LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
CENTRUM FÜR INFORMATIONS- UND SPRACHVERARBEITUNG
STUDIENGANG COMPUTERLINGUISTIK

Bachelor's Thesis

in Computational Linguistics

at the Ludwig-Maximilians-Universität München

Faculty of Languages and Literature

Simulating Circuits in Quantum Natural Language Processing using Tensor Networks

submitted by
Adrian Maurice Mülthaler

Supervisors: M.Sc. Korbinian Staudacher, M.Sc. Florian Krötz and M.Sc. Jakob Murauer
Examiner: Dr. Axel Wisiorek
Work period: 26.03.2024 - 04.06.2024

Abstract

English version:

Quantum natural language processing deals with the implementation of natural language models on quantum hardware. It remains uncertain whether these models benefit from a quantum advantage or can be efficiently simulated on classical hardware. Tensor networks emerge as a promising tool in the efficient approximation of quantum states. This thesis explores the feasibility and potential advantages of employing tensor network simulation for quantum natural language processing. Using a matrix product state architecture, we investigate the complexity of simulating circuits obtained from the Categorical Compositional Distributional framework. Our results indicate an exponential complexity when these simulations are performed non-approximated. To analyze the impact of approximation on the performance of quantum natural language processing models, we introduce a binary classification task and train on two datasets, focusing on the difference between training with and without approximating the simulation. Our findings indicate that training with approximated simulation is possible, yet it exhibits greater instability and slower convergence.

German version:

Quantum natural language processing beschäftigt sich mit der Implementierung natürlicher Sprache auf Quantenhardware. Es ist noch unsicher, ob diese Modelle einen Quantenvorteil haben oder ob sie effizient durch klassische Computer simuliert werden können. Tensor-Netzwerke erweisen sich als vielversprechendes Werkzeug zur effizienten Approximation von Quantenzuständen. Diese Arbeit untersucht die Realisierbarkeit und potenziellen Vorteile der Verwendung von Tensor-Netzwerk-Simulationen für Quantum Natural Language Processing. Unter Verwendung eines Matrix Product States Simulators untersuchen wir die Komplexität der Simulation von Quanten-Schaltkreisen welche vom Categorical Compositional Distributional Framework generiert werden. Unsere Ergebnisse deuten auf eine exponentielle Komplexität hin, wenn diese Simulationen nicht approximiert werden. Wir untersuchen die Auswirkungen der Approximation auf ein binäres Klassifikationsproblem. Wir trainieren auf zwei Datensätzen und konzentrieren uns auf den Unterschied zwischen dem Training mit und ohne Approximation der Simulation. Unsere Ergebnisse zeigen, dass das Training mit approximierter Simulation möglich ist, jedoch zu einer langsameren und instabileren Trainingskonvergenz führt.

Contents

1. Introduction

The convergence of natural language processing (NLP) with quantum computing has ignited significant interest and speculation. Assertions have been made regarding the potential exponential computational advantages of quantum natural language processing (QNLP) over its classical counterparts [Zeng & Coecke, 2016]. However, the practical realization of this alleged quantum superiority confronts substantial challenges [Preskill, 2018].

1.1. Motivation

The Categorical Compositional Distributional (framework) (DisCoCat) introduced by Coecke, Sadrzadeh, and Clark [2010] for QNLP, claims to be quantum-native [Coecke, de Felice, Meichanetzidis, & Toumi, 2020]. This means that the authors believe they found a 'quantum' model for language, that would 'fit' on quantum hardware but have an exponential blow-up on classical hardware [Zeng & Coecke, 2016]. This model has potential, because it manages to combine distributional linguistics, based on statistical patterns of cooccurence of words, with the underlying linguistic structure of sentences.
While perfect quantum computers are unarguably exponentially difficult to simulate, the noisy intermediate-scale quantum (NISQ) hardware available at the moment suffers from decoherence with an exponentially decaying fidelity [Zhou, Stoudenmire, & Waintal, 2020]. This limits the degree of entanglement that can be reached and opens up the possibility of classical simulation.
There is a lot of excitement around tensor networks at the moment because they represent a method to approximate a quantum state by preserving only its most important properties. This way, certain classes of quantum systems can be simulated more efficiently [Biamonte & Bergholm, 2017].

1.2. Research Question

The quantum advantage in complexity presented by Zeng and Coecke [2016] exists under the assumption of the availability of quantum random access memory (qRAM) [Giovannetti, Lloyd, & Maccone, 2008]. qRAM would make it possible to store arbitrary quantum states, with memory calls having only linear complexity.
Due to the unavailability of this qRAM and fault-tolerant quantum gate hardware, DisCoCat currently relies on variational quantum circuits running on currently available NISQ devices.
Because it is still unsure when (or even if) qRAM will materialize, the question arises if the variational QNLP algorithms running on NISQ hardware really profit from a quantum advantage or if they can be efficiently simulated classically.
To address this, this thesis will

1. Analyze the tensor network simulation complexity of certain circuits from the DisCoCat framework

2. Analyze results of a simple classification task with different simulation fidelity.

The latter will show if a lower simulation fidelity (and with that a more efficient simulation) can already bring acceptable machine learning results, or if a high fidelity is needed for good results in QNLP.

1.3. Thesis structure

This thesis will first give a brief and partial introduction to quantum computing in Chapter 2 and to tensor network simulation in Chapter 3 in order to provide the reader with the necessary basics needed to understand the rest of the thesis. Further reading is required to build a comprehensive understanding of these fields. Then in Chapter 4 will be an introduction to the novel field of QNLP, more specifically to the DisCoCat framework, which is used in the experiment.
After presenting related work in Chapter 5, the thesis will explain the conducted experiments in Chapter 6. Our research is conducted utilizing the DisCoCat framework for creating quantum circuits, which are then simulated by a self-constructed tensor network simulator, which will be detailed in the first part of Chapter 6. In the second part, we analyze the complexity of quantum circuits used for QNLP, focusing on the increase in bond dimensions. The third part consists of our analysis on employing approximation for simulating circuits used to solve a binary classification task.
The thesis will finally culminate in Chapter 7 with the conclusion of the experiments.

2. Quantum Computation

Quantum computation makes use of unique properties from quantum mechanics, which potentially allows quantum computers to perform certain types of calculations exponentially faster than classical computers. It is a new field at the intersection of physics, mathematics and computer science. It offers a paradigm shift in the approach of computation on classical computers. This chapter will only discuss some key aspects of quantum computation, that are important for this thesis. For a more comprehensive introduction, readers are directed to the work of Nielsen and Chuang [2010] and Homeister [2022], upon which this chapter is primarily built.

2.1. Qubits

Just like a classical bit can be in a state of either 0 or 1, a quantum bit, or qubit, also possesses a state. The state of a qubit is normally written in the *Bra-Ket* (or *Dirac*) notation. Here, $|\psi\rangle$ (read *Ket*) represents a column vector. A qubit is represented by the column vector $\begin{pmatrix}\alpha\\\beta\end{pmatrix}$. The basis vectors $\begin{pmatrix}1\\0\end{pmatrix}$ and $\begin{pmatrix}0\\1\end{pmatrix}$ are written by convention as the binary number of the position of the 1 in the vector so $|0\rangle$ and, respectively $|1\rangle$.
So, the state of a qubit generally looks as follows:

$$|\psi\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix} = \alpha\begin{pmatrix}1\\0\end{pmatrix} + \beta\begin{pmatrix}1\\0\end{pmatrix} = \alpha|0\rangle + \beta|1\rangle$$

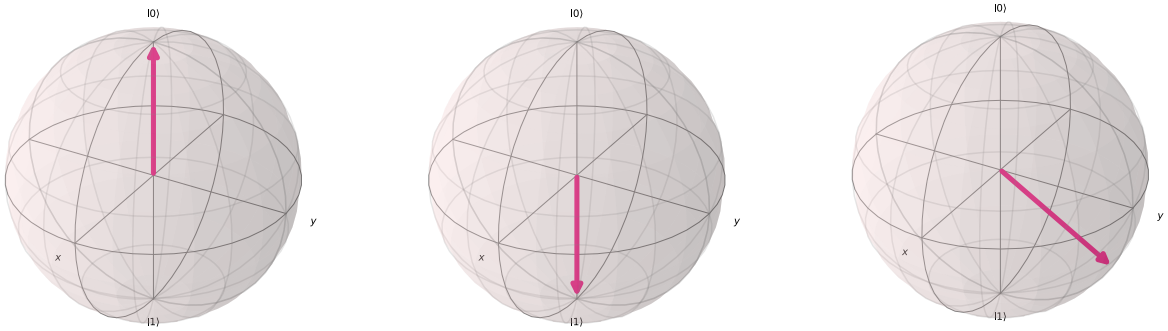
Where $\alpha, \beta \in \mathbb{C}$ are called the amplitudes of the state and $|\alpha|^2 + |\beta|^2 = 1$ must hold. This means that the squared absolute amplitudes give a probability distribution.

2.1.1. Superposition

Other than a classical bit, a qubit can also be in a state that is between $|0\rangle$ and $|1\rangle$ called a superposition. These "in-between states" cannot be read out directly from the quantum hardware. Instead, the probability of obtaining each basis state is given by the squared absolute amplitude. If the qubit is for example in the state of $|\psi\rangle = \frac{1}{2}|0\rangle + i\frac{\sqrt{3}}{2}|1\rangle$, then the probability of measuring "*0*" is 0.25 and the probability of measuring "*1*" is 0.75 because $|\frac{1}{2}|^2 = \frac{1}{4}$ and $|i\frac{\sqrt{3}}{2}|^2 = \frac{3}{4}$.

2.1.2. Bloch Sphere

Any quantum state, meaning a vector of two complex numbers, can be visualized on a *Bloch sphere*. This is a three-dimensional unit sphere with X, Y and Z axes. The basis states are on the Z-axis, with $|0\rangle$ on the north pole and $|1\rangle$ on the south pole. The state of one qubit can be rotated to any point on the surface of the Bloch sphere.
Figure 2.1 shows the two basis states and the example state from 2.1.1. The probabilities of measuring either "0" or "1" can be observed on the Bloch sphere by noting their proximity to the north or south poles. Our example state $|\psi\rangle = \frac{1}{2}|0\rangle + i\frac{\sqrt{3}}{2}|1\rangle$ is nearer to the south pole because the probability of measuring "*1*" was 0.75.



(a) $|\psi\rangle = |0\rangle$ (b) $|\psi\rangle = |1\rangle$ (c) $|\psi\rangle = \frac{1}{2}|0\rangle + i\frac{\sqrt{3}}{2}|1\rangle$

Figure 2.1.: Bloch sphere representation for 3 example states.

2.1.3. Quantum Register

When working with multiple qubits, the state of the system is represented by the tensor product $\otimes$. The tensor product between two vector spaces $\mathbb{C}^n$ and $\mathbb{C}^m$ represents a mapping of $\mathbb{C}^n \otimes \mathbb{C}^m \to \mathbb{C}^{n \cdot m}$. This means, that a quantum register consisting of n qubits is represented by a state vector with 2^n dimensions, because every qubit lies in $\mathbb{C}^2$.
For example, a quantum register containing two qubits, both in the state $|0\rangle$, is in the state

$$|\psi\rangle = |0\rangle \otimes |0\rangle = |00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

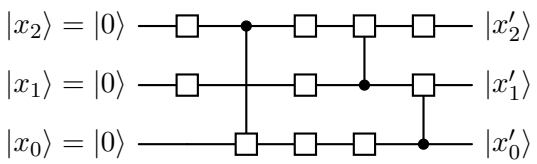
This is normally the initial state for a quantum register. Here $|00\rangle$ represents a column vector with a dimension of 4. The Ket-notation again is written as the binary position of the 1 in the corresponding vector.

2.2. Gates

Like in classical computing, manipulating the state of qubits is essential for computing to be possible. The operations performed on the qubits to alter their state are called gates. Mathematically, operations on qubits are matrices, which are required to be unitary. A unitary matrix is any matrix U that holds the equation $U^\dagger = U^{-1}$. The *dagger* operator $\dagger$ represents the complex conjugate and transpose of a matrix. So $U^\dagger = (U^*)^T$.
The unitary matrices are important for quantum computation because of their norm preserving property. This means that the vector representation stays on the surface of the Bloch sphere and that the sum of the probabilities remains 1.
Another important property is that these operations are reversible. This is a big difference to classical computation, where some operations are inherently irreversible.

2.2.1. Quantum Circuits

Quantum circuits can be represented by a diagrammatic language describing the operations performed on each qubit. There, each qubit is represented by a horizontal line and the operations are represented as boxes that these lines go through. Vertical lines, connecting a dot and a gate, represent controlled operations. The information flow is going from left to right.
The circuit width (the amount of horizontal lines) shows the number of qubits in the circuit, that are mathematically connected by the tensor product. The circuit depth (the length of the circuit) shows the amount of operations performed on the qubits. A quantum circuit generally looks like this:



Normally, the boxes would have a description on them, to depict what gate is applied.

2.2.2. Single Qubit Gates

Single qubit gates are operations performed on one qubit. Mathematically, they are $\mathbb{C}^{2\times 2}$ matrices. In the circuits, they are represented by single boxes. Each gate can be seen as a rotation on the Bloch sphere.
The most important gates are the *Pauli*-gates (X, Y, Z) and the *Hadamard* gate (see Table 2.1). The Hadamard gate is important, because it brings a qubit from a basis state to a superposition with equal probabilities for "*0*" and "*1*". E.g. $H \cdot |0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$

	X	**Y**	**Z**	**H**
Matrix	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Rotation	π around X-axis	π around Y-axis	π around Z-axis	$\frac{\pi}{2}$ around Y-axis followed by π around X-axis

Table 2.1.: Most common quantum gates

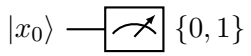
While each of the Pauli-gates are rotations by π on one of the axes of the Bloch sphere, it is possible to rotate by an arbitrary rotation angle θ. The three respective rotation matrices for each axis are:

$$R_X(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \quad R_Y(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \quad R_Z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}.$$

With these rotation gates, a qubit can be brought to any state on the Bloch sphere.

Measurement

Like mentioned earlier in 2.1.1 it is impossible to read out the state of a qubit in superposition. The only way of getting information of the qubit is by measuring it.



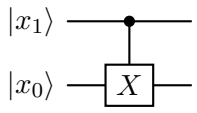
After the measurement, the state of $|x_0\rangle$ is irretrievably lost. The measurement generates an output of either "*0*" or "*1*" and $|x_0\rangle$ is now in one of the basis states in which the measurement took place, i.e. in $|0\rangle$ or $|1\rangle$ respectively.
Even though the original state is lost, it is possible to estimate the probabilities by repeated execution of the same circuit.

2.2.3. Multi-Qubit Gates

To enable qubit interaction, it is necessary to employ gates that operate on multiple qubits. The interplay between qubits is of course crucial for deploying quantum algorithms and with this quantum computation.

Controlled Gates

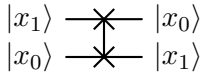
The most important multi-qubit gates are controlled gates. In a controlled gate, a *control qubit* controls the application of a gate on the *target qubit*. An example would be the Controlled-X gate, also called the *CNOT* gate:



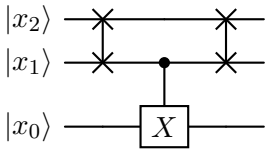
If $|x_1\rangle$ is in a basis state, the X-gate is only applied to $|x_0\rangle$ if $|x_1\rangle$ is $|1\rangle$. If $|x_1\rangle$ is in a superposition, only the amplitudes of $|1\rangle$ are important for the control. This will be explained further in 2.2.4.
The gate applied to the target can be any valid quantum gate. This will become important in 2.3, where controlled rotation gates are used.

SWAP Gate

A SWAP gate operates on two qubits and does nothing else than swapping their states. On a circuit, a swap gate looks as follows:



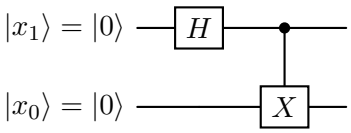
This is important because on real quantum hardware, not all qubits are connected, and they need to be switched next to each other in order to apply gates between them. An example of this would be:



Here, qubit $|x_2\rangle$ is switched next to $|x_0\rangle$ with the aim of performing a CNOT between them, after which $|x_2\rangle$ is switched back. This will also become important in Chapter 3.

2.2.4. Entanglement

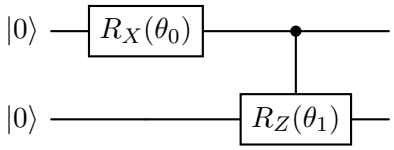
Entanglement is next to superposition, the second important phenomenon from quantum mechanics that can be used for computation. In 2.2.3 it was said, that if the control qubit is in superposition the amplitudes of $|1\rangle$ are important for the target gate. This means that the two qubits are now connected and cannot be described individually, i.e. as a tensor product between two individual qubits. The manipulation of the one qubit can influence the measure probabilities of the other.
One example for entanglement can be obtained by the following circuit



Here the Hadamard gate sets $|x_1\rangle$ in a superposition of equal probabilities, so to the state $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. After the CNOT gate, the qubit $|x_0\rangle$ has a probability of 0.5 of being flipped. The state of the two qubits now is $|x_1x_0\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. This state is one of the four *Bell states* (Appendix B). The state shows us that if qubit $|x_1\rangle$ is measured (destroying the superposition), the result for qubit $|x_0\rangle$ is also certain. So after $|x_1\rangle$ is measured, $|x_0\rangle$ is brought to a basis state. This happens instantaneously, even if the qubits are spatially separated.

2.3. Parameterized Quantum Circuits

Parameterized quantum circuits, also called variational circuits, are an essential part of *quantum machine learning (QML)*. This thesis will mainly deal with this kind of circuits. Their main characteristic is that they primarily employ rotation gates, which were presented in 2.2.2. Each rotation gate features an adjustable angle of rotation within the range of $[0, 2\pi]$. The following circuit is a simple example for a parameterized quantum circuit:



In QML, variational quantum circuits are normally used in a hybrid quantum-classical setup. The rotation angles are used as parameters in encoding data on a quantum device [various authors, 2023]. The quantum circuit is executed and measured, and the output is then used for a classical optimization algorithm, training the rotation parameters. This forms the training process.

Expressibility in the context of parameterized quantum circuits refers to the ability of expressing a wide range of quantum states on the Bloch sphere. The expressibility is given by the distribution uniformity of the states when sampling the parameters [Sim, Johnson, & Aspuru-Guzik, 2019].

Entangling capability is the average entanglement of the states a variational circuit produces. The Meyer-Wallach measure is used can be used to quantify this entanglement [Sim et al., 2019]. A circuit with a high expressibility and entangling capability is a good candidate for QML [various authors, 2023].

An **ansatz** (plural ansätze) refers to the specific method of choosing and arranging the rotation gates within a quantum circuit. They can be applied multiple times, with each block being called a layer. The ansätze that are used in this work can be found in Appendix A.

3. Tensor Network Simulation

This chapter provides a brief introduction into tensor networks, while a more comprehensive understanding is detailed in the work of Biamonte and Bergholm [2017].
Tensor networks offer a robust framework for representing and analyzing intricate quantum systems and high-dimensional data. They can provide an efficient approximation to certain classes of quantum states. They have an associated graphical language that makes them easy to describe and practical to work with.
As described in 2.1.3, an n-party quantum state is represented by a state vector with 2^n dimensions. The memory complexity grows exponentially with the amount of qubits. The objective of tensor network simulation is to find an alternative representation that avoids exponential blow up as the number of qubits increases.
A tensor is a multilinear map that can be represented as a multidimensional array of, in our case, complex numbers. The *order* of the tensor indicates the number of dimensions or indices, and the *shape* of the array indicates the number of elements in the respective dimensions. So a scalar is an order-0 tensor, because we need no indices. A vector is an order-1 tensor, because we need one index to find a specific component, and this vector can be of any shape (number of dimensions of that vector).
Tensor networks are a collection of tensors connected by tensor contractions. These contractions are obtained by summing over their indices. A matrix multiplication of two $\mathbb{C}^{N\times N}$ matrices, for example is a tensor contraction, that looks like this:

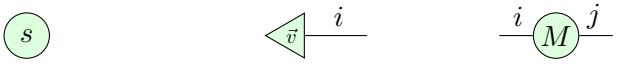
$$C_{i,j} = \sum_k A_{i,k} B_{k,j}. \tag{3.1}$$

Tensor networks can be arranged in many different architectures. Some of the best known applications are 1D Matrix Product States (MPS), Tensor Trains (TT), Tree Tensor Networks (TTN) and many more.

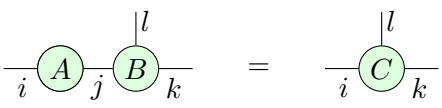
3.1. Tensor Diagram Notation

Tensors can be represented through a simple yet powerful graphical language, which was first introduced by Penrose [1971].
Here every tensor is represented by a node, with edges that represent its indices, order. Here are some examples:

Scalar: Vector: Matrix:



Contraction can be seen visually as pulling two nodes together through their connected edges. Remember, this is just the summation over the indices, that connect the two tensors. A tensor contraction of the form $\sum_j A_{i,j} B^l_{j,k} = C^l_{i,k}$ can be represented as:



When two or more disconnected edges are adjacent in the same diagram, they are multiplied together using the tensor product. This observation serves to evoke the earlier introduced

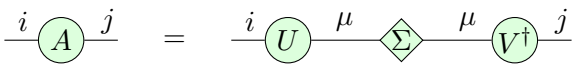
quantum circuit model from Chapter 2.2.1. Essentially, this model is a restricted subclass of tensor networks, where the qubit input is a vector, the gates are matrices and the layers are connected by the tensor product.

3.2. Singular Value Decomposition

The *singular value decomposition (SVD)* is an important method of factorizing matrices widely used in machine learning and numerical simulation algorithms. This is the most common technique for decomposition in tensor networks [Rieser, Köster, & Raulf, 2023].
The SVD decomposes a matrix into three well-defined matrices: a diagonal matrix containing the singular values and two unitary matrices. The equation for SVD of a matrix $A_{i,j}$ is the following:

$$A_{i,j} = U_{i,\mu}\Sigma_{\mu,\mu}V^{\dagger}_{\mu,j} \tag{3.2}$$

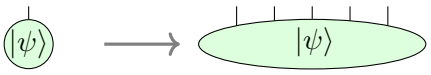
where U and V are unitary and Σ is real, non-negative and diagonal. Σ contains the singular values $\{\sigma_k\}_k$ of A on its diagonal, which are arranged in a decreasing order.
In the diagrammatic representation, the same SVD would look like:



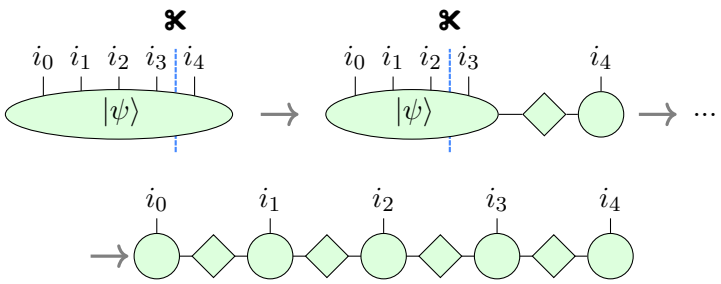
To determine the optimal rank approximation, the matrix Σ can be trimmed, by discarding the smallest singular values. The Eckart-Young-Mirsky theorem [Eckart & Young, 1936; Mirksy, 1960] says that this approximation is optimal.
When employing SVD on quantum states, the Schmidt decomposition [Nielsen & Chuang, 2010, p. 109] can be attained through appropriate mathematical transformations [Biamonte & Bergholm, 2017; Zhou et al., 2020]. This means that the singular values $\{\sigma_k\}_k$ correspond to the Schmidt coefficients. This implies that $\sum_k \sigma_k^2 = 1$ holds and that in some sense the number of non-zero singular values gives the 'amount' of entanglement [Nielsen & Chuang, 2010, p. 110]. An unentangled quantum state would have only one non-zero singular value $\sigma_0 = 1$.
The general strategy for tensor network algorithms with SVD is to either keep only a finite number $\chi \in \mathbb{N}$ of the singular values or to choose a cutoff value $\xi \in [0, 1]$.

3.3. Matrix Product States

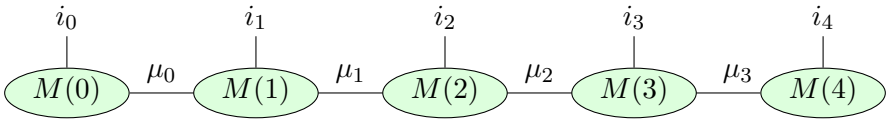
A matrix product state (MPS) is a linear-chain (one-dimensional) tensor network, representing a quantum state.
By reshaping a state vector of n dimensions, we can get an order-n tensor where each leg has dimension 2. Diagrammatically, this would look as follows for a state vector $|\psi\rangle \in \mathbb{C}^{32}$ representing 5 qubits:



In order to get to an MPS from this high order tensor, recursive application of the SVD is needed. One can start on either side by separating one leg from the rest. This process is repeated traversing the entire tensor.



Now, the diagonal matrices can be contracted into either adjacent tensor. This results in the classical representation for an MPS:



The quantum state is now described by N tensors, where N is the amount of qubits present in our system. The amplitudes for any $|i_0 i_1 i_2 ... i_{N-1}\rangle$ can be obtained by calculating the tensor contraction over those N tensors.

$$|\psi\rangle = \sum_{i_0 ... i_{N-1}} \sum_{\mu_0 \cdots \mu_{N-2}} M(0)^{i_0}_{\mu_0} M(1)^{i_1}_{\mu_0,\mu_1} M(2)^{i_2}_{\mu_1,\mu_2} ... M(N-1)^{i_{N-1}}_{\mu_{N-2}} |i_0 i_1 i_2 ... i_{N-1}\rangle \tag{3.3}$$

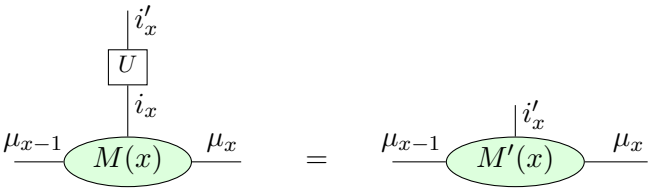
Here, the *physical* indices $i_n \in \{0, 1\}$ span the space for our two basis states ($|0\rangle$ and $|1\rangle$) while the *bond* indices $\mu_n \in \mathbb{N}$ provide a measure of the degree of entanglement [Zhou et al., 2020].

3.3.1. Single Qubit Gates

The application of a single qubit gate U to a qubit in the MPS is just done by applying the operation matrix to the tensor. The resulting tensor is calculated by

$$M'(x)^{i'_x}_{\mu_{x-1},\mu_x} = \sum_{i_x} U_{i'_x,i_x} M(x)^{i_x}_{\mu_{x-1},\mu_x}. \tag{3.4}$$

For better comprehension, this can be represented diagrammatically as the contraction of a gate with the qubit it is applied to.

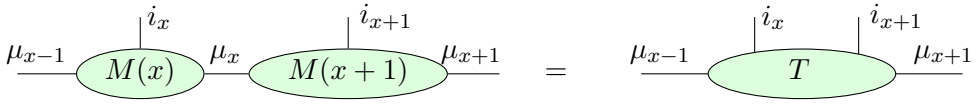


This is done by an exact calculation and does not affect the bond dimensions (μ_{x-1}, μ_x) [Zhou et al., 2020], which means that the tensor does not grow in number of entries by this calculation. In this step, no approximation is needed.

3.3.2. Multi Qubit Gates

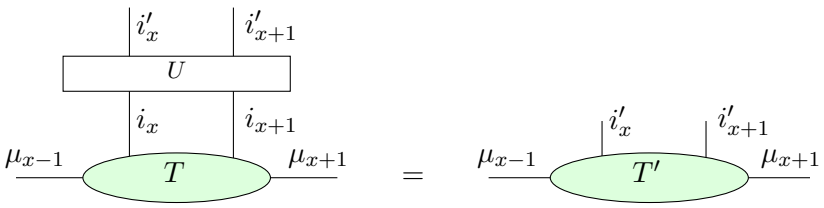
Although a 2-qubit gate on an MPS can only be applied to adjacent qubits, it is possible to bring their representing tensors next to each other by iteratively applying the SWAP gate, presented in Chapter 2.2.3.
The first step in a 2-qubit operation between the qubit at position x and $x+1$ is to contract the two tensors at these position over their bond dimension to form a 2-qubit tensor [Zhou et al., 2020].

$$T^{i_x,i_{x+1}}_{\mu_{x-1},\mu_{x+1}} = \sum_{\mu_x} M(x)^{i_x}_{\mu_{x-1},\mu_x} M(x+1)^{i_{x+1}}_{\mu_x,\mu_{x+1}} \tag{3.5}$$

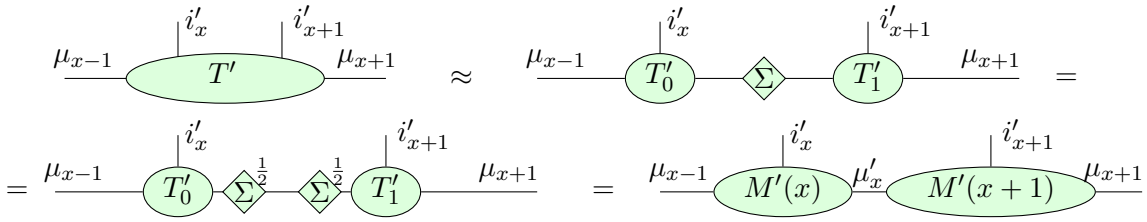


The operation matrix (which for a 2-qubit gate normally is written as a four by four matrix) is now rewritten to an order-4 tensor. This operation tensor is then connected to the physical indices of the contracted tensor T.

$$(T')^{i'_x,i'_{x+1}}_{\mu_{x-1},\mu_{x+1}} = \sum_{i_x,i_{x+1}} U^{i'_x,i'_{x+1}}_{i_x,i_{x+1}} T^{i_x,i_{x+1}}_{\mu_{x-1},\mu_{x+1}} \tag{3.6}$$



The 2-qubit tensor is then split with SVD into two 1-qubit tensors again. The matrix Σ can be contracted into either tensor. In this work, the convention is, that Σ is subdivided and contracted into both adjacent tensors, because we believe, this is beneficial for spreading the resulting error of the subsequently introduced approximations evenly.



If the gate produces entanglement, the number of resulting singular values after this operation is always $2\mu_x$, with μ_x being the bond dimension between the qubits before the application of the gate. If the gate does not produce entanglement between the qubits, the number of non-zero singular values will not be greater than the original μ_x.
This means that the bond dimensions double each time two qubits are entangled, if no approximation methods are used.

3.3.3. Approximation

When splitting the tensor T', some kind of the before mentioned approximation can take place, by only keeping a fixed amount χ of singular values or by allowing an error rate ξ.
By only keeping the χ highest singular values on each 2-qubit gate application, the maximal degree of entanglement is bounded. The bond dimensions in the system are capped at χ. Then, the overall cost of computing a 2-qubit gate scales as χ^3 [Zhou et al., 2020].
The other approximation method is by allowing an error rate $\xi \in [0, 1]$. As it was mentioned in 3.2, all the squared singular values add up to exactly 1. The smallest n singular values are discarded until $\sum_n {\sigma'_n}^2 > \xi$. The sum of the squared remaining singular values (approximately $1 - \xi$) is called the *fidelity* of the gate.

3.3.4. Advantage

As introduced, MPS is efficient for single qubit gates, as the computing complexity only scales linear with the circuit depth of 1-qubit gates. Contrarily, the complexity grows exponentially with the degree of entanglement of the qubits, in other words their bond dimensions. For each fully entangled qubit the bond dimension will be doubled, resulting in a complexity of $2^{\lfloor n/2 \rfloor}$. Yet this complexity can be reduced by truncating the space.
Although this brings much computing advantage, it initiates an error, which scales exponentially. Still, Zhou et al. [2020] show that for a modest $\chi = 64$, one can arrive at a gate fidelity of over 0.985 even at a circuit depth of 200.

4. DisCoCat

The **Cat**egorical **Co**mpositional **Dis**tributional model, short DisCoCat, was first introduced by Clark, Coecke, and Sadrzadeh [2008]. Its aim is to unify a distributional theory of meaning with a compositional theory of meaning.
Distributional refers to a vector space model of meaning [Schütze, 1998] in which similar words will be represented through similar vectors in a high-dimensional 'meaning-space'. Normalized vectors are similar if their inner product is close to one. Such methods can be seen as so-called bag-of-words methods, as they do not account for the structure and order the words appear in.
Compositional refers to a rule-based syntactic model in which the underlying structure words appear in is important. For this, any kind of grammar is suitable, which forms a compact closed category. An example of such a grammar is the *pregroup grammar*, which is introduced in 4.1.
This chapter will explain the DisCoCat model to a limited extent, without going into detail about the mathematical foundation, namely the *Category theory* the model relies on. For a more comprehensive understanding, refer to Coecke et al. [2020, 2010].

4.1. Pregroup Grammar

The pregroup grammar goes back to Lambek [1999, 2007]. It is a type of *categorical* grammar, which is an attempt to describe the structure of natural language and its syntax by assigning categories to the words.
A pregroup is a partially ordered monoid[1] $(P, 1, \leq, \cdot, -^r, -^l)$, in which each element $p \in P$, called *basic type*, has a left adjoint p^l and a right adjoint p^r. The right and left adjoint satisfy

$$p^l \cdot p \leq 1 \leq p \cdot p^l \qquad \text{and} \qquad p \cdot p^r \leq p^r \cdot p$$

In the pregroup algebra, the two kinds of adjoint act as left and right inverses under the multiplication of basic types. This means that the juxtaposition of adjacent adjoint types causes

$$p^l \cdot p \to 1 \to p \cdot p^l \qquad \text{and} \qquad p \cdot p^r \to 1 \to p^r \cdot p.$$

Here, each left-hand side is called a *contraction* (because the types get reducted to the unit type 1) and each right-hand side is called an *expansion* [Meichanetzidis, Toumi, de Felice, & Coecke, 2023].
In order to formalize grammar of natural languages, one starts by defining a set of grammatical roles. Each basic type can be viewed as an atomic linguistic structure. For example, one could introduce the types: n as the noun type, s as the sentence type. Compound types can be built in order to represent more linguistic structures. An adjective, for example, is a structure that demands a noun at its right-hand side, after which it again acts as a noun. Hence, an adjective can be described as $n \cdot n^l$, which makes the contraction with a noun possible:

$$(n \cdot n^l) \cdot n \to n \cdot (n^l \cdot n) \to n \cdot 1 \to n.$$

Any sequence that gets reduced to an s type can be seen as a grammatical correct sentence. If we look for example at the sentence "*Alice loves black cats*" and we map every token to its corresponding type, $\{\text{Alice} : n, \text{cats} : n, \text{loves} : n^r \cdot s \cdot n^l, \text{black} : n \cdot n^l\}$, we can reduce the tag

[1] A partially ordered monoid is a tuple $(P, \leq, \cdot)$ where $(P, \leq)$ is a partially ordered set and $(P, \cdot)$ is a monoid with the identity element 1 [Clark et al., 2008].

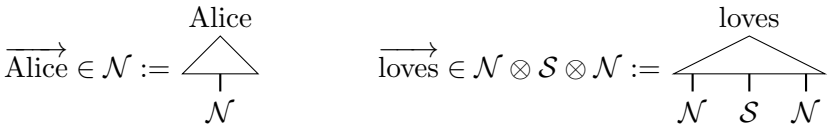
sequence like this:

$$\begin{aligned} \text{Alice} \quad \text{loves} \quad \text{black} \quad \text{cats} \\ n \cdot (n^r \cdot s \cdot n^l) \cdot (n \cdot n^l) \cdot n \rightarrow \\ n \cdot n^r \cdot s \cdot n^l \cdot n \cdot n^l \cdot n \rightarrow \\ (n \cdot n^r) \cdot s \cdot n^l \cdot n \cdot (n^l \cdot n) \rightarrow \\ 1 \cdot s \cdot (n^l \cdot n) \cdot 1 \rightarrow \\ 1 \cdot s \cdot 1 \cdot 1 = s \end{aligned}$$

This means that this is a grammatical correct sentence in our syntax.

4.2. The Quantum Model

For the purpose of combining the compositional model with the distributional, the meaning of words get embedded into vectors. The meaning of a noun, for example, is represented by a vector in the vector space $\mathcal{N}$. The meanings of transitive verbs like "loves" are represented in the space $\mathcal{N} \otimes \mathcal{S} \otimes \mathcal{N}$. We write these vectors in a diagrammatic notation, which, as already established in Chapters 2 and 3, is usual in quantum information.



The grammatical type reduction, given by the syntactic model of the pregroup grammar, constructs a linear map by "wiring up" the vectors with so-called *cups*. This process can be seen in Figure 4.1 [Zeng & Coecke, 2016].

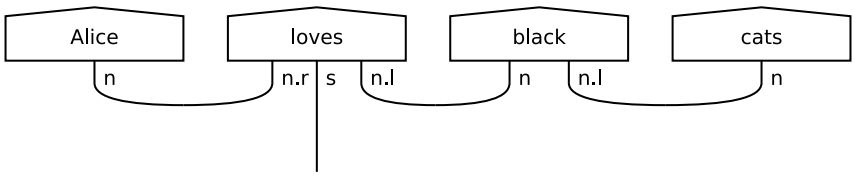
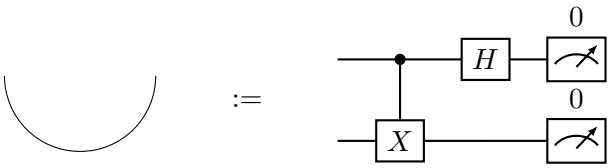


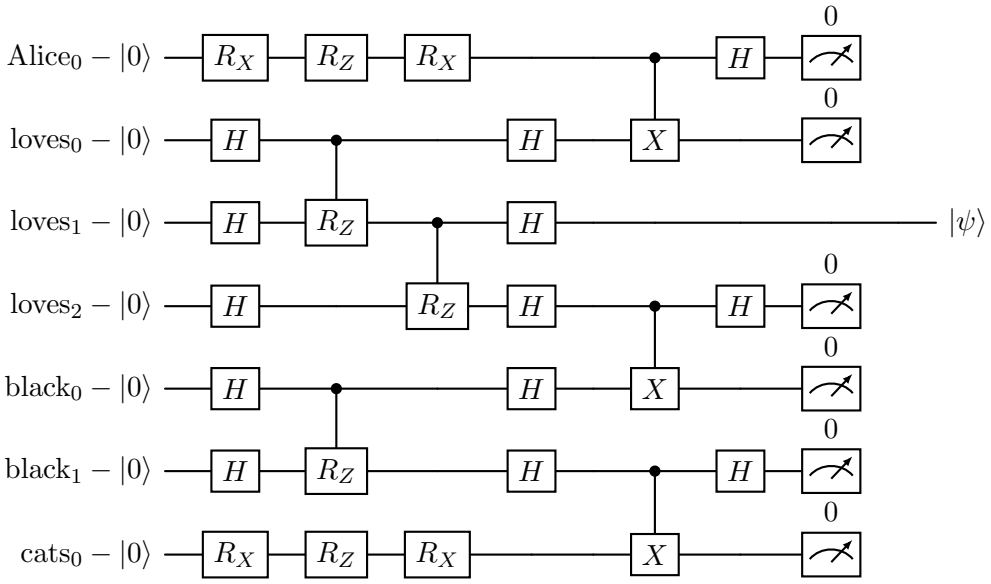
Figure 4.1.: String diagram for the sentence "*Alice loves black cats*".

In this case, pregroup grammar determines how the words are connected in a *string diagram*. As written by Coecke et al. [2020]: "Grammar is what mediates the flows of meaning between words". This constitutes the big advantage of this NLP model: word meanings are given by vectors, while the grammar controls how these vectors interact with each other, returning the meaning of the whole sentence.
When applying this model to quantum circuits, the cups are mapped to *Bell measurements*, which are the opposite of the Bell state presented in 2.2.4, since they form the identity when concatenated. It is crucial, that the Bell measurements need to be post selected: when running the circuits on quantum hardware, only those results can be used for optimization in which all the measurements emerging from these cups result in "0".

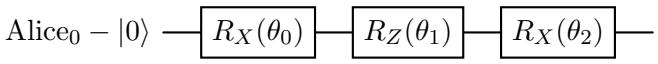


It is important to note that for the dimension of atomic types like $\mathcal{N}$, the number of used qubits can be chosen. The following circuit shows how a quantum circuit for the example sentence "Alice loves black cats" looks like. Here, every basic type is mapped to one qubit. On the single

qubit words, the meaning is encoded in three rotation gates. Also, one layer of the IQP[2] ansatz is used.



As one can see, the circuit outputs one qubit wire. This means the output vector $|\psi\rangle \in \mathbb{C}^2$ can be used for binary classification because it gives a probability distribution between $|0\rangle$ and $|1\rangle$. If one would choose a higher amount of qubits for the s type, a multiclass classification would be possible, as $\mathcal{S}$ would be higher dimensional.
This model is combined with quantum machine learning. The parameters for every rotation gate, are trained like in variational circuits by using the circuit outcome $|\psi\rangle$ with a classical optimizer. For example, the word "Alice" in the circuit above would have three parameters that can be learned:



Because the word "Alice" has only one output wire of the basic type n (4.1) and n is assigned only one qubit, the meaning of "Alice" is encoded in single qubit rotation gates. The amount of rotation gates for these 1-qubit cases, is one selectable hyperparameter.
If a word is assigned multiple qubits, the circuit depth is controlled by the ansatz and the amount of ansatz layers.
By modifying these hyperparameters (the amount of qubits for each atomic type, the ansatz, the number of ansatz layers and the number of single qubit rotation gates), one can adjust the amount of trainable rotation angles for each word.

[2]IQP: Instantaneous quantum polynomial, see Appendix A

5. Related Work

Although there has not been much research on tensor network simulation of QNLP circuits, there has been a considerable amount of research on these fields individually in recent years.

5.1. Tensor Network Simulation

The use of tensor networks for simulating multipartite quantum systems dates back to Vidal [2003]. In this work, the author demonstrated that employing tensor networks results in simulation complexity that grows linearly with the number of qubits and exponentially with the amount of entanglement. This indicates that a system with restricted entanglement can be efficiently simulated using tensor networks.
Since then, tensor networks have taken a central role in the representation of quantum wave functions, as they enable the efficient representation of specific classes of quantum states. This had the effect that many specially constructed sampling problems, originally believed to offer a significant complexity advantage when solved on quantum hardware, conversely could be computed efficiently on classical hardware.
An example of this is, when Pednault, Gunnels, Nannicini, Horesh, and Wisnieff [2019] managed to simulate a circuit in a matter of days, while Arute et al. [2019] claimed that they had achieved "quantum supremacy" on a task that would take 10,000 years on classical hardware. In their approach, the authors partition the quantum circuit into subcircuits, simulating each of them independently. This allows for greater parallelization and the possibility of storing some of the results on secondary storage. In order to apply entangling gates between different subcircuits, they used a method called *contraction deferral*, which made it possible to contract tensors, which are not connected by an edge in the tensor network.
A more recent example is the work of Tindall, Fishman, Stoudenmire, and Sels [2024]. With their tensor network, they were able to perform the efficient simulation of a kicked Ising quantum system much more accurate than on the 127 qubit quantum hardware. Their tensor network imitates the architecture of the "Eagle Processor", a heavy-hexagon quantum processor. They demonstrate that the use of *belief propagation* tensor networks is effective for solving many-body dynamics problems. When implementing belief propagation, the vertices in the graph share a communication channel over which they can pass messages. Each vertex stores information about adjacent nodes, called its "belief". At each time step, the nodes update their beliefs. For a more detailed explanation of the belief propagation process, refer to Leifer and Poulin [2008].

5.2. QNLP

Zeng and Coecke [2016] show how the DisCoCat model can be used to calculate sentence similarity on quantum hardware. The inner product of two quantum circuits' output vectors can be calculated in order to get a measure for their similarity.
A large part of the experiments conducted in this thesis is based on the work of Meichanetzidis et al. [2023] to compare their findings with ours. They created a small dataset (which is also used in this work, see Appendix D.1) and implemented a binary classification task on real quantum hardware. They generated their dataset by creating random sentences with the help of a context-free grammar. Although the data is synthetic, they labelled the sentences by hand, creating a semantically non-trivial classification task. They used the *simultaneous perturbation stochastic approximation* (SPSA) algorithm [Spall, 1998] as optimizer. This optimizer, which is also used in our work, estimates the gradients through stochastic methods. They ran their circuits on

NISQ devices, provided by IBMQ. By showing a decreasing loss on multiple iterations, this was the first paper to show that an NLP task can be performed on quantum devices.

5.3. NLP using Tensor Networks

In recent years, there has been significant development in employing tensor networks for probabilistic modeling of sequence data, such as text for NLP or DNA sequences for bioinformatics. Unlike this thesis, the following works did not focus on the simulation of quantum circuits, but rather on the direct implementation of sequences using tensor networks.
Miller, Rabusseau, and Terilla [2021] applied a uniform Matrix Product State (u-MPS) model for the task of probabilistic sequence modeling. An u-MPS is a tensor network where all cores of an MPS are tensors with identical shapes. They demonstrated that, despite its recurrent nature, this tensor network model can be processed in a highly parallel manner, unlike recurrent neural networks (RNNs) which struggle with parallelization. The authors introduced a novel approach, allowing the trained u-MPS to sample from a variety of conditional distributions defined by regular expressions. They evaluated their model on Tomita grammars, such as the parity of bit-strings (determining whether a bit-string contains an even or odd number of 1s). Their results showed that their model outperforms the baselines including HMMs, LSTMs, and Transformers.
Harvey, Yeung, and Meichanetzidis [2023] presented experimental results of binary classification of sequences. They defined 5 models, which implement tree-like, hierarchical tensor networks. For optimization, they used the AdamW algorithm. By choosing their tensors to be unitary, they were able to conduct experiments not only on classical hardware but also on a quantum computer. On both, they demonstrated the efficient implementation of their models for NLP datasets. They worked on two NLP datasets, one for detecting whether short news titles are clickbaits and one for sentiment analysis on Rotten Tomatoes movie reviews.

6. Experiments

The conducted experiments are divided into two parts. The first set of experiments, described in 6.2, investigates the simulation complexity of circuits obtained from the DisCoCat framework. More specifically, the bond dimensions of the tensor networks are examined. The second set of experiments, described in 6.3, investigates the trainability of the circuits, specifically in regard to approximating the simulation. For this purpose, a simple classification task is introduced.

6.1. Experimental Setup

To conduct our experiments, we need to establish a pipeline. First, a parser must analyze a given sentence to determine its grammatical structure and produce the corresponding string diagram. Next, this string diagram must be mapped to a quantum circuit, which lastly needs to be simulated with a tensor network. Each of these steps is detailed in the following sections.

6.1.1. Parser

To syntactically parse sentences, we make extensive use of the lambeq package [Kartsaklis et al., 2021]. There are two parsers from the lambeq package that are used and compared in this work: *the Cups reader* and the *BobCat parser*.
The Cups Reader is a trivial parser. Each word is assigned the $s^r \cdot s$ type. This means each two adjacent words are connected by an s-wire. At the start of each sentence, a special "START" token of type s needs to be inserted, so that the s^r of the first word can be contracted. The output is then again of type s. This important for this work, as this grammar outputs a "tensor train", which is structurally similar to the MPS architecture of our MPS simulator. Because the information of the sentence is passed along each word, this will also be referenced as the *sequential* grammar model. In Figure 6.1 you can see how a string diagram for this model looks like.

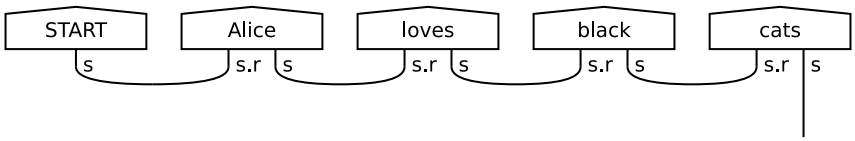


Figure 6.1.: String diagram from the Cups reader for the sentence "*Alice loves black cats*".

The BobCat parser is built upon the work of Clark [2021]. It uses a combination of Transformer-based models and a symbolic grammar to parse sentences. It employs Combinatory Categorial Grammar (CCG), a type-driven grammar formalism similar to pregroup grammar. Consequently, the BobCat parser can represent sentences in pregroup grammar, producing string diagrams like those shown in Figure 4.1.
This parser is used the most in this work, as it provides a non-trivial grammatical structure, that models the information flow of words in a sentence.

6.1.2. Creating Quantum Circuits

To create the quantum circuits, the lambeq package is used again. Setting specific hyperparameters is necessary for mapping the string diagrams to circuits.
In our experiments, we chose between 4 different **ansätze**: Instantaneous quantum polynomial (IQP), Strongly Entangled, Sim 14 and Sim 15 (Appendix A). For the ansätze the amount of

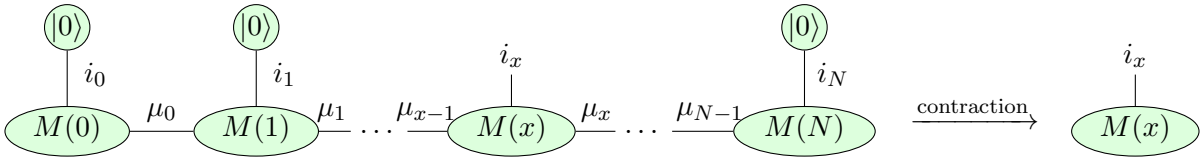
layers $\mathbf{n_{layers}}$ needs to be specified. Additionally, the number of rotation gates used for single qubit words $\mathbf{n_{single}}$ needs to be chosen.
The number of used qubits for each atomic type is defined: $\mathbf{q_s}$ for sentences, $\mathbf{q_n}$ for nouns and $\mathbf{q_p}$ for prepositional phrases. The number of qubits for the sentence type will be mostly 1, as the output can then be used for binary classification.
With these hyperparameters, the quantum circuits are created and can be passed to the simulator.

6.1.3. Simulator

The simulator is build using the TensorNetwork library [Roberts et al., 2019]. For the architecture, we used MPS, because it is a simple yet effective architecture that excels at handling near-neighbor entanglements. The TensorNetwork library relies on NumPy [Harris et al., 2020] for parallelization of large matrix operations. Additionally, for finding the most efficient contraction order, the opt_einsum [Smith & Gray, 2018] package is used.
Our MPS simulator is complete in the sense that all gates that occur in the resulting circuits from lambeq package can be directly applied. For controlled gates acting on non-adjacent qubits within the network, the SWAP gate is applied iteratively to bring these qubits next to each other. Instead of reversing the SWAP operations to return the qubits to their original positions, we update the qubit positions in our mapping. Subsequent gates are then applied to the qubits at their new positions. This approach eliminates unnecessary SWAP gates.
In our simulator, we implemented the possibility of truncating the SVD. For this, either the allowed truncation error $\boldsymbol{\xi}$ or the maximum number of singular values $\boldsymbol{\chi}$ can be specified. This allows for control over the accuracy of the simulation.
A series of tests are deployed for the simulator to ensure correct execution of circuits. This involves executing many small circuits and comparing their results with the expected outcomes.

Measurement

When implemented naively, the measurement gate is costly in the MPS simulator. Measuring one qubit has an effect on the entire system. This means the information must be passed along the entire circuit in order to renormalize the system. This entails many 2-qubit operations. As mentioned in 4.2 all cups are mapped to post-selection of "0". This means all qubits that are not part of the output wire need to be measured, which accounts for many measurements.
To address this, we implemented a method of post-selection which is only possible, when simulating. Instead of applying the measurements, we memorize which qubits should have been measured. When contracting the tensor network at the end to obtain the amplitudes (such as in Equation 3.3), we set all the physical indices of these qubits to $i = 0$. We do this by connecting a two-dimensional $|0\rangle$ vector to their corresponding tensor. This amounts to choosing the "0-layer" of those qubits. The diagram below shows this process for an example where all qubits except the one qubit at position x need to be measured and post-selected.



After the contraction, the amplitudes for our result qubits remain. These need to be normalized, as their squared absolute values do not add up to 1. The "loss" of amplitudes and need for renormalization can be interpreted as discarding results where not all post-selection measurements resulted in "0" on real quantum hardware.
To verify this approach, tests were conducted, confirming that the results from the "classical" measurement method are consistent with those obtained using our approach.
After the post-selection, the simulator returns a probability distribution for 2^{q_s} classes, thus completing the pipeline.

6.2. Analyzing Simulation Complexity

The first part of the experiments focuses on the simulation complexity of circuits with varying numbers of layers and qubits. As primary metric, we look at the maximum bond dimensions of the tensor networks during execution of the circuit.
For this task, we selected three example sentences. One is a short sentence commonly used as an example in QNLP, while the other two are longer sentences extracted from real-world contexts. The sentences are as follows:

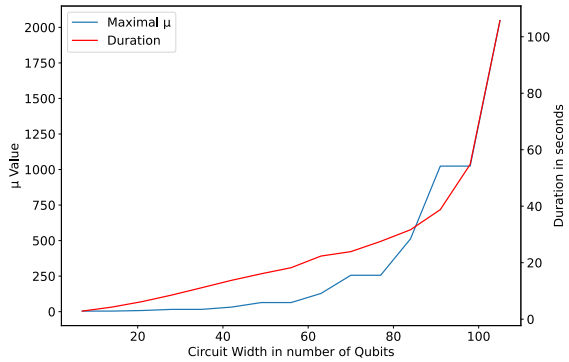
1. "Alice loves Bob."
2. "A quantum computer is a computer that takes advantage of quantum mechanical phenomena."[1]
3. "All animals are equal, but some animals are more equal than others."[2]

The string diagrams corresponding to the pregroup parse of these sentences can be seen in Appendix C.
The goal of this analysis is to see how much the maximum singular values increase with increasing circuit width and depth. This is done for both the sequential and the pregroup grammar model. To see the real increase of bond dimensions, the circuit simulations are performed without approximation, meaning that at every SVD all the singular values are kept. As discussed in 3.3.4, this implies that the bond dimensions are expected to grow exponentially with the number of entangled qubits.
All experiments in this section were conducted on the "Atos Quantum Learning Machine", provided by the Leibniz-Rechenzentrum. This hardware operates with 384 Intel(R) Xeon(R) Platinum 8260L cores, of which a maximum of ten were used simultaneously for each experiment. It is equipped with 6.5 TB of RAM.

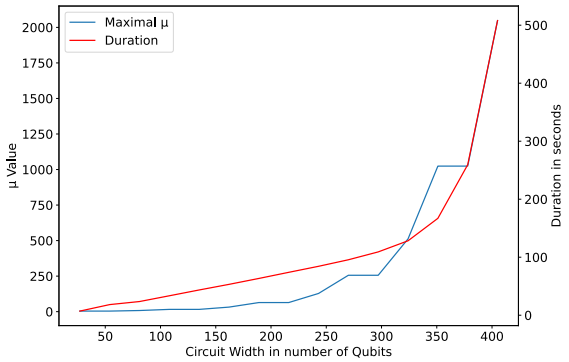
6.2.1. Results

In Figure 6.2, the results using the Cups reader are presented. We use one layer of the IQP ansatz. We observe a steep increase in both the maximum number of singular values μ and the simulation time across all three example sentences. These values were obtained by running the simulation with a progressively increasing q_s from 1 to 15. When applying more ansatz layers, the trend remains similar, but the μ values become even higher.
We did the same using the BobCat parser, and the results are shown in Figure 6.3. In this setup, the number of qubits for all basic types are increased by one at each step. Note that sentence 1 was increased to 15 qubits per basic type, while sentence 2 was only increased to 6 qubits and sentence 3 was only increased to 5 qubits per atomic type. The next increment was not calculated due to prohibitively long simulation times and memory usage rising to the hundreds of gigabytes.
To analyze the impact of circuit depth, we used sentence 3 as an example and progressively increased the number of ansatz layers, keeping the number of qubits for atomic types fixed at 3. This was done using the BobCat parser and the IQP ansatz. The results are presented in Figure 6.4.
To further investigate the bond dimensions for increasing circuit depth within the sequential syntax model, we compared multiple ansätze. This was done because the IQP ansatz employs only next-neighbor 2-qubit gates, whereas the Strongly Entangled ansatz or the Sim 14 ansatz incorporate more distant entanglements (see Appendix A). When using the IQP ansatz, the maximum singular values increases only from 1 layer to 2 layers, after which it remained constant. The Strongly Entangled ansatz led to fluctuating μ_{max}-values, while the Sim 14 ansatz showed constant μ_{max}-values. These results are illustrated in Figure 6.5.

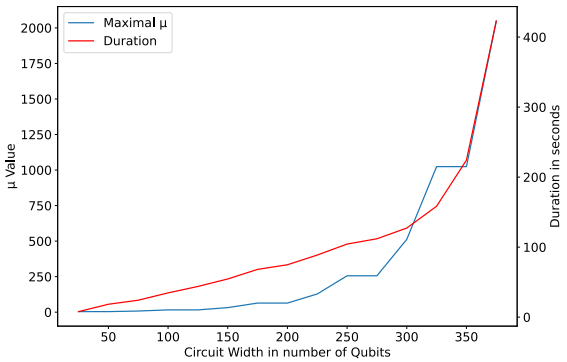
[1] Taken from https://en.wikipedia.org/wiki/Quantum_computing
[2] Taken from George Orwell's "Animal Farm"



(a) Sentence 1



(b) Sentence 2

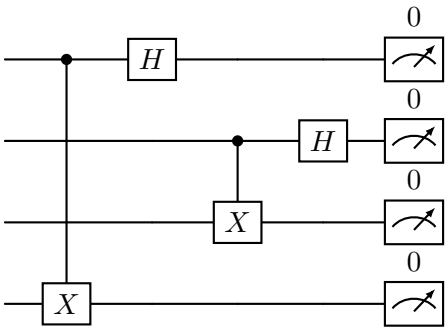


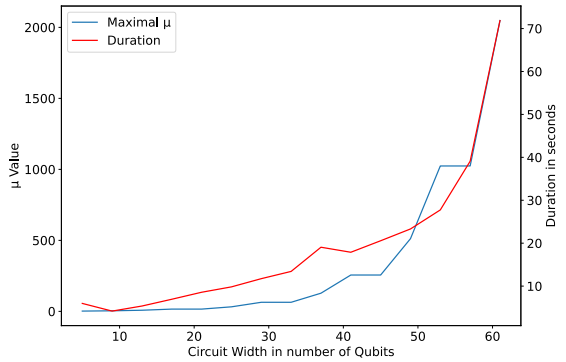
(c) Sentence 3

Figure 6.2.: The increase in μ-values and simulation duration with increasing circuit depth. Here we used the sequential grammar and 1 layer of the IQP ansatz.

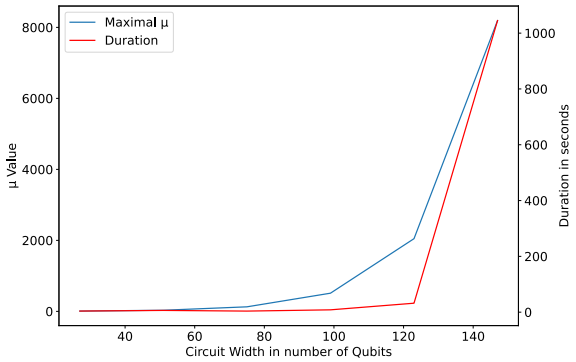
6.2.2. Discussion

The results reveal a strong correlation between the amount of used qubits per atomic type and the simulation complexity. In all cases, for pregroup grammar or sequential grammar, increasing the number of qubits per atomic type results in an exponential increase in both the maximum singular value and simulation time. This suggests that a great circuit width, arising from a higher number of qubits per basic type and longer, more complex sentences, become exponentially more challenging to simulate with our non-approximated method. Although this result was to be expected, we find it notable that even the sequential grammar model, which we thought would be very suitable for MPS simulation, shows this exponential increase in complexity. We believe this complexity increase when using the sequential grammar is caused by the Bell measurements, produced by the cups in the string diagrams. A cup between two 2-qubit words looks like this on a quantum circuit:

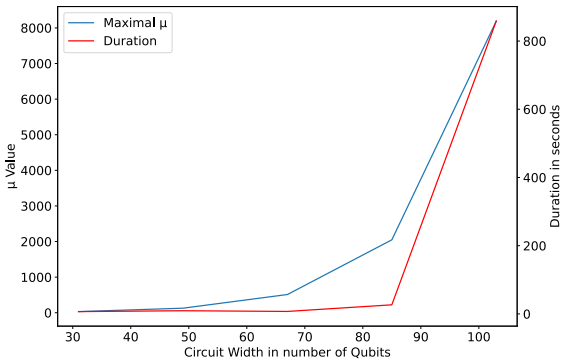




(a) Sentence 1



(b) Sentence 2



(c) Sentence 3

Figure 6.3.: The increase in μ-values and simulation duration with increasing circuit depth. Here we used the pregroup grammar and 1 layer of the IQP ansatz.

This means that using $q_s = 2$ leads to one control gate, which needs 2 SWAP gates. The control gates resulting from a cup in the string diagram are connected in the following way on a circuit of n qubits:

$$(0, n-1), (1, n-2), (2, n-3), \ldots, (\frac{n}{2} - 1, \frac{n}{2}).$$

This means the overall number of SWAP-gates needed can be calculated by:

$$(n-2) + (n-4) + (n-6) + \cdots + 2 = \frac{n(n-2)}{4} \tag{6.1}$$

This indicates that the number of required SWAP operations grows quadratically with the circuit width and thus with q_s. Although the SWAP gates do not affect the rise in the maximum bond dimension directly, it does have an effect on the time complexity.
Nevertheless, to pinpoint the cause of the exponential increase in μ_{max}, further research is necessary.
When comparing between the sentences, the graphs in Figure 6.2 display almost the same pattern. Notably, all sentences reach a μ_{max} of 2048 when $q_s = 15$. This occurs despite the total number of qubits differing among the sentences: sentence 1 only has a circuit width of 105 qubits, while sentence 2 and sentence 3 have a circuit width of approximately 400 qubits. This indicates that, when using the sequential grammar model with the IQP ansatz, the exponential complexity growth is more closely correlated with q_s, rather than with sentence length or circuit width.
When examining the results of the pregroup grammar model shown in Figure 6.3, we observe a different pattern. The more complex sentences 2 and 3 show a μ_{max} of 8192 when using 6 or 5 qubits per atomic type, respectively. In contrast, the simpler sentence 1 only reaches a

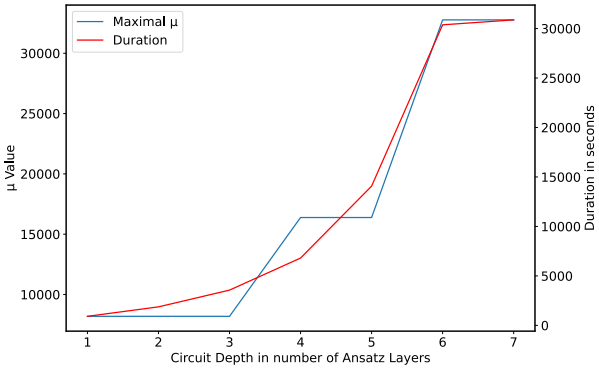
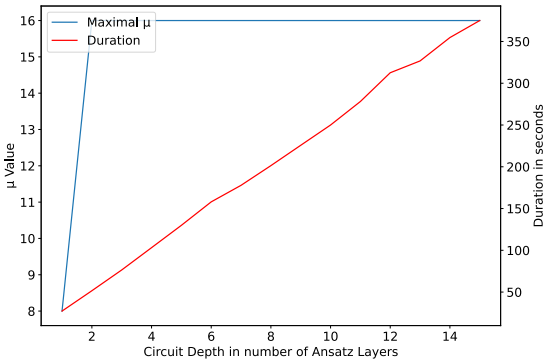
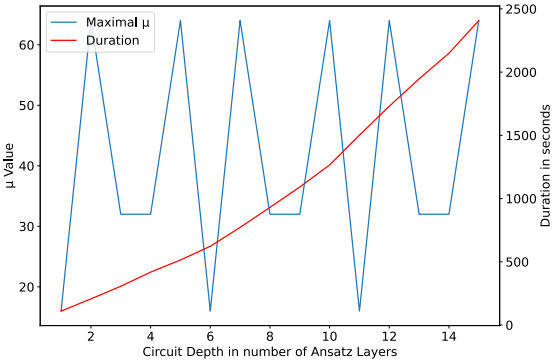


Figure 6.4.: The increase in μ-values and simulation duration with increasing circuit depth. We utilized sentence 3 and fixed the number of Qubits per atomic type to 3. Employing the BobCat parser, this resulted in a circuit width of 93 qubits. We used the IQP ansatz.

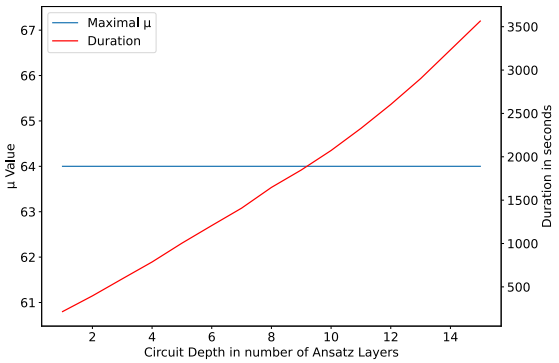
μ_{max} of 2048 again. The graph of sentence 1 resembles the one from Figure 6.2, as the pregroup grammar and sequential grammar exhibit similar adjacent connectivity for this sentence: the pregroup grammar has two cups and an outgoing s-wire from "loves" (Figure C.1), while the sequential grammar has three cups and one outgoing s-wire from "Bob". However, the other sentences with more complex pregroup grammar structures show a much steeper increase in μ_{max}, indicating that the pregroup grammar model has a higher simulation complexity than the sequential grammar model.
When using the pregroup grammar model, increasing the number of layers seems to have an exponential effect on simulation complexity. This is evident in Figure 6.4 where more IQP layers led to higher degrees of entanglement and consequently exponentially higher bond dimensions. In Figure 6.5 it is visible that with the sequential grammar model, the μ-value is not growing, but rather fluctuating or remaining constant, when the number of layers is increased within the interval $[1, 15]$. For all ansätze, the time complexity appears to be linear. This suggests that the number of layers has a small effect on simulation complexity of circuits implementing the sequential grammar model. However, further investigation is needed on circuits with greater width and depth and circuits using other ansätze to draw definitive conclusions.
In summary, when using a non-trivial grammar (which was the aim of the DisCoCat framework), the simulation complexity for an MPS simulator is growing exponentially with the circuit width and depth. It is an open question if a different architecture, which better imitates the grammatical connection of words in a sentence, yields better results.



(a) IQP ansatz



(b) Strongly Entangled ansatz



(c) Sim 14 ansatz

Figure 6.5.: The increase in μ-values and simulation duration with increasing circuit depth. We utilized sentence 3 and fixed the number of Qubits per atomic type to 3. Employing the sequential grammar, this resulted in a circuit width of 75 qubits. Each graph illustrates the results of increasing a different ansatz.

6.3. Analyzing Machine Learning Results

In this part of the experiment, we aimed to determine whether training is possible with approximated simulation. A series of tests with varying hyperparameters were conducted.
At the outset of each training, all parameters were initialized randomly within the range $[0, 2\pi]$.
For training, we used a virtual machine provided by the Munich Network Management Team, equipped with 45 GB RAM. We operated on ten vCPU cores.

6.3.1. Datasets

We used two datasets, that can be found in their entirety in Appendix D. Both are annotated for binary classification tasks.
The first dataset, referred to as "The Big Lebowski / Romeo & Juliet" dataset (or D.1 dataset), is from Meichanetzidis et al. [2023]. This dataset is chosen because it was already shown that training on this data is possible. As mentioned in Chapter 5, they created 52 sentences using a context-free grammar. They annotated these sentences by hand, labelling them either as false (0) or true (1). It is important to note that a perfect training accuracy is impossible for this dataset, as it contains two contradictory examples.
To see results for more complex data, we also present our own dataset, named "Animals/Plants" dataset (or D.2 dataset). This dataset was generated using GPT-3.5[3]. It contains 100 English sentences, categorized thematically as either animals (0) or plants (1). To ensure somewhat similar syntactic structures, we restricted the language model to use only nouns, intransitive verbs, transitive verbs and the pronouns who/that.

6.3.2. Cost Function

For the cost function, we chose binary cross-entropy loss. Although we considered mean squared error, cross-entropy loss is more commonly used for binary classification and demonstrated better results in some tests.
Given a dataset Δ, each sentence σ has an accompanying label y_σ. Predictions are made by parsing all sentences, mapping them to quantum circuits, and then simulating these circuits using the parameters $\vec{\theta}$.
The predicted label $\hat{y}_{\sigma,\vec{\theta}} \in [0, 1]$ for each sentence is computed by squaring the absolute amplitude of $|1\rangle$ from the simulation output.
Using the binary cross-entropy loss, our cost function for the entire dataset is defined as:

$$L(\vec{\theta}) = -\frac{1}{|\Delta|} \sum_{\sigma \in \Delta} [y_\sigma \cdot \log(\hat{y}_{\sigma,\vec{\theta}}) + (1 - y_\sigma) \cdot \log(1 - \hat{y}_{\sigma,\vec{\theta}})]. \tag{6.2}$$

By minimizing this cost function, the optimal parameters $\vec{\theta}^* = argmin_{\vec{\theta}} L(\vec{\theta})$ can be found.

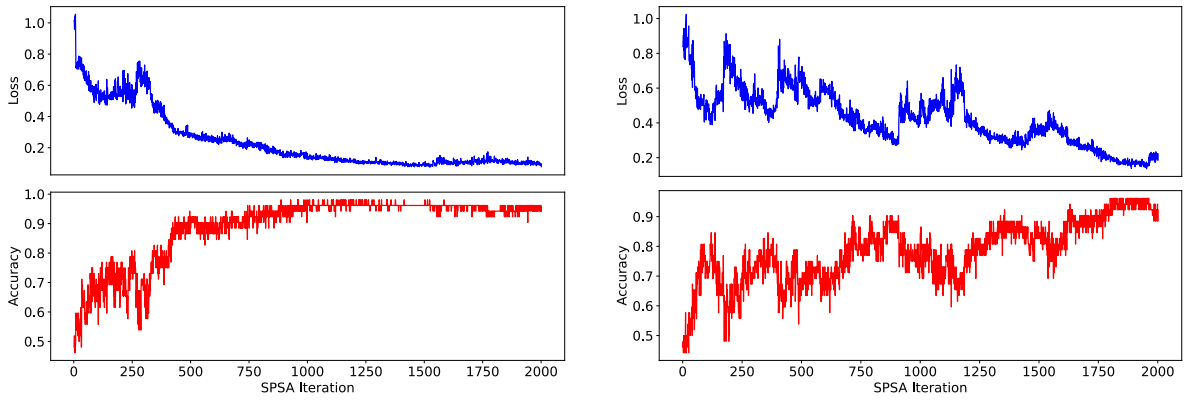
6.3.3. Optimizer

On real quantum hardware, obtaining parameter gradients is very costly due to the necessity of averaging noisy measurement outputs over multiple circuit runs to obtain the expectation value required for partial derivatives [Wiedmann et al., 2023]. Although tensor networks theoretically do not face this issue, we wanted to concentrate on the simulation of real hardware behavior. Therefore, to minimize the cost function, we use the gradient-free *simultaneous perturbation stochastic approximation (SPSA)* algorithm [Spall, 1998]. This method does not require the calculation of exact gradients. Instead, it uses stochastic approximation to estimate the gradients. SPSA perturbs all parameters and evaluates the cost function with the perturbed parameters. The difference in the loss of the perturbed parameters is then used to approximate the gradients. After some hyperparameter search, we found a **perturbation magnitude** of 0.06 with a **learning rate** of 0.15 to be particularly effective.

[3]OpenAI. (2024). ChatGPT (Apr 16 version) [Large language model]. https://chatgpt.com

6.3.4. Results

In Figure 6.6 we compare the training convergence of the D.1 dataset. On the left side, we show the results when training with exact simulation. The maximum number of singular values (μ_{max}) reached 32, occurring 58 times. On the right side, we show the results of training with the same hyperparameters but using an approximation where χ was set to 16. In this case, the maximum number of singular values was halved to 16, occurring 212 times. The non-approximated case achieved a training accuracy of 0.98, whereas the approximated case achieved 0.96. 2000 training iterations took about 55 hours for both cases.

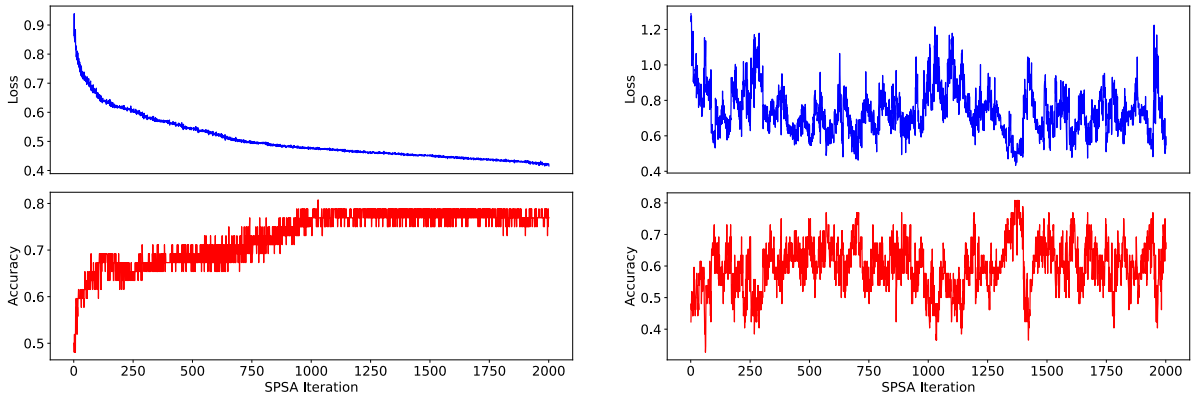


(a) Without approximation. $\mu_{max} = 32$.

(b) With approximation: $\chi = 16$.

Figure 6.6.: Convergence of cost function (Top) and Accuracy (Bottom) while training on dataset D.1. The used syntax is the BobCat parser. For the circuit creation, we used 7 layers of the IQP ansatz and the amount of qubits for the basic types are: $q_s = 1$ and $q_n = 2$. This results in 231 learnable parameters.

When reducing the amount of trainable parameters, for instance by setting q_n to 1 with $n_{single} = 2$, the training curve for accuracy flattens at a lower point, reaching a maximum training accuracy of 0.81. The training time shortens noticeable to 1 hour. Conversely, when increasing the amount of parameters, such as setting $q_n = 3$, resulting in 371 parameters, no clear training direction was observed, during the 139 hours of training. These results are illustrated in Figure 6.7.

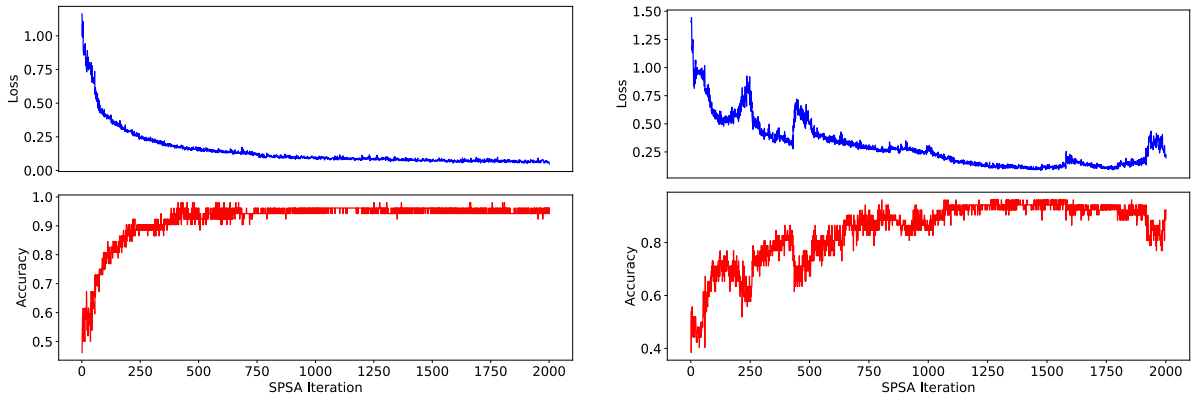


(a) $q_n = 1, n_{layers} = 1, n_{single} = 2$
$\rightarrow$ **25** parameters

(b) $q_n = 3, n_{layer} = 7$
$\rightarrow$ **371** parameters

Figure 6.7.: Convergence of cost function (Top) and Accuracy (Bottom) while training on dataset D.1. The left side shows training with one qubit for the atomic type n, while the right side used three qubits for the atomic type n. The simulations for both were exact.

We also conducted experiments, examining the impact of simulation approximation on training when using the Strongly Entangled ansatz, the results of which can be found at 6.8. These tests ran for about 21 hours. The non-approximated case reached a maximum training accuracy of 0.98, and the approximated case one of 0.96.



(a) Without approximation. $\mu_{max} = 64$.

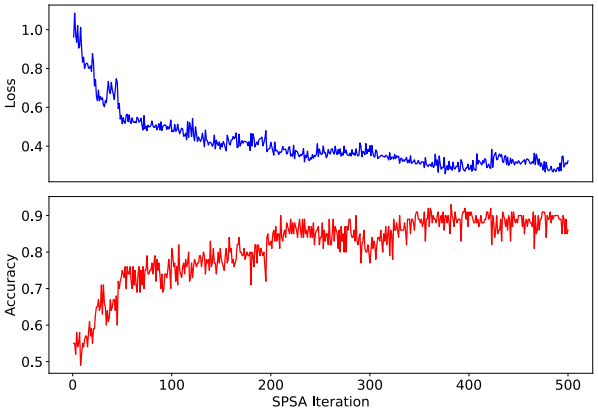
(b) With approximation: $\chi = 32$.

Figure 6.8.: Convergence of cost function (Top) and Accuracy (Bottom) while training on dataset D.1. The hyperparameters are the same as for Figure 6.6 except that here we use 1 layer of the Strongly Entangled ansatz. This results in 141 parameters that are learned.

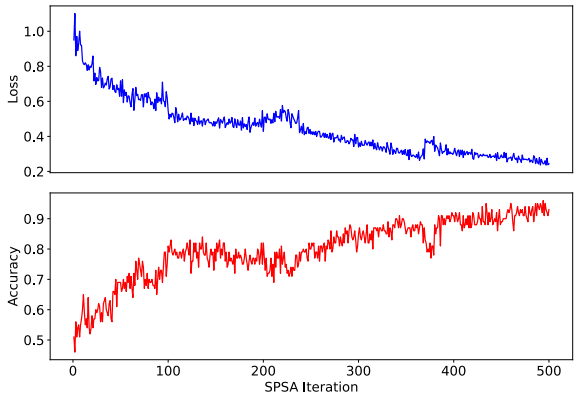
Dataset D.2 was more challenging to train, as it consists of more and longer sentences. This is why we had to limit the number of iterations to 500, all running for more than 180 hours. Using the BobCat parser, we employed 3 layers of the IQP ansatz and 2 qubits for each atomic type except s. In the non-approximated case, we observed a maximum bond dimension of $\mu_{max} = 256$, which occurred 68 times. From this we made two approximations: one with $\chi = 128$ and one with $\chi = 64$. The result of these trainings are depicted in Figure 6.9. While the non-approximated case reached a maximum training accuracy of 0.93, the $\chi = 128$ case reached 0.96 and the $\chi = 64$ case reached 0.95.
We also trained both datasets using the Cups reader. The results are found in 6.10. The training on dataset D.2 was interrupted after 881 SPSA iterations, when perfect training accuracy was achieved. This took 50 hours. The training on dataset D.1 took 8 hours and achieved a maximum training accuracy of 0.96.

6.3.5. Discussion

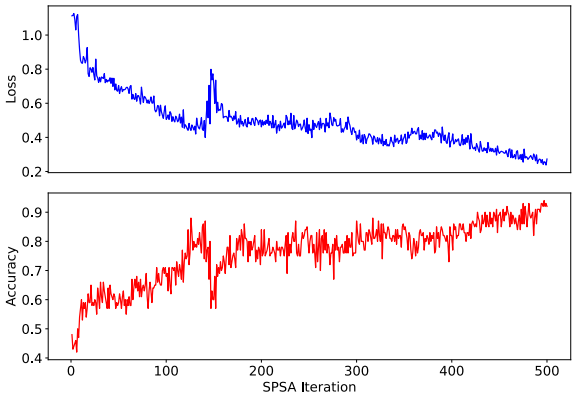
These results show that training is possible when the circuits are approximated. Especially results like 6.8, 6.6 or 6.9 show how training is more ineffective when we approximate, but ultimately reaches similar results. The training curve is more unstable and needs longer to reach its maximum, when we use approximation methods.
Comparing result 6.7a with 6.6a indicates that more parameters could lead to better training. However, when too many parameters are utilized, the training does not work effectively. This can be observed in 6.7b. There are many possible reasons for this: more qubits may require more ansatz layers, the learning rate or perturbation magnitude might have been chosen poorly, or the optimization landscape may be affected by *barren plateaus*. Barren plateaus pose a significant problem in QML, as the optimization landscape flattens exponentially with the number of qubits [Arrasmith, Cerezo, Czarnik, Cincio, & Coles, 2021].
As a result, we were unable to conduct experiments with larger hyperparameters. These results could have been particularly interesting, given that higher degrees of entanglement would likely lead to a more significant difference in simulation time between approximated and non-approximated iterations. For circuits of our size, our approximation methods do not significantly affect simulation time, because matrix operations of this smaller scale can easily be parallelized



(a) Without approximation. $\mu_{max} = 256$.



(b) With approximation: $\chi = 128$.



(c) With approximation: $\chi = 64$.

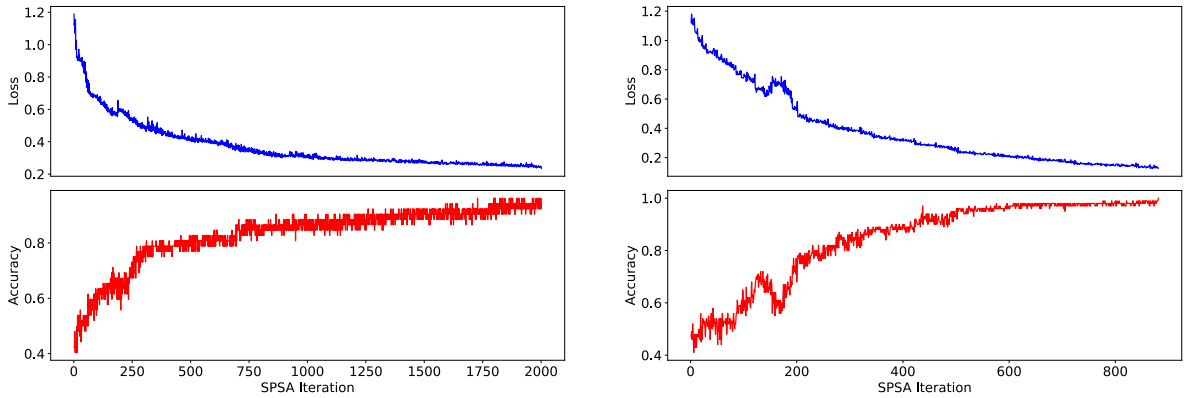
Figure 6.9.: Convergence of cost function (Top) and Accuracy (Bottom) while training on dataset D.2. The used syntax is the BobCat parser. For the circuit creation, we used 3 layers of the IQP ansatz and the amount of qubits for the basic types are: $q_s = 1$ and $q_n = 2$. This results in 1755 learnable parameters.

on one or multiple CPUs. Nonetheless, our results are meaningful, as they demonstrate the trainability using approximated MPS simulation. We assume that these approximation methods can be employed for greater hyperparameters, where the time complexity is more affected.

We also observed promising results when utilizing the Cups reader for training. Despite requiring more iterations to reach high accuracies than with pregroup grammar, it is significantly faster. Nevertheless, employing pregroup grammar appears to facilitate a steeper learning curve and faster training (in terms of number of SPSA iterations).

The training of dataset D.2 using the pregroup grammar demonstrates the feasibility of training even with more complex sentences. This experiment also indicates that a greater approximation generally leads to a slower training convergence. Training without approximation plateaued at a training accuracy of about 0.9 and a loss of about 0.35 already after 350 iterations. With an approximation of $\chi = 128$, the training reached a similar accuracy level later, after about 400 iterations, with a still slowly increasing trend afterward. With an approximation of $\chi = 64$, the training curve reached these high accuracies only in the last 50 iterations, reaching its maximum at iteration 497. Despite the slower convergence, the approximated simulations achieved an even higher maximum training accuracy after 500 iterations. We believe this is due to a fortuitous coincidence where the SPSA perturbation produced particularly favorable results in some iterations of the approximated cases, and we do not consider this an indication that approximated simulation generally achieves higher training performance.

Our results seen in Figure 6.10 attest that for a binary classification task, the sequential grammar yields highly effective results. This method was particularly successful on dataset D.2



(a) Training on dataset D.1
$n_{layers} = 7, n_{single} = 2$
$\rightarrow$ 78 parameters

(b) Training on dataset D.2
$n_{layers} = 3, n_{single} = 3$
$\rightarrow$ 744 parameters

Figure 6.10.: Convergence of cost function (Top) and Accuracy (Bottom). Here, the Cups reader is used on both datasets. This means only s-wires are used. q_s is set to 1 again.

achieving perfect training accuracy after only 881 iterations. After 500 iterations, it had already reached multiple training accuracies of 0.95 and above, indicating faster convergence than our experiments using the pregroup grammar, shown in Figure 6.9. In dataset D.2 the classification task relies more on vocabulary differences rather than subtle grammatical distinctions, which we believe is why the sequential grammar outperformed the pregroup grammar in this case. Conversely, in dataset D.1, the grammatical structure of the sentence is crucial for classification accuracy. For instance is “Dude annoys Walter” false, while “Walter annoys Dude” is true. This dependency on grammatical structure likely explains why the pregroup grammar led to much faster convergence in this dataset.
We did not conduct experiments on the approximation of circuits used in the sequential grammar, as the highest bond dimension, produced from this model with $q_s = 1$, was only 4.

7. Conclusion

In this concluding chapter, we give a brief summary of the thesis, discuss encountered limitations and provide the reader with an outlook to possible further research.

7.1. Summary

This thesis aimed to analyze the possibility of tensor network simulation for QNLP. To address this, we analyzed the simulation complexity of our own-build MPS simulator, especially focusing on the rise in bond dimension. Additionally, we conducted a binary classification task on two datasets, with which we analyzed the impact of approximating the quantum circuits, by limiting the number of singular values.
The simulation complexity appears to grow exponentially for increasing circuit width. This means that when using the DisCoCat framework to create parameterized quantum circuits from sentences, the time complexity of simulating these circuits is expected to grow exponentially with the number of used qubits for the atomic types. This is the case for our simple MPS architecture, a more sophisticated tensor network architecture may capture the syntactical structure of sentences better.
The results of the classification task suggest that approximating the simulation of the quantum circuits can yield results nearly on par with exact simulation. This finding potentially enables more efficient simulation of parameterized circuits in QNLP. However, our results also indicate, that approximation affects trainability and excessive approximation could possibly lead to poor training outcomes.
Furthermore, we conducted experiments using a sequential grammar model due to its structural similarity to the MPS architecture. This grammar demonstrated great trainability for binary classification tasks, with much shorter training durations. It is still unclear whether the use of complex grammars like the pregroup grammar offers substantial advantages, especially for more sophisticated NLP tasks.
We conclude that simulating circuits in QNLP using a simple tensor network architecture proves to be challenging. However, our results demonstrate that approximated simulations can still achieve good training convergence. This approach potentially reduces the simulation complexity to a manageable level, making the simulation more feasible.

7.2. Limitations

While our research provides valuable insights into the simulation of quantum circuits using tensor network methods and their application in QNLP, several limitations must be acknowledged.
Although we demonstrated an exponential increase in bond dimension with increasing circuit width, especially for the pregroup grammar, it remains uncertain what exactly causes it. We proposed that the cups from the string diagrams might cause the increase in bond dimensions when using sequential grammar, but this hypothesis needs to be validated through further simulation analysis.
We conducted our experiments with a restricted number of qubits per atomic grammar type due to difficulties in training, possibly caused by barren plateaus. The question of whether approximation circuits with much higher degrees of entanglement still results in good training results remains open.
Even though our approximated simulation demonstrated a good trainability for binary classification tasks, further investigation is needed to assess the impact of stronger approximations.

Excessive approximation of larger circuits with higher degrees of entanglement could lead to suboptimal training results. Our experiments were limited to a specific circuit width and a simple NLP task, and approximating larger circuits used for more complex tasks may produce different outcomes.
Due to our aim to simulate real quantum hardware, with which the direct calculation of gradients is very challenging, we confined ourselves to an optimization method that only approximates gradients, namely SPSA. Using exact gradient methods with tensor networks could significantly improve efficiency and convergence speed.
Moreover, our own-build simulator could benefit from further improvements. For instance, our MPS simulator applies all gates sequentially, which can be inefficient. There is potential to improve our simulator by parallelizing the application of gates where possible. Also, our implementation is limited to a maximum of 10 CPU cores working in parallel. Utilizing more CPU cores could significantly reduce the simulation time of circuits with high degrees of entanglement.

7.3. Future Work

There is a lot of possible further research in this area. While we focused on analyzing the simulation complexity using an MPS architecture, other tensor network architectures could produce different results. Although, architectures that imitate a general pregroup syntax might be challenging to develop, an analysis of recurring structures in the pregroup grammar could provide more information. Additionally, investigating the performance of other grammars with more regular structures could be beneficial. An example for this would be using the Stairs reader (also implemented in the lambeq package), which maybe could be efficiently simulated using tree tensor networks (TTNs).
In this work, we only explored binary classification tasks. Future work could focus on multiclass classification tasks or other NLP tasks and analyze the effect of approximation for these more complex training tasks.
The exponential increase in bond dimensions needs further investigation. Experiments should be conducted, that pinpoint which part of the circuits contribute the most to the fast increase: the ansatz layers or the Bell measurements. This investigation could help to find more efficient tensor network architectures or improved approximation methods.
Furthermore, the impact of circuit optimization could be examined. For instance, employing the ZX-calculus to optimize the quantum circuits before simulation could potentially enhance performance. This approach might reduce the overall complexity and improve the efficiency of simulating parameterized circuits in QNLP.

Bibliography

Arrasmith, A., Cerezo, M., Czarnik, P., Cincio, L., & Coles, P. J. [2021]. Effect of barren plateaus on gradient-free optimization. *Quantum*, *5*, 558. Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften. doi:10.22331/q-2021-10-05-558

Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., ... Martinis, J. M. [2019]. Quantum supremacy using a programmable superconducting processor. *Nature*, *574*[7779], 505–510. Number: 7779 Publisher: Nature Publishing Group. doi:10.1038/s41586-019-1666-5

Biamonte, J., & Bergholm, V. [2017, July 31]. Tensor networks in a nutshell. doi:10.48550/arXiv.1708.00006. arXiv: 1708.00006[cond-mat,physics:gr-qc,physics:hep-th,physics:math-ph,physics:quant-ph]

Clark, S. [2007]. Combining symbolic and distributional models of meaning.

Clark, S. [2021, September 28]. Something old, something new: Grammar-based CCG parsing with transformer models. doi:10.48550/arXiv.2109.10044. arXiv: 2109.10044[cs]

Clark, S., Coecke, B., & Sadrzadeh, M. [2008]. A compositional distributional model of meaning.

Coecke, B. [2020, February 28]. The mathematics of text structure. doi:10.48550/arXiv.1904.03478. arXiv: 1904.03478[quant-ph]

Coecke, B., de Felice, G., Meichanetzidis, K., & Toumi, A. [2020, December 7]. Foundations for near-term quantum natural language processing. doi:10.48550/arXiv.2012.03755. arXiv: 2012.03755[quant-ph]

Coecke, B., Sadrzadeh, M., & Clark, S. [2010, March 23]. Mathematical foundations for a compositional distributional model of meaning. arXiv. arXiv: 1003.4394[cs,math]. Retrieved March 13, 2024, from http://arxiv.org/abs/1003.4394

Eckart, C., & Young, G. [1936]. The approximation of one matrix by another of lower rank. *Psychometrika*, *1*[3], 211–218. doi:10.1007/BF02288367

Giovannetti, V., Lloyd, S., & Maccone, L. [2008]. Quantum random access memory. *Physical Review Letters*, *100*[16], 160501. doi:10.1103/PhysRevLett.100.160501. arXiv: 0708.1879[quant-ph]

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... Oliphant, T. E. [2020]. Array programming with NumPy. *Nature*, *585*[7825], 357–362. doi:10.1038/s41586-020-2649-2

Harvey, C., Yeung, R., & Meichanetzidis, K. [2023, August 15]. Sequence processing with quantum tensor networks. doi:10.48550/arXiv.2308.07865. arXiv: 2308.07865[quant-ph]

Homeister, M. [2022]. *Quantum Computing verstehen: Grundlagen – Anwendungen – Perspektiven*. doi:10.1007/978-3-658-36434-2

Hubregtsen, T., Pichlmeier, J., Stecher, P., & Bertels, K. [2021]. Evaluation of parameterized quantum circuits: On the relation between classification accuracy, expressibility, and entangling capability. *Quantum Machine Intelligence*, *3*[1], 9. doi:10.1007/s42484-021-00038-w

Kartsaklis, D., Fan, I., Yeung, R., Pearson, A., Lorenz, R., Toumi, A., ... Coecke, B. [2021, October 8]. Lambeq: An efficient high-level python library for quantum NLP. doi:10.48550/arXiv.2110.04236. arXiv: 2110.04236[quant-ph]

Lambek, J. [1999]. Type grammar revisited. In A. Lecomte, F. Lamarche, & G. Perrier [Eds.], *Logical aspects of computational linguistics* [pp. 1–27]. doi:10.1007/3-540-48975-4_1

Lambek, J. [2007]. From word to sentence: A pregroup analysis of the object pronoun who(m). *Journal of Logic, Language and Information*, *16*[3], 303–323. doi:10.1007/s10849-006-9035-9

Leifer, M. S., & Poulin, D. [2008]. Quantum graphical models and belief propagation. *Annals of Physics*, *323*[8], 1899–1946. doi:10.1016/j.aop.2007.10.001

Lorenz, R., Pearson, A., Meichanetzidis, K., Kartsaklis, D., & Coecke, B. [2023]. QNLP in practice: Running compositional models of meaning on a quantum computer. *Journal of Artificial Intelligence Research*, *76*, 1305–1342. doi:10.1613/jair.1.14329. arXiv: 2102.12846[quant-ph]

Meichanetzidis, K., Toumi, A., de Felice, G., & Coecke, B. [2023]. Grammar-aware sentence classification on quantum computers. *Quantum Machine Intelligence*, *5*[1], 10. doi:10.1007/s42484-023-00097-1. arXiv: 2012.03756[quant-ph]

Miller, J., Rabusseau, G., & Terilla, J. [2021, April 23]. Tensor networks for probabilistic sequence modeling. doi:10.48550/arXiv.2003.01039. arXiv: 2003.01039[quant-ph,stat]

Mirksy, L. [1960]. Symmetric gauge functions and unitarily invariant norms. *The Quarterly Journal of Mathematics*, *11*[1], 50–59. doi:10.1093/qmath/11.1.50

Murauer, J. [2023, July 31]. *Analyzing word predictions by quantum natural language processing* [Master, Ludwig-Maximilians-University Munich, Munich].

Nielsen, M. A., & Chuang, I. L. [2010, December 9]. Quantum computation and quantum information: 10th anniversary edition [Higher education from cambridge university press]. doi:10.1017/CBO9780511976667

Pednault, E., Gunnels, J. A., Nannicini, G., Horesh, L., & Wisnieff, R. [2019, October 22]. Leveraging secondary storage to simulate deep 54-qubit sycamore circuits. doi:10.48550/arXiv.1910.09534. arXiv: 1910.09534[quant-ph]

Penrose, R. [1971]. Applications of negative dimensional tensors. In *Combinatorial mathematics and its applications* [pp. 221–244]. Retrieved from https://homepages.math.uic.edu/~kauffman/Penrose.pdf

Preskill, J. [2018]. Quantum computing in the NISQ era and beyond. *Quantum*, *2*, 79. Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften. doi:10.22331/q-2018-08-06-79

Rieser, H.-M., Köster, F., & Raulf, A. P. [2023]. Tensor networks for quantum machine learning. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *479*[2275], 20230218. Publisher: Royal Society. doi:10.1098/rspa.2023.0218

Roberts, C., Milsted, A., Ganahl, M., Zalcman, A., Fontaine, B., Zou, Y., ... Leichenauer, S. [2019, May 3]. TensorNetwork: A library for physics and machine learning. arXiv. arXiv: 1905.01330[cond-mat,physics:hep-th,physics:physics,stat]. Retrieved May 25, 2024, from http://arxiv.org/abs/1905.01330

Schütze, H. [1998]. Automatic word sense discrimination. *Computational Linguistics*, *24*[1], 97–123. Place: Cambridge, MA Publisher: MIT Press. Retrieved May 21, 2024, from https://aclanthology.org/J98-1004

Sim, S., Johnson, P. D., & Aspuru-Guzik, A. [2019]. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Advanced Quantum Technologies*, *2*[12], 1900070. doi:10.1002/qute.201900070. arXiv: 1905.10876[quant-ph]

Smith, D. G. a., & Gray, J. [2018]. Opt_einsum - a python package for optimizing contraction order for einsum-like expressions. *Journal of Open Source Software*, *3*[26], 753. doi:10.21105/joss.00753

Spall, J. [1998]. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronic Systems*, *34*[3], 817–823. Conference Name: IEEE Transactions on Aerospace and Electronic Systems. doi:10.1109/7.705889

Tindall, J., Fishman, M., Stoudenmire, M., & Sels, D. [2024]. Efficient tensor network simulation of IBM's eagle kicked ising experiment. *PRX Quantum*, *5*[1], 010308. doi:10.1103/PRXQuantum.5.010308. arXiv: 2306.14887[quant-ph]

various authors. [2023]. *Qiskit textbook*. Github. Retrieved from https://github.com/Qiskit/textbook

Vidal, G. [2003, February 25]. Efficient classical simulation of slightly entangled quantum computations. doi:10.1103/PhysRevLett.91.147902. arXiv: quant-ph/0301063

Wiedmann, M., Hölle, M., Periyasamy, M., Meyer, N., Ufrecht, C., Scherer, D. D., ... Mutschler, C. [2023, September 17]. An empirical comparison of optimizers for quantum machine learning with SPSA-based gradients. In *2023 IEEE international conference on quantum computing and engineering (QCE)* [pp. 450–456]. doi:10.1109/QCE57702.2023.00058. arXiv: 2305.00224[quant-ph]

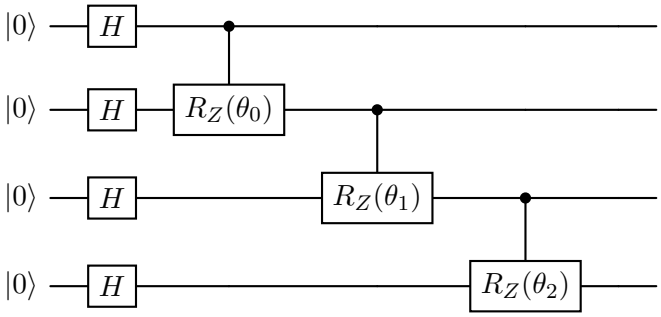
Zeng, W., & Coecke, B. [2016]. Quantum algorithms for compositional natural language processing. *Electronic Proceedings in Theoretical Computer Science*, *221*, 67–75. doi:10.4204/EPTCS.221.8. arXiv: 1608.01406[quant-ph]

Zhou, Y., Stoudenmire, E. M., & Waintal, X. [2020]. What limits the simulation of quantum computers? *Physical Review X*, *10*[4], 041038. Publisher: American Physical Society. doi:10.1103/PhysRevX.10.041038

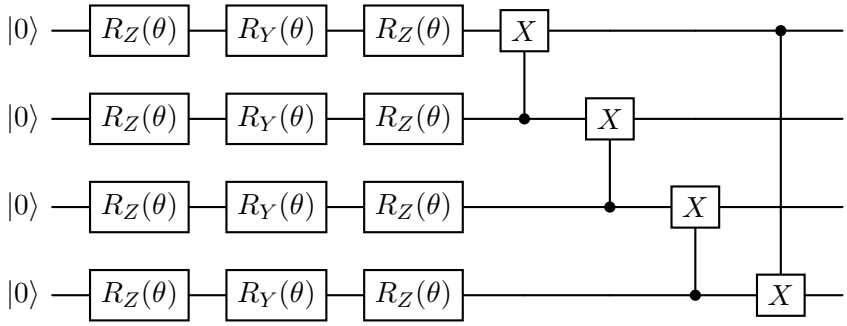
A. Ansätze

Here, the ansätze that are used in this work are shortly presented. Each ansatz is shown as an example of 1 layer on a 4-qubit circuit.

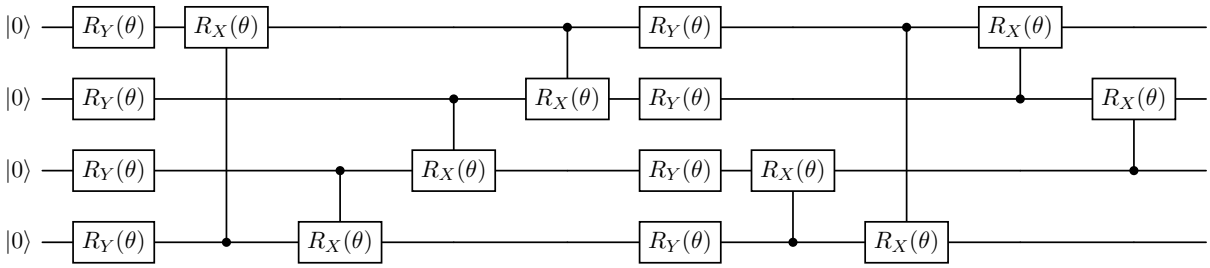
Instantaneous Quantum Polynomial (IQP) Ansatz



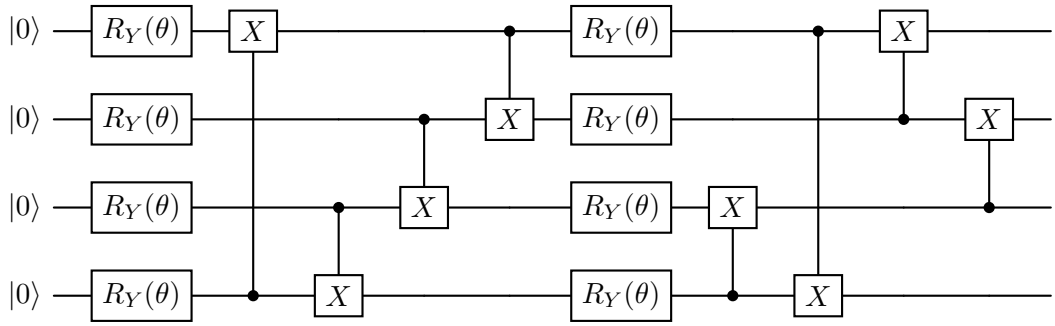
Strongly Entangled Ansatz



Sim 14 Ansatz

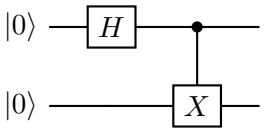
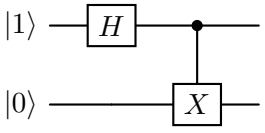
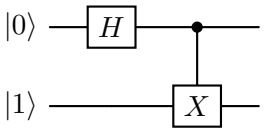
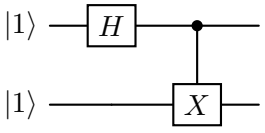


Sim 15 ansatz



B. Bell States

Here we present the four Bell states, with their corresponding circuits.

Bell State	**Dirac Notation**	**Circuit**
$\|\Phi^+\rangle$	$\frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$	
$\|\Phi^-\rangle$	$\frac{1}{\sqrt{2}}(\|00\rangle - \|11\rangle)$	
$\|\Psi^+\rangle$	$\frac{1}{\sqrt{2}}(\|01\rangle + \|10\rangle)$	
$\|\Psi^-\rangle$	$\frac{1}{\sqrt{2}}(\|01\rangle - \|10\rangle)$	

C. String Diagrams

Here we provide the reader with the string diagrams for the sentences used in the experiments in Chapter 6.2 using the pregroup grammar.

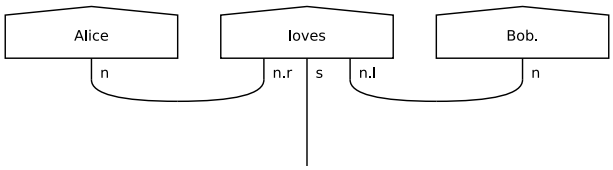


Figure C.1.: String diagram from the BobCat parser for the sentence "*Alice loves Bob.*".

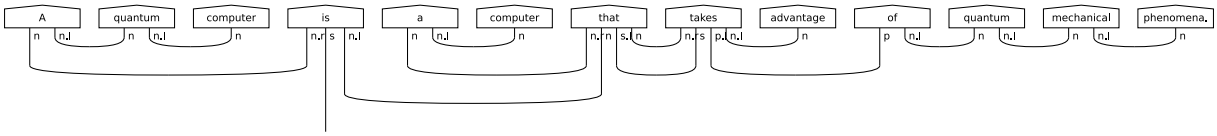


Figure C.2.: String diagram from the BobCat parser for the sentence "*A quantum computer is a computer that takes advantage of quantum mechanical phenomena.*".

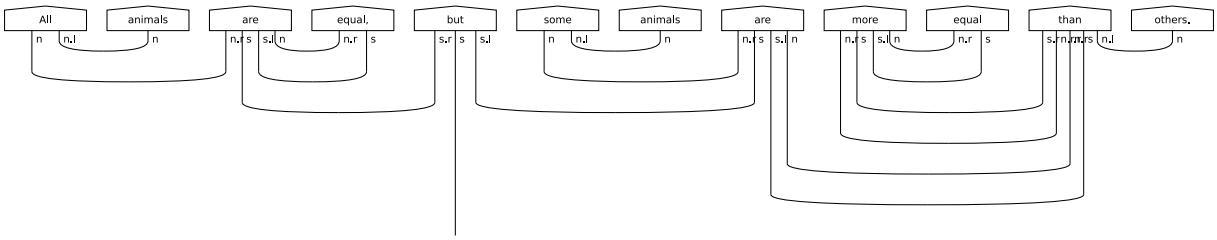


Figure C.3.: String diagram from the BobCat parser for the sentence "*All animals are equal, but some animals are more equal than others.*".

D. Datasets

D.1. The Big Lebowski / Romeo & Juliet

This dataset was taken from [Meichanetzidis et al., 2023]. To create this small scale corpus, they used a context-free grammar and the following vocabulary:

- $N = \{\text{Dude}, \text{Walter}, \text{Romeo}, \text{Juliet}\}$
- $TV = \{\text{loves}, \text{annoys}, \text{kills}\}$
- $IV = \{\text{abides}, \text{bowls}, \text{dies}\}$
- $RPRON = \{\text{who}\}$

These sentences were annotated by hand, resulting in the following 52 labeled sentences.

Dude who loves Walter bowls, 1
Dude bowls, 1
Dude annoys Walter, 0
Walter who abides bowls, 0
Walter loves Walter, 1
Walter annoys Dude, 1
Walter bowls, 1
Walter abides, 0
Dude loves Walter, 1
Dude who bowls abides, 1
Walter who bowls annoys Dude, 1
Dude who bowls bowls, 1
Dude who abides abides, 1
Dude annoys Dude who bowls, 0
Walter annoys Walter, 0
Dude who abides bowls, 1
Walter who abides loves Walter, 0
Walter who bowls bowls, 1
Walter loves Walter who abides, 0
Walter annoys Walter who bowls, 0
Dude abides, 1
Dude loves Walter who bowls, 1
Walter who loves Dude bowls, 1
Dude loves Dude who abides, 1
Walter who abides loves Dude, 0
Dude annoys Dude, 0
Walter who annoys Dude bowls, 1
Walter who annoys Dude abides, 0
Walter loves Dude, 1
Dude who bowls loves Walter, 1
Romeo dies, 1
Romeo loves Juliet, 0
Juliet who dies dies, 1
Romeo loves Romeo, 0
Juliet loves Romeo, 0
Juliet dies, 1
Juliet kills Romeo who dies, 0
Juliet dies, 1
Romeo who loves Juliet dies, 1
Romeo dies, 1
Juliet who dies dies, 1
Romeo loves Juliet, 1
Juliet who dies loves Juliet, 0
Romeo kills Juliet who dies, 0
Romeo who kills Romeo dies, 1
Romeo who dies dies, 1
Romeo who loves Romeo dies, 0
Romeo kills Juliet, 0
Romeo who dies kills Romeo, 1
Juliet who dies kills Romeo, 0
Romeo loves Romeo, 0
Romeo who dies kills Juliet, 0

D.2. Animals / Plants

This dataset was obtained from GPT-3.5 the following prompt: "Write a csv file containing 100 sentences with binary labels. The sentences should either be in the category animals (label 0) or plants (label 1). The sentences should have a simple syntax. Use only nouns, transitive and intransitive verbs and the pronouns who/that. Example: The lion who drinks dies. Do not start all sentences with 'the'." The 100 resulting sentence/label pairs can be seen below. It is noteworthy, that the dataset contains duplicate sentences. Removing the duplicates leaves 68 unique sentences.

The cat that sleeps purrs, 0
Dogs that bark scare birds, 0
The horse who eats runs fast, 0
Cows that graze give milk, 0
Birds that sing greet mornings, 0
The fish who swims explores the ocean, 0
Owls that hunt are nocturnal, 0
The rabbit that hops enjoys carrots, 0
Monkeys that climb entertain us, 0
The bear who roars protects its territory, 0
Plants that bloom attract bees, 1
Trees that sway provide shade, 1
The flower that blossoms smells sweet, 1
Vines that climb cover walls, 1
The grass that grows needs sunlight, 1
Tulips that bloom add color to gardens, 1
The fern that unfurls loves damp soil, 1
Roses that bloom have thorns, 1
Sunflowers that face the sun thrive, 1
The oak that stands tall offers acorns, 1
The snake who slithers hunts mice, 0
Birds that migrate cover long distances, 0
The koala who eats eucalyptus sleeps a lot, 0
Frogs that leap catch insects, 0
The lion that roars marks its territory, 0
Butterflies that flutter are colorful, 0
The penguin who waddles cannot fly, 0
Ants that work together build strong colonies, 0
The elephant who trumpets communicates with others, 0
The sunflower that turns follows the sun, 1
The tomato plant that grows yields juicy fruits, 1
Cacti that thrive in deserts store water, 1
The palm tree that sways survives storms, 1
The orchid that blooms is delicate, 1
Seeds that sprout become new plants, 1
The fern that unfurls loves damp soil, 1
Grains that grow provide sustenance, 1
The oak that stands tall offers acorns, 1
The cheetah who runs fast catches prey, 0
Bees that buzz pollinate flowers, 1
The snail that crawls leaves a trail, 0
The giraffe who grazes has a long neck, 0
The crocodile that lurks in water is stealthy, 0
The eagle that soars has keen eyesight, 0
Cacti that thrive in deserts store water, 1
The spider who spins creates intricate webs, 0
Penguins that slide on ice are playful, 0
The owl that hoots is nocturnal, 0
The pumpkin plant that grows yields orange fruits, 1
The monkey who swings from trees is agile, 0
The lizard that basks in the sun warms up, 0
Plants that produce oxygen support life, 1
The salmon that swims upstream spawns, 0
The butterfly that flutters has delicate wings, 0
Chickens that cluck lay eggs, 0
The bamboo that grows quickly is versatile, 1
The snail that crawls leaves a shiny trail, 0
The whale that sings communicates with others, 0
The rose that blooms is a symbol of love, 1
The rabbit who hops enjoys fresh greens, 0
The tomato plant that grows yields plump tomatoes, 1
Owls that hunt are skilled predators, 0
The beaver that builds dams constructs with precision, 0
The sunflower that turns follows the sun, 1
The spider that weaves creates intricate webs, 0
The rabbit who hops enjoys carrots, 0
The oak that stands tall offers acorns, 1
The elephant who trumpets communicates with others, 0
The fox that hunts is cunning, 0
The tulip that blooms adds beauty to gardens, 1
The ant that works diligently builds a strong colony, 0
The lizard that basks in the sun warms up, 0
Cows that graze give milk, 0
Plants that bloom attract bees, 1
The snake who slithers hunts mice, 0
The lion that roars marks its territory, 0
The sunflower that turns follows the sun, 1
The tomato plant that grows yields juicy fruits, 1
Cacti that thrive in deserts store water, 1
The palm tree that sways survives storms, 1
The orchid that blooms is delicate, 1
Seeds that sprout become new plants, 1
The fern that unfurls loves damp soil, 1
Grains that grow provide sustenance, 1
The oak that stands tall offers acorns, 1
Bees that buzz pollinate flowers, 1
Cacti that thrive in deserts store water, 1
The spider who spins creates intricate webs, 0
The owl that hoots is nocturnal, 0
Plants that produce oxygen support life, 1
The rose that blooms is a symbol of love, 1
The elephant who trumpets communicates with others, 0
The rabbit who hops enjoys fresh greens, 0
The sunflower that turns follows the sun, 1
The beaver that builds dams constructs with precision, 0
The tulip that blooms adds beauty to gardens, 1
The ant that works diligently builds a strong colony, 0
The cheetah who sprints is a fast runner, 0
Plants that thrive in shade require less sunlight, 1
The butterfly that flutters by brings joy to gardens, 0

E. List of Acronyms

DisCoCat Categorical Compositional Distributional (framework)

NLP natural language processing

QNLP quantum natural language processing

NISQ noisy intermediate-scale quantum

qRAM quantum random access memory

QML quantum machine learning

SVD singular value decomposition

MPS matrix product state

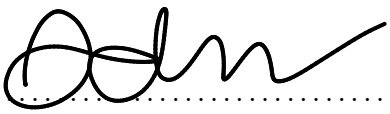
SPSA simultaneous perturbation stochastic approximation

Declaration of Authorship / Selbstständigkeitserklärung

English:
I hereby declare that this bachelor's thesis is my own work, I have marked all citations and I have documented all sources and materials used.

German:
Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

Munich, 04.06.2024



Adrian Maurice Mülthaler