terms of fidelity and total loss reduction.

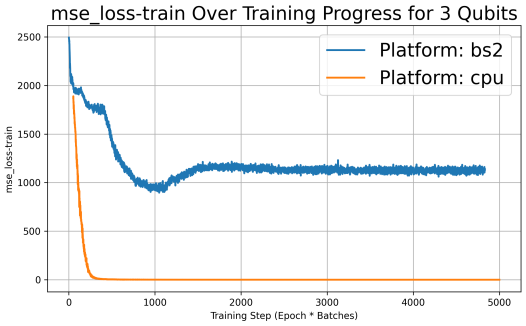
(a) Fidelity of training for 3 qubits comparison.



(b) Training loss plot.



(c) MMD loss during training.



(d) MSE loss during training.

Figure 4.4: Training metrics visualized. (a) Fidelity comparison of 3 qubits, (b) Training loss plot, (c) MMD loss during training, and (d) MSE loss during training.

4.4. Comparison with State of the Art

Table 4.4 presents a comparison of different models, where the first two columns correspond to implementations on the BrainScaleS-2 (BS2) system and the remaining columns represent classical neural network-based approaches. The results from the BS2 implementation in this work surpass those reported in [7] for 3 and 4 qubits, despite using only 10^3 samples compared to the 10^6 samples used in their work. This highlights the efficiency of the proposed method in achieving high fidelity with significantly fewer samples. The

Among the models compared, the highest fidelity is achieved by the variational autoencoder (VAE) implemented on a continuous artificial neural network, as reported by [26]. This suggests that while spiking neuromorphic implementations are promising, classical deep learning architectures still provide the best performance in terms of fidelity.

Qubits	SVAE BS2	RBM BS2	SVAE CPU	VAE CPU
2	0.89	0.99	0.99	0.99
3	0.89	0.85	0.99	0.99
4	0.84	0.69	0.97	0.98
5	0.16	N.A.	0.90	0.98
6	N.A.	N.A.	0.82	0.98
7	N.A.	N.A.	0.78	0.98
8	N.A.	N.A.	0.50	0.98

Table 4.4: Fidelity comparison of different models. The first two columns represent implementations on the BrainScaleS-2 (BS2) system, while the last two correspond to classical neural network-based approaches. The highlighted values indicate the 3- and 4-qubit fidelities in BS2-based models.

5

Conclusion

This thesis explored the intersection of quantum state tomography and spiking neural networks (SNNs) through the development of a fully spiking variational autoencoder (FSVAE). By employing the BrainScaleS-2 (BS2) neuromorphic hardware as the decoder, this thesis aimed to harness the advantages of spiking computation for efficient quantum state reconstruction and generative modelling.

5.1. Summary of Contributions

Key contributions of this work include:

- The integration of spiking neurons in both encoder and decoder to achieve a fully spiking VAE architecture, demonstrating compatibility with neuromorphic platforms like BS2.
- The introduction of spiking latent space sampling methodologies, adapting standard VAE methods for binary spiking frameworks.
- Experimental validation of the architecture for quantum state reconstruction, with performance benchmarks on fidelity and efficiency.

5.2. Findings and Impact

The results demonstrate that spiking VAEs can represent entangled quantum states with competitive fidelity, addressing challenges in energy-efficient quantum simulations. For instance, the fidelity of the reconstructed 3 and 4-qubit Greenberger-Horne-Zeilinger (GHZ) states approached 0.85 and 0.82 respectively on BS2 , improving the prior benchmarks on BS2 for restricted Boltzmann machines (RBMs). This suggests the viability of spiking neuromorphic systems for encoding quantum correlations and generative tasks, with the potential for scaling to larger quantum systems.

5.3. Future Work

While this thesis establishes a foundational framework, several avenues remain open for further exploration:

- Exploring the feasibility of CNN-based architectures in place of the VAE encoder and decoder.
- Investigating techniques to further reduce MSE loss in BrainScaleS-2.
- Optimization of the BS2's calibration processes to reduce hardware-induced noise and enhance fidelity for high-dimensional quantum states.
- Investigating the scalability of spiking VAEs to larger quantum systems and their application in quantum many-body simulations.
- Exploring cross-disciplinary applications, such as reinforcement learning in quantum settings, to evaluate the broader utility of spiking VAEs.

References

[1] Isaac L. Chuang and M. A. Nielsen. “Prescription for experimental determination of the dynamics of a quantum black box”. en. In: *Journal of Modern Optics* 44.11-12 (Nov. 1997). arXiv:quant-ph/9610001, pp. 2455–2467. ISSN: 0950-0340, 1362-3044. DOI: 10.1080/09500349708231894. URL: http://arxiv.org/abs/quant-ph/9610001 (visited on 10/09/2024).

[2] Marcus Cramer et al. “Efficient quantum state tomography”. en. In: *Nature Communications* 1.1 (Dec. 2010), p. 149. ISSN: 2041-1723. DOI: 10.1038/ncomms1147. URL: https://www.nature.com/articles/ncomms1147 (visited on 08/22/2024).

[3] Daniel F. V. James et al. “Measurement of qubits”. en. In: *Physical Review A* 64.5 (Oct. 2001), p. 052312. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.64.052312. URL: https://link.aps.org/doi/10.1103/PhysRevA.64.052312 (visited on 08/22/2024).

[4] Joseph M Lukens et al. “A practical and efficient approach for Bayesian quantum state estimation”. en. In: *New Journal of Physics* 22.6 (June 2020), p. 063038. ISSN: 1367-2630. DOI: 10.1088/1367-2630/ab8efa. URL: https://iopscience.iop.org/article/10.1088/1367-2630/ab8efa (visited on 08/22/2024).

[5] Sanjaya Lohani et al. “Machine learning assisted quantum state estimation”. en. In: *Machine Learning: Science and Technology* 1.3 (Sept. 2020), p. 035007. ISSN: 2632-2153. DOI: 10.1088/2632-2153/ab9a21. URL: https://iopscience.iop.org/article/10.1088/2632-2153/ab9a21 (visited on 08/22/2024).

[6] Andrea Rocchetto et al. “Learning hard quantum distributions with variational autoencoders”. en. In: *npj Quantum Information* 4.1 (June 2018), p. 28. ISSN: 2056-6387. DOI: 10.1038/s41534-018-0077-z. URL: https://www.nature.com/articles/s41534-018-0077-z (visited on 09/11/2024).

[7] Stefanie Czischek et al. “Spiking neuromorphic chip learns entangled quantum states”. en. In: *SciPost Physics* 12.1 (Jan. 2022), p. 039. ISSN: 2542-4653. DOI: 10.21468/SciPostPhys.12.1.039. URL: https://scipost.org/10.21468/SciPostPhys.12.1.039 (visited on 09/02/2024).

[8] Peter Cha et al. “Attention-based quantum tomography”. en. In: *Machine Learning: Science and Technology* 3.1 (Mar. 2022), 01LT01. ISSN: 2632-2153. DOI: 10.1088/2632-2153/ac362b. URL: https://iopscience.iop.org/article/10.1088/2632-2153/ac362b (visited on 08/22/2024).

[9] Giacomo Torlai and Roger G. Melko. “Machine-Learning Quantum States in the NISQ Era”. en. In: *Annual Review of Condensed Matter Physics* 11.1 (Mar. 2020), pp. 325–344. ISSN: 1947-5454, 1947-5462. DOI: 10.1146/annurev-conmatphys-031119-050651. URL: https://www.annualreviews.org/doi/10.1146/annurev-conmatphys-031119-050651 (visited on 08/22/2024).

[10] Chuangtao Chen et al. “Reconstructing a quantum state with a variational autoencoder”. en. In: *International Journal of Quantum Information* 19.08 (Dec. 2021), p. 2140005. ISSN: 0219-7499, 1793-6918. DOI: 10.1142/S0219749921400050. URL: https://www.worldscientific.com/doi/abs/10.1142/S0219749921400050 (visited on 09/12/2024).

[11] Giacomo Torlai et al. “Neural-network quantum state tomography”. en. In: *Nature Physics* 14.5 (May 2018), pp. 447–450. ISSN: 1745-2473, 1745-2481. DOI: 10.1038/s41567-018-0048-5. URL: https://www.nature.com/articles/s41567-018-0048-5 (visited on 08/22/2024).

[12] Andreas Baumbach et al. “Quantum many-body states: A novel neuromorphic application”. en. In: *Neuro-Inspired Computational Elements Conference*. Virtual Event USA: ACM, Mar. 2022, pp. 104–106. ISBN: 978-1-4503-9559-5. DOI: 10.1145/3517343.3517379. URL: https://dl.acm.org/doi/10.1145/3517343.3517379 (visited on 08/22/2024).

[13] Robert Klassert et al. “Variational learning of quantum ground states on spiking neuromorphic hardware”. en. In: *iScience* 25.8 (Aug. 2022), p. 104707. ISSN: 25890042. DOI: 10.1016/j.isci.2022.104707. URL: https://linkinghub.elsevier.com/retrieve/pii/S2589004222009798 (visited on 09/04/2024).

[14] Christian Pehle et al. *The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity*. en. arXiv:2201.11063 [cond-mat, q-bio]. Feb. 2022. URL: http://arxiv.org/abs/2201.11063 (visited on 08/22/2024).

[15] Zhaokun Zhou et al. *Spikformer V2: Join the High Accuracy Club on ImageNet with an SNN Ticket*. en. arXiv:2401.02020 [cs]. Jan. 2024. URL: http://arxiv.org/abs/2401.02020 (visited on 09/17/2024).

[16] Ali Samadzadeh et al. “Convolutional Spiking Neural Networks for Spatio-Temporal Feature Extraction”. en. In: *Neural Processing Letters* 55.6 (Dec. 2023). Publisher: Springer Science and Business Media LLC, pp. 6979–6995. ISSN: 1370-4621, 1573-773X. DOI: 10.1007/s11063-023-11247-8. URL: https://link.springer.com/10.1007/s11063-023-11247-8 (visited on 09/17/2024).

[17] Hiromichi Kamata, Yusuke Mukuta, and Tatsuya Harada. *Fully Spiking Variational Autoencoder*. en. arXiv:2110.00375 [cs]. Dec. 2021. URL: http://arxiv.org/abs/2110.00375 (visited on 09/17/2024).

[18] Qiugang Zhan et al. *ESVAE: An Efficient Spiking Variational Autoencoder with Reparameterizable Poisson Spiking Sampling*. en. arXiv:2310.14839 [cs]. Aug. 2024. URL: http://arxiv.org/abs/2310.14839 (visited on 09/18/2024).

[19] Max Born. “Zur Quantenmechanik der Sto□vorg□nge”. de. In: *Zeitschrift f□r Physik* 37.12 (Dec. 1926), pp. 863–867. ISSN: 1434-6001, 1434-601X. DOI: 10.1007/BF01397477. URL: http://link.springer.com/10.1007/BF01397477 (visited on 10/09/2024).

[20] Marcus Cramer et al. “Efficient quantum state tomography”. en. In: *Nature Communications* 1.1 (Dec. 2010), p. 149. ISSN: 2041-1723. DOI: 10.1038/ncomms1147. URL: https://www.nature.com/articles/ncomms1147 (visited on 08/22/2024).

[21] Christian Reimer et al. “Generation of multiphoton entangled quantum states by means of integrated frequency combs”. In: *Science* 351 (Mar. 2016), pp. 1176–1180. DOI: 10.1126/science.aad8532.

[22] Warren Mcculloch and Walter Pitts. “A Logical Calculus of Ideas Immanent in Nervous Activity”. In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 127–147.

[23] F. Rosenblatt. *The perceptron - A perceiving and recognizing automaton*. Tech. rep. 85-460-1. Ithaca, New York: Cornell Aeronautical Laboratory, Jan. 1957.

[24] Pavithra. Dec. 2023. URL: https://siliconlotus.net/listing/decoding-the-perceptron-foundation-of-neural-networks/.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[26] Juan Carrasquilla et al. “Reconstructing quantum states with generative models”. en. In: *Nature Machine Intelligence* 1.3 (Mar. 2019), pp. 155–161. ISSN: 2522-5839. DOI: 10.1038/s42256-019-0028-1. URL: https://www.nature.com/articles/s42256-019-0028-1 (visited on 08/22/2024).

[27] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML]. URL: https://arxiv.org/abs/1312.6114.

[28] Klaus M. Stiefel and Jay S. Coggan. “The energy challenges of artificial superintelligence”. In: *Frontiers in Artificial Intelligence* 6 (2023). ISSN: 2624-8212. DOI: 10.3389/frai.2023.1240653. URL: https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2023.1240653.

[29] Anders Sandberg and Nick Bostrom. *Whole Brain Emulation: A Roadmap*. Tech. rep. 2008-3. Future of Humanity Institute, Oxford University, 2008. URL: https://www.fhi.ox.ac.uk/reports/2008-3.pdf.

[30] IEA. *Electricity 2024*. Licence: CC BY 4.0. Paris, 2024. URL: https://www.iea.org/reports/electricity-2024.

[31] Neural Academy. *The Action Potential*. Accessed: 2024-10-22. 2018. URL: https://www.youtube.com/watch?v=6qS83wD29PY.

[32] Paul Davidovits. *Physics in Biology and Medicine*. 5th. San Diego, CA: Academic Press, 2019. ISBN: 9780128137161. URL: https://www.sciencedirect.com/book/9780128137161/physics-in-biology-and-medicine.

[33] A. L. Hodgkin and A. F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of Physiology* 117.4 (1952), pp. 500–544. DOI: https://doi.org/10.1113/jphysiol.1952.sp004764. eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1952.sp004764. URL: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764.

[34] Mihai Alexandru Petrovici. *Form Versus Function: Theory and Models for Neuronal Substrates*. en. Springer Theses. ISSN: 2190-5053, 2190-5061. Cham: Springer International Publishing, 2016. ISBN: 978-3-319-39551-7 978-3-319-39552-4. DOI: 10.1007/978-3-319-39552-4. URL: http://link.springer.com/10.1007/978-3-319-39552-4 (visited on 09/17/2024).

[35] Jiankun Chen et al. *SAR Image Classification Based on Spiking Neural Network through Spike-Time Dependent Plasticity and Gradient Descent*. June 2021. DOI: 10.48550/arXiv.2106.08005.

[36] Christof Koch and Idan Segev, eds. *Methods in Neuronal Modeling: From Ions to Networks*. 2nd. Computational Neuroscience Series. Cambridge, MA: MIT Press, 1998. ISBN: 9780262112310.

[37] Richard Gerum and Achim Schilling. "Integration of Leaky-Integrate-and-Fire Neurons in Standard Machine Learning Architectures to Generate Hybrid Networks: A Surrogate Gradient Approach". In: *Neural computation* 33 (July 2021), pp. 1–26. DOI: 10.1162/neco_a_01424.

[38] Kenneth Stewart et al. "Encoding Event-Based Data With a Hybrid SNN Guided Variational Autoencoder in Neuromorphic Hardware". en. In: *Neuro-Inspired Computational Elements Conference*. arXiv:2104.00165 [cs]. Mar. 2022, pp. 88–97. DOI: 10.1145/3517343.3517372. URL: http://arxiv.org/abs/2104.00165 (visited on 10/18/2024).

[39] Christian Pehle et al. "The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity". In: *Frontiers in Neuroscience* 16 (2022). ISSN: 1662-453X. DOI: 10.3389/fnins.2022.795876. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.795876.

[40] Hadi Vafaii, Dekel Galor, and Jacob L. Yates. *Poisson Variational Autoencoder*. en. arXiv:2405.14473 [cs, q-bio]. May 2024. URL: http://arxiv.org/abs/2405.14473 (visited on 09/18/2024).

[41] *SIC-POVM - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/SIC-POVM. [Accessed 23-09-2024].

[42] Jason K Eshraghian et al. "Training spiking neural networks using lessons from deep learning". In: *Proceedings of the IEEE* 111.9 (2023), pp. 1016–1054.

[43] Philipp Spilger et al. "hxtorch.snn: Machine-learning-inspired Spiking Neural Network Modeling on BrainScaleS-2". en. In: *Neuro-Inspired Computational Elements Conference*. San Antonio TX USA: ACM, Apr. 2023, pp. 57–62. ISBN: 978-1-4503-9947-0. DOI: 10.1145/3584954.3584993. URL: https://dl.acm.org/doi/10.1145/3584954.3584993 (visited on 09/19/2024).

[44] Jason Ansel et al. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation". In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. URL: https://pytorch.org/assets/pytorch2-2.pdf.

A

Calculations

A.1. Calculations Measurement operators

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$M^{(0)} = \frac{1}{4}\left(I + \vec{s}^{(0)} \cdot \vec{\sigma}\right)$$

$$= \frac{1}{4}\left(I + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{bmatrix}\right)$$

$$= \frac{1}{4}\left(I + (0 \cdot \sigma_x + 0 \cdot \sigma_y + 1 \cdot \sigma_z)\right)$$

$$= \frac{1}{4}\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\right)$$

$$= \frac{1}{4}\begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 0 \end{bmatrix}$$

$$M^{(1)} = \frac{1}{4}\left(I + \vec{s}^{(1)} \cdot \vec{\sigma}\right)$$

$$= \frac{1}{4}\left(I + \begin{bmatrix} \frac{2\sqrt{2}}{3} \\ 0 \\ -\frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{bmatrix}\right)$$

$$= \frac{1}{4}\left(I + \left(\frac{2\sqrt{2}}{3} \cdot \sigma_x + 0 \cdot \sigma_y + -\frac{1}{3} \cdot \sigma_z\right)\right)$$

$$= \frac{1}{4}\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \frac{2\sqrt{2}}{3}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + \left(-\frac{1}{3}\right)\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\right)$$

$$= \frac{1}{4}\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & \frac{2\sqrt{2}}{3} \\ \frac{2\sqrt{2}}{3} & 0 \end{bmatrix} + \begin{bmatrix} -\frac{1}{3} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}\right)$$

$$= \frac{1}{4}\begin{bmatrix} 1 - \frac{1}{3} & \frac{2\sqrt{2}}{3} \\ \frac{2\sqrt{2}}{3} & 1 + \frac{1}{3} \end{bmatrix} = \frac{1}{4}\begin{bmatrix} \frac{2}{3} & \frac{2\sqrt{2}}{3} \\ \frac{2\sqrt{2}}{3} & \frac{4}{3} \end{bmatrix}$$

$$M^{(2)} = \frac{1}{4}\left(I + \vec{s}^{(2)} \cdot \vec{\sigma}\right)$$

$$= \frac{1}{4}\left(I + \begin{bmatrix} -\frac{\sqrt{2}}{3} \\ \frac{\sqrt{2}}{3} \\ -\frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{bmatrix}\right)$$

$$= \frac{1}{4}\left(I + \left(-\frac{\sqrt{2}}{3} \cdot \sigma_x + \frac{\sqrt{2}}{3} \cdot \sigma_y + -\frac{1}{3} \cdot \sigma_z\right)\right)$$

$$= \frac{1}{4}\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \left(-\frac{\sqrt{2}}{3}\right)\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + \frac{\sqrt{2}}{3}\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} + \left(-\frac{1}{3}\right)\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\right)$$

$$M^{(3)} = \frac{1}{4}\left(I + \vec{s}^{(3)} \cdot \vec{\sigma}\right)$$

$$= \frac{1}{4}\left(I + \begin{bmatrix} -\frac{\sqrt{2}}{3} \\ -\frac{\sqrt{2}}{3} \\ -\frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{bmatrix}\right)$$

$$= \frac{1}{4}\left(I + \left(-\frac{\sqrt{2}}{3} \cdot \sigma_x + -\frac{\sqrt{2}}{3} \cdot \sigma_y + -\frac{1}{3} \cdot \sigma_z\right)\right)$$

$$= \frac{1}{4}\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & -\frac{\sqrt{2}}{3} \\ -\frac{\sqrt{2}}{3} & 0 \end{bmatrix} + \begin{bmatrix} 0 & -\frac{\sqrt{2}}{3}i \\ \frac{\sqrt{2}}{3}i & 0 \end{bmatrix} + \begin{bmatrix} -\frac{1}{3} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}\right)$$

B

Other results

In this appendix all the other experiments done are described here.

B.1. Initial experiment

A SVAE is initially build using the original paper of the VAE but then with spiking layers. This was to prove that it was possible to build a SVAE on the cpu that was able to learn the probabilities of a quantum state. In Fig. B.1 is seen that a quantum computer or simulator runs an experiment S amount of shots and saves the result of the experiments in a database. The database is build on top of the database library of pytorch and can be used by the pytorch framework [44]. The encoder and decoder are build using a three layer network consisting of two spiking layers and a simple output layer that sums the spikes and calculates the spiking probabilities per output neuron. The model uses no optimisation yet and the results are seen in B.2.

```
# Define hyperparameters
shots = 100_000 # amount of shots taken by the quantum simulator
beta = 0.819
num_steps = 200
num_epochs = 1
learning_rate = 1e-3
num_workers = 0
shuffle = False
split = [0.6, 0.2, 4**n *500]

# Reproduction of paper (qubits, training batch size, total samples, batch size samples)
parameters = [
    (3, 100, 20_000, 20_000),
    (4, 200, 100_000, 40_000),
    (5, 500, 4**5 * 500,40_000),  # 4^5 * 500
    (6, 600, 4**6 * 500, 40_000),  # 4^6 * 500
    (7, 800, 4**7 * 500, 40_000),  # 4^7 * 500
    (8, 1000, 4**8 * 500,80_000)  # 4^8 * 500
]
```

Listing B.1: Settings

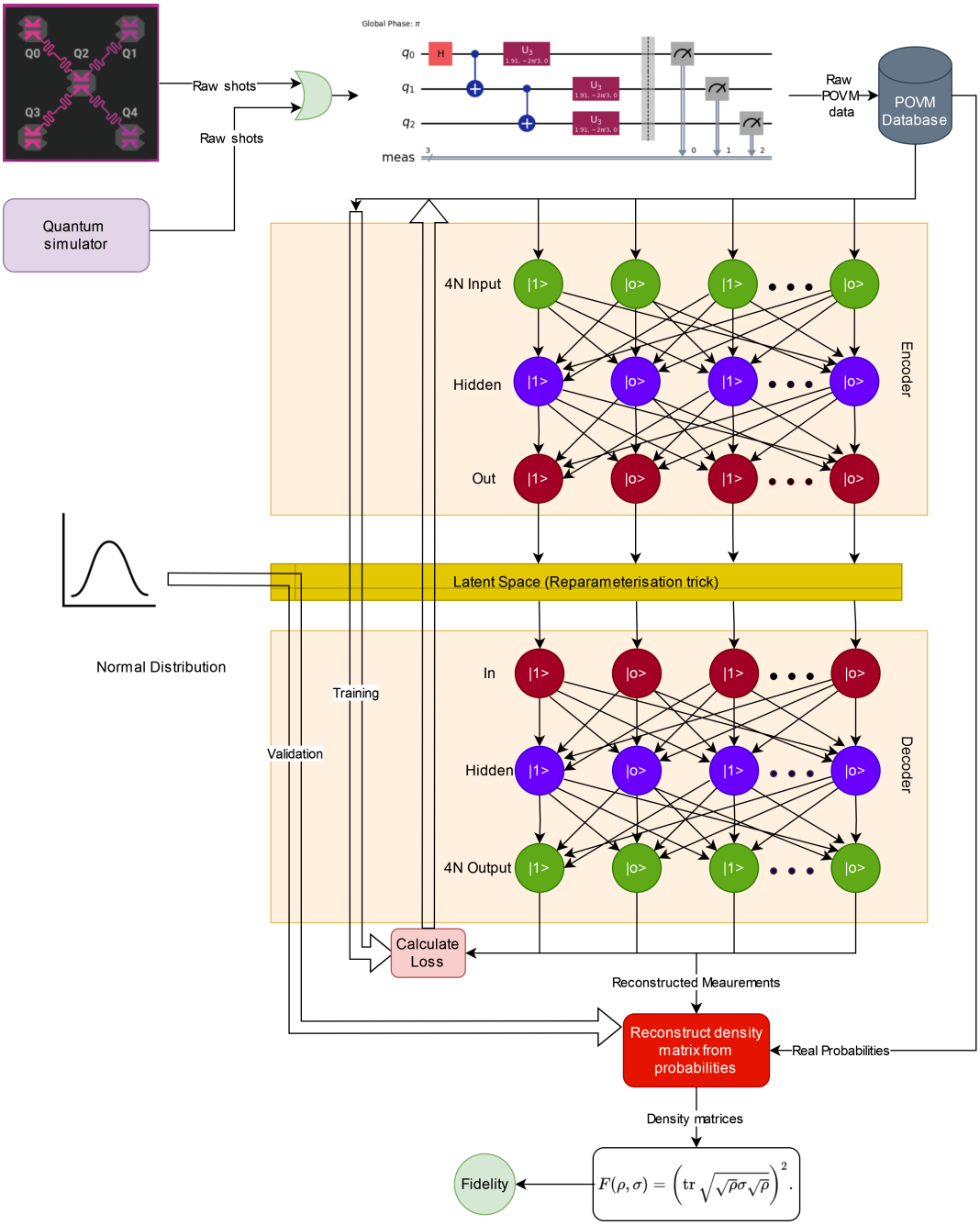


Figure B.1: SVAE first build

B.2. Results of the initial SVAE

This is the result of the initial experiment

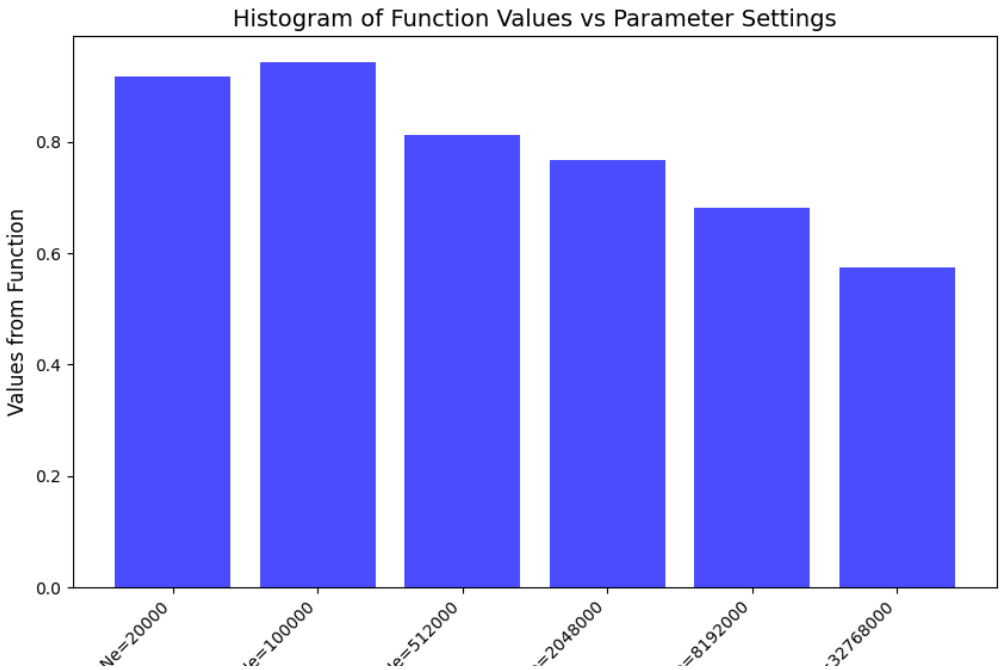
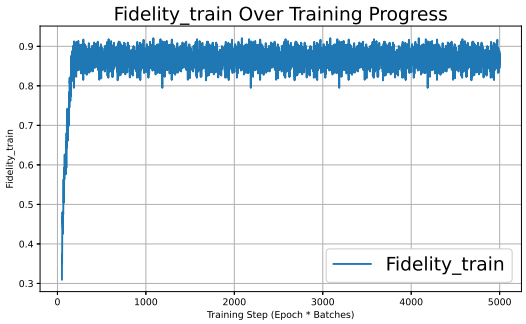


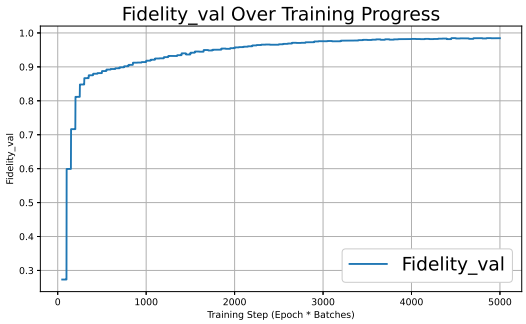
Figure B.2: Experiment with 3 to 8 qubits and $10 * 10^4$ samples from the quantum simulator

B.3. Training details per qubit

3 Qubits



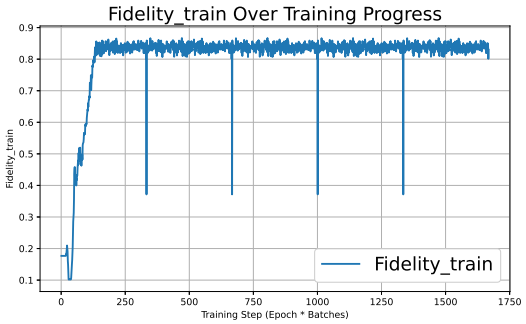
(a) Fidelity_train for 3 Qubits



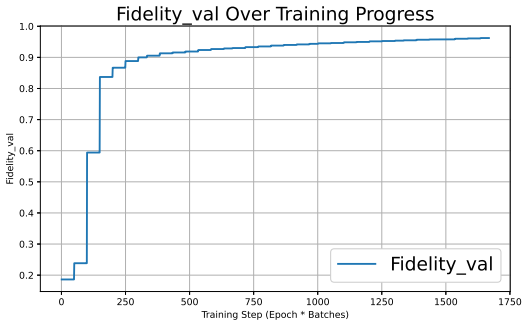
(b) Fidelity_val for 3 Qubits

Figure B.3: Fidelity analysis for 3 qubits: Training vs Validation.

4 Qubits



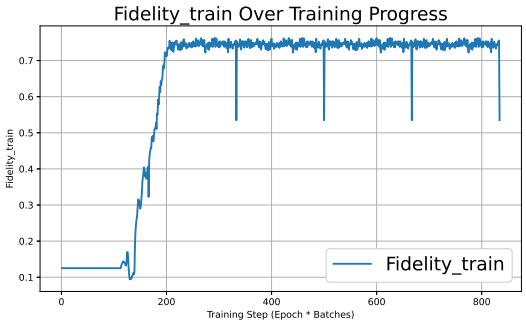
(a) Fidelity_train for 4 Qubits



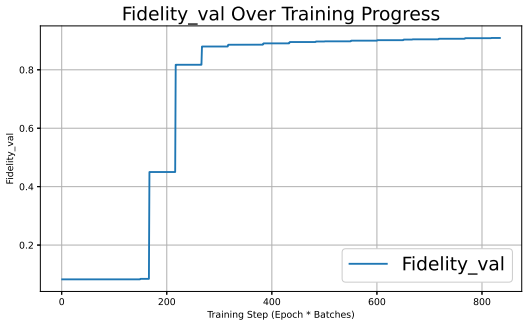
(b) Fidelity_val for 4 Qubits

Figure B.4: Fidelity analysis for 4 qubits: Training vs Validation.

5 Qubits



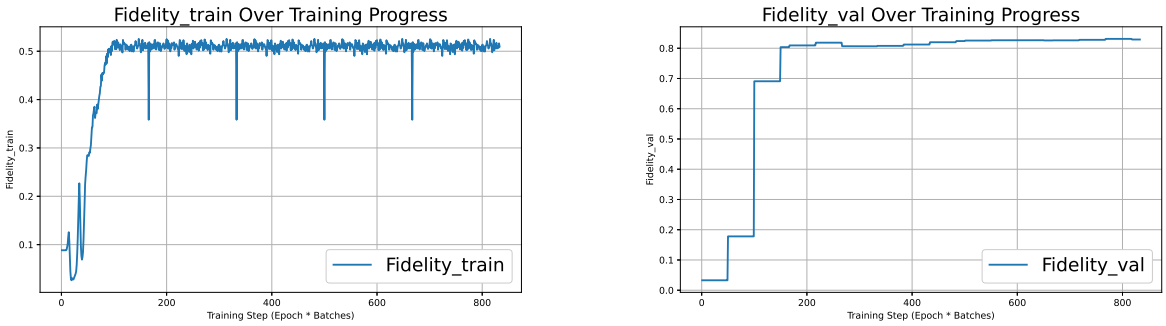
(a) Fidelity_train for 5 Qubits



(b) Fidelity_val for 5 Qubits

Figure B.5: Fidelity analysis for 5 qubits: Training vs Validation.

6 Qubits

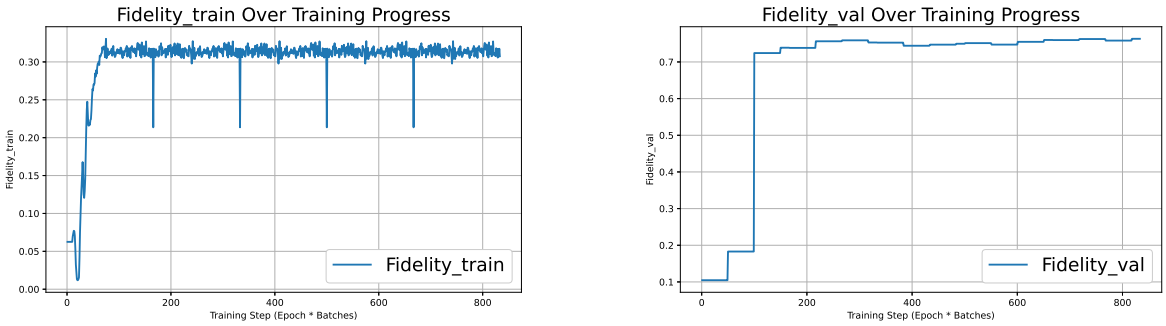


(a) Fidelity_train for 6 Qubits

(b) Fidelity_val for 6 Qubits

Figure B.6: Fidelity analysis for 6 qubits: Training vs Validation.

7 Qubits



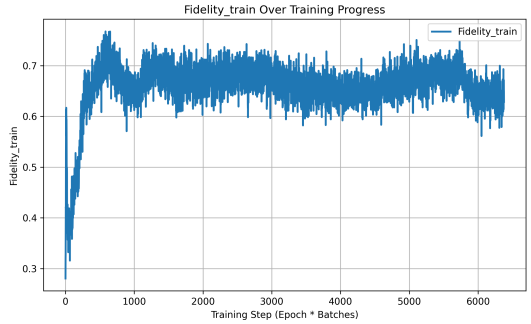
(a) Fidelity_train for 7 Qubits

(b) Fidelity_val for 7 Qubits

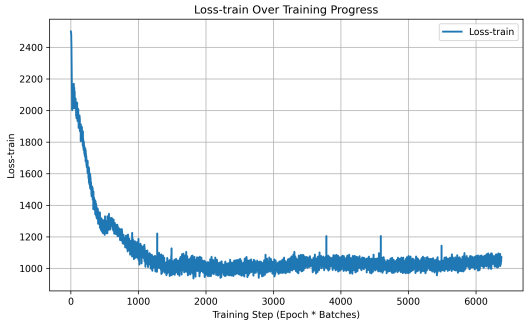
Figure B.7: Fidelity analysis for 7 qubits: Training vs Validation.

B.4. BS2 optimisations

B.4.1. Result of training 3 qubits



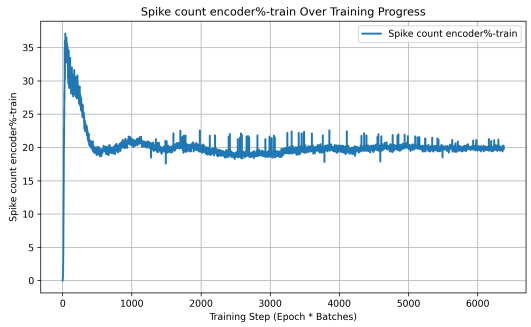
(a) Fidelity_train



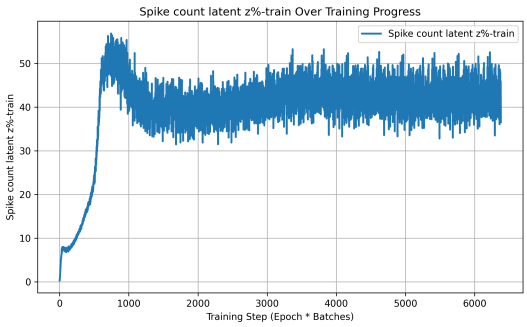
(b) Loss-train

(c) MMD Loss-train

(d) MSE Loss-train



(e) Spike Count Encoder-train

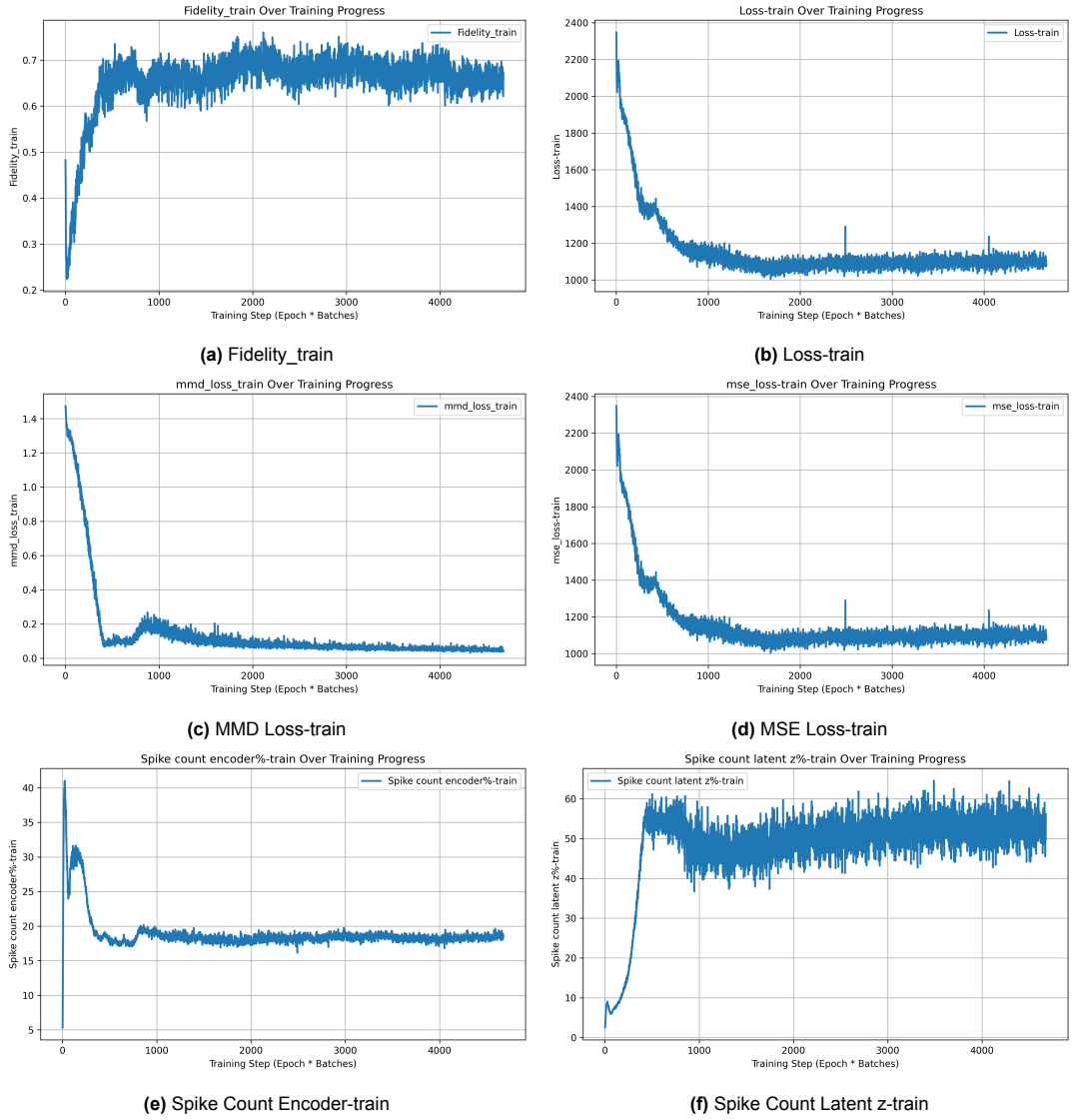


(f) Spike Count Latent z-train

Figure B.8: Plots from folder: training_metrics_2024-12-14

B.4.2. Result of 3 qubits with increased hidden size

The hidden size was increased from 20 * n to 32 * n.



(a) Fidelity_train

(b) Loss-train

(c) MMD Loss-train

(d) MSE Loss-train

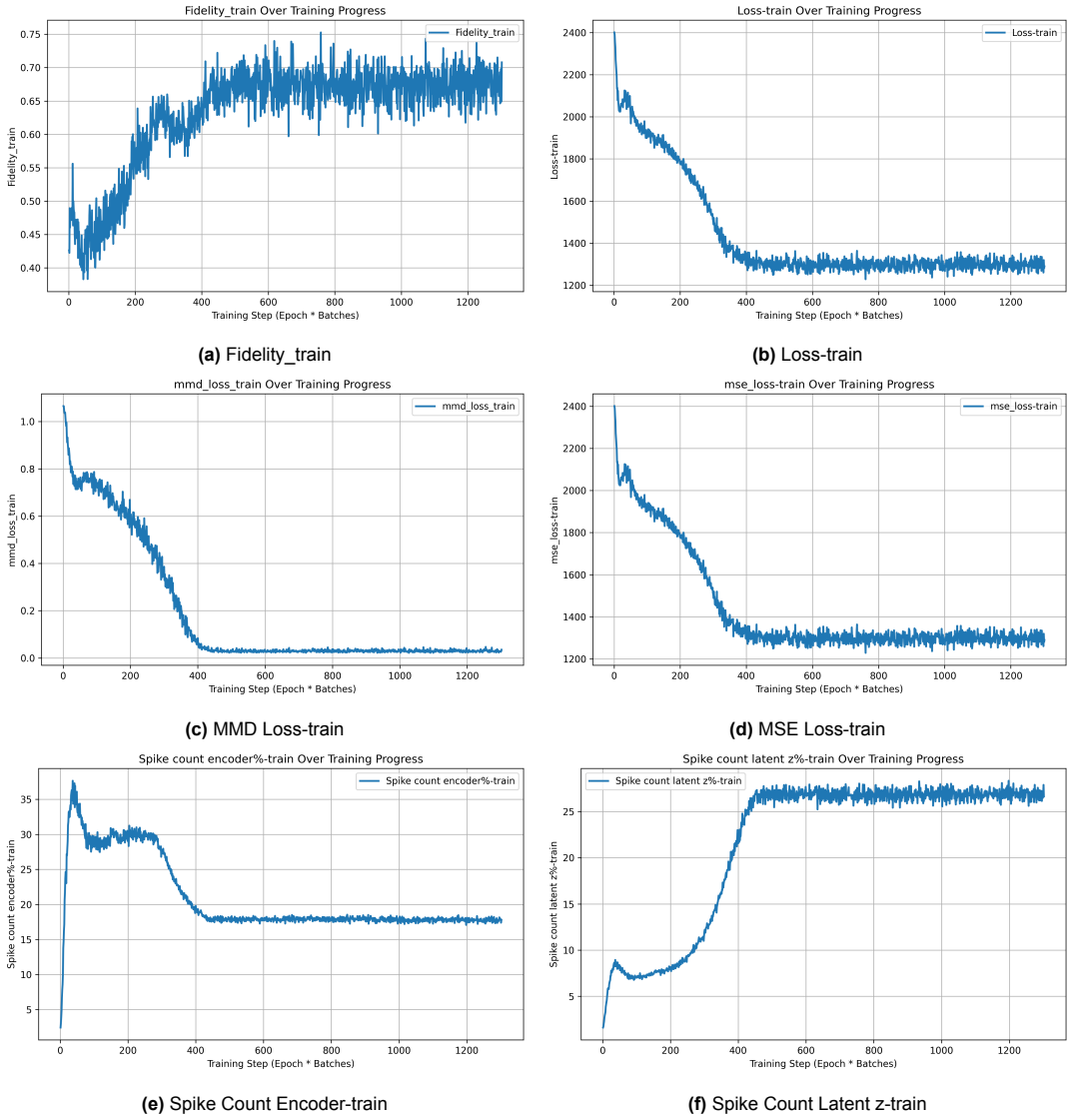
(e) Spike Count Encoder-train

(f) Spike Count Latent z-train

Figure B.9: Plots from folder: training_metrics_2024-12-15

B.4.3. Result of 3 qubits with increased latent size

The hidden size was increased from 2 * 2**N to 3 * 2**N This test was done on 16/12/2024



(a) Fidelity_train

(b) Loss-train

(c) MMD Loss-train

(d) MSE Loss-train

(e) Spike Count Encoder-train

(f) Spike Count Latent z-train

Figure B.10: Plots from folder: training_metrics_2024-12-18