UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fakultät für Elektrotechnik, Informatik und Mathematik
Arbeitsgruppe Quanteninformatik

Classical Simulation of Quantum Circuits with Restricted Boltzmann Machines

Master's Thesis

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Jannes Stubbemann

submitted to:

Jun. Prof. Dr. Sevag Gharibian

and

Prof. Dr. Eyke Hüllermeier

Paderborn, September 30, 2020

Declaration

(Translation from German)

I hereby declare that I prepared this thesis entirely on my own and have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

Original Declaration Text in German:

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

City, Date

Signature

Contents

List of Figures

1 Nomenclature and Notations

This chapter introduces mathematical notations and expressions used throughout this study. Its focus is on linear algebra, the mathematical foundation of *quantum mechanics*. It is thought as a lookup reference for the following chapters.

Vectors are represented in the *Bra-ket* notation. *Ket*-vectors $|\psi\rangle$ denote column vectors:

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}. \tag{1.1}$$

The *bra* $\langle\psi|$ of a vector $|\psi\rangle$ is its hermitian conjugate, $\langle\psi| = |\psi\rangle^\dagger$. It is the transpose of the vector with complex-conjugated entries. For the ket vector $|\psi\rangle$ above, the corresponding bra-vector would be:

$$\langle\psi| = \begin{pmatrix} \alpha^* & \beta^* \end{pmatrix}, \tag{1.2}$$

where the complex-conjugate of some complex number $\alpha = a + bi$ is defined as $\alpha^* = a - bi$. Using this notation, the *inner product* of two vectors $|\psi\rangle$ and $|\phi\rangle$ can be written as:

$$\langle\psi \,|\, \phi\rangle = \sum_j \psi_j^* \phi_j. \tag{1.3}$$

The *outer product* $|\psi\rangle \langle\phi|$ defines the matrix A with entries a_{ij} given by:

$$a_{ij} = \psi_i \phi_j^*. \tag{1.4}$$

The class of matrices of special interest in quantum computing are *unitary* matrices. A complex-valued square matrix U is unitary if:

$$UU^\dagger = U^\dagger U = I. \tag{1.5}$$

Unitary matrices are norm preserving and thus represent rotations in the vector space. The *tensor product* $\cdot \otimes \cdot$ of two matrices $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ and $B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_2 2 \end{pmatrix}$ is defined as:

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{pmatrix} \tag{1.6}$$

$$= \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{pmatrix}. \tag{1.7}$$

The dimensions of the tensor product of two matrices with dimensions $n_A \times m_A$ and $n_B \times m_B$ is $n_A n_B \times m_A m_B$. The tensor product for vectors is defined accordingly. The *Kroenecker Delta* δ_{ij} sometimes occurs in quantum physics. It is defined as:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} . \tag{1.8}$$

and by that defines the dot product of two vectors $|\psi_i\rangle$ and $|\psi_j\rangle$ from a set of orthonormal vectors.

2 Quantum Computing

This chapter gives an introduction to the field of *Quantum Computation*. It covers the basics of qubits, quantum gates and circuits, and gives an overview of quantum complexity classes. The chapter is based on [37], which offers a profound introduction into the field.

2.1 Qubits

While classical computers harness classical physical phenomena like electrical current to perform calculations, quantum computers harness *quantum mechanical* effects. The simplest quantum mechanical system is a quantum bit, or *qubit* for short. It is the fundamental building block of a quantum computer.

Mathematically, a qubit is a two-dimensional complex-valued unit vector, also called the *state vector* or *wave function*:

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2.1}$$

with *amplitudes* $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. Rewriting $|\psi\rangle$ as

$$|\psi\rangle = e^{i\gamma} \left(\cos\frac{\theta}{2} |0\rangle + e^{i\rho} \sin\frac{\theta}{2} |1\rangle \right) \tag{2.2}$$

with $\theta, \rho, \gamma \in \mathbb{R}$ allows an intuitive interpration of a qubit as a point on the surface of the three-dimensional unit sphere, called the *Bloch Sphere*, which is visualized in figure **??**. The factor $e^{i\gamma}$ in front is called a *global phase* and can be ignored:

$$|\psi\rangle = \cos\frac{\theta}{2} |0\rangle + e^{i\rho} \sin\frac{\theta}{2} |1\rangle . \tag{2.3}$$

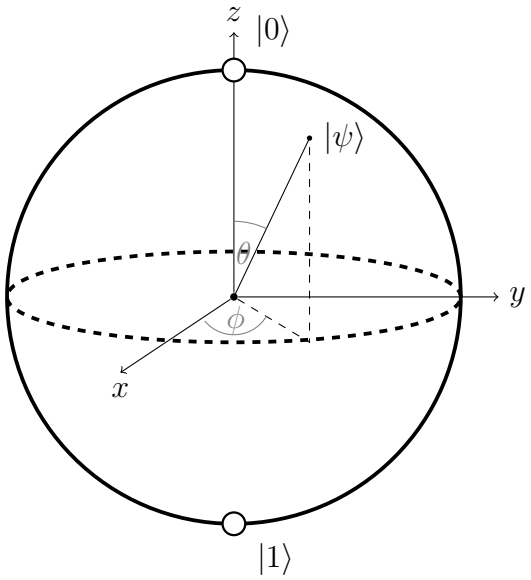


Figure 2.1: Bloch Sphere representation of a qubit state $|\psi\rangle$. $|\psi\rangle$ points to an arbitrary direction on the three dimensional unit sphere.

The states on the north and south pole of the Bloch Sphere are called the *computational basis states*, defined as:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.4}$$

and

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.5}$$

These states correspond to the classical states 0 and 1 of a classical bit. Thus, another way of writing the wave function of a qubit, which is also the most practical one for quantum computation, is a linear combination of the two computational basis states:

$$|\psi\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle. \tag{2.6}$$

A qubit whose amplitudes α and β are both non-zero is in a so-called *superposition* of the two computational basis states. A qubit in a superposition can be interpreted as being in both states $|0\rangle$ and $|1\rangle$ at the same time. This property is one of the underlying reasons why the classical simulation of quantum systems is computationally so hard, and it is one of the sources of the potential computational power of quantum computers. Nevertheless, it is not possible to read the values of the amplitudes α and β directly.

The qubit can only be in a superposition as long as it is completely isolated from its environment. Such a state is called a *coherent* state. As soon as the qubit

interacts with its environment, as it is necessary to read its value, it collapses randomly into one of the two computational basis states. All the information that was stored in the amplitudes gets lost in this process.

In a quantum computer, the qubits are in a coherent state, and potentially in a superposition, during the computation. At the end of the computation, a so-called *projective measurement* is performed to read the states of the individual qubits. A projective measurement is given by *operators* $\mathbf{M} = \{M_i\}$, in this case $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$. The probabilities $p(0)$ and $p(1)$ of observing a qubit in the $|0\rangle$ or $|1\rangle$ state are then defined by the amplitudes:

$$\begin{aligned} p(0) &= \langle\psi| M_0^\dagger M_0 |\psi\rangle && (2.7)\\ &= \langle\psi| M_0 |\psi\rangle && (2.8)\\ &= \alpha \langle 0 \,|\, 0\rangle \langle 0| \alpha |0\rangle + \beta \langle 1 \,|\, 0\rangle \langle 0| \beta |1\rangle && (2.9)\\ &= \alpha^2 \langle 0 \,|\, 0\rangle \langle 0 \,|\, 0\rangle + \beta^2 \langle 1 \,|\, 0\rangle \langle 0 \,|\, 1\rangle && (2.10)\\ &= \alpha^2 && (2.11) \end{aligned}$$

and

$$\begin{aligned} p(1) &= \langle\psi| M_1^\dagger M_1 |\psi\rangle && (2.12)\\ &= \langle\psi| M_1 |\psi\rangle && (2.13)\\ &= \alpha \langle 0 \,|\, 1\rangle \langle 1| \alpha |0\rangle + \beta \langle 1 \,|\, 1\rangle \langle 1| \beta |1\rangle && (2.14)\\ &= \alpha^2 \langle 0 \,|\, 1\rangle \langle 1 \,|\, 0\rangle + \beta^2 \langle 1 \,|\, 1\rangle \langle 1 \,|\, 1\rangle && (2.15)\\ &= \beta^2 && (2.16) \end{aligned}$$

as $M_i^\dagger M_i = M_i$ and $\langle i \,|\, j\rangle = \delta_{ij}$.

After the measurement, the state of the qubit will be either

$$\begin{aligned} \left|\psi^\dagger\right\rangle &= \frac{M_0 |\psi\rangle}{\sqrt{p(0)}} && (2.17)\\ &= \frac{|0\rangle\langle 0| (\alpha |0\rangle + \beta |1\rangle)}{\sqrt{\alpha^2}} && (2.18)\\ &= \frac{\alpha |0\rangle \langle 0 \,|\, 0\rangle + \beta |0\rangle \langle 0 \,|\, 1\rangle}{\alpha} && (2.19)\\ &= \frac{\alpha |0\rangle}{\alpha} && (2.20)\\ &= |0\rangle && (2.21) \end{aligned}$$

or

$$\left|\psi^{\dagger}\right\rangle = \frac{M_1 \left|\psi\right\rangle}{\sqrt{p(1)}} \tag{2.22}$$

$$= \frac{\left|1\right\rangle \left\langle 1\right| \left(\alpha \left|0\right\rangle + \beta \left|1\right\rangle\right)}{\sqrt{\beta^2}} \tag{2.23}$$

$$= \frac{\alpha \left|1\right\rangle \left\langle 1 \,|\, 0\right\rangle + \beta \left|1\right\rangle \left\langle 1 \,|\, 1\right\rangle}{\beta} \tag{2.24}$$

$$= \frac{\beta \left|1\right\rangle}{\beta} \tag{2.25}$$

$$= \left|1\right\rangle \tag{2.26}$$

respectively, implying that the state of the qubit collapsed into one of the two computational basis states. Thus, any succinct measurement will result in the same outcome. In order to perform multiple measurements on the same state, it has to be prepared multiple times.

Two special qubit states, which often occur in the domain of quantum computation, are the $|-\rangle$ and $|+\rangle$ state defined as:

$$\left|-\right\rangle = \frac{1}{\sqrt{2}} \cdot \left|0\right\rangle - \frac{1}{\sqrt{2}} \cdot \left|1\right\rangle \tag{2.27}$$

and

$$\left|+\right\rangle = \frac{1}{\sqrt{2}} \cdot \left|0\right\rangle + \frac{1}{\sqrt{2}} \cdot \left|1\right\rangle . \tag{2.28}$$

Those states differ in a *relative phase*. Both have the same measurement statistics, lying between the states $|0\rangle$ and $|1\rangle$ on the Bloch Sphere:

$$p(0) = \left\langle +\right| M_0 \left|+\right\rangle \tag{2.29}$$

$$= \frac{1}{\sqrt{2}} \left\langle 0 \,|\, 0\right\rangle \left\langle 0\right| \frac{1}{\sqrt{2}} \left|0\right\rangle + \frac{1}{\sqrt{2}} \left\langle 1 \,|\, 0\right\rangle \left\langle 0\right| \frac{1}{\sqrt{2}} \left|1\right\rangle \tag{2.30}$$

$$= \frac{1}{2} \left\langle 0 \,|\, 0\right\rangle \left\langle 0 \,|\, 0\right\rangle + \frac{1}{2} \left\langle 1 \,|\, 0\right\rangle \left\langle 0 \,|\, 1\right\rangle \tag{2.31}$$

$$= \frac{1}{2} \tag{2.32}$$

and

$$p(1) = \langle +| M_1 |+\rangle \tag{2.33}$$
$$= \frac{1}{\sqrt{2}} \langle 0 \,|\, 1\rangle \langle 1| \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} \langle 1 \,|\, 1\rangle \langle 1| \frac{1}{\sqrt{2}} |1\rangle \tag{2.34}$$
$$= \frac{1}{2} \langle 0 \,|\, 1\rangle \langle 1 \,|\, 0\rangle + \frac{1}{2} \langle 1 \,|\, 1\rangle \langle 1 \,|\, 1\rangle \tag{2.35}$$
$$= \frac{1}{2} \tag{2.36}$$

for the $|+\rangle$ state. The same probabilities hold for the $|-\rangle$ state. Again, the state will be destroyed on measurement and be either $|0\rangle$ or $|1\rangle$ afterward, depending on the measurement outcome.

2.2 Multiple Qubits and Entanglement

The state vector of a single qubit lies in a two-dimensional space, assigning a complex-valued amplitude to each of the both possible measurement outcomes $|0\rangle$ and $|1\rangle$. The state vector of a multi-qubit system is defined by the *tensor product* of the state vectors of the individual subsystems. For a two-qubit system consisting of $|\psi_1\rangle = \alpha_1 |0\rangle + \beta_1 |1\rangle$ and $|\psi_2\rangle = \alpha_2 |0\rangle + \beta_2 |1\rangle$, the state vector is given by:

$$|\psi_{1,2}\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \tag{2.37}$$
$$= (\alpha_1 |0\rangle + \beta_1 |1\rangle) \otimes (\alpha_2 |0\rangle + \beta_2 |1\rangle) \tag{2.38}$$
$$= \alpha_1\alpha_2 |0\rangle |0\rangle + \alpha_1\beta_2 |0\rangle |1\rangle + \beta_1\alpha_2 |1\rangle |1\rangle + \beta_1\beta_2 |1\rangle |1\rangle \tag{2.39}$$
$$= \alpha_1\alpha_2 |00\rangle + \alpha_1\beta_2 |01\rangle + \beta_1\alpha_2 |10\rangle + \beta_1\beta_2 |11\rangle \tag{2.40}$$
$$= \alpha_1\alpha_2 |0\rangle + \alpha_1\beta_2 |1\rangle + \beta_1\alpha_2 |2\rangle + \beta_1\beta_2 |3\rangle , \tag{2.41}$$

where $|0\rangle = |00\rangle$, $|1\rangle = |01\rangle$, $|2\rangle = |10\rangle$ and $|3\rangle = |11\rangle$. In general, the wave function of a n-qubit system is given by a 2^n dimensional vector:

$$|\psi\rangle = \sum_{i=0}^{2^n} a_i |i\rangle . \tag{2.42}$$

Therefore, a classical simulation of a n-dimensional qubit system has to keep track of 2^n complex amplitudes. This makes it impossible to simulate quantum systems above a certain size. For the classical simulation of a system consisting of 500 qubits, the state vector's dimension is already larger than the number of atoms in the universe.

As in the single qubit case, the probability to observe state $|n\rangle$ is given by a_n^2. It is also possible to only measure a subset of the qubits, leaving the other qubits in the normalized state:

$$|\psi\prime\rangle = \frac{(M_m \otimes I)\,|\psi\rangle}{\sqrt{p(m)}}. \tag{2.43}$$

A phenomenon that can appear in multi-qubit systems is *entanglement*. Consider for instance the two-qubit state

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{2.44}$$

This state can be created with a very simplistic quantum program as shown later in section **??**. Note that this state is a proper two-qubit state as it fulfills the normalization property $a_{00}^2 + a_{11}^2 = \frac{1}{\sqrt{2}}^2 + \frac{1}{\sqrt{2}}^2 = \frac{1}{2} + \frac{1}{2} = 1$.

Nevertheless, there are no two single-qubit states $|a\rangle$ and $|b\rangle$ such that $|\psi\rangle = |a\rangle\,|b\rangle$. The two qubits of $|\psi\rangle$ are *entangled* with each other. Measuring one of the qubits immediately determines the state of the other qubit, i.e. measuring the first qubit as $|0\rangle$ happens with probability:

$$\begin{aligned} p_1(0) &= \langle\psi|\,(M_0 \otimes I)^\dagger (M_o \otimes I)\,|\psi\rangle && (2.45)\\ &= \frac{1}{\sqrt{2}}\,\langle 00|\,\frac{1}{\sqrt{2}}\,|00\rangle && (2.46)\\ &= \frac{1}{2}, && (2.47) \end{aligned}$$

leaving the system in the post-measurement state:

$$\begin{aligned} |\psi\prime\rangle &= \frac{(M_0 \otimes I)\psi}{\sqrt{p_1(0)}} && (2.48)\\ &= \frac{\frac{1}{\sqrt{2}}\,|00\rangle}{\frac{1}{\sqrt{2}}} && (2.49)\\ &= |00\rangle\,. && (2.50) \end{aligned}$$

The same maths apply for the case of measuring the first qubit as $|1\rangle$.

Even if the state of the second qubit is not determined before, it will be either $|0\rangle$ or $|1\rangle$ from the moment on the *other* qubit has been measured.

The exponential growth of the state space dimension and entanglement are two properties of quantum systems which make them hard to simulate classically. Known quantum algorithms with quantum speedups like Shor's algorithm create entanglement in smart ways to perform calculations which seem to be impossible for classical computers.

2.3 Quantum Gates

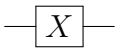
It is physically possible to modify the state vector of a quantum system, even when it is in a coherent state. This opens the theoretical possibility to perform calculations based on the laws of quantum physics and build quantum computers.

The quantum state can be modified through *quantum gates*. Quantum physics allows the gates only to be linear and reversible. Thus, quantum gates can be mathematically described as unitary matrices. Gates vary in the number of qubits on which they act and the effects they have on the qubits' state.

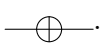
2.3.1 Single-Qubit Gates

Single qubit gates are linear mappings that can be applied to the state vector of a single qubit. Mathematically, single-qubit gates can be described by 2×2 unitary matrices. The fact that these matrices are unitary makes sure the state $|\phi\rangle = G\,|\psi\rangle$ after gate G has been applied to state $|\psi\rangle$ is a proper quantum state which preserves the normalization constraint $\alpha^2_{|\phi\rangle} + \beta^2_{|\phi\rangle} = 1$.

An example of a single qubit gate is the NOT or X gate, also denoted as:



or



.

The X gate is the quantum analog of the classical NOT gate. Like the classical version, the X gate swaps the states $|0\rangle$ and $|1\rangle$ when applied to them. It is even more general, as it maps a single-qubit state of the form

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.51}$$

to the state

$$|\phi\rangle = \beta\,|0\rangle + \alpha\,|1\rangle \tag{2.52}$$

by swapping the amplitudes for $|0\rangle$ and $|1\rangle$. The X gate has the following matrix representation:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{2.53}$$

A single qubit gate is applied to a quantum state by matrix multiplication:

$$|\phi\rangle = X|\psi\rangle \tag{2.54}$$
$$= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} |\psi\rangle \tag{2.55}$$
$$= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2.56}$$
$$= \begin{pmatrix} \beta \\ \alpha \end{pmatrix} \tag{2.57}$$

Another example of a single qubit gate without any classical analog is the *Hadamard gate* or H gate, described by the unitary:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{2.58}$$

Notice that the H gate is a unitary as it is reversible:

$$H^\dagger H = I. \tag{2.59}$$

The Hadamard gate maps between the so-called Z and X bases, mapping the states:

$$H|0\rangle = |+\rangle \tag{2.60}$$
$$H|1\rangle = |-\rangle \tag{2.61}$$

and back

$$H|+\rangle = |0\rangle \tag{2.62}$$
$$H|-\rangle = |1\rangle. \tag{2.63}$$

The Hadamard gate is a good example of how the Bloch Sphere visualization can help to understand a qubit state's transformation. The application of the Hadamard gate to the $|+\rangle$ is shown in figure **??**.

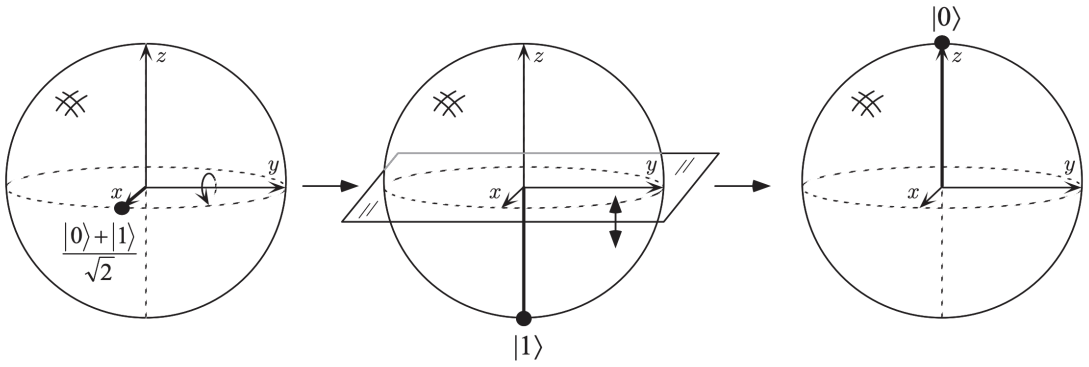


Figure 2.2: Visualization of the Hadamard gate on the Bloch sphere, acting on the $|+\rangle$ state [37].

An overview of some common single-qubit gates is given in figure **??**.

uu Operator	Gate	Matrix
X	X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Y	Y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Z	Z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
H	H	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
S	S	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
T	T	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix}$

Table 2.1: Overview of common single-qubit gates.

There are indefinitely many possible quantum gates in theory. Still, a universal finite gate set consisting of a few gates is sufficient to construct arbitrary unitary transformations [30]. This is the same as in the classical case where NAND gates suffice to build up arbitrary classical computation circuits [43] and good news for the fabrication of quantum computers with reusable components.

Nevertheless, single-qubit gates are not sufficient to build all possible unitary transformations on multi-qubit systems. For this purpose, multi-qubit gates are a necessity.

2.3.2 Multi-Qubit Gates

Single-qubit gates are not sufficient to create universal gate sets for quantum computation. In order to run arbitrary quantum programs, multi-qubit gates like *controlled* gates are needed. Controlled quantum gates are simple extensions of single-qubit gates. Every single-qubit gate can be implemented as a controlled version of it with one *control* and one *target* qubit. Only if the control qubit is in the $|1\rangle$ state, the gate is applied to the target qubit. As the control qubit can be in a superposition state, it is possible to apply and not apply the gate to the target qubit at the same time.

The prototypical two-qubit gate is the controlled NOT or CNOT gate. It is the controlled version of the X gate described before. As the CNOT gate acts on a two-qubit state, it can be described by a 4×4 unitary matrix. Each column describes the mapping of one of the four base state $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. The matrix representation of the CNOT gate is the following:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{2.64}$$

The matrix can be read as follows: The first two columns describe that the vectors $|00\rangle$ and $|01\rangle$ will not change when the gate is applied to these states. Only when the first qubit is in the $|1\rangle$ state, will the state of the second qubit be swapped. Thus $|10\rangle$ will be mapped to $|11\rangle$ and $|11\rangle$ to $|10\rangle$.

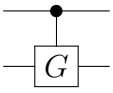
This shows how the matrix representation of generalized two-qubit gates can be derived: As long as the control qubit is off, it does not affect the state. Thus, the controlled gate matrix is the same as the identity with 1s on the diagonal and 0s elsewhere in the upper left corner. Only when the first qubit is in the $|1\rangle$ state, the matrix differs from the identity. For example, the controlled version of the Z gate:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.65}$$

is given by:

$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{2.66}$$

A controlled gate G is represented by a vertical line ending in a black dot indicating the control qubit:



With two-qubit operations at hand, it is possible to build a universal gate set. A commonly used gate set is the so-called *Clifford + T* set, consisting of the gates $CNOT$, H, S and T [21]. The Solovay-Kitaev theorem [30] guarantees that this set can efficiently approximate any unitary operation. In this study, another universal gate set will be used, composed of the CZ, $\sqrt{X}$, $\sqrt{Y}$ and T gates [3].

2.4 Quantum Circuits

The standard model for computation in theoretical computer science is the Turing Machine [48]. Invented by Alan Turing in 1936, it is a powerful tool to understand the limits of (classical) computation. The theoretical nature of the Turing machine gives it unlimited computational resources, like infinite memory, which do not reflect the properties of realizable computer architectures.

Another computation model that does not suffer from the gap between theory and practice is the *circuit model* [42]. In the classical circuit model, each bit is represented by a wire, and operations (or gates) are represented by different shapes acting on those wires. The circuit model is as powerful as a Turing Machine. It is the model of choice in quantum computing.

Quantum programs are described by quantum circuits. Each qubit is represented by a wire and each gate by a rectangular on those wires. Figure **??** describes a very simple quantum circuit which flips a single qubit initialized in the $|0\rangle$ state by applying a X gate before measuring it.

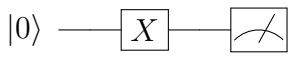


Figure 2.3: A simple quantum circuit applying an X gate to a qubit initialized in the $|0\rangle$ state before measuring it.

The circuit is being read from left to right. Important to note is that for calculating the resulting state, the matrix representation of the gates are multiplied from the left side to the current state $|\psi\rangle$ In the example above the final state of the program represented at the end of the circuit is:

$$|\phi\rangle = X\,|\psi\rangle \tag{2.67}$$
$$= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} |\psi\rangle \tag{2.68}$$
$$= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2.69}$$
$$= \begin{pmatrix} \beta \\ \alpha \end{pmatrix}. \tag{2.70}$$

In practice, circuits consist of several qubits and multiple gates, that are applied to them. It is possible to apply gates to different qubits sequentially as well as in parallel, as demonstrated in figure **??**:

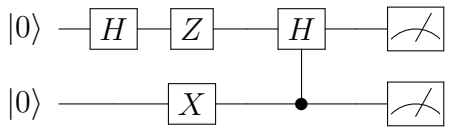


Figure 2.4: A quantum circuit acting on two qubits. The effect of parallel applications of single-qubit gates is defined by the tensor product of the gates.

Note that, when calculating the state of the quantum program above, the state of the system is given by the tensor product of the individual qubit states. For the circuit from figure **??**, the initial state $|\psi_{init}\rangle$ before any gate has been applied is:

$$|\psi_{init}\rangle = |0\rangle \otimes |0\rangle \tag{2.71}$$
$$= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.72}$$
$$= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2.73}$$
$$= |00\rangle\,. \tag{2.74}$$

When a gate gets applied to only a subset of the qubits, as in the case of the first H gate in the circuit above, the unitary applied to the multi-qubit state is implicitly the tensor product of the gate on that qubit and identity matrices on the remaining qubits. For the given circuit, the state $|\psi_{H_1}\rangle$ after the first H gate on the first qubit is calculated as:

$$|\psi_{H_1}\rangle = (H \otimes I)\,|00\rangle \tag{2.75}$$

$$= \left(\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \otimes \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\right)|00\rangle \tag{2.76}$$

$$= \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{pmatrix}|00\rangle \tag{2.77}$$

$$= \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{pmatrix}\begin{pmatrix}1\\ 0\\ 0\\ 0\end{pmatrix} \tag{2.78}$$

$$= \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 0\\ 1\\ 0\end{pmatrix} \tag{2.79}$$

$$= |+0\rangle\,. \tag{2.80}$$

As expected, applying a H gate to the first qubit, it should be in the $|+\rangle$ state while the second qubit remains in the $|0\rangle$ state.

Similar to the first gate, the unitary applied to the state, when multiple gates are applied to different qubits at the same time, is constructed by the tensor product of those gates. The resulting state $|\psi_{Z_1X_2}\rangle$ after the Z gate is applied to the first qubit, and the X gate is applied to the second qubit, is calculated as:

$$
\begin{aligned}
|\psi_{Z_1X_2}\rangle &= (Z \otimes X)\,|+0\rangle && (2.81)\\
&= \left(\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix} \otimes \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\right)|+0\rangle && (2.82)\\
&= \begin{pmatrix}0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 0 & -1\\ 0 & 0 & -1 & 0\end{pmatrix}|+0\rangle && (2.83)\\
&= \begin{pmatrix}0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 0 & -1\\ 0 & 0 & -1 & 0\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{2}}\\ 0\\ \frac{1}{\sqrt{2}}\\ 0\end{pmatrix} && (2.84)\\
&= \begin{pmatrix}0\\ \frac{1}{\sqrt{2}}\\ 0\\ -\frac{1}{\sqrt{2}}\end{pmatrix} && (2.85)\\
&= |-1\rangle\,. && (2.86)
\end{aligned}
$$

Accordingly, applying a Z gate to the $|+\rangle$ state brings the first qubit to the $|-\rangle$ state, while the X gate on the second qubit moves the qubit state from the $|0\rangle$ to the $|1\rangle$ state.

The final state $|\psi_{final}\rangle$ of the circuit is calculated by applying the 4×4 matrix representation of the controlled H gate to $|-1\rangle$. In this case, the second qubit is the controlled qubit:

$$|\psi_{final}\rangle = CH\,|-1\rangle \tag{2.87}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix} |-1\rangle \tag{2.88}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{2.89}$$

$$= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.90}$$

$$= |11\rangle\,. \tag{2.91}$$

The first qubit has been swapped from the $|-\rangle$ to the $|1\rangle$ state as the second qubit is in the $|1\rangle$ state and the H gate has been applied. The circuit **??** thus maps the initial state $|00\rangle$ to $|11\rangle$.

Another simple quantum circuit, which entangles two qubits with each other, is shown in figure **??**.

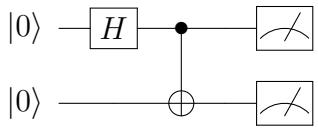


Figure 2.5: A quantum circuit to create maximally entangled 2-qubit states.

The Hadamard gate on the first qubit maps the system from the initial $|00\rangle$ into the $|+0\rangle$ state. Afterward, the first qubit, which is currently between the $|0\rangle$ and $|1\rangle$ state on the Bloch Sphere, is used as the control qubit in the $CNOT$ gate. This has an interesting effect on the second qubit, as the X gate is applied and not applied at the same time, entangling the two qubits with each other. The final state before measurement is $\frac{|00\rangle+|11\rangle}{2}$, which has been discussed in section **??**. This state is also known as one of the four *Bell states*. They represent maximally entangled two-qubit states. Moreover, they play an important role in the analysis of *quantum communication*.

The quantum circuit demonstrates how entanglement can be created by applying controlled gates with qubits in superposition states. Entanglement plays an essential role in the construction of so-called *random quantum circuits* in recent quantum supremacy experiments.

2.5 Quantum Computational Complexity

It is essential to understand the theoretical capabilities and limitations of quantum computers to understand for which kind of problems they provide advantages over classical computers.

For many years, it has been assumed that the extended Church Turing thesis holds true. It states that a probabilistic Turing machine can efficiently simulate any realistic model of computation. The thesis is challenged by quantum computers, because they can potentially solve specific problems exponentially faster than classical computers [17].

The class of problems which can be solved efficiently by a quantum computer is called **BQP**, shorthand for *bounded-error quantum polynomial time* [8]. A decision problem is in **BQP** if there exists a quantum program that solves the decision problem in 2/3 of the cases and runs in polynomial time. This is the quantum analog of the *bounded-error probabilistic polynomial time* or **BPP** [20], which is decidable by a probabilistic Turing machine in polynomial time and widely believed to be the same as **P**.

Currently, there are a few problems known to be in **BQP**, which are suspected not to be in **P**, providing evidence for the superiority of quantum computers. One of these problems is *factorization*, the problem of decomposing a composite integer into its prime factors. This problem has been known to be in *NP* before. Though, it is also known that factorization is not an **NP**-hard problem, indicating that quantum computers are not able to provide an exponential speedup for every problem that is not efficiently solvable on a classical computer. Indeed, classical algorithms might even exist, which can solve factorization on a classical computer efficiently, which have not been discovered yet.

While **BQP** includes **P** and intersects with **NP**, it can be shown that it is strictly included in **PSPACE** [8]. The relationship of these complexity classes is visualized in figure 2.6.

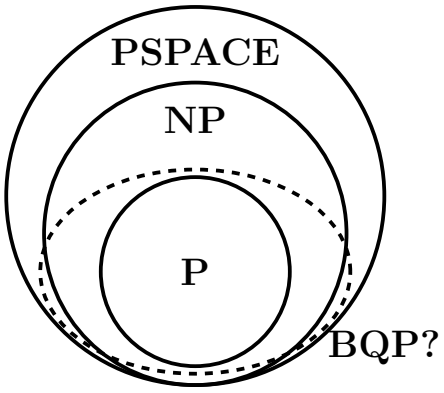


Figure 2.6: Relation of Complexity Classes to each other.

While it is hard to prove some fundamental relationships between complexity

classes like the famous $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ problem, it is believed that quantum computers can solve specific problems with practical applications like integer factorization [44] or the simulation of quantum systems [17] exponentially faster than classical computers.

The moment in time a physical quantum computer can outperform a classical computer on a specific problem for the first time has been coined by John Preskill in 2012 as *quantum supremacy* [40]. Recently, Google announced their quantum supremacy results with a 54 qubit quantum computer, providing the first physical evidence that it might be possible to build quantum computers with advantages over classical computers [3].

3 Random Circuit Sampling

Despite four decades of research in quantum computing, there was no physical proof of whether it would be possible to realize quantum computers that have an advantage over classical ones. This only changed recently, when a team from Google and the UCSB claimed to have demonstrated *quantum supremacy* with a 54 superconducting qubit quantum processor [3]. For this purpose, they chose the problem of *random circuit sampling*, a problem designed explicitly as a quantum supremacy experiment [9].

This chapter will motivate and introduce the theoretical framework of random circuit sampling, also used as the benchmark for Restricted Boltzmann Machines in this study. Further, the results of Google's quantum supremacy experiments will be discussed.

3.1 Quantum Supremacy

John Preskill has coined the term *quantum supremacy* in 2012 [40]. It describes the point in time when a physical quantum computer outperforms a classical computer on some task for the first time. The problem solved by the quantum computer does not need to be useful. Its only purpose is to prove that a quantum computer can be realized, which has an advantage over classical computers on *some* problem.

There exist different proposals for quantum supremacy experiments [44, 1, 9]. Since quantum computers do not promise to outperform classical computers on every problem, as discussed in section **??**, but only provide a practical advantage for problems which lie in **BQP** and outside of **P**, quantum supremacy experiments must be based upon a strong theoretical foundation [8].

One way to demonstrate quantum supremacy would be to run Shor's algorithm for integer factorization [44] on a physical quantum computer on some number which would not be feasible to decompose with known classical algorithms on existing supercomputers [33]. The issue with this approach is that many qubits are necessary to represent such numbers while today it is possible to build quantum processors with about 50 qubits of reasonable quality [3].

Another famous proposal for a quantum supremacy experiment is *Boson Sampling* [1], which is based on the fact that calculating the permutant of a matrix is computationally hard.

The approach that a team from Google and UCSB took last year is called *Random Circuit Sampling* (RCS) [9, 3]. In this approach, random quantum circuits

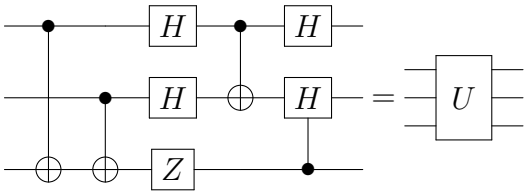
of specific structures that create highly entangled states are run on a quantum processor and are simulated classically. For enough qubits and depth, performing the classical simulations would take years on the biggest existing supercomputers [3]. If the quantum computer can generate outputs for such circuits which cannot be simulated classically anymore while their output can still be verified to be correct, this would demonstrate quantum supremacy.

A metric that allows the verification on random circuit instances that cannot be simulated is the *cross entropy difference*.

3.2 Cross Entropy Difference

The challenge of quantum supremacy is that one has to be able to verify the results of the quantum computer on instances which cannot be calculated classically anymore. In the case of integer factorization, this is a simple task as the test for the correctness consists of simple multiplications which can be performed efficiently. Since integer factorization needs more qubits than currently available in existing quantum computer hardware, other supremacy experiments like RCS provide a framework to demonstrate quantum supremacy much earlier.

The first observation to understand the concept of RCS is that every quantum circuit acting on n qubits can be described by a single 2^n dimensional unitary matrix [37], as illustrated in figure **??**. This follows from the fact that quantum gates are described by unitary transformations and the matrix product, as well as the tensor product of two unitaries, is a unitary again.



(3.1)

Figure 3.1: Every Circuit can be described by one big unitary acting on n qubits. W.l.o.g. the resulting states corresponds to the first row of the unitary U.

A quantum circuit consisting of gates chosen uniformly at random will act like a uniformly random unitary U [9]. Since the input state of the circuit can be assumed to be initialized in the $|0\rangle^{\otimes n}$ state, the amplitudes of the final quantum state of the circuit correspond to the first column of the random unitary U.

This column's entries are complex numbers with real and imaginary parts distributed by an unbiased Gaussian with respect to the normalization constraints. The output probabilities of the random circuit are given by the squared norms of these entries. They are thus distributed as the square of a Gaussian which is

an exponential distribution, also known as a Porter-Thomas [39]. So the probabilities $p(x_j) = |\langle x_j | \psi_{RC} \rangle|^2$ to observe bitstrings $x_j \in \{x_j\}_{j=1}^{N}$ with $N = 2^n$ are distributed according to

$$Pr(p) = e^{-Np} \tag{3.2}$$

as visualized in figure **??**.

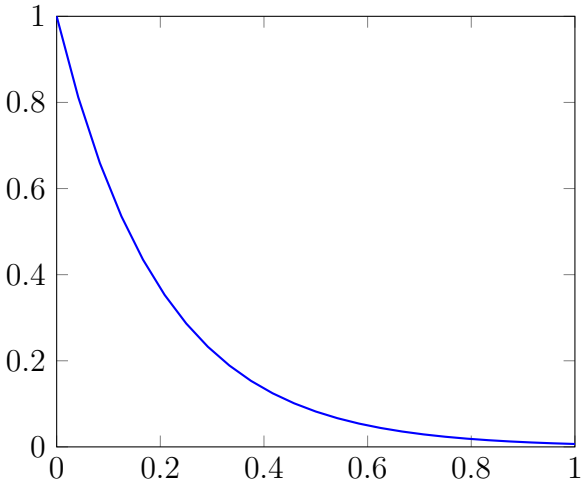


Figure 3.2: Distribution $P(p_{x_j})$ of output probabilities p_{x_j} of a quantum circuit. The distribution $P(p_{x_j}) = e^{-Np}$ is distributed as Porter-Thomas. A lot of output strings will have a low probability to be observed and a few output strings will dominate the observed outputs [9].

As quantum computers suffer from decoherent noise [49], it is important to understand if noisy quantum computers can still outperform classical computers on the RCS task. Knowing that the outputs are Porter-Thomas distributed will help with that.

The task is to understand how samples $S = \{x_1, \ldots, x_m\}$ drawn from a perfect execution of U compare to a set $S' = \{x'_1, \ldots, x'_m\}$ of samples drawn from a potentially noisy execution of U.

A well motivated measure for the difference between the underlying probability distributions p_U and p'_U of samples S and S' is the *cross entropy* [31], defined as

$$H(p'_U, p_U) = -\sum_{j=1}^{N} p'_U(x_j|U) \log p_U(x_j). \tag{3.3}$$

Of interest is the expected quality of the potentially noisy execution over different random circuit instances:

$$\mathbb{E}_U[H(p'_U, p_U)] = -\mathbb{E}_U[\log p_U(x_j)] \qquad = \mathbb{E}\left[\sum_{j=1}^{N} p'_U(x_j|U) \log \frac{1}{p_U(x_j)}\right] \tag{3.4}$$

The output of the noisy execution can be assumed to be statistically uncorrelated with the output of the perfect execution [9]. Thus, and since the output distribution for a fixed x_j over many random circuit instances U also has the shape of the Porter-Thomas distribution [22], $\mathbb{E}_U[\log \frac{1}{p_U(x_j)}] = -\mathbb{E}_U[\log p_U(x_j)]$ from equation 3.4can be computed independently as:

$$-\mathbb{E}_U[\log p_U(x_j)] \approx -\int_0^\infty N e^{-NP} \log p \, dp \tag{3.5}$$

$$= \log N + \gamma, \tag{3.6}$$

where $\gamma \approx 0.577$ is the Euler constant. Using the fact that $\sum_{j=1}^{N} p'_U(x_j) = 1$ for any distribution $p_U\prime$, the average cross entropy $\mathbb{E}_\mathbb{U}[H(p'_U, p_U)]$ between the Porter-Thomas distribution and any other uncorrelated distribution is given as

$$\mathbb{E}_U[H(p'_U, p_U)] = \log N + \gamma. \tag{3.7}$$

This is equivalent to the cross entropy $H_0 = \log N + \gamma$ between the Porter-Thomas and the uniform distribution. Thus, on average, a worst-case noisy execution will be as good compared to a perfect execution as sampling every bitstring with probability $1/N$. H_0 differs from the entropy $H(p_U)$ of the Porter-Thomas distribution, which corresponds to the cross entropy with itself, only by a -1 term:

$$H(p_U) = -\int pN^2 e^{-Np} \log p \, dp \tag{3.8}$$

$$= \log N - 1 + \gamma, \tag{3.9}$$

This directly leads to a benchmark for any algorithm A_U sampling output strings from a random circuit U, given either by a classical simulation or by a (noisy) quantum computer. The quality of A_U can be calculated as the difference between the entropy of the uniform distribution and the cross entropy of the output samples from A_U with the Porter-Thomas distribution. This metric is defined as the *cross entropy difference* [9]:

$$\Delta H(p_A) \equiv H_0 - H(p_A, p_U) \tag{3.10}$$

$$= \sum_j \left(\frac{1}{N} - p_A(x_j|U)\right) \log \frac{1}{p_U(x_j)}. \tag{3.11}$$

The cross entropy difference is zero for an algorithm sampling from the uni-

form distribution and one for an algorithm sampling from the underlying Porter-Thomas distribution of the circuit.

In order to calculate $\Delta H(p_A)$, a perfect simulation of the random circuit is necessary as the values of $p_U(x_j)$ have to be known. This is not possible in the quantum supremacy regime. As discussed shortly, it will be possible to extrapolate the cross entropy difference $\Delta H(p_A)$ of A_U to the quantum supremacy regime with high precision based on samples of its cross entropy difference on smaller circuit instances which can still be simulated classically [9].

Another justification for the cross entropy difference is that the l_1 distance between the uniform and Porter-Thomas distribution

$$l_1(p,q) = \|p-q\|_1 \tag{3.12}$$

$$= \sum_{j=1}^{N} \mid p_i - q_i \mid \tag{3.13}$$

is $\frac{2}{e}$ and thus independent of the number of qubits n. Therefore, a constant number of samples is sufficient to calculate the cross entropy $H(p_U)$ difference.

Experimentally, the cross entropy difference

$$\alpha = \mathbb{E}_U[\Delta H(p'_U)] \tag{3.14}$$

can be obtained by the execution of several random circuits U. Quantum supremacy is achieved by a physical quantum computer when its average cross entropy

$$1 \geq \alpha > C \tag{3.15}$$

is greater than the average performance C of the best known classical algorithm A^*:

$$C = \mathbb{E}_U[\Delta H(p^*)]. \tag{3.16}$$

In the regime where perfect simulations are possible, $C = 1$ and quantum supremacy cannot be achieved [9]. For sufficiently many qubits, perfect simulations will not be possible anymore and $1 > C \geq 0$ with C decreasing exponentially with the number of gates $g >> n$. For a typical set of samples S_{exp} drawn from the execution of a quantum circuit, the central limit theorem implies that:

$$\alpha \simeq H_0 - \frac{1}{m}\sum_{j=1}^{m} \log \frac{1}{p_U(x_j^{exp})}. \tag{3.17}$$

The experimental setup to estimate the cross entropy difference of any approximate simulation of a quantum computer thus is [9]:

1. Select random circuit U.

2. Take sufficiently many samples $S_{exp} = \{x_1, \ldots, x_m\}$, m in range $10^3 - 10^6$.

3. Compute the quantities $\log \frac{1}{p_U(x_j^{exp})}$ with the aid of a sufficiently large classical computer.

4. estimate α using equation 3.17.

3.3 Extrapolating to the Supremacy Regime

Keeping the qubits in a coherent state during a calculation and applying quantum gates exactly is a difficult task [3]. Qubits will suffer from decoherent noise, which will destroy the quantum states. This will happen in particular in the early implementations of quantum computers.

Before large scale error-corrected quantum computers will be available, so-called noisy intermediate-scale quantum (NISQ) computers present the first stage of quantum computing hardware. Such noisy devices will already provide an advantage over classical computers on specific problems. If the noise can be kept below a certain threshold, a NISQ device can demonstrate quantum supremacy on the RCS task [36].

In the presence of noise, the quantum state ρ generated by a physical quantum computer after the execution of a random circuit U can be represented as

$$\rho = \tilde{\alpha} U |\psi_0\rangle \langle\psi_0| U^\dagger + (1 - \tilde{\alpha})\sigma \tag{3.18}$$

with $\tilde{\alpha}$ being the circuit *fidelity* and noise term σ being an orthogonal state to the true state with $\langle\psi_0| U^\dagger \sigma U |\psi_0\rangle = 0$. The corresponding average cross entropy difference is:

$$\alpha = \mathbb{E}_U \left[H_0 + \sum_j \langle x_j| \rho |x_j\rangle \log p_U(x_j) \right] \tag{3.19}$$

$$= \tilde{\alpha} + (1 - \tilde{\alpha}) H_0 + \mathbb{E}_U \left[(1 - \tilde{\alpha}) \sum_j \langle x_j| \sigma |x_j\rangle \log p_U(x_j) \right]. \tag{3.20}$$

The states generated by the random circuits are maximally entangled [9]. A single bit or phase flip introduced by an additional X or Z gate in the circuit completely destroys such a state. Instead of being Porter-Thomas distributed, the output would be almost uniformly distributed, as shown in figure **??**.

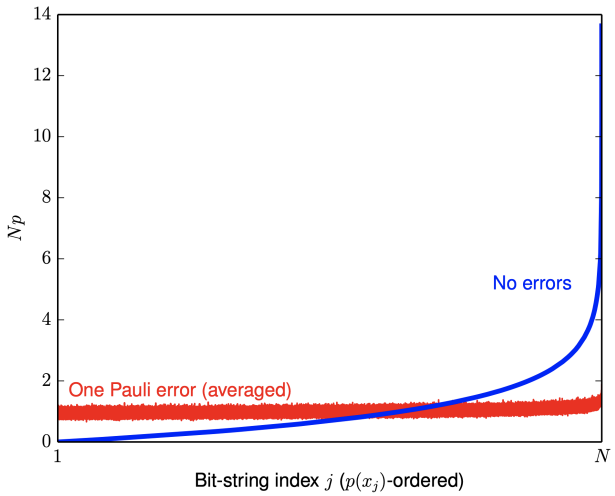


Figure 3.3: Blue: Output bitstrings of a random circuit ordered by their probabilities. Red: Averaged simulated output distribution with one Pauli error on all possible positions in a 5×4 qubits random circuit with depth 40. The distribution is almost uniform and almost uncorrelated with the real output distribution. The small tild at the end can be explained by the fact that Z gates close to the end of the circuit do not have an effect on the output distribution [9].

In other words, it is not possible to infer any information about the output probabilities without a perfect simulation of the circuit. This justifies the assumption that the actual output probabilities and the output of the quantum computer are (almost) uncorrelated. This assumption of no correlation, in turn, leads to the conclusion that the circuit fidelity $\tilde{\alpha}$ of a random circuit U is approximately equal to the average cross entropy:

$$\alpha = \mathbb{E}_U[\Delta H(p_{exp})] \approx \tilde{\alpha}. \tag{3.21}$$

This means that the fidelity α of a quantum device can be extrapolated to the quantum supremacy regime from estimates on random circuit instances which can still be simulated classically.

Even further, the cross entropy difference can also aid to estimate the single and two-qubit gate and errors of a quantum device:

$$\alpha \approx e^{-r_1 g_1 - r_2 g_2 - r_{init} n - r_{mes} n} \tag{3.22}$$

with $r_1, r_2 \ll 1$ being the Pauli error rates for one and two-qubit gates, $r_{init}, r_{mes} \ll 1$ the initialisation and measurement error and $g_1, g_2 \gg 1$ being the numbers of one and two-qubit gates. This relation has been numerically confirmed by Boixo et al. [9], indicating that the relation between the cross entropy difference and the circuit fidelity can be used to extrapolate the cross entropy difference of a

quantum device to the quantum supremacy regime [9].

So, even on the RCS task, a noisy quantum computer is able to demonstrate quantum supremacy, by calculating the fidelity of the quantum computer on random circuit instances still possible to simulate classically, which can be extrapolated to the supremacy regime using equation 3.21. If the fidelity is greater zero in the regime where no classical simulation is possible anymore, quantum supremacy has been demonstrated [9].

3.4 Random Circuit Design

The derivation of the cross entropy difference based on the assumption that the output probabilities of random circuit instances will be distributed according to the Porter-Thomas distribution [9]. As it is known that circuits of low depth as well as so-called *Clifford Circuits* can be simulated classically efficiently [21], it is crucial to understand which requirements the structures of the random circuits has to fulfill in order to be hard to simulate classically and generate exponential output distributions [9].

In a fully connected architecture, random circuits approximate a pseudo-random distribution with logarithmic depth [22]. Fully connected in this statement means that two-qubit gates can be applied to pairs of any two-qubits of the circuit. In architectures like superconducting qubits such as Google's Sycamore processor, the qubits are laid out on a 2D lattice, allowing two-qubit gates only to be applied to neighboring qubits on that lattice [4]. Circuits on fully connected architectures of logarithmic depths are known to be translatable into circuits of depth $\sqrt{n}$ on 2D lattices [5]. Boxio et al. used numerical simulations to optimize strategies to generate random circuits with minimized time to converge to a Porter-Thomas distribution, leading to the following algorithm [9]:

1. Initialize in the state $|0\rangle^{\otimes n}$.
2. Apply a Hadamard gate to each qubit.
3. Apply a random circuit with a stack of depth d, where each layer has the following two clock cycles:
   a) Apply a clock cycle of random single-qubit gates to all qubits.
   b) Apply a clock cycle of two-qubits gates.

The gate set consists of the single-qubit gates $\{\sqrt{X}, \sqrt{Y}, T\}$ with

$$\sqrt{X} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix} \tag{3.23}$$

and

$$\sqrt{Y} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \tag{3.24}$$

being the "square root of X" and the "square root of Y" gates:

$$\sqrt{X}^2 = \left(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix} \right)^2 \tag{3.25}$$

$$= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{3.26}$$

$$= X \tag{3.27}$$

and

$$\sqrt{Y}^2 = \left(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \right)^2 \tag{3.28}$$

$$= \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{3.29}$$

$$= Y. \tag{3.30}$$

The circuit can be separated into different layers, where each layer consists of one layer of controlled Z (CZ) gates and one layer of random single-qubit gates. The single-qubit gates are chosen such that each single-qubit gate on qubit q in the current cycle differs from the single-qubit gate on q at the previous cycle.

The two-qubit CZ gates are applied as demonstrated in figure **??**.

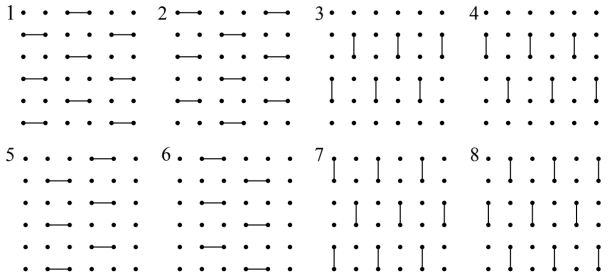


Figure 3.4: Order in which CZ gates are applied to the qubits. As the qubits are aligned on a 2D lattice, only neighboring qubits can be coupled through CZ gates. The pattern repeats every 8 cycles.

Boxio et al. could numerically show that the output distribution of random circuits generated this way converge to Porter-Thomas after a square root number of cycles in the number of qubits [9].

3.5 Experiments the Sycamore Processor

Built upon the theoretical framework of random circuit sampling, a team from Google and the UCSB conducted a quantum supremacy experiment on the 54 superconducting qubit *Sycamore* processor, shown in figure **??** [3].

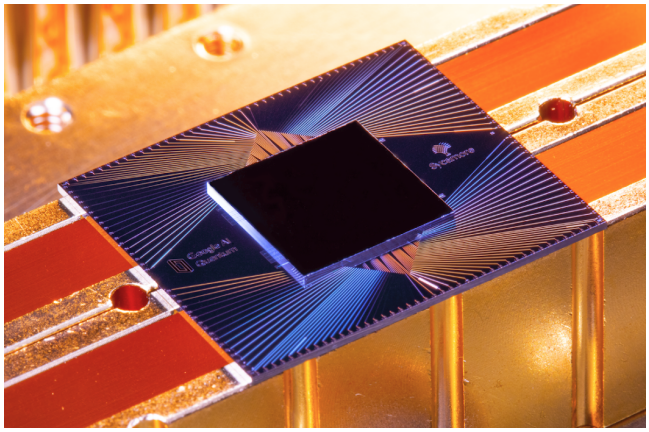


Figure 3.5: The Sycamore processor used in the quantum supremacy experiments of Google and UCSB. It consists of 54 qubits of which one was malfunctioning [3].

For the experiments, they generated circuits with $n = 10$ to $n = 53$ qubits and $m = 12$ to $m = 20$ cycles. The full cycles generated according to the procedure given above can be classically simulated up to $n = 53$ qubits with $m = 14$ cycles in a reasonable amount of time. Beyond that regime, simplified circuit architectures, called *elided* and *patched*, were used. These circuits are split into two sub-circuits with only weak entanglement in the elided and no entanglement in the patched version. Such circuits can be classically simulated for up to $n = 53$ qubits with $m = 20$ cycles to verify the fidelity of the Sycamore processor. For every circuit parameters, 20 circuits have been generated. For each circuit, $0.5 - 2.5$ million samples have been drawn from the quantum processor [3].

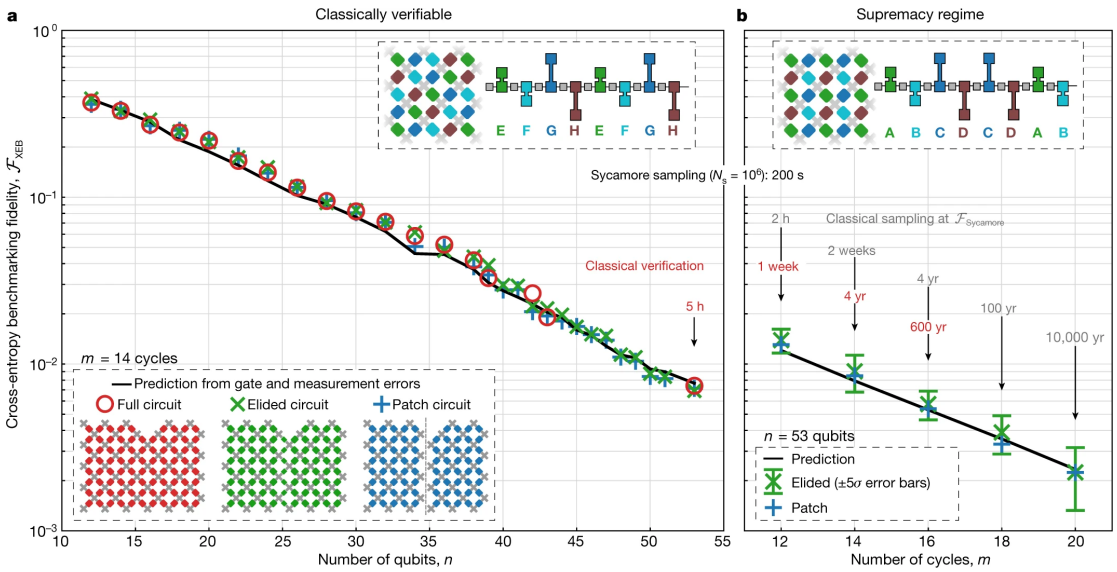


Figure 3.6: Results from the quantum supremacy experiments with the Sycamore processor [3]. The full circuits could only be simulated up to $n = 53$ qubits and $m = 12$ cycles. Experiments on elided and patched circuits match the values of the extrapolated cross entropy fidelity.

The team reports an average circuit fidelity $> 0.1\%$ with 5σ confidence for the largest elided circuits. All results fit the extrapolated expected fidelity from smaller circuits, making the team conclude that similar fidelities would be confirmed for the full circuits. As classical simulations of these circuits would take up to 100,000 years according to the team with a hybrid Schroedinger-Feynman algorithm, this would imply that quantum supremacy has been achieved [3]. The reported fidelities of the quantum supremacy for the different circuit sizes and depths are reported in figure **??**.

Shortly after these results have been announced, researchers from IBM claimed that the classical simulations for the full circuits on 53 qubits and 20 cycles could have been performed on the Summit supercomputer within a few days only with an optimized memory usage [38]. Experimental proof for these claims is still to be given.

Even if these circuits can be simulated within a few days, the quantum processor would still outperform the classical simulation as it only takes about 200ms to generate the 2 million samples for those circuits from it. As quantum supremacy is a loosely defined term, the discussion might continue whether quantum supremacy has been demonstrated by the Google and UCSB team. Nevertheless, the results proof that it is possible to build a quantum processor of 53 qubits which can execute quantum circuits consisting of up to 1,113 single-qubit and 430 two-qubit gates. For the future, the Google team expects a double exponential growth of the computational power of quantum computers as the classical simulation costs grow exponentially with the computational volume (gates and qubits) and quantum

hardware improvements will probably follow a quantum processor equivalent of Moore's law [3].

This study does not focus on quantum circuits running on physical quantum devices, but rather on the classical simulation. New and improved classical approximate simulations of quantum systems with neural networks recently proved to work well on many problem instances. Understanding their performance and necessary computational resources on the RCS task might provide useful insights about the capabilities and limitations of neural networks for the classical simulation of quantum systems.

4 Boltzmann Machines

The following chapter gives an introduction to Boltzmann machines and their applications to the classical simulation of quantum computing.

An overview of the architecture and mathematical properties of Boltzmann machines are given in the first section. The restricted Boltzmann machine is motivated as a special kind of Boltzmann machine with useful mathematical properties in the second part of this chapter. Afterward, the concepts of Gibbs sampling and supervised learning are explained. In the last section, a constructive approach is given on how restricted Boltzmann machines can be applied to the classical simulation of quantum computing.

The introduction to Boltzmann machines and restricted Boltzmann machines as well as the introduction to Gibbs sampling are based on [34] and [18] which provide a more throughout introduction into the topic. The work of Jónsson, Bauer and Carleo [28] is the foundation of the last section of this chapter.

4.1 Characteristics

The concept of the Boltzmann machine has first been proposed in the 1980s as a model for parallel distributed computing [27]. Boltzmann machines are physically inspired by the Ising Spin model and can be interpreted as energy-based recurrent neural networks representing probability distributions over vectors $\boldsymbol{d}_i \in \{0,1\}^n$ [2].

A Boltzmann machine is a network of stochastic units (or neurons) $X = V \cup H$ which are segmented into *visible* neurons $V = \{v_1, \ldots, v_n\}$ and *hidden* neurons $H = \{h_1, \ldots, h_m\}$. The joint state of the visible neurons $\boldsymbol{v} = (v_1 \ldots v_n) \in \{0,1\}^n$ represents n-dimensional data points $\boldsymbol{d}_i \in \{0,1\}^n$. The hidden neurons increase the expressiveness of the Boltzmann machine by acting as non-linear feature detectors to model dependencies between the visible neurons [24].

The neurons are connected to each other by weighted links W_{ij} and poss biases a_i (visible) or b_i (hidden), respectively. In general, the neurons of such a Boltzmann machine can be fully connected. A graphical representation of a fully connected Boltzmann machine is shown in figure **??**.

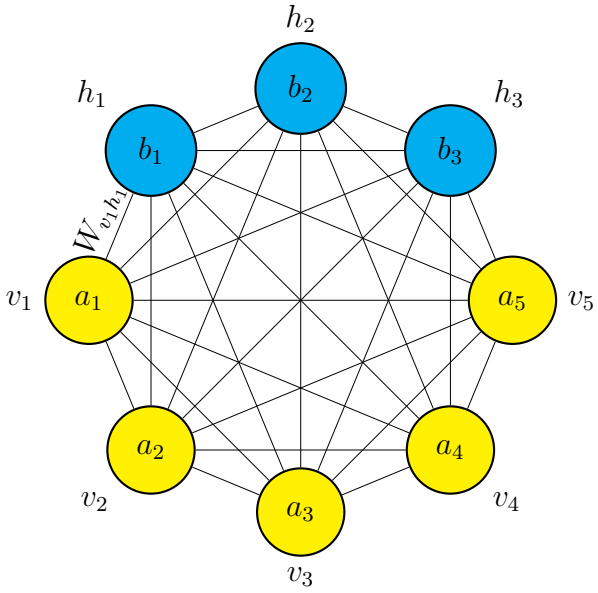


Figure 4.1: Graphical representation of a fully connected Boltzmann machine with 5 visible neurons (yellow) v_1 to v_5 and 3 hidden neurons (blue) h_1 to h_3. Each neuron posses a bias a_1 to a_5 and b_1 to b_3 respectively. The connection weight between two neurons i and j is given by W_{ij}.

Each configuration $\boldsymbol{c} = (v_1, \ldots, v_n, h_1, \ldots, h_m)$ of neuron states of the Boltzmann machine is associated with an energy $E(\boldsymbol{c})$ value, which is defined by the parameters $\mathcal{W}$, consisting of the weights and biases $\mathcal{W} = \{a_i, b_i, W_{ij}\}$:

$$E(\boldsymbol{c}; \mathcal{W}) = -\sum_{v_i \in V} a_i v_i - \sum_{h_i \in H} b_i h_i - \sum_{x_i, x_j \in X} W_{x_i, x_j} x_i x_j. \tag{4.1}$$

When sampling configurations from the Boltzmann machine (discussed in more detail in section 4.3), the Boltzmann machine prefers low energy states over states with high energies. The *stationary probability* of a configuration $\boldsymbol{c}$ with energy $E(\boldsymbol{c}; \mathcal{W})$ is given by the so-called *Gibbs-Boltzmann distribution* [19]:

$$p(\boldsymbol{c}; \mathcal{W}) = \frac{\mathrm{e}^{-E(\boldsymbol{c}; \mathcal{W})}}{Z(\mathcal{W})}, \tag{4.2}$$

where $Z(\mathcal{W})$ is the normalizing partition function

$$Z(\mathcal{W}) = \sum_{\boldsymbol{c\prime} \in C} \mathrm{e}^{-E(\boldsymbol{c\prime}; \mathcal{W})}. \tag{4.3}$$

In a training phase (discussed in section 4.4), the parameters of the Boltzmann machine can be adapted in such a way that the marginal probability distribution $p(\boldsymbol{v}; \mathcal{W})$ of the visible neurons

$$p(\boldsymbol{v}; \mathcal{W}) = \sum_{\boldsymbol{h}_k \in \{0,1\}^m} p(\boldsymbol{v}, \boldsymbol{h}_k; \mathcal{W}), \tag{4.4}$$

which traces out the hidden unit states by summing over all possible configurations of them, resembles the probability distribution of data points $\boldsymbol{d}_i$ in a training set $D = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_l\}$.

For a fully connected Boltzmann machine, this representation consists of an exponential number of summands and cannot be calculated efficiently. So-called *Restricted Boltzmann Machines* have a specific architecture with a restricted connectivity, which allows the efficient calculation of the marginal probability.

4.2 Restricted Boltzmann Machines

The Restricted Boltzmann machine (RBM) is a type of Boltzmann machine with a specific architecture and properties [45]. Since their invention, RBMs have been applied to a variety of machine learning tasks. They played a key role in the development of deep learning architectures as building blocks of so-called Deep Belief networks [7, 26]. RBMs are also the kind of Boltzmann machines which are being used in this study of the simulation of quantum circuits.

In its restricted form, the neurons of the Boltzmann machine are separated into two layers; one being the visible layer containing the visible neurons $v_i \in V$ and the other layer being the hidden layer containing the hidden neurons $h_j \in H$. Each neuron of the RBM is only allowed to be connected to the neurons from the other layer. Intra-layer connections are not allowed. As a result, the graph of the RBM is bipartite, as shown in figure **??**.

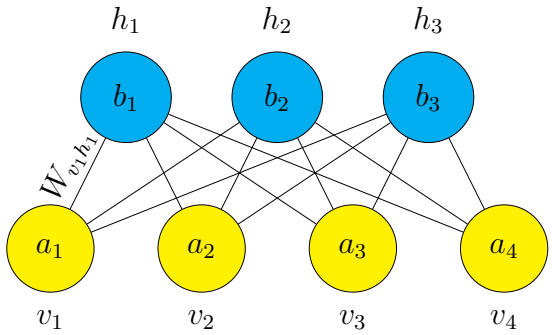


Figure 4.2: Graphical representation of a RBM with 5 visible neurons and 3 hidden ones. There are only connections between the two layers and no connection between two neurons from the same layer.

The marginal probability $p(\boldsymbol{v}; \mathcal{W})$ of the visible neuron states in a RBM has the form:

$$p(\boldsymbol{v};\mathcal{W}) = \sum_{\boldsymbol{h}_k\in\{0,1\}^m} p(\boldsymbol{v},\boldsymbol{h}_k;\mathcal{W}) \tag{4.5}$$

$$= \frac{1}{Z(\mathcal{W})} \sum_{\boldsymbol{h}_k\in\{0,1\}^m} \mathrm{e}^{-E(\boldsymbol{v},\boldsymbol{h}_k;\mathcal{W})} \tag{4.6}$$

$$= \frac{1}{Z(\mathcal{W})} \sum_{h_1\in\{0,1\}} \cdots \sum_{h_m\in\{0,1\}} \mathrm{e}^{\sum_{v_i} b_i v_i} \prod_{j=1}^{m} \mathrm{e}^{h_j(b_j+\sum_{i=1}^{n} W_{ij}v_i)} \tag{4.7}$$

$$= \frac{\mathrm{e}^{\sum_{v_i} b_i v_i}}{Z(\mathcal{W})} \sum_{h_1\in\{0,1\}} \mathrm{e}^{h_1(b_1+\sum_{i=1}^{n} W_{i1}v_i)} \cdots \sum_{h_m\in\{0,1\}} \mathrm{e}^{h_m(b_m+\sum_{i=1}^{n} W_{im}v_i)} \tag{4.8}$$

$$= \frac{\mathrm{e}^{\sum_{v_i} b_i v_i}}{Z(\mathcal{W})} \prod_{i=1}^{m} \sum_{h_i\in\{0,1\}} \mathrm{e}^{h_i(b_i+\sum_{i=1}^{n} W_{ij}v_i)} \tag{4.9}$$

$$= \frac{\mathrm{e}^{\sum_{v_i} b_i v_i}}{Z(\mathcal{W})} \prod_{i=1}^{m} (1 + \mathrm{e}^{b_i+\sum_{i=1}^{n} W_{ij}v_i}). \tag{4.10}$$

This quantity consists of only a polynomial number of terms in the number of hidden units of the RBM. Consequently, it can be calculated efficiently. This makes the RBM a compact representation of a probability distribution over vectors $\boldsymbol{d}_i \in D$ inferred from a dataset D.

Even though the RBM has a limited connectivity between its units, it is a universal function approximator [32]. It can model any distribution over $\{0,1\}^m$ arbitrary well with m visible and k+1 hidden units, where k denotes the cardinality of the support set of the target distribution. That is the number of input elements from $\{0,1\}^m$ that have a non-zero probability of being observed. This also implies a worst-case exponential number of hidden units for distributions with an extensive support set [32]. Even fewer units can be sufficient depending on the patterns in the support set [35].

4.3 Gibbs Sampling

Boltzmann machines are generative models that represent probability distributions over their configurations. This means that it is possible to draw configurations from a Boltzmann machine according to their (marginal) probabilities given in equations 4.4 and 4.10.

Although it is required to calculate $Z(\mathcal{W})$ for the exact probabilities of each configuration, it is not necessary to calculate the energies for all 2^{n+m} possible configurations of a Boltzmann machine to draw samples from it.

Instead, Boltzmann machines can be seen as *Markov chains* with a stationary probability distribution. With a stochastic process called *Gibbs sampling*, the samples can be drawn efficiently according to this stationary distribution for

RBMs.

Gibbs sampling belongs to the class of so-called *Metropolis-Hastings* algorithms. By that, it is a *Monte Carlo Markov Chain* (MCMC) algorithm [23]. It is a simple algorithm to produce samples from the joint probability distribution of multiple random variables like neuron state configurations of a Boltzmann machine, which can be considered as a Markov chain.

A Markov chain is a discrete stochastic process of configurations of random variables $C = \{\boldsymbol{c}^{(t)}\}$ at time steps $t = 1, \ldots, T$ which take values in a set Ω (for Boltzmann machines $\Omega = \{0,1\}^{m+n}$) and for which for all time steps t and for all configurations $\boldsymbol{c}_j, \boldsymbol{c}_i, \boldsymbol{c}_{i-1}, \ldots, \boldsymbol{c}_0 \in \Omega$ the *Markov property*

$$p_{ij}^{(t)} := P(\boldsymbol{c}^{(t+1)} = c_j \mid \boldsymbol{c}^{(t)} = \boldsymbol{c}_i, \ldots, \boldsymbol{c}^{(0)} = c_0) \tag{4.11}$$

$$= P(\boldsymbol{c}^{(t+1)} = c_j \mid \boldsymbol{c}^{(t)} = \boldsymbol{c}_i) \tag{4.12}$$

holds. This means that the next state of the system only depends on the current state and not on the system's history. A Markov chain can be represented as a (finite) graph, as shown in figure **??**.

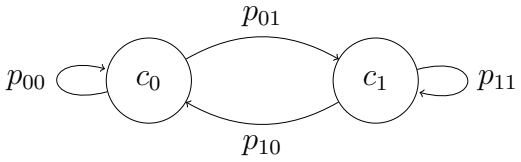


Figure 4.3: A Markov chain with two states c_0 and c_1. The Markov chain is described by the 2×2 matrix $\boldsymbol{P} = \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix}$.

In the case of Boltzmann machines, the transition probabilities p_{ij} are time independent and given by the ratios of configuration probabilities. For RBMs the transition probability p_{ij} from configuration c_i to c_j is given by:

$$p_{ij} = \frac{\mathrm{e}^{\sum_{v_i \in \boldsymbol{c_i}} b_i v_i} \prod_{i=1}^{m} (1 + \mathrm{e}^{b_i + \sum_{i=1}^{n} W_{ij} v_i})}{\mathrm{e}^{\sum_{v_i \in \boldsymbol{c_j}} b_i v_i} \prod_{i=1}^{m} (1 + \mathrm{e}^{b_i + \sum_{i=1}^{n} W_{ij} v_i})}, \tag{4.13}$$

which can be calculated efficiently as the $Z(\mathcal{W})$ terms from equation 4.10 cancel out.

In each time step t during the Gibbs sampling, the state of a single randomly chosen neuron of the RBM is flipped so that the configurations $\boldsymbol{c}^{(t)}$ and $\boldsymbol{c}^{(t+1)}$ only differ in the state of one neuron. With probability p_{ij} configuration $\boldsymbol{c}_j$ is kept as the new configuration. With probability $1 - p_{ij}$ the Boltzmann machine will stay in its current configuration $\boldsymbol{c}_i$. This corresponds to a random walk on the Markov chain defined by the RBM transition probabilities. The algorithm is given in algorithm **??**.

Algorithm 1 Gibbs Sampling

Require: $bm(c)$: function returning the energy of a Boltzmann machine for configuration c
Require: T: time steps
1: $t \leftarrow 0$
2: $c^{(0)} \leftarrow randomize(\{0,1\}^{m+n})$ (random initialization)
3: **repeat**
4: $\quad r \leftarrow random(m+n)$
5: $\quad E_i \leftarrow bm(c^{(t)})$
6: $\quad E_j \leftarrow bm(\{c_1, \ldots, \bar{c_r}, \ldots, c_{m+n}\})$
7: $\quad$ **if** $random(1) < \frac{E_i}{E_j}$ **then**
8: $\quad\quad c^{(t+1)} \leftarrow \{c_1, \ldots, \bar{c_r}, \ldots, c_{m+n}\}$
9: $\quad t \leftarrow t+1$
10: **until** $t = T$
11: **return** $c^{(T)}$

The Markov chain for configurations of a Boltzmann machine is known to converge to its so-called *stationary distribution* π, that is

$$\pi^T = \pi^T \boldsymbol{P} \tag{4.14}$$

with $\boldsymbol{P} = (p_{ij})$ being the transition matrix of the Markov chain with the transition probabilities as its entries. Once the Markov chain reaches its stationary distribution, all subsequent states will be distributed according to this distribution. Running Gibbs sampling for sufficiently many time steps T will sample configurations according to the Gibbs-Boltzmann distribution given in equation 4.10.

Although Gibbs sampling is a simple algorithm, it is an important algorithm in the context of Boltzmann machines. Different versions of Gibbs sampling have been developed for RBMs, like (persistent) contrastive divergence [25, 47] or parallel tempering [15]. In this study, Gibbs sampling will be used in its simplest version, as described in this chapter.

4.4 Supervised Learning

The probability distribution over vector spaces given by a Boltzmann machine can be trained to resemble the distribution of data points $\boldsymbol{d}_i$ in a dataset $D = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_d\}$. This can be done either in a *unsupervised* or in a *supervised* manner.

In both cases the parameters $\mathcal{W} = \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{W}\}$ of the Boltzmann machine are updated minimizing an objective function $O(\mathcal{W})$, which depends on the parameters of the Boltzmann machine and depicts the overlap of the current and the target distribution in an iterative process called *gradient descent*.

Before the training phase, the Boltzmann machine is initialized with random parameters $\theta_i \in \mathcal{W}_0$ and a training set $D = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_d\}$ is given. In the case of

supervised learning of Boltzmann machines, the training set consists of tuples of configurations and their corresponding target energy values $\boldsymbol{d}_i = (\boldsymbol{c}_i, t_i)$.

In the batch version of gradient descent, the training set D is split into subsets $D_1, \ldots, D_l$, called *batches*.

For each such batch D_i, the average negative log overlap $\langle L(\mathcal{W}) \rangle$ for all $\boldsymbol{c}_j \in D_i$ is computed. Afterwards, the gradients $\frac{\partial L}{\partial \delta_i}$ are calculated and the parameters updated into the direction of the steepest descent:

$$\theta_i = \theta_i - \alpha \frac{\partial L}{\partial \delta_i} \tag{4.15}$$

with *learning rate* $\alpha \ll 1$ to keep the step sizes moderate. The learning rate prevents to big step sizes in steep areas of the objective function $L(\mathcal{W})$ This process is repeated for a defined number of iterations to minimize the objective function $O(\mathcal{W})$. A graphical representation of this process is shown in figure **??**.

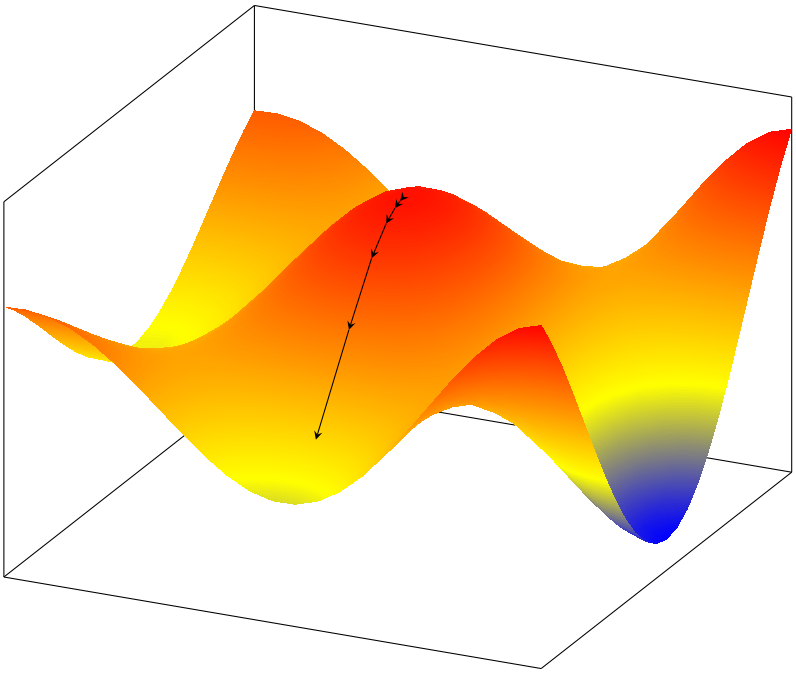


Figure 4.4: Iterations of a gradient descent method. At each iteration, the randomly initialized parameters get updated in direction of the steepest descent of the function to be optimized. After enough iterations, a minimum is of the function is reached.

The simple update rule in 4.15 has several drawbacks, however. First, The gradient and thus the step size will become smaller as closer the function approaches its minimum, leading to only small improvements and many steps necessary. Second, the function's shape will usually have (many) local minima, which should be avoided. Several variants of gradient descent have been developed to over-

come these issues [41]. Two of these methods are *AdaMax* [41] and *Stochastic Reconfiguration* [46].

4.4.1 Adamax

AdaMax is a special version of *Adam* [29].It combines the advantages of AdaGrad [16] and RMSProp [6], two other popular gradient descent methods. AdaMax is computationally efficient, has little memory requirements, and proves to be well suited for problems with much noise and sparse gradients.

Instead of using the raw gradient, the algorithm updates exponential moving averages of the gradient (mt) and the squared gradient (vt) to include more information about the shape of the error function into the parameter updates. The hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages. The moving averages themselves are estimates of the 1st moment (the mean) and the 2nd raw moment of the gradient. The update rule for the parameters θ_{ij} for Adamax are summarized in algorithm **??**.

Algorithm 2 Adamax

Require: α: step size
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta_i \in \mathcal{W}$
Require: $\mathcal{W}_0$: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$u_0 \leftarrow 0$ (Initialize the exponential weighted infinity norm)
$t \leftarrow 0$ (Initialize time step)
while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_{\mathcal{W}} f_t(\mathcal{W}_{t-1})$ (Get gradients w.r.t. objective function at time step t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $u_t \leftarrow max(\beta_2 \cdot u_{t-1}, |g_t)$ (Update the exponentially weighted infinity norm)
    $\mathcal{W}_t \leftarrow \mathcal{W}_{t-1} - (\alpha/(1 - \beta_1^t)) \cdot m_t/u_t$ (Update parameters)
return θ_t (Resulting parameters)

4.4.2 Stochastic Reconfiguration

Another popular gradient descent method to optimize the parameters of an RBM to represent quantum states is *Stochastic reconfiguration* []. The idea behind this method is that even a small change in the parameters of the RBM might lead to a big change in the output distribution of the RBM. Instead of small steps in the parameter space as in the pure version of gradient descent, the step size is kept small with respect to the change in the output distribution represented by the RBM. Stochastic reconfiguration is equivalent to the *natural gradient descent* method [].

The update rule for the parameters W of the RBM in the Stochastic Reconfiguration (SR) method is:

$$\theta_i' = \theta_i - \alpha \boldsymbol{S}^{-1} \boldsymbol{f} \tag{4.16}$$

with

$$\boldsymbol{f}_i = \langle \Delta_i^* \frac{\partial O}{\partial \theta_i} \rangle - \langle \Delta_i^* \rangle \langle \frac{\partial O}{\partial \theta_i} \rangle \tag{4.17}$$

$$\boldsymbol{S}_{ij} = \langle \Delta_i^* \Delta_j \rangle - \langle \Delta_i^* \rangle \langle \Delta_j \rangle \tag{4.18}$$

where $\langle x \rangle$ denotes the mean of a random variable x and Δ_i denotes the log-derivative of of the wave function $\Delta_i(\boldsymbol{c}) = \frac{1}{\psi(\boldsymbol{v})} \frac{\partial \psi(\boldsymbol{c})}{\partial \theta_i}$ with respect to some paramter $\theta_i \in \mathcal{W}$.

Using the covariance matrix $\boldsymbol{S}$ of the derivatives of the wave function keeps the KL divergence between the RBM before and after the update low, leading to a smooth transition through the error landscape [46].

Both AdaMax and Stochastic reconfiguration are modifications to the standard gradient descent method which try to reduce oscillations during the training and time to convergence by including more information about the shape of the error function into the changes of the gradients. This study will compare both methods for the classical simulation of quantum computing with RBMs.

4.5 Application to Quantum Computing

Machine learning techniques have become a successful approach for the classical simulation of quantum systems [12, 14, 11]. Carleo and Troyer [12] were the first to give a compact presentation of the wavefunction of a many-body quantum system with an RBM. The same framework has later been used by Jónsson, Bauer, and Carleo for the classical simulation of the quantum Fourier and Hadamard transform [28]. Their work gives a constructive approach on how the parameters of a complex-valued RBM representing the quantum state

$$|\psi_{\mathcal{W}}(\boldsymbol{v})\rangle = \frac{\mathrm{e}^{\sum_{v_i} b_i v_i}}{Z(\mathcal{W})} \prod_{i=1}^{m} (1 + \mathrm{e}^{b_i + \sum_{i=1}^{n} W_{ij} v_i}) \tag{4.19}$$

can be adapted to apply a universal set of quantum gates to the quantum state $|\psi_{\mathcal{W}}\rangle$.

RBMs allow an exact application only of unitary gates diagonal in the computational basis. For non-diagonal gates more hidden layers would be necessary, making the calculation of the quantum state intractable [11]. Nevertheless, it is possible to train a Boltzmann machine in a supervised fashion, such that its parameters are adapted to apply a non-diagonal gate to its state approximately.

The rules for the applications of diagonal and non-diagonal gates to an RBM state from [28] laid out in following two sections.

4.5.1 Diagonal Gates

Diagonal gates can be applied to an RBM quantum state by solving a set of linear equations. This section describes the rules for the applications of single-qubit Z rotations, controlled Z rotations as well as the Pauli X, Y and Z gates.

Single-Qubit Z rotations

The action of the single Z rotation of angle θ is given by the 2×2 unitary matrix

$$R_l^Z = \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{e}^{i\theta} \end{pmatrix}. \tag{4.20}$$

Its action on qubit l yields $\langle \boldsymbol{v} \mid R_l^Z(\theta) \mid \psi_W \rangle = \mathrm{e}^{i\theta v_l} \left| \psi_W(\boldsymbol{v}) \right\rangle$. Considering a RBM machine with weights $\mathcal{W}' = \{\alpha', \beta', W'\}$, the action of the $R^Z(\theta)$ gate is exactly reproduced if $\mathrm{e}^{v_l a_l} \mathrm{e}^{i\theta v_l} = \mathrm{e}^{v_l a'_l}$ is satisfied. This is the case for

$$a'_j = a_j + \delta_{jl} i\theta. \tag{4.21}$$

The action of the Z rotation thus simply modifies the bias of the visible neuron l of the RBM.

Controlled Z rotations

The action of a controlled Z rotations acting on two given qubits l and m is determined by the 4×4 unitary matrix:

$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \mathrm{e}^{i\theta} \end{pmatrix} \tag{4.22}$$

where θ is a given rotation angle. This gate is diagonal and can compactly be written as

$$\langle \boldsymbol{v} \mid CZ(\theta) \mid \psi_W \rangle = \mathrm{e}^{i\theta v_l v_m} \left| \psi_W(v_1 \dots v_n) \right\rangle. \tag{4.23}$$

As the RBM architecture does not allow direct interaction between visible neurons, the CZ gate requires the insertion of a dedicated extra hidden unit h_c, which is connected to the qubits l and m:

$$\langle \boldsymbol{v} \mid CZ(\theta) \mid \psi_W \rangle = \mathrm{e}^{\Delta a_l B_l + \Delta a_m Z_m} \sum_{h_c} \mathrm{e}^{W_{lc} B_l h_c + W_{mc} B_m h_c} \tag{4.24}$$

$$= \mathrm{e}^{\Delta a_l B_l + \Delta a_m B_m} \times (1 + \mathrm{e}^{W_{lc} B_l + W_{mc}} B_m) \, |\psi_W(\boldsymbol{v})\rangle \,, \tag{4.25}$$

where the new weights W_{lc} and W_{mc} and visible units biases $a_l\prime = a_l + \Delta a_l$, $a_m\prime = a_m + \Delta a_m$ are determined by the equation:

$$\mathrm{e}^{\Delta a_l B_l + \Delta a_m B_m} (1 + \mathrm{e}^{W_{lc} B_l + W_{mc} B_m}) = C \times \mathrm{e}^{i\theta B_l B_m} \tag{4.26}$$

for all the four possible values of the qubits values $B_l, B_m = \{0, 1\}$ and where C is an arbitrary (finite) normalization. A possible solution for this system is:

$$W_{lc} = -2\mathrm{A}(\theta) \tag{4.27}$$

$$W_{mc} = 2\mathrm{A}(\theta) \tag{4.28}$$

$$\Delta a_l = i\frac{\theta}{2} + \mathrm{A}(\theta) \tag{4.29}$$

$$\Delta a_m = i\frac{\theta}{2} - \mathrm{A}(\theta) \tag{4.30}$$

where $\mathrm{A}(\theta) = arccosh(\mathrm{e}^{-i\frac{\theta}{2}})$. Modifying the RBM's parameters accordingly will apply a CZ gate to its current state.

Pauli X gate

The X gate just flips the qubit l and the RBM amplitudes are:
$\langle \boldsymbol{v} \mid X_l \mid \psi_W \rangle = \langle v_1 \dots \bar{v_l} \dots v_N \mid \psi_W \rangle$.
Since $\bar{v_l} = (1 - v_l)$, it must be satisfied that

$$(1 - v_l) W_{lk} + b_k = v_l W_{lk}\prime + b_k\prime \tag{4.31}$$

and

$$(1 - B_l) a_l = B_l a_l\prime + C \tag{4.32}$$

hold for all the (two) possible values of $B_l = \{0, 1\}$. The solution to these equations is:

$$W_{lk}\prime = -W_{lk} \tag{4.33}$$

$$b_k\prime = b_k + W_{lk} \tag{4.34}$$

$$a_l\prime = -a_l \tag{4.35}$$

$$C = a_l, \tag{4.36}$$

whereas all the a_j and the other weights W_{jk} with $j \neq i$ are unchanged.

Pauli Y gate

A similar solution is also found for the Y gate, with the noticeable addition of extra phases with respect to the X gate:

$$W_{lk}\prime = -W_{lk} \tag{4.37}$$

$$b_k\prime = b_k + W_{lk} \tag{4.38}$$

$$a_l\prime = -a_l + i\pi \tag{4.39}$$

$$C = a_l + \frac{i\pi}{2}, \tag{4.40}$$

whereas all the a_j and other weights W_{jk} with $j \neq l$ are unchanged.

Pauli Z gate

The Pauli Z gate is a special case of the Z rotation with $\theta = \pi$. Thus the only change necessary is to set the bias a_l of the visible neuron l to

$$a_l\prime = a_l + i\pi. \tag{4.41}$$

4.5.2 Non-diagonal Gates

Exact applications of non-diagonal gates to Boltzmann machine quantum states would require introducing a second hidden layer [11]. In contrast to an RBM with just one hidden layer, this would make the calculation of the represented quantum state inefficient.

Thus, there are no rules for applying non-diagonal gates to an RBM state similar to those for diagonal gates. Nevertheless, the parameters of the RBM can be adapted in a supervised learning process instead.

The training set consists of samples drawn from the RBM with its current set of parameters. In the sampling process, the RBM's energy function can be adapted to resemble the energy states after a non-diagonal unitary gate G has been applied to its state.

For any non-diagonal unitary gate

$$G = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{4.42}$$

applied to qubit l, samples from the state $|\phi(\boldsymbol{v})\rangle = G_l\,|\psi(\boldsymbol{v})\rangle$ can be drawn according to

$$|\,|\phi(\boldsymbol{v}_{v_l=0})\rangle\,|^2 = |a\cdot|\psi(\boldsymbol{v}_{v_l=0})\rangle + c\cdot|\psi(\tilde{\tilde{\boldsymbol{v}}}_{\tilde{\tilde{v}}_l=1})\rangle|^2 \tag{4.43}$$

when the state of qubit l is sampled to be 0 and

$$||\phi(\boldsymbol{v}_{v_l=1})\rangle|^2 = |b\cdot|\psi(\boldsymbol{v}_{v_l=0})\rangle + d\cdot|\psi(\boldsymbol{v}_{v_l=1})\rangle|^2. \tag{4.44}$$

when the state is sampled to be 1. This way, the samples are drawn with respect to their probabilities given by $||\phi\rangle|^2$.

Afterward, the samples are used to minimize the log-likelihood $L(\mathcal{W})$ of the overlap of the two quantum states ψ and ϕ, given by:

$$\begin{aligned} L(\mathcal{W}) &= -\log O(\mathcal{W}) && (4.45)\\ &= -\log\sqrt{\frac{|\langle\psi_{\mathcal{W}}\,|\,\phi\rangle|^2}{\langle\psi_{\mathcal{W}}\,|\,\psi_{\mathcal{W}}\rangle\,\langle\phi\,|\,\phi\rangle}} && (4.46)\\ &= -\log\sqrt{\left\langle\frac{\phi(\boldsymbol{v})}{\psi_{\mathcal{W}}(\boldsymbol{v})}\right\rangle_{\psi}\left\langle\frac{\psi_{\mathcal{W}}(\boldsymbol{v})}{\phi(\boldsymbol{v})}\right\rangle_{\phi}^{*}} && (4.47)\end{aligned}$$

with

$$\langle F(\boldsymbol{v})\rangle_A := \frac{\sum_{\boldsymbol{v}} F(\boldsymbol{v})|A(\boldsymbol{v})|^2}{\sum_{\boldsymbol{v}}|A(\boldsymbol{v})|^2} \tag{4.48}$$

with some gradient descent method. For Boltzmann machines, the gradients $\frac{\partial L}{\partial\theta_i} = \partial_{p_i}L(\mathcal{W})$ are of the form:

$$\partial_{p_i}L(\mathcal{W}) = \langle\mathcal{O}_k^*(\boldsymbol{v})\rangle_\psi - \frac{\langle\frac{\phi(\boldsymbol{v})}{\psi(\boldsymbol{v})}\mathcal{O}_k^*(\boldsymbol{v})\rangle_\psi}{\langle\frac{\phi(\boldsymbol{v})}{\psi(\boldsymbol{v})}\rangle_\psi}, \tag{4.49}$$

with $\mathcal{O}_k(\boldsymbol{v}) = \partial_{p_k}\log\psi_{\mathcal{W}}(\boldsymbol{v})$ being the variational derivatives of the RBMs wave function with respect to its parameters. These are simple to compute, since

$$\partial_{a_i}\log\psi_{\mathcal{W}}(\boldsymbol{v}) = v_i \tag{4.50}$$

$$\partial_{b_i}\log\psi_{\mathcal{W}}(\boldsymbol{v}) = h_i \tag{4.51}$$

$$\partial_{W_{ij}}\log\psi_{\mathcal{W}}(\boldsymbol{v}) = v_i h_j. \tag{4.52}$$

The quantum state $\psi_{\mathcal{W}}$, represented by the RBM, approximates the target state ϕ.

In summary, the procedure to apply non-diagonal quantum gates to a quantum state $|\psi(\mathcal{W})\rangle$ represented by an RBM is as follows:

1. Draw samples $\boldsymbol{d}_i = (\boldsymbol{c}_i, t_i)$ with Gibbs sampling and transition probabilities given by equation 4.43 and 4.44.
2. Calculate the gradients $\frac{\partial L}{\partial \theta_i}$ of the loss function defined in equation 4.49.
3. Use a gradient descent method like AdaMax or SR to update the parameters $\theta_i \in \mathcal{W}$.

This process is repeated for a predefined number of iterations. Using this approach and AdaMax as the optimization method, Jónsson et al. reported a per gate error of 10^{-3} [28].

5 Experiments

This study investigates the capabilities and limitations of restricted Boltzmann machines for the classical simulation of quantum computing. For this purpose, the approach of Janson et al. [28] has been adapted to simulate random circuit instances [9]. These kinds of experiments have recently been conducted on a 53 superconducting qubit quantum computer [3].

This chapter will describe the experimental setup. It further represents and includes an interpretation of the results.

5.1 Setup

In order to run the experiments, Netket, an open-source library for the classical simulation of quantum many-body systems [10], has been extended to simulate quantum circuits given in the QASM format [13]. The resulting software is published at [] as an open-source tool for the classical simulation of quantum circuits. The experiments have been conducted on the Noctua Cluster at the University of Paderborn.

Several parameters of the RBM's training process have been varied. For the choice of the optimization algorithm, AdaMax and Stochastic Reconfiguration are tested against each other. Performances of different numbers of training iterations ,as well as training samples, are compared. Further, the CZ gate is applied in two fashions: Once by following the rules from [] by introducing an additional hidden unit to the RBM on each gate. In other experiments, the CZ gate is applied by training the parameters of the RBM the same way as for non-diagonal gates. In the first case, the RBM is initialized without any hidden units. In the latter case, it is initialized with so many hidden units that each pair of visible units is connected by one hidden unit.

The training process is once performed with *random restarts* and once without. In the experiments with random restarts, the RBM is trained several times with a slight Gaussian noise modification to its current parameters and the best performing resulting parameters are chosen as the after gate parameters.

The gate set of the circuits consists of the T, $\sqrt{X}$, $\sqrt{Y}$, and CZ gate. While the CZ gate is applied in different manners as described above, the T gate is a diagonal gate an can be applied exactly to the RBM state. The $\sqrt{X}$ and $\sqrt{Y}$ gates are non-diagonal and are thus applied by adopting the parameters of the RBM in a training phase. The training samples have been chosen $i.i.d$ uniformly at random. The number of samples has been chosen in such a way that with a

high probability $n, n^2, n^3, 2^n, 0.95 \times 2^n, 0.9 \times 2^n$ and 0.85×2^n respectively different samples are included in the training set. The number of samples is computed by the maths of the coupon collectors problem. The number of training iterations has been chosen as $1,000$, $10,000$, and $100,000$.

For each parameter setting, five RBMs have been trained on 20 different instances of random circuits of the same size. The total variation distance as well as the cross-entropy fidelity have been used as a performance measure. The training parameters have first been tested on random circuit instances with four qubits and five, ten, 15, and 20 cycles each.

The experiments resulting experiments are *AdaMax-restarts-learned*, *SR-restarts-learned*, *SR-restarts-not-learned* and *SR-no-restarts-learned*.

The study compares three different setups for SR with one setup for RBMs trained with AdaMax. The reason to vary the parameters for SR but not for AdaMax is twofold: Once, limited access to the cluster did not allow to test all possible parameter combinations. Second, prestudies, conducted as part of this study, suggested that AdaMax without random restarts was inferior to SR concerning the TVD between the RBMs state and the true output distribution of the circuits tested. RBMs trained with AdaMax in those settings repeatedly could not reduce the log overlap below a certain threshold during the training

5.2 Results

In total, 33,600 classical simulations of random quantum circuits have been performed on the Noctua cluster over 8 weeks. In each run, the total variation distance (TVD) between the RBM's output probability and the true output probabilities have been measured. The log overlaps of the RBM's state and the distribution on the training and test sets have been recorded during the training process. This allows for a detailed comparison of the different training methods and the influence of the different training parameters on the accuracy of the RBM. Further, the cross-entropy fidelity on the circuits' mean outcome probabilities has been calculated. The following four sections present the results of the four different training setups.

5.2.1 Stochastic Reconfiguration with Random Restarts and CZ-Gates Learned

Stochastic Reconfiguration (SR) has been successfully used to train RBMs to represent the wave function of many-body quantum systems before []. In this study, SR has been applied once with random restarts and without. This section summarizes the results of RBMs that have been trained with the SR optimizer and five random restarts.

Figure 5.1 shows the averaged output distribution of all RBMs and the averaged true output distributions of all circuits. With an increasing number of cycles, the

true output distributions approach a Porter-Thomas shape, even though they are not exactly Porter-Thomas distributed. The TVD of the average output of all RBMs considered is X (sigma = X) for 5, X () for 10, X () for 15 and X () for 20 cycles. The corresponding cross-entropy is X for 5, X for 10, X for 15 and X for 20 cycles.

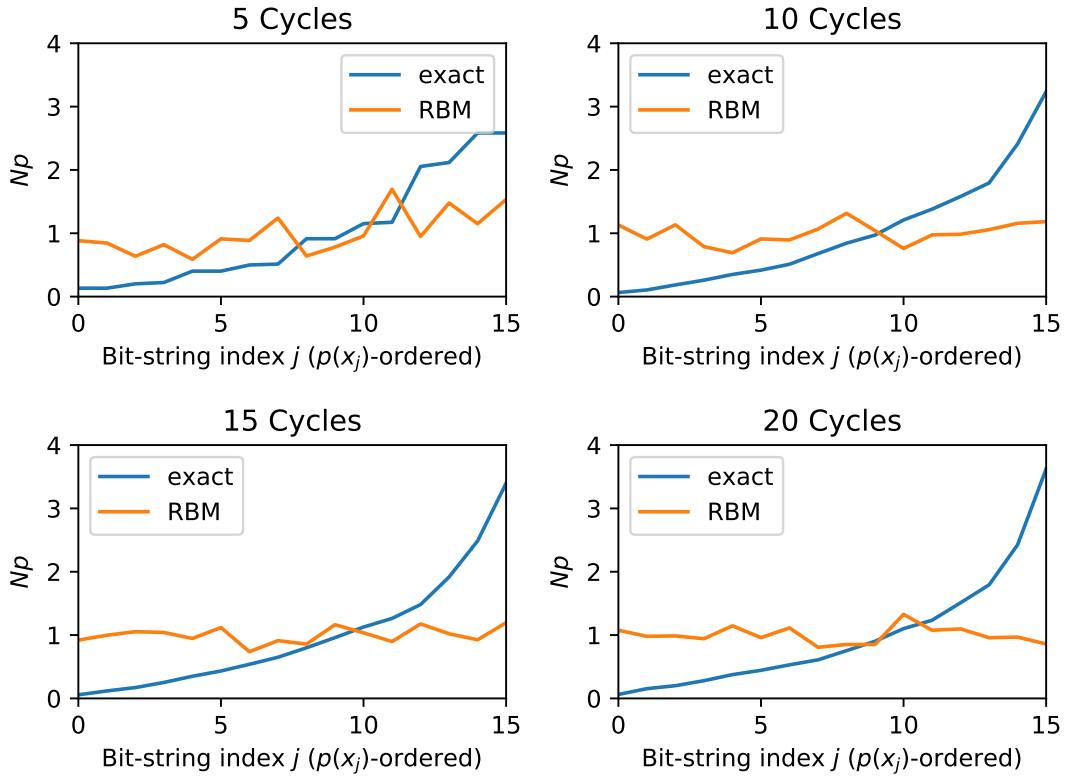


Figure 5.1: Average output probabilities of Stochastic Reconfiguration with Restarts and the CZ gates learned. The true output distribution approaches a Porter-Thomas shape with increasing number of cycles.

In figure **??**, only the output distribution of the best performing RBMs with respect to the TVD are selected and their outputs are averaged. The averaged best output distribution for the SR methods with random restarts and CZ gates learned has a TVD of X () for 5, X () for 10, X () for 15, and X for 20 () cycles. The cross-entropy fidelity is X for 5, X for 10, X for 15 and X for 20 cycles each.

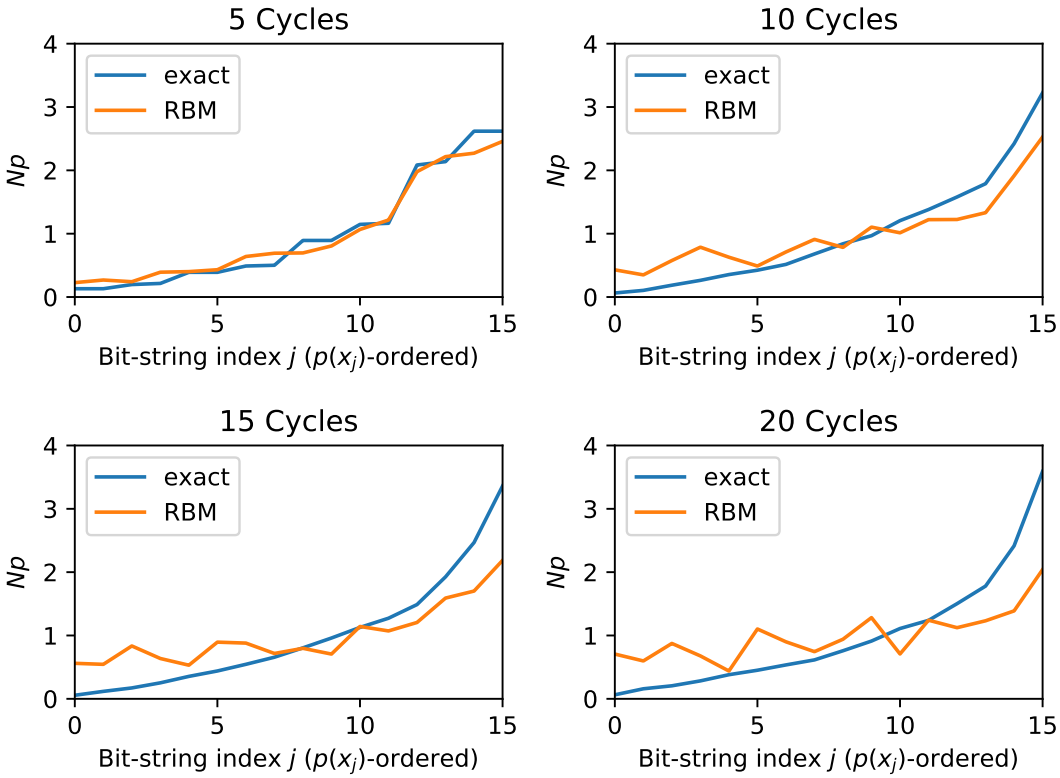


Figure 5.2: Average output probabilities of Stochastic Reconfiguration with Restarts and the CZ gate learned, only RBMs with lowest TVD for each circuit are considered.

Figure ?? and figure ?? give a more detailed overview of the influence of the number of training samples and training iterations on the TVD and cross-entropy fidelity.

There is no correlation between the number of training iterations and the performance of the RBMs. For 5 cycles, a higher number of training samples correlates with a lower TVD, independent of the number of training iterations. The RBMs perform best on circuits with a depth of 5 cycles when trained with 100,000 iterations and 303 samples. In that case, it achieves a mean TVD of 0.50. The mean TVD is similar when trained with 303 samples and 1.00 iterations, where it is 0.51. When trained with fewer samples, the TVD is generally lowest when trained with 1,000 iterations and highest when trained with 10,000 iterations for all number of samples tested.

On 10 cycles, RBMs trained for 1,000 training iterations achieve the lowest TVDs in the range of 43 to 54 samples. For 303 samples, 100,000 iterations lead to the lowest TVD (0.59). Except for the case of 47 samples, 10,000 iterations correlate with the highest TVD.

For 15 and 20 cycles, again 10,000 training iterations correlate with the lowest

TVD in most cases. 1,000 and 100,000 iterations perform similar, with the lowest TVDs on these circuits achieved with 50 samples and 100,000 iterations on 15 cycles (0.62) and 47 samples and 100,000 iterations on 20 cycles (0.68).

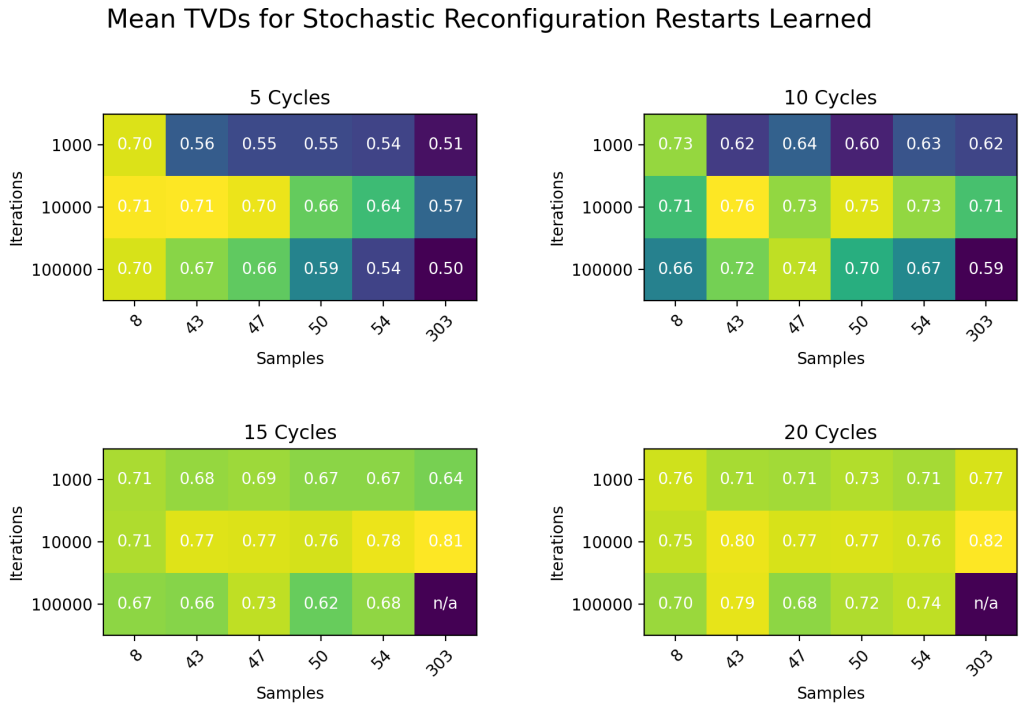


Figure 5.3: TVD of Stochastic Reconfiguration with Restarts Learned for the combinations of iterations and samples tested. For 100,000 iterations and 303 samples, the experiments did not finish within the time limit.

A lower TVD does not always correlate with a higher cross-entropy fidelity as figure 5.4 shows. On 5 cycles, the highest fidelity is achieved with 303 samples and 10,000 iterations, on 10 cycles with 303 samples and 100,000 iterations, 8 samples and 1,000 iterations on 15 cycles, and 47 samples and 100,000 iterations on 20 cycles each. Overall, there is a tendency for a higher cross-entropy fidelity when trained with more samples for 5 and 10 cycles. For 15 cycles, there is no clear correlation between the cross-entropy fidelity and the training parameters. For circuits with 20 cycles, 47 samples achieved the highest fidelity for all number of training samples.

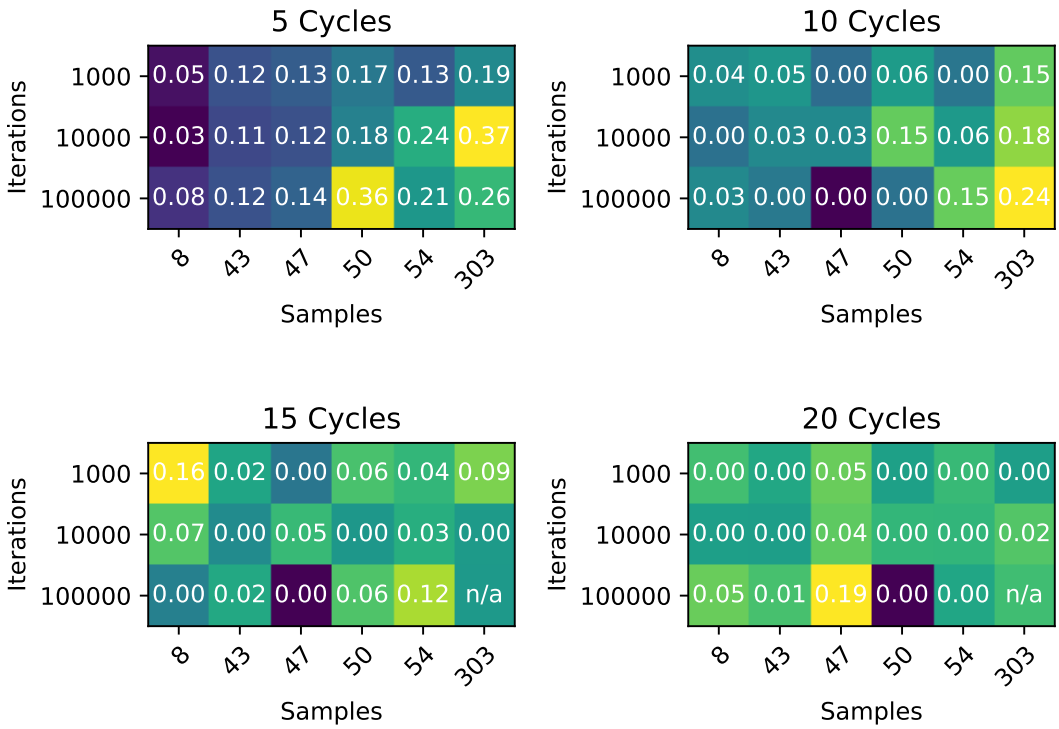


Figure 5.4: Cross-entropy Fidelity of Stochastic Reconfiguration with Restarts and the CZ gates learned for the combinations of iterations and samples tested.

Figure 5.5 to 5.7 detail the mean log overlap of the RBMs during the training process when trained with 10,000 iterations for 8, 47, and 303 samples. The mean overlap of the RBM's state and the distribution implied by the training and test data sets are measured for each training iteration.

For 8 samples, the log overlap on the training set can be reduced to almost 0 within the first few training iterations on all three gates. For the $\sqrt{Y}$ gate there is a second small dip after about 2,500 training iterations. The log overlap on the test set stays slightly below 0.60 for the single-qubit gates and at about 0.80 for the CZ gate.

Averaged over all three gates, the log overlap is slightly above 0 on the training data during the whole process. There is a small improvement within the first few training iterations and another small improvement at around 2,500 training iterations. The overall log overlap on the test data is about 0.6 and does not change much during the training process either. It slightly increases within the first few iterations and goes down a little bit again at about 2,500 training iterations.

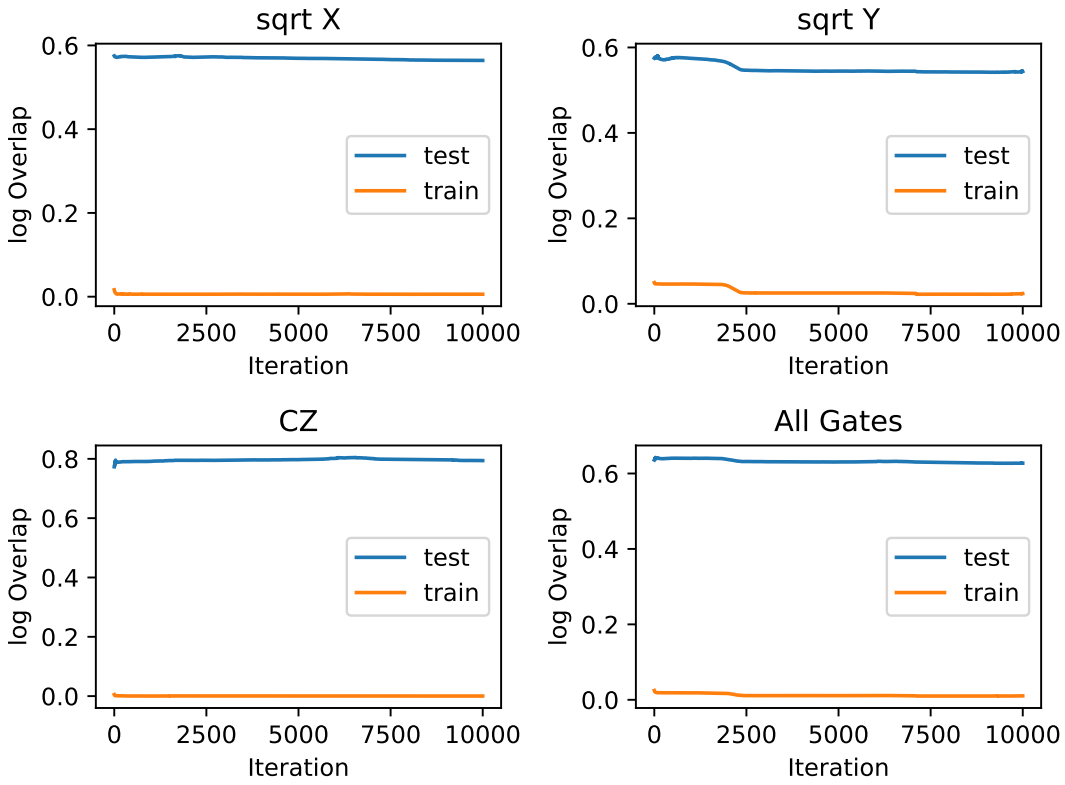


Figure 5.5: Training Overlap of Stochastic Reconfiguration with Restarts and the CZ gates learned for 8 samples.

For 47 samples, the change in the average training and test overlap is fluctuating during all 10,000 training iterations on all gates. For the $\sqrt{X}$ gate, the log overlap on the test set steadily decreases from about 0.35 to about 0.30. In the same period, the overlap on the training data decreases from about 0.24 to about 0.21 in that period.

For the $\sqrt{Y}$ gate, the training and test overlap decrease by about 0.50 within the first few iterations. Afterward, the training overlap slightly increases again until about the 2,500th iteration before it decreases again until the 10,000th iteration. On the test data, the overlap follows a similar same pattern. The training overlap is slightly below 0.275 after 10,000 iterations. The test overlap is at about 0.325 at the end of the 10,000 iterations.

For the CZ gate, there is also an initial drop in the training and testing overlaps. Afterward, the training overlap oscillates at around 0.5 for the first 5.00 iterations before it slowly increases to about 0.10 after 10,000 iterations. The test overlap oscillates around about 0.16 for the first 5,000 iterations and slowly increases to about 0.17 within the following 5,000 iterations.

Averaged over all gates, the training and test error both drop within the first few iterations to about 0.20 on the test data and about 0.27 on the training data.

Afterward, the overlap slowly decreases to about 0.19 and 0.24 over the period of the 10,000 iterations.

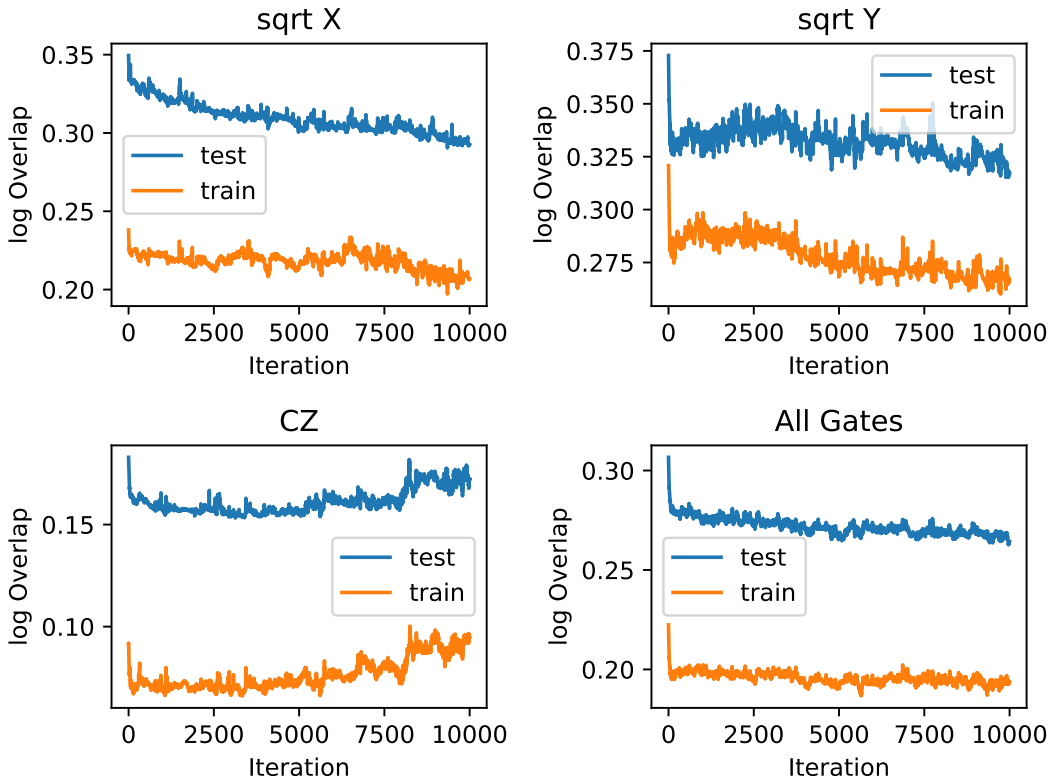


Figure 5.6: Training Overlap of Stochastic Reconfiguration with Restarts Learned for 47 samples.

With 303 samples, the mean test and training overlaps oscillate even more. The overlap on the training and the test data is very close.

For the $\sqrt{X}$ gate, the overlap drops from about 0.28 to 0.23 within the first very few iterations. Afterward, it slowly decreases further to about 0.22 throughout the 10,000 iterations.

On the $\sqrt{Y}$ gate, the overlap has the biggest decrease from about 0.28 to 0.25 within the first few iterations. Afterward, it keeps steady to about the 2,500th iteration before it drops to 0.24 at about 5,000 iterations. Then, it increases again to about 0.255 at about 7,500 iterations and stays in that range for the remaining iterations.

On the CZ gate, the overlap starts at about 0.09 and drops to about 0.06 within the first few iterations. For the remaining training process, it slowly increases again to about 0.07.

Averaged over all gates, the training and testing overlap is about 0.22 at the beginning of the training, drops to 0.19 within the first very few iterations and

hits a minimum of about 0.18 at about 6.000 iterations before it increases again. Starting at about 9.000 iterations, it decreases again to about 0.18.

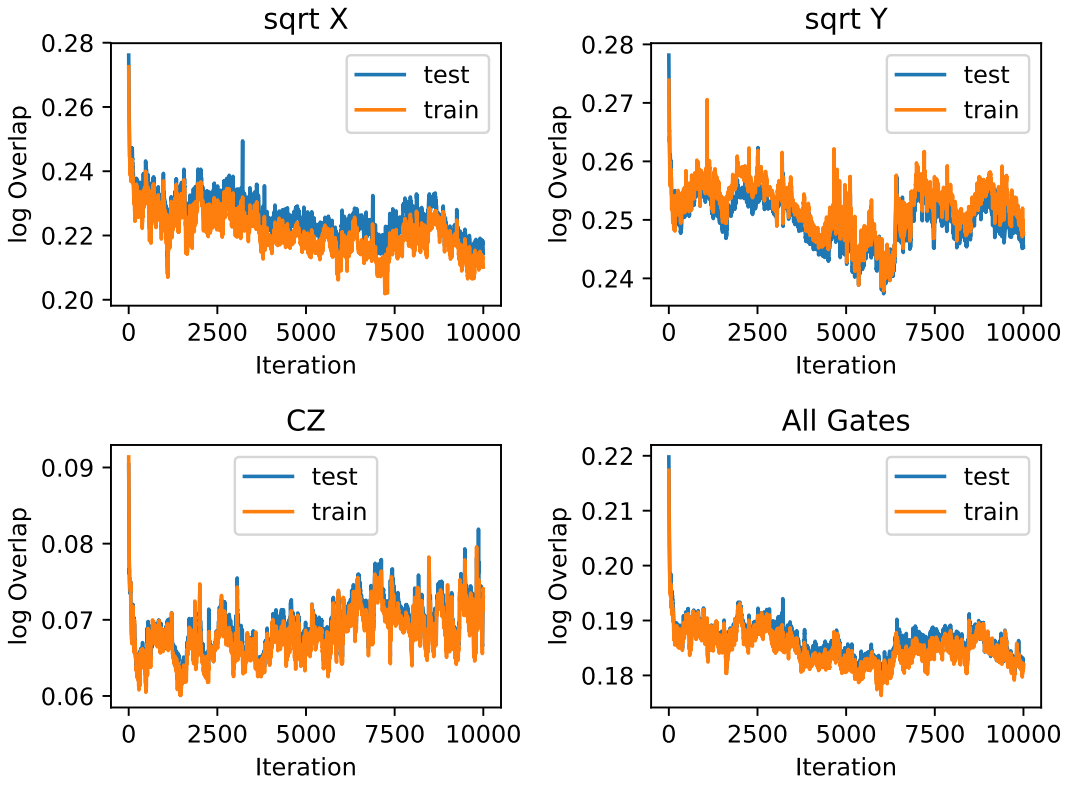


Figure 5.7: Training Overlap of Stochastic Reconfiguration with Restarts and the CZ gates learned for 303 samples.

5.2.2 Stochastic Reconfiguration without Random Restarts and CZ Gates Learned

The last section showed the results for RBMs that were trained with the Stochastic Reconfiguration method and random restarts. The following section shows the results of RBMs trained without restarts.

Figure 5.8 shows the averaged output distribution of all RBMs and the averaged true output distributions of all circuits. As before, the true output distributions approach a Porter-Thomas shape with an increasing number of cycles. The TVD of the average output of all RBMs trained with SR without restarts is X (sigma = X) for 5, X () for 10, X () for 15 and X () for 20 cycles. The corresponding cross-entropy is X for 5, X for 10, X for 15 and X for 20 cycles.

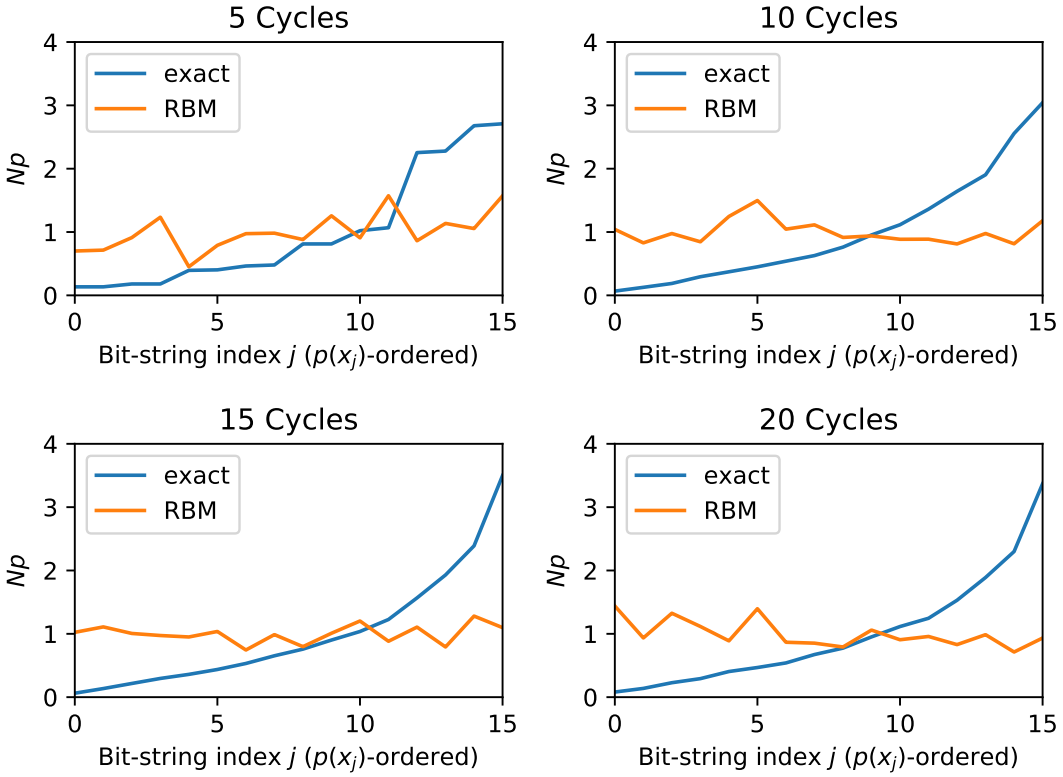


Figure 5.8: Average output probabilities of Stochastic Reconfiguration without Restarts Learned. The true output distribution approaches a Porter-Thomas shape with increasing number of cycles.

In figure **??**, only the output distribution of the best performing RBMs with respect to the TVD are selected and their outputs averaged. The averaged best output distribution for the SR methods without random restarts and CZ gates learned has a TVD of X () for 5, X () for 10, X () for 15, and X for 20 () cycles. The cross-entropy fidelity is X for 5, X for 10, X for 15 and X for 20 cycles each.

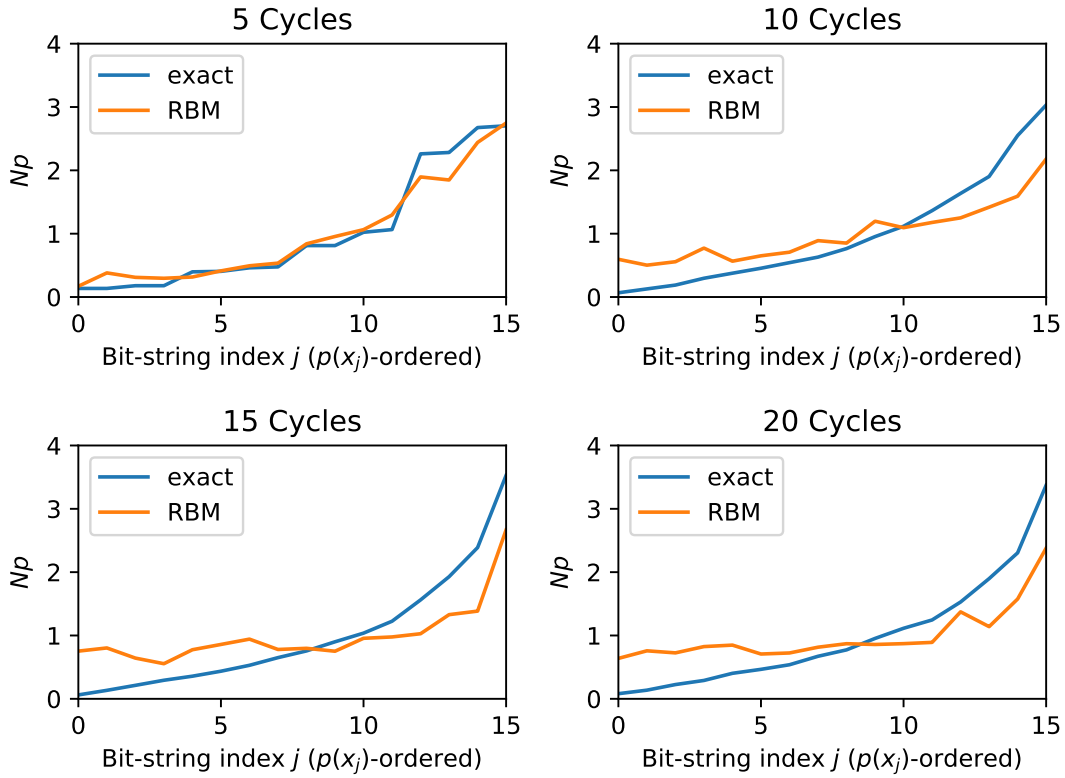


Figure 5.9: Average output probabilities of Stochastic Reconfiguration without Restarts Learned, only RBMs with lowest TVD for each circuit are considered. The true output distribution approaches a Porter-Thomas shape with increasing number of cycles.

Figure ?? and figure ?? give a more detailed overview of the influence of the number of training samples and training iterations on the TVD and cross-entropy fidelity achieved by the RBMs. Also here, there seems to be no correlation between the number of training iterations and the performance of the RBMs.

For 5 cycles, a higher number of training samples correlates with an overall lower TVD for 10,000 and 100,000 iterations, but not so for 1,000 iterations. The RBM performs best on circuits with a depth of 5 cycles when trained with 100,000 iterations and 303 samples. In that case, it achieves a mean TVD of 0.43. Overall, the performance is independent of the number of training iterations, when trained with less than 303 samples. In these cases, the performance is very similar for 1,000, 10,000, and 100,000 iterations. The only exception is for 54 samples, where 100,000 iterations lead to a mean TVD of 0.55, while the mean TVD is at 0.64 when trained with 1,000 and 0.62 when trained for 10,000 iterations.

On 10 cycles, 1,000 training iterations achieves the lowest TVDs in the range from 43 samples and more. 100,000 iterations lead to a lower TVD than 10,000 iterations in all cases.

For 15 and 20 cycles, again 10,000 training iterations correlate with the highest TVD in most cases. The performance is similar for all number of training samples. The lowest TVDs are achieved when trained for 1,000 iterations. Also in these cases, the performance is similar for all tested number of training samples with the exception that it is significantly higher for 8 samples.

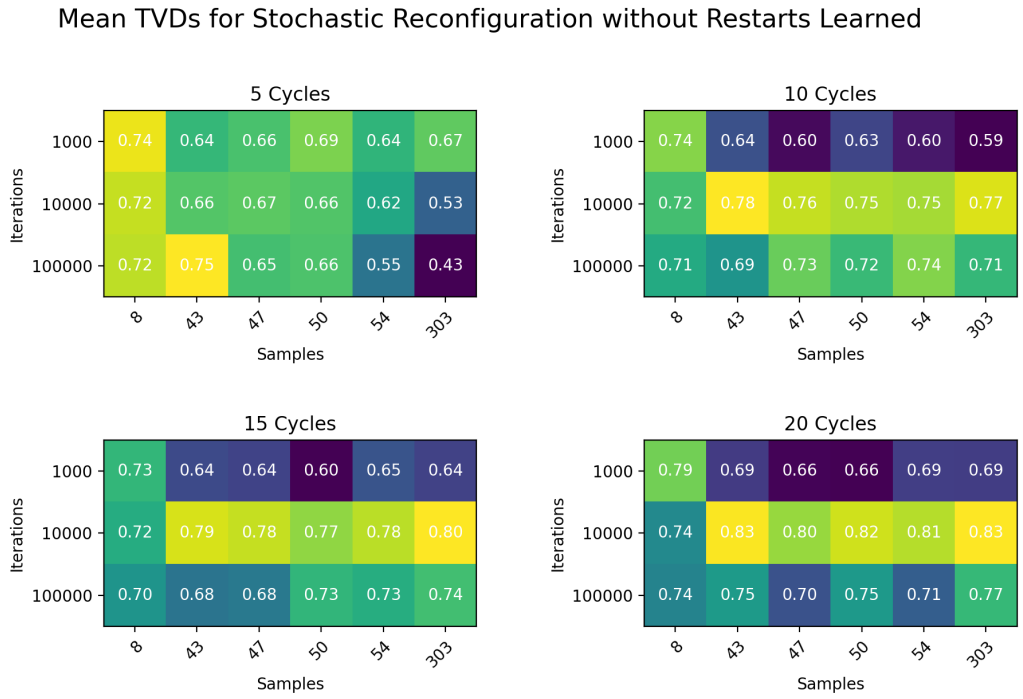


Figure 5.10: TVD of Stochastic Reconfiguration without Restarts Learned for the combinations of iterations and samples tested. For 100,000 iterations and 303 samples, the experiments did not finish within the time limit.

As for the case with restarts, a lower TVD does not always correlate with a higher cross-entropy fidelity as figure 5.11 implies. On 5 cycles, the highest fidelity is achieved with 303 samples and 10,000 iterations, on 10 cycles with 43 samples and 1,000 iterations, with 47 samples and 100,000 iterations on 15 cycles, and with 303 samples and 0.000 iterations on 20 cycles. There is no clear tendency that would show a correlation between the number of training iterations or samples and the cross-entropy fidelity.

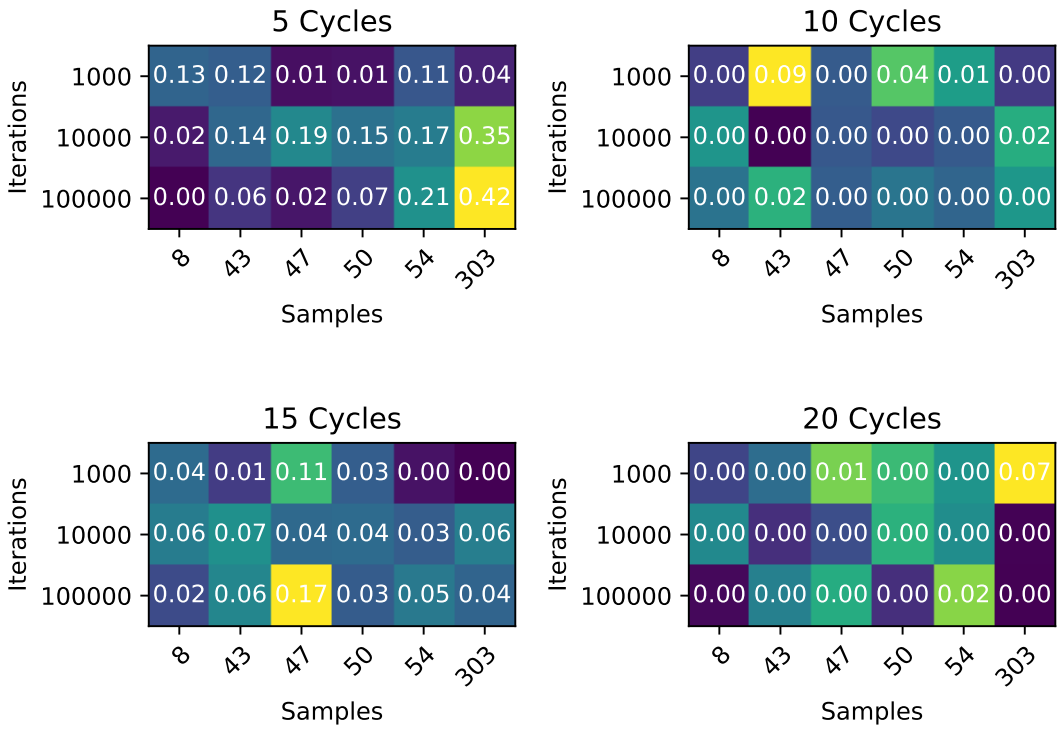


Figure 5.11: Cross-entropy Fidelity of Stochastic Reconfiguration without Restarts Learned for the combinations of iterations and samples tested. For 100,000 iterations and 303 samples, the experiments did not finish within the time limit.

Figure **??** to **??** detail the mean log overlap of the RBMs during the training process when trained with 10,000 iterations for 8, 47, and 303 samples. The mean overlap of the RBM's state and the distribution implied by the training and test data sets are measured for each training iteration.

For 8 samples, the log overlap on the training set can be reduced to about 0 for all three gates after about 2,500 iterations. on all different gates. For the $\sqrt{Y}$ gate, the overlap plateaus at about 0.1 after about 1,000 iterations before it starts to decrease again from 2,000 iterations on.

The log overlap on the test goes down to about 0.50 for the $\sqrt{X}$ gate within the first 500 iterations and stays at that level for the remaining training process. For the $\sqrt{Y}$ gate, the testing overlap decreases to about 0.55 within the first about 500 iterations. Afterward, it increases again to about 0.60 where it stays for the remaining iterations. For the CZ gate, it goes down to about 1.0 within the first 500 iterations and stays at that level afterward.

Averaged over all gates, the overlap goes down close to 0 within the first 2,500 training iterations. The test overlap reaches about 0.65 after 500 iterations at

which it stays for the remaining training iterations. It only slightly increases again over the course of the next about 2,000 iterations.

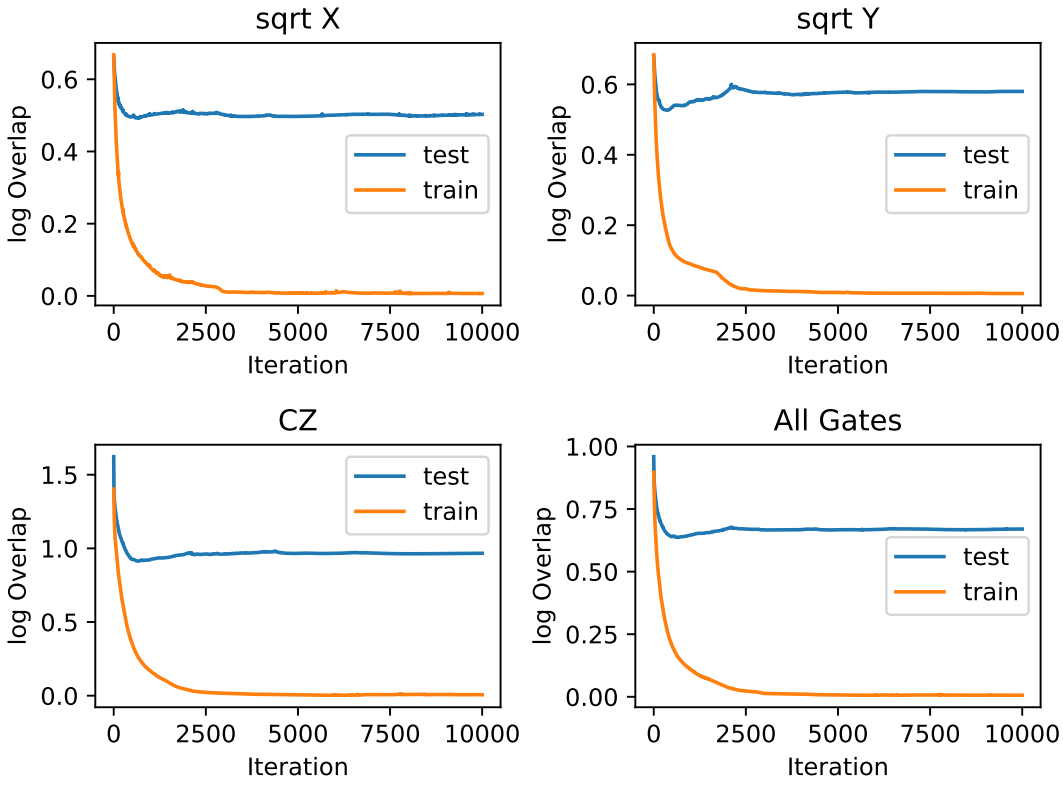


Figure 5.12: Training Overlap of Stochastic Reconfiguration without Restarts Learned for 8 samples.

For 47 samples, the change in the average training and test overlap is slightly more fluctuating than for 8 qubits. For the $\sqrt{X}$ gate, the log overlap on the training set steadily decreases on the test data set from about 0.70 to about 0.19 within the first 2,500 training iterations. The overlap on the test data decreases to about 0.25 within the same time. Afterward, the training overlap increases again to about 0.20 within the next 1.500 iterations. At the same time, the overlap on the test data increases to about 0.3 and stays at that level afterward.

On the $\sqrt{Y}$ gate, the training overlap decreases to about 0.20 within the first 2,500 training iterations and stays at that level from thereon. The test overlap decreases to about 0.29 within the first 2,500 iterations and also stays there for the remaining part of the training process.

On the CZ gate, the training overlap decreases to about 0.1 in the first 3.000 training iterations. It does not change much afterward. The overlap on the test data goes down to about 0.2 within the first 3.500 iterations and stays at that level.

Averaged over all gates, the overlap on the test data goes down to about 0.18 within the first 2,500 training iterations. The test overlap goes down to about 0.25 in the same time. Both metrics stay at those values afterward.

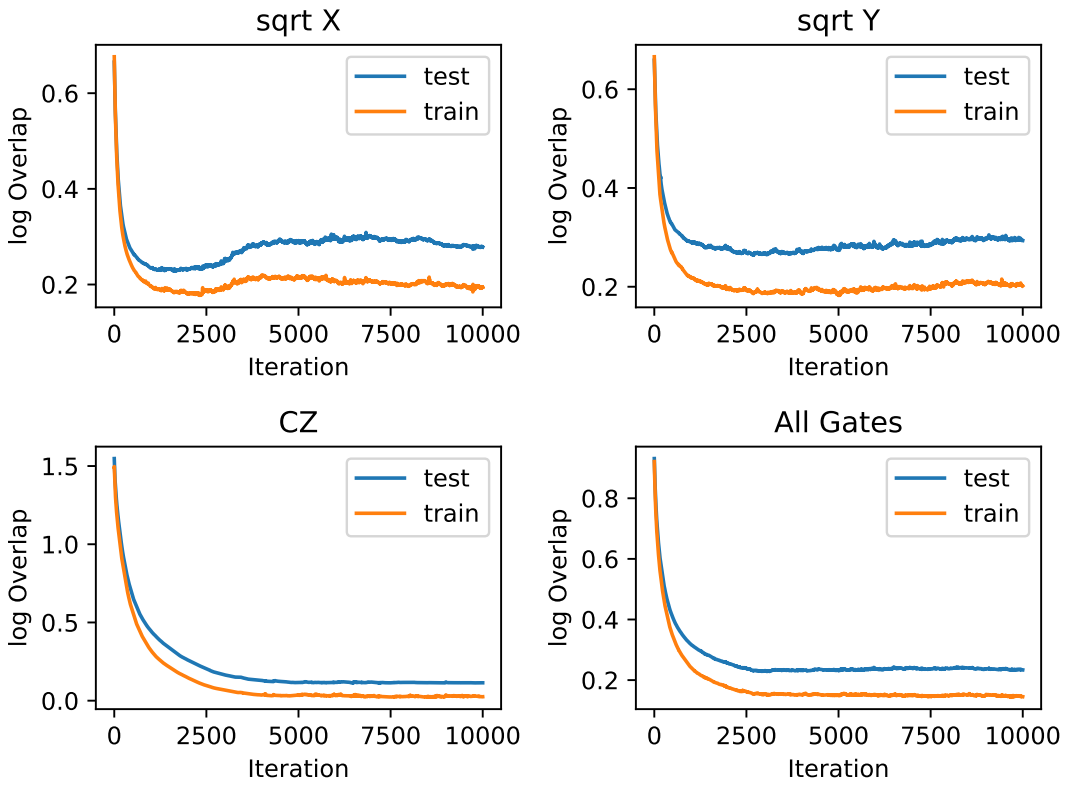


Figure 5.13: Training Overlap of Stochastic Reconfiguration without Restarts Learned for 47 samples.

With 303 samples, the curves of the log overlap on the training data look very similar to the ones with 47 samples. A noticeable difference can be observed for the test overlap, which is about the same as the training overlap in all cases.

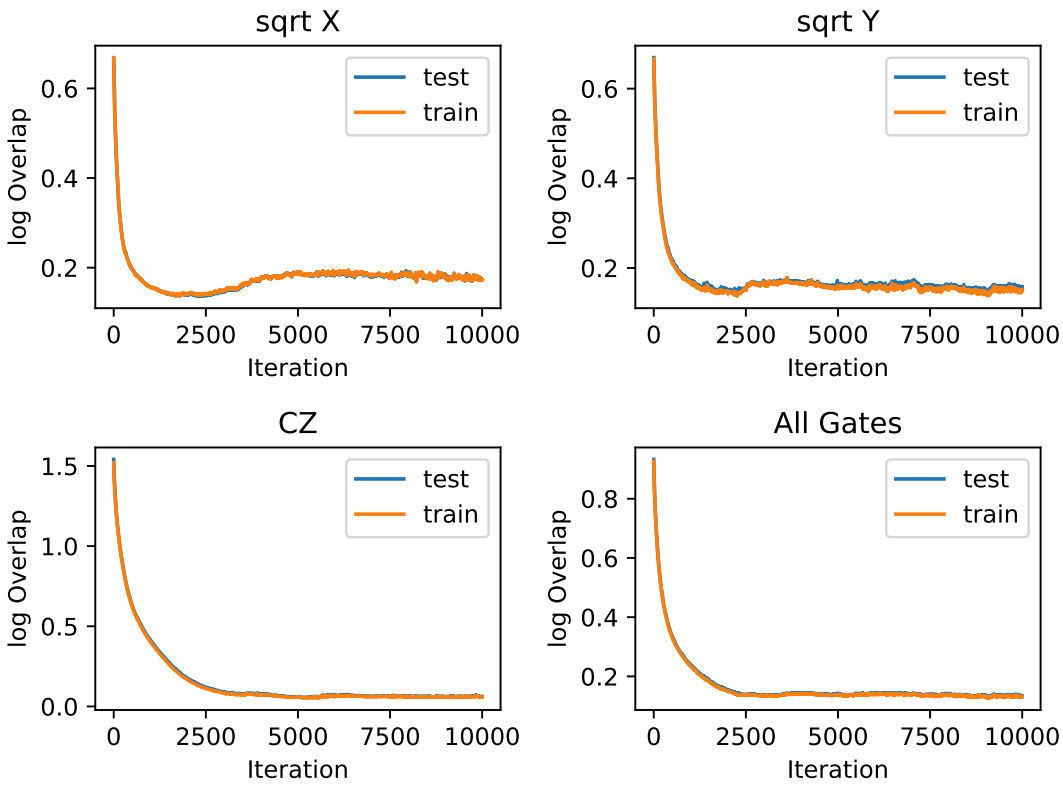


Figure 5.14: Training Overlap of Stochastic Reconfiguration without Restarts Learned for 303 samples.

5.2.3 Stochastic Reconfiguration with Random Restarts and CZ Gates Applied Exactly

In previous works, the CZ gate had been applied to the RBM state exactly. This study compares the performance of RBMs when the CZ gate is applied exactly and learned. The comparison might shed some light on how the training process of a gate is influenced by the number of qubits the gate is acting on. It further can give a hint on whether the addition of hidden units by applying the CZ gate exactly influences the training process of the other gates. The following section shows the results of RBMs trained with restarts and the CZ gates applied with the rules from section X.

Figure 5.15 shows the averaged output distribution of all RBMs and the averaged true output distributions of all circuits. As in the cases before, the true output distributions approach a Porter-Thomas shape with an increasing number of cycles. The TVD of the average output of all RBMs is X (sigma = X) for 5, X () for 10, X () for 15 and X () for 20 cycles. The corresponding cross-entropy is X for 5, X for 10, X for 15 and X for 20 cycles.

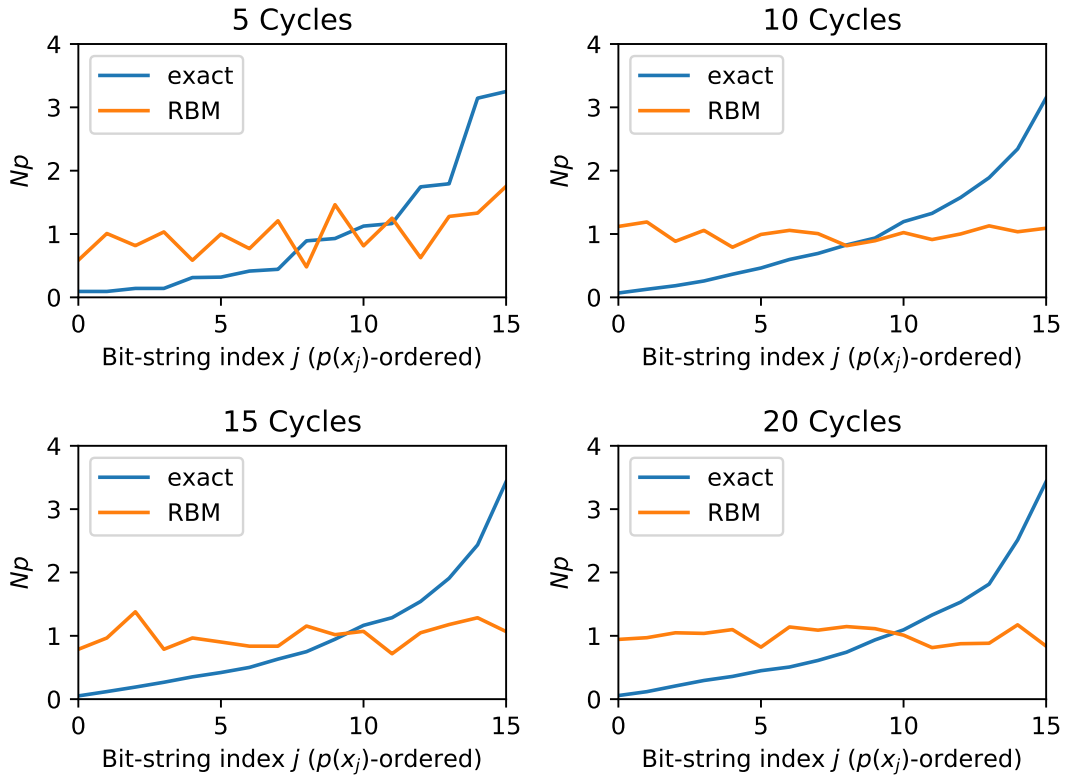


Figure 5.15: Average output probabilities of Stochastic Reconfiguration with Restarts Exact. The true output distribution approaches a Porter-Thomas shape with increasing number of cycles.

In figure **??**, only the output distribution of the best performing RBMs with respect to the TVD are selected and their outputs averaged once again. The averaged best output distribution has a TVD of X () for 5, X () for 10, X () for 15, and X for 20 () cycles. The cross-entropy fidelity is X for 5, X for 10, X for 15 and X for 20 cycles each.

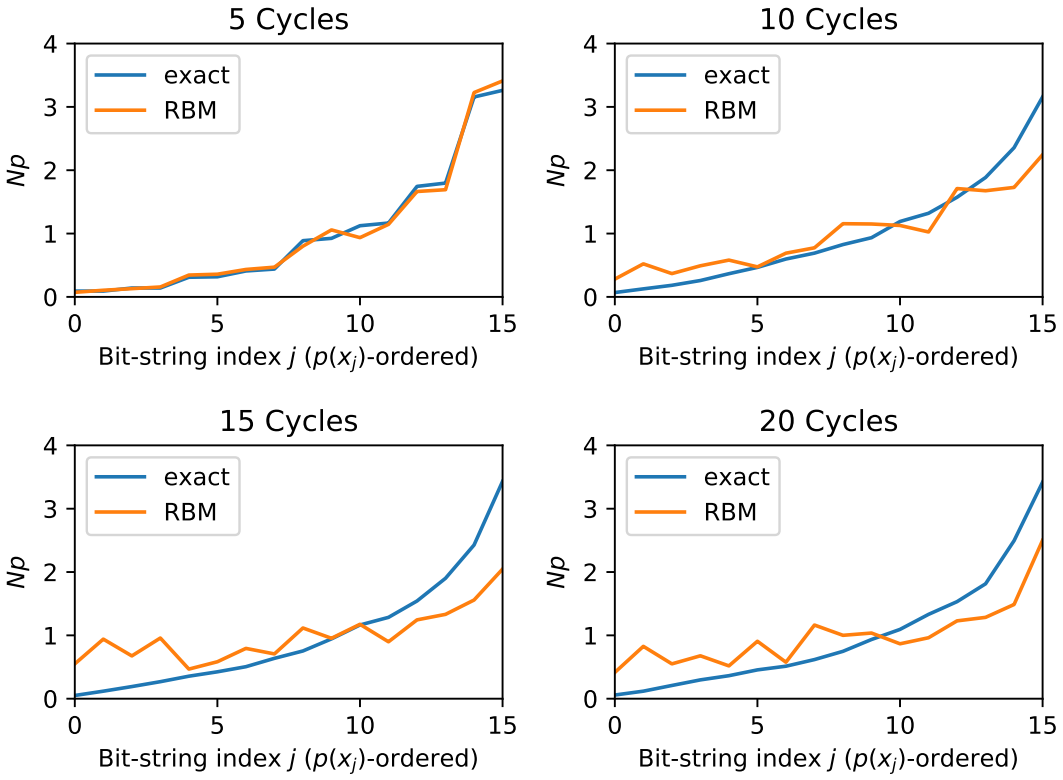


Figure 5.16: Average output probabilities of Stochastic Reconfiguration with Restarts Exact, only RBMs with lowest TVD for each circuit are considered. The true output distribution approaches a Porter-Thomas shape with increasing number of cycles.

Figure **??** and figure **??** detail the influence of the number of training samples and training iterations on the TVD and cross-entropy fidelity achieved by the RBMs.

For 5 cycles, a higher number of training samples correlates with an overall lower TVD. The RBM performs best on circuits with a depth of 5 cycles when trained with 100,000 iterations and 303 samples. In that case, it achieves a mean TVD of 0.49. Overall, the performance is similar for 1,000 and 100,000 iterations.

On 10 cycles, the lowest TVD is also achieved by RBMs trained with 303 samples and 100,000 iterations. A higher number of samples correlates with an overall lower TVD for 100,000 iterations. For 1,000 iterations, the TVD is about 0.60 for

43 samples onwards and 0.74 for 8 samples. For 10,000 iterations, the TVD varies around about 0.75 independent of the number of samples.

For 15 and 20 cycles, again 10,000 training iterations correlate with the highest TVD in most cases. The performance is similar for all number of training samples. The lowest TVDs are achieved with 100,000 iterations and 47 or 54 samples. With 10,000 iterations, a higher number of training samples correlates with a higher TVD. For 1,000 and 100,000 iterations, there is no clear difference in the number of training samples to be recognized.

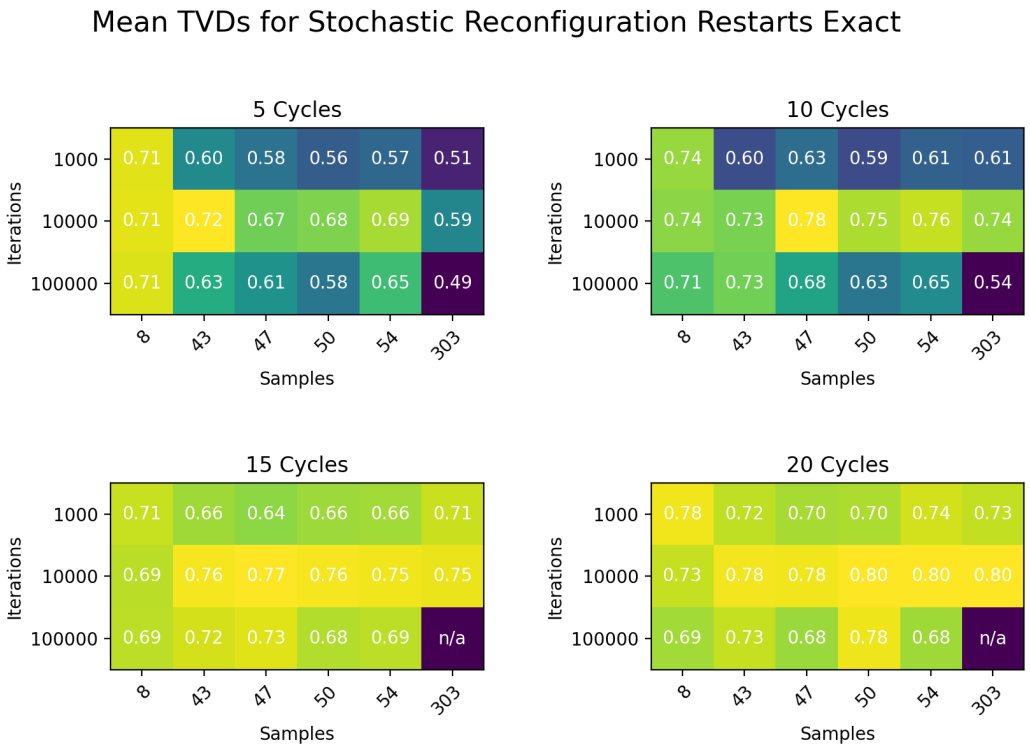


Figure 5.17: TVD of Stochastic Reconfiguration with Restarts Exact for the combinations of iterations and samples tested. For 100,000 iterations and 303 samples, the experiments did not finish within the time limit.

Figure 5.11 shows the average cross-entropy fidelity achieved. On 5 and 10 cycles, the highest fidelity is achieved with 303 samples and 100,000 iterations. On 15 cycles, the highest fidelity is achieved with 50 samples on 100,000 iterations and 303 samples and 10,000 iterations. On 20 cycles, the highest fidelity is achieved on RBMs trained with 47 samples for 100,000 iterations. There is again no clear tendency that would show a correlation between the number of training iterations or samples and the cross-entropy fidelity.

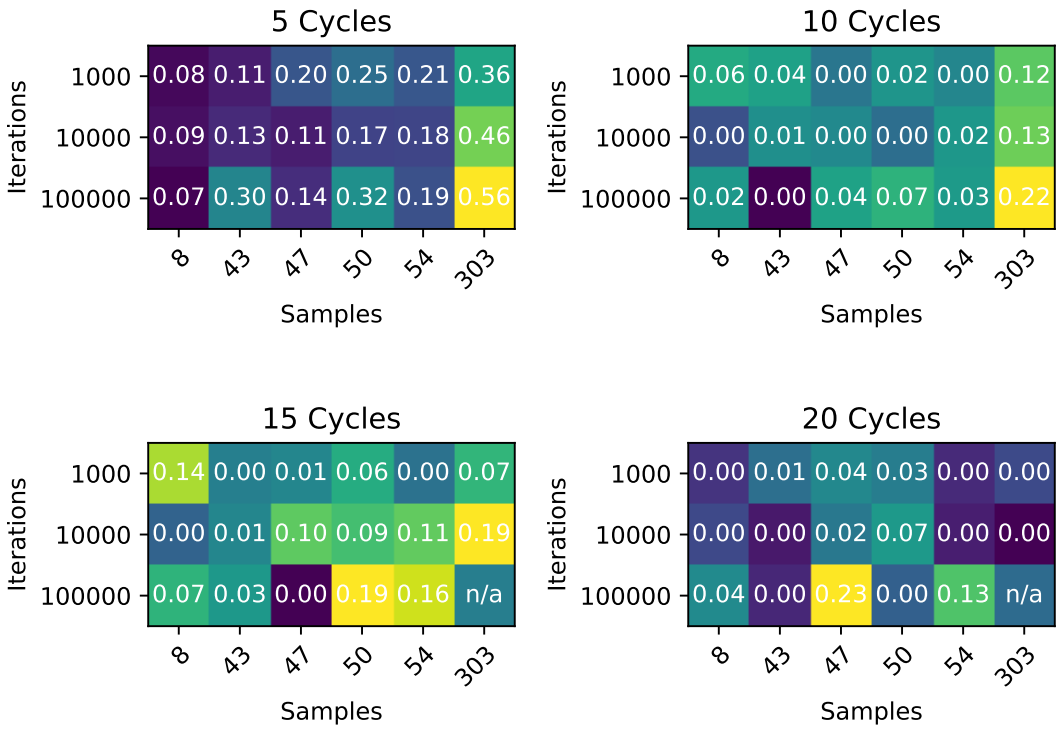


Figure 5.18: Cross-entropy Fidelity of Stochastic Reconfiguration with Restarts Exact for the combinations of iterations and samples tested. For 100,000 iterations and 303 samples, the experiments did not finish within the time limit.

Figure ?? to ?? detail the mean log overlap of the RBMs during the training process when trained with 10,000 iterations for 8, 47, and 303 samples. The mean overlap of the RBM's state and the distribution implied by the training and test data sets are measured for each training iteration.

For 8 samples, not much is happening in the training process at all. The training errors on the $\sqrt{X}$ and $\sqrt{Y}$ gate are both close to 0 during the whole training process. The test error is about 0.60.

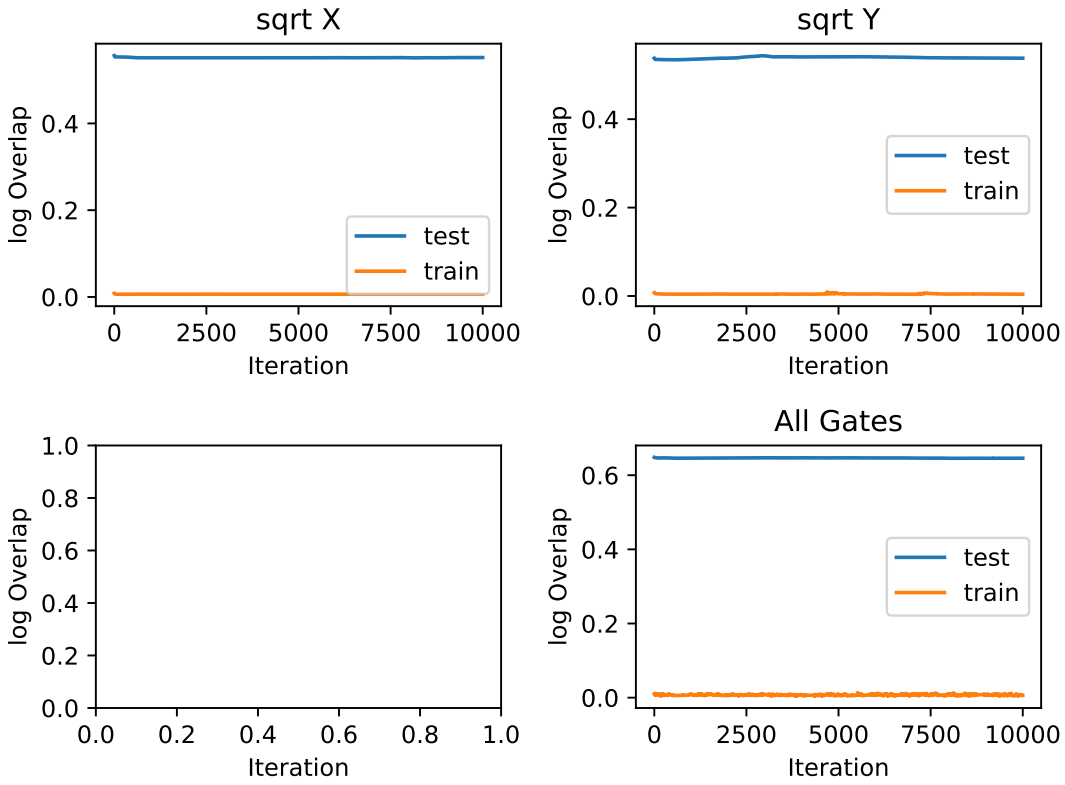


Figure 5.19: Training Overlap of Stochastic Reconfiguration with Restarts Exact for 8 samples.

For 47 samples, the change in the average training and test overlap is oscillating. For the $\sqrt{X}$ gate, the log overlap on the training set oscillates around about 0.16 during most of the training process. It reaches its lowest value of about 0.13 after 3.000 iterations. The overlap on the test data oscillates around about 0.21 during the whole process. Initially, the training error drops from 0.175 to 0.16 within the first few iterations. The training overlap drops from 0.25 to 0.21 at that time.

On the $\sqrt{Y}$ gate, the training overlap decreases to about 0.17 within the first few training iterations and oscillates around that value from thereon. The test overlap decreases to about 0.28 within the first few iterations and slowly increases to about 0.3 throughout the training process.

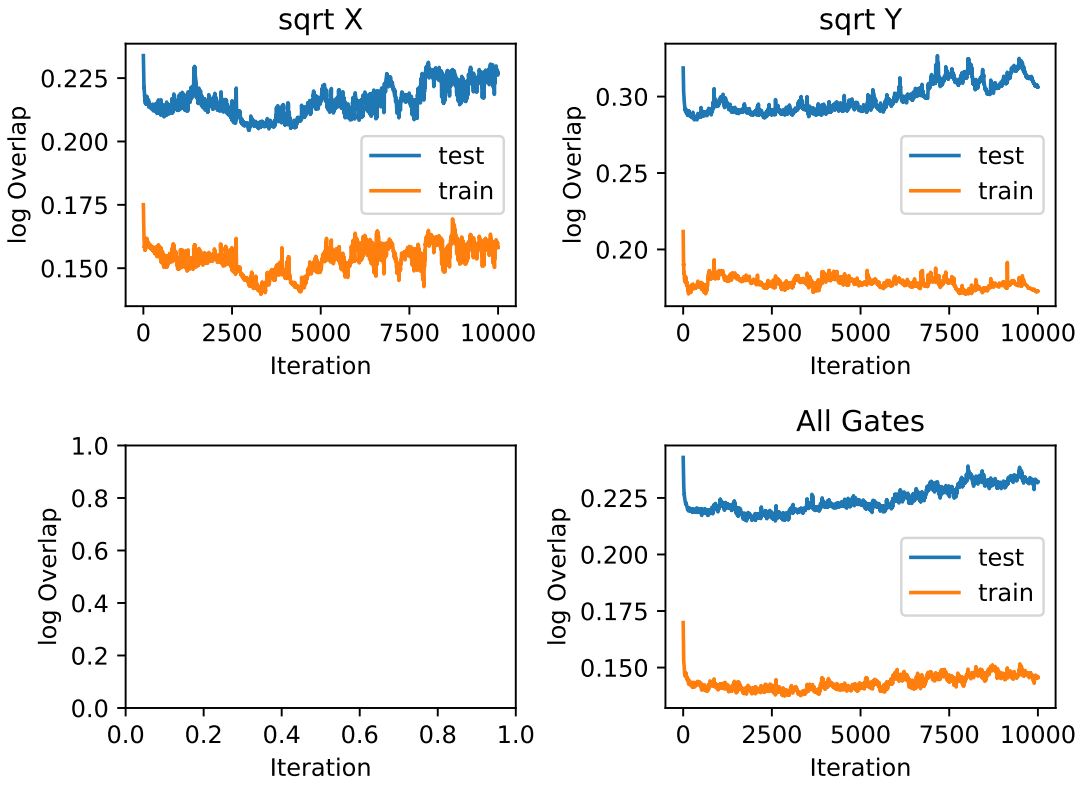


Figure 5.20: Training Overlap of Stochastic Reconfiguration with Restarts Exact for 47 samples.

With 303 samples, the oscillation in the test and training overlap is even more than for 47 samples. For the $\sqrt{X}$ gate, both values drop to about 0.21 within the first 1,000 training iterations. Afterward, the overlaps increase to about 0.25 at 6.000 iterations and decrease to about 0.24 in the remaining part of the training process again.

The process looks similar for the $\sqrt{Y}$ gate. The overlap goes down to about 0.22 within the first 1,000 iterations. Afterward, it goes up to about 0.24 again and stays at that level for the following training iterations.

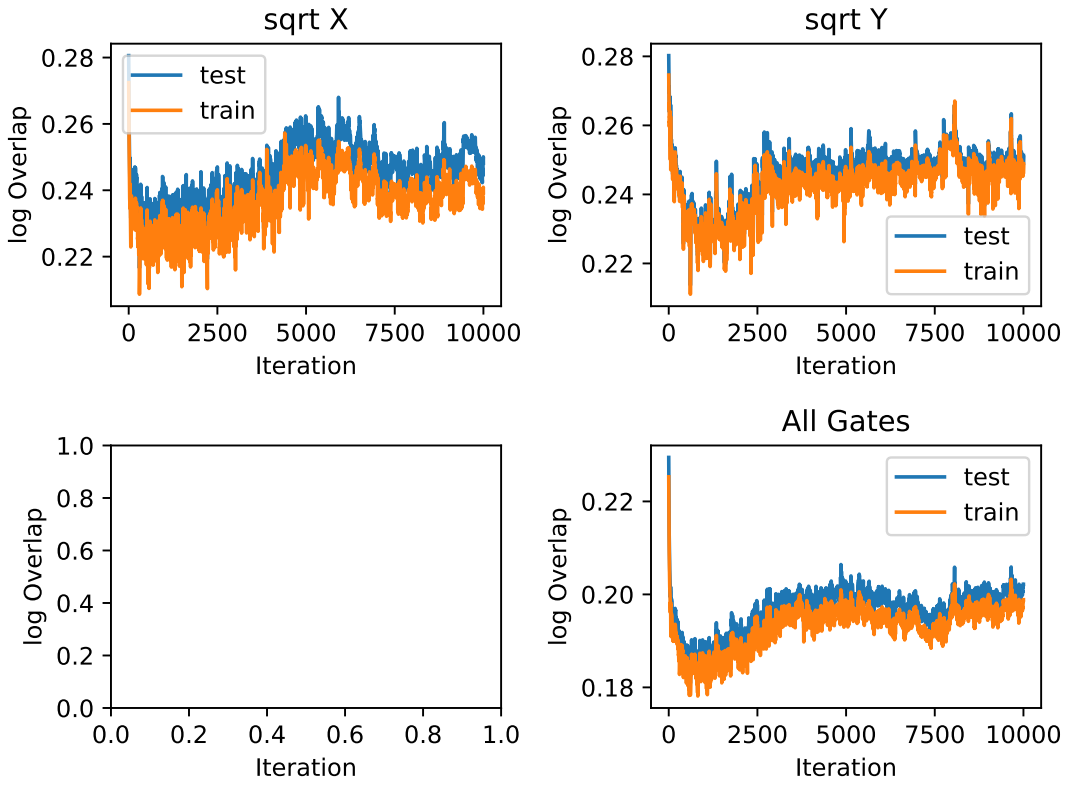


Figure 5.21: Training Overlap of Stochastic Reconfiguration with Restarts Exact for 303 samples.

5.2.4 AdaMax with Random Restarts and CZ Gates Learned

AdaMax had also been used for training RBMs for the classical simulation of quantum circuits before. The following section presents the results of RBMs trained with AdaMax and five random restarts. The CZ gates are also applied with the learning approach in these cases.

Figure 5.22 shows the averaged output distribution of all RBMs trained with AdaMax and the averaged true output distributions of all circuits. As in the other cases, the true output distributions approach a Porter-Thomas shape with an increasing number of cycles. The TVD of the average output of all RBMs trained with AdaMax is X (sigma = X) for 5, X () for 10, X () for 15 and X () for 20 cycles. The corresponding cross-entropy is X for 5, X for 10, X for 15 and X for 20 cycles.

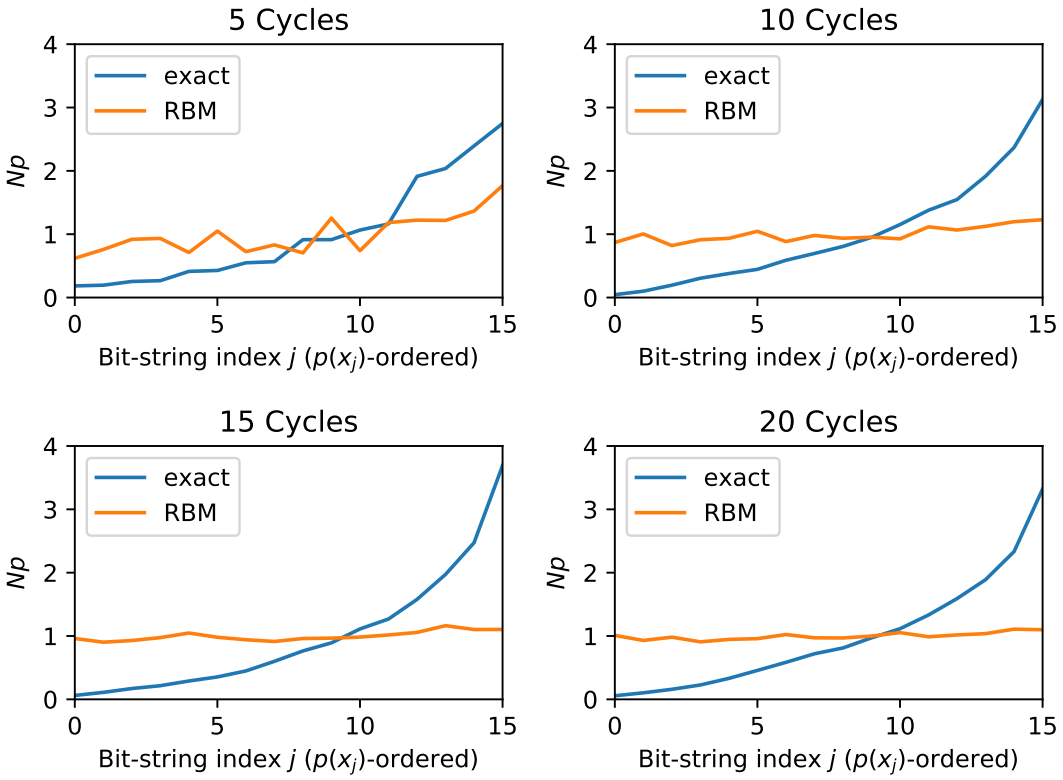


Figure 5.22: Average output probabilities of AdaMax with Restarts Learned. The true output distribution approaches a Porter-Thomas shape with increasing number of cycles.

In figure **??**, only the output distribution of the best performing RBMs with respect to the TVD are selected and their outputs averaged once again. The averaged best output distribution has a TVD of X () for 5, X () for 10, X () for 15, and X for 20 () cycles. The cross-entropy fidelity is X for 5, X for 10, X for

15 and X for 20 cycles each.

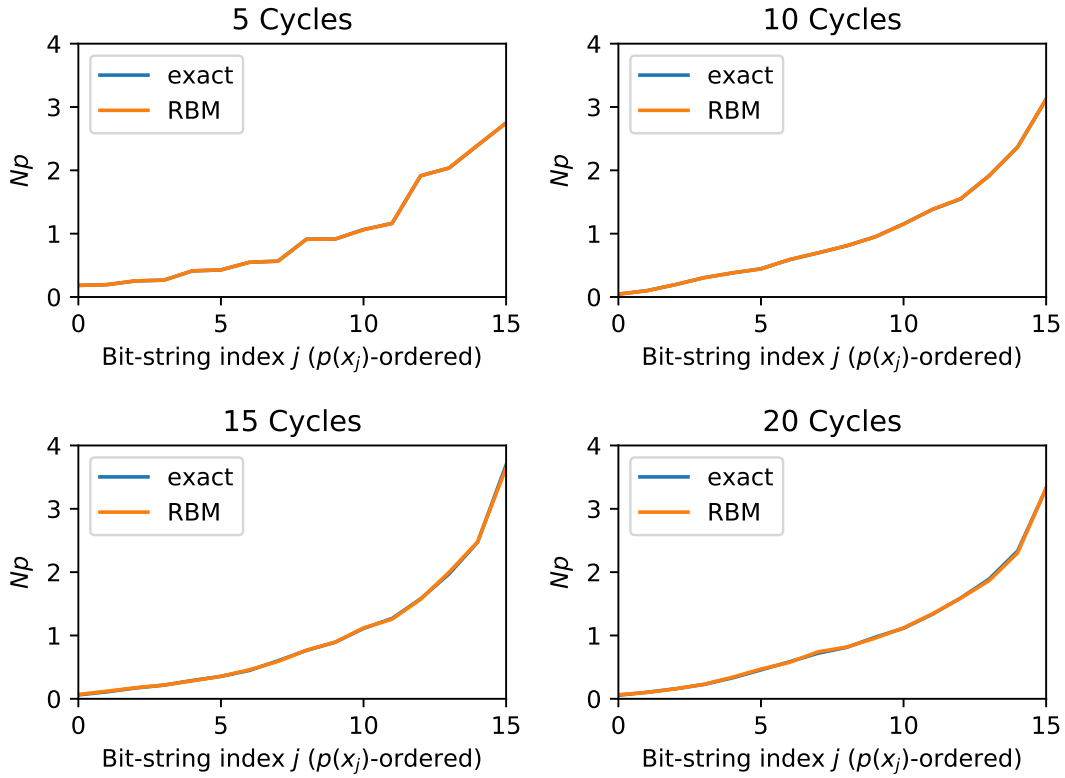


Figure 5.23: Average output probabilities of AdaMax with Restarts Learned, only RBMs with lowest TVD for each circuit are considered. The true output distribution approaches a Porter-Thomas shape with increasing number of cycles.

Figure **??** and figure **??** give a more detailed overview of the influence of the number of training samples and training iterations on the TVD and cross-entropy fidelity.

Except for the case of 8 samples, for AdaMax a higher number of training samples as well as a higher number of training iterations corresponds to a lower TVD. For 5 and 10 cycles, the TVD is lowest when trained with 303 samples and 100,000 iterations. In these cases, it approaches 0.

For 15 and 20 cycles, the training process could not finish within the given timeframe with this combination of training parameters. On these circuits, the lowest TVDs have been achieved with 303 samples and 10,000 iterations.

Overall, the difference between 10,000 iterations and 100,0000 iterations is smaller than between 1,000 iterations and 10,000 iterations. For 43 to 54 samples, the performance is similar in all cases. Nevertheless, the TVD is lower in all but one cases with 54 samples than with 43 samples.

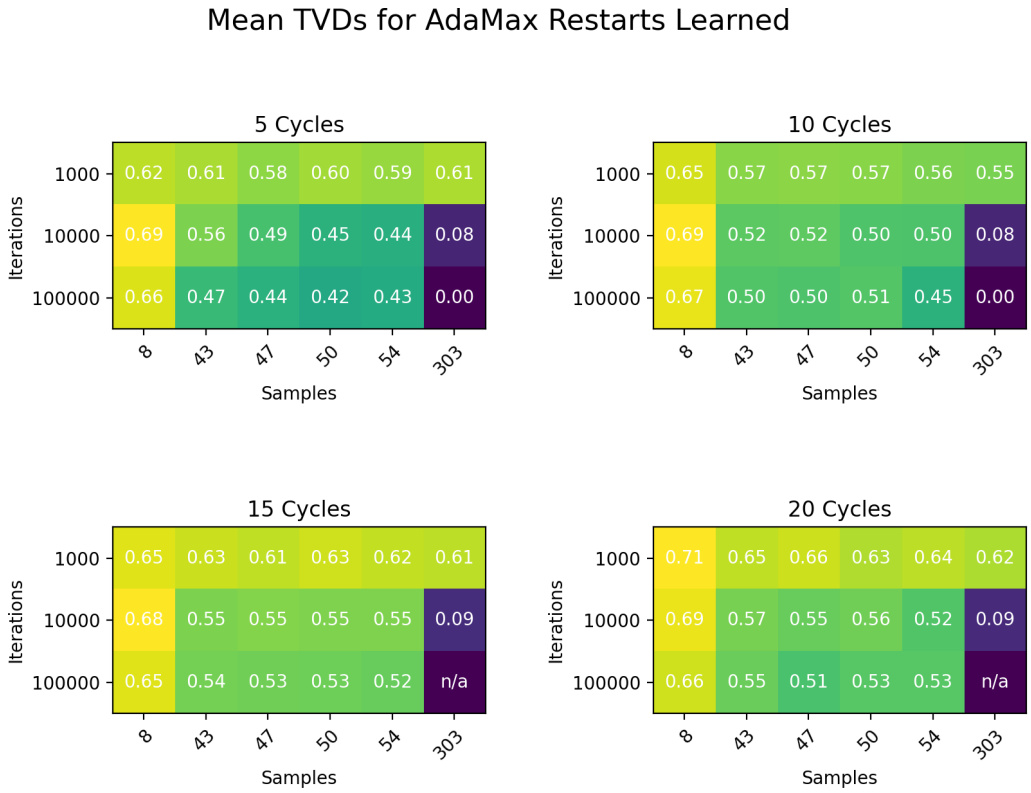


Figure 5.24: TVD of Stochastic Reconfiguration with Restarts Learned for the combinations of iterations and samples tested. For 100,000 iterations and 303 samples, the experiments did not finish within the time limit.

Figure 5.25 shows the average cross-entropy fidelity achieved. In contrast to the case of TVD, a higher number of training samples and training iterations does not correlate with higher fidelities. Only in the case of 5 cycles, this correlation can be observed for 10,000 and 100,000 iterations.

The highest fidelities for 5 and 10 cycles are achieved for 303 samples and 100,000 iterations. For 15 and 20 cycles, they are achieved with 303 samples and 10,000 iterations.

In the case of 15 cycles, the overall highest fidelity of 0.90 had been measured. The highest fidelity is 0.65 for 5, 0.75 for 10, and 0.73 for 20 cycles. This once again demonstrates that a lower TVD does not translate to a higher fidelity directly.

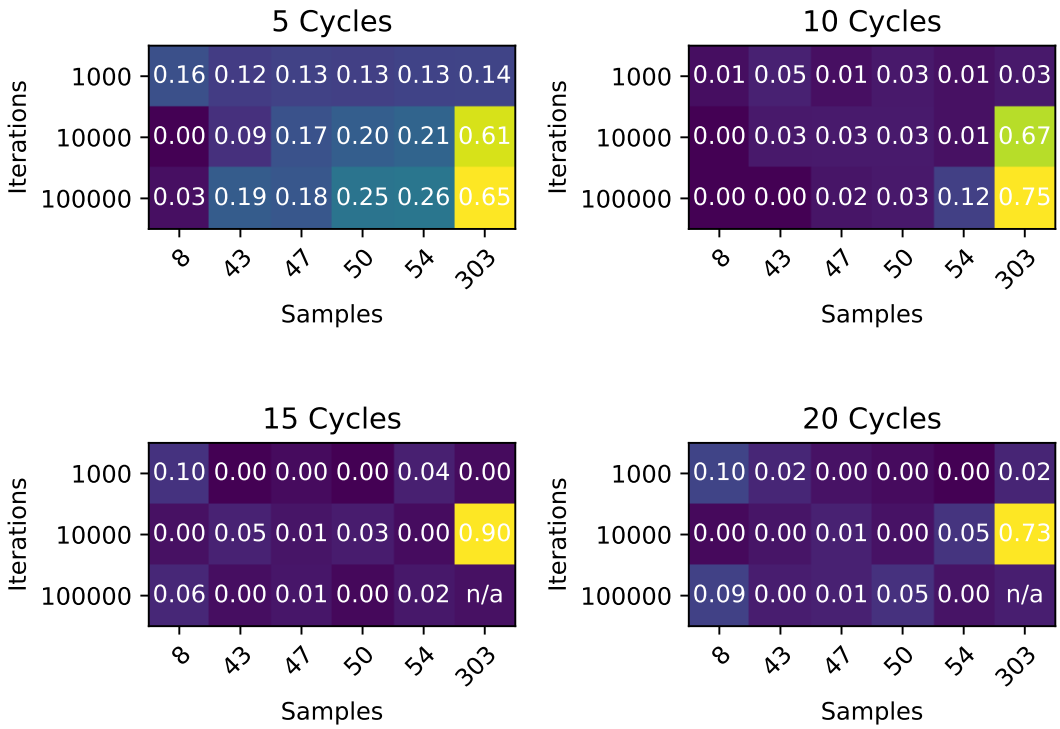


Figure 5.25: Cross-entropy Fidelity of Stochastic Reconfiguration with Restarts Learned for the combinations of iterations and samples tested. For 100,000 iterations and 303 samples, the experiments did not finish within the time limit.

Figure ?? to ?? detail the mean log overlap of the RBMs during the training process when trained with 10,000 iterations for 8, 47, and 303 samples. The mean overlap of the RBM's state and the distribution implied by the training and test data sets are measured for each training iteration. For 8 samples, not much is happening in the training process at all. The training errors on all three gates close to 0 during the whole training process. The test error is at about 0.40 on the single-qubit gates and about 0.9 on the CZ gate.

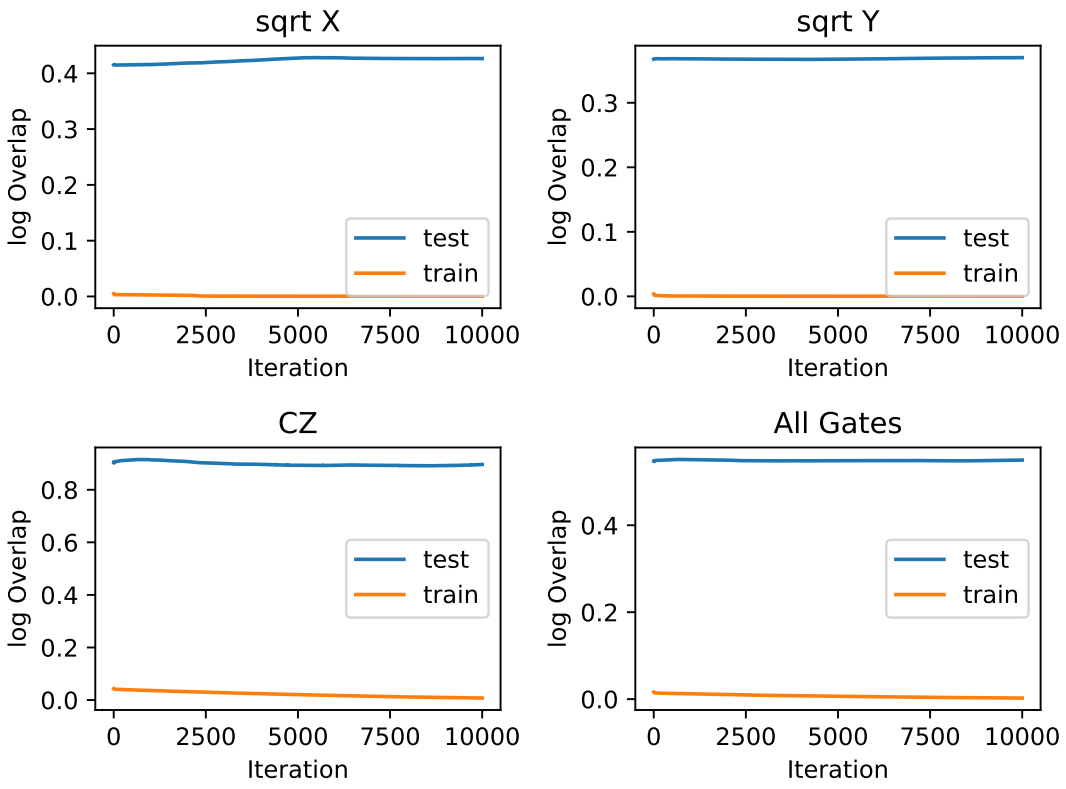


Figure 5.26: Training Overlap of AdaMax with Restarts Learned for 8 samples.

For 47 samples, the log overlap on the training data drops to close to 0 within the first few iterations of training. There is only a small improvement to be observed in the $\sqrt{Y}$ gate afterward. The overlap on the test data drops to about 0.1 in the case of $\sqrt{X}$ gates within the first few iterations. Afterward, there is a slight tendency upwards but the overlap stays in about that range. In the case of $\sqrt{Y}$ gates, the test overlap drops to about 0.11 within the first few iterations. Afterward, it slightly increases again before it reaches 0.11 after about 2,500 iterations again.

The log overlap looks different for the CZ gate. After an initial drop of about 0.02 to 0.12, the training overlap slowly decreases until it reaches a value close to 0 after about 10,000 iterations. The training overlap takes a similar course. During the training process, it goes down to about 0.18 after starting at about 0.33.

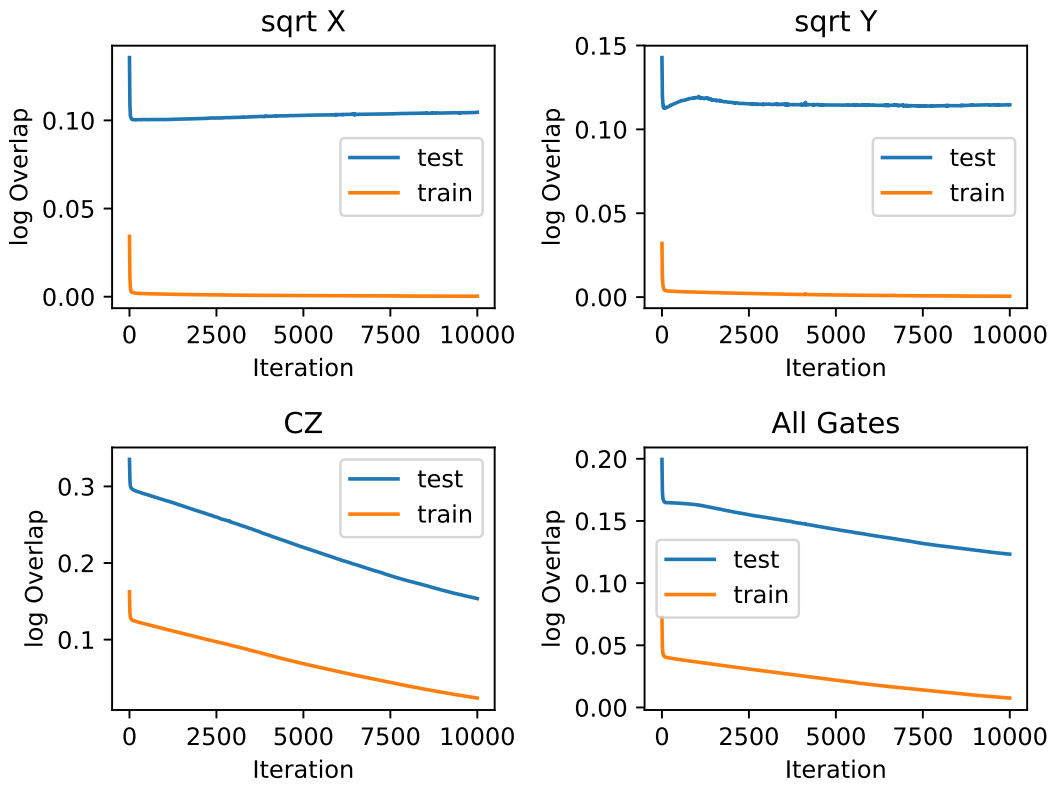


Figure 5.27: Training Overlap of AdaMax with Restarts Learned for 47 samples.

For 303 training samples, the course of the training overlap is very similar to the case of 47 samples. A difference can be observed in the test overlap, which is very close to the training overlap throughout the whole training process.

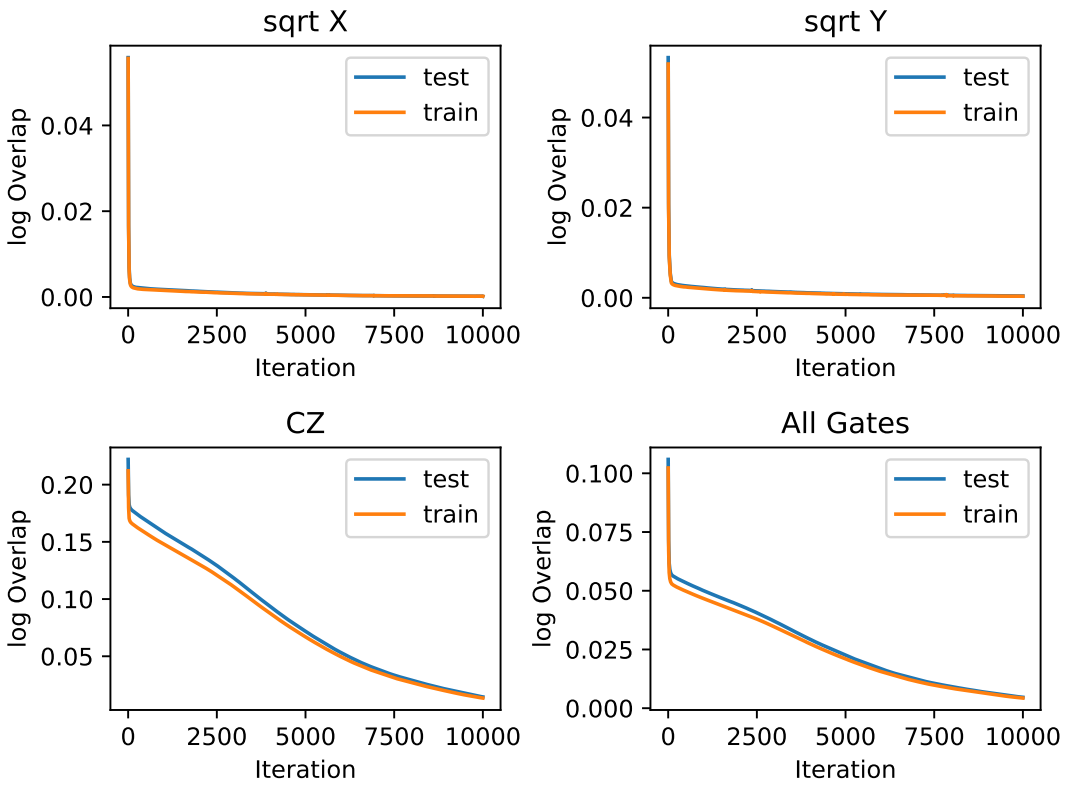


Figure 5.28: Training Overlap of AdaMax with Restarts Learned for 303 samples.

Bibliography

[1] Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. *Theory of Computing*, 9(4):143–252, 2013.

[2] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines*. *Cognitive Science*, 9(1):147–169, 1985.

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando Brandao, David Buell, Brian Burkett, Yu Chen, Jimmy Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Michael Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew Harrigan, Michael Hartmann, Alan Ho, Markus Rudolf Hoffmann, Trent Huang, Travis Humble, Sergei Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, Dave Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod Ryan McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin Jeffery Sung, Matt Trevithick, Amit Vainsencher, Benjamin Villalonga, Ted White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574:505–510, 2019.

[4] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland, and John M. Martinis. Logic gates at the surface code threshold: Superconducting qubits poised for fault-tolerant quantum computing, 2014.

[5] Robert Beals, Stephen Brierley, Oliver Gray, Aram W Harrow, Samuel Kutin, Noah Linden, Dan Shepherd, and Mark Stather. Efficient distributed quantum computing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 469(2153):20120686, 2013.

[6] Yoshua Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *corr abs/1502.04390*, 2015.

[7] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[8] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *in Proc. 25th Annual ACM Symposium on Theory of Computing, ACM*, pages 11–20, 1993.

[9] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14(6):595–600, Apr 2018.

[10] Giuseppe Carleo, Kenny Choo, Damian Hofmann, James E. T. Smith, Tom Westerhout, Fabien Alet, Emily J. Davis, Stavros Efthymiou, Ivan Glasser, Sheng-Hsuan Lin, Marta Mauri, Guglielmo Mazzola, Christian B. Mendl, Evert van Nieuwenburg, Ossian O'Reilly, Hugo Théveniaut, Giacomo Torlai, Filippo Vicentini, and Alexander Wietek. Netket: A machine learning toolkit for many-body quantum systems. *SoftwareX*, page 100311, 2019.

[11] Giuseppe Carleo, Yusuke Nomura, and Masatoshi Imada. Constructing exact representations of quantum many-body systems with deep neural networks. *Nature communications*, 9(1):1–11, 2018.

[12] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017.

[13] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. Open quantum assembly language, 2017.

[14] Dong-Ling Deng, Xiaopeng Li, and S Das Sarma. Quantum entanglement in neural network states. *Physical Review X*, 7(2):021021, 2017.

[15] Guillaume Desjardins, Aaron Courville, Yoshua Bengio, Pascal Vincent, and Olivier Delalleau. Parallel tempering for training of restricted boltzmann machines. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 145–152. MIT Press Cambridge, MA, 2010.

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[17] Richard P Feynman. Simulating physics with computers. *Int. J. Theor. Phys*, 21(6/7), 1982.

[18] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. pages 14–36, 01 2012.

[19] Josiah Willard Gibbs. *Elementary Principles in Statistical Mechanics: Developed with Especial Reference to the Rational Foundation of Thermodynamics*. Cambridge Library Collection - Mathematics. Cambridge University Press, 2010.

[20] John Gill. Computational complexity of probabilistic turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[21] Daniel Gottesman. The heisenberg representation of quantum computers, 1998.

[22] Aram W. Harrow and Richard A. Low. Random quantum circuits are approximate 2-designs, 2008.

[23] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

[24] Geoffrey Hinton. *Boltzmann Machines*, pages 132–136. Springer US, Boston, MA, 2010.

[25] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[26] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[27] Geoffrey E. Hinton and Terrence J. Sejnowski. Analyzing cooperative computation. In *Proceedings of the Fifth Annual Conference of the Cognitive Science Society, Rochester NY*, 1983.

[28] Bjarni Jónsson, Bela Bauer, and Giuseppe Carleo. Neural-network states for the classical simulation of quantum computing, 2018.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] A.Y. Kitaev, A. Shen, M.N. Vyalyi, and M.N. Vyalyi. *Classical and Quantum Computation*. Graduate studies in mathematics. American Mathematical Society, 2002.

[31] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[32] Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural computation*, 20(6):1631–1649, 2008.

[33] Enrique Martin-Lopez, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, and Jeremy L. O'Brien. Experimental realisation of shor's quantum factoring algorithm using qubit recycling, 2011.

[34] Guido Montufar. Restricted boltzmann machines: Introduction and review, 2018.

[35] Guido Montufar and Nihat Ay. Refinements of universal approximation results for deep belief networks and restricted boltzmann machines. *Neural computation*, 23(5):1306–1319, 2011.

[36] Charles Neill, Pedran Roushan, K Kechedzhi, Sergio Boixo, Sergei V Isakov, V Smelyanskiy, A Megrant, B Chiaro, A Dunsworth, K Arya, et al. A blueprint for demonstrating quantum supremacy with superconducting qubits. *Science*, 360(6385):195–199, 2018.

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

[38] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, and Robert Wisnieff. Leveraging secondary storage to simulate deep 54-qubit sycamore circuits, 2019.

[39] C. E. Porter and R. G. Thomas. Fluctuations of nuclear reaction widths. *Phys. Rev.*, 104:483–491, Oct 1956.

[40] John Preskill. Quantum computing and the entanglement frontier, 2012.

[41] Sebastian Ruder. An overview of gradient descent optimization algorithms., 2016. cite arxiv:1609.04747Comment: Added derivations of AdaMax and Nadam.

[42] John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1997.

[43] Henry Maurice Sheffer. A set of five independent postulates for boolean algebras, with application to logical constants. *Transactions of the American Mathematical Society*, 14(4):481–488, 1913.

[44] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, Oct 1997.

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

[46] S. Sorella. Green function monte carlo with stochastic reconfiguration, 1998.

[47] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071, 2008.

[48] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 01 1937.

[49] H.D. Zeh. On the interpretation of measurement in quantum theory. *Found. Phys.*, 1:69–76, 1970.