Politecnico di Torino

EURECOM
Sophia Antipolis

amadeus

POLITECNICO DI TORINO
INSTITUT EURECOM

Double Master Degree in Computer Engineering and Data Science

---

Experimental Quantum Natural Language Processing for the Travel Industry

Supervisors
Prof. Bartolomeo Montrucchio[1]
Prof. Marios Kountouris[2]
Dr. Alix Lhéritier[3]
Nicolas Bondoux[3]
Dr. Mourad Boudia[3]

[1]Politecnico di Torino
[2]EURECOM
[3]Amadeus IT Group

Candidate
Massimiliano Pronesti

Academic Year 2022 - 2023

To my family, girlfriend, teachers and friends,
who taught me to achieve my dreams through patience and commitment.

Abstract

In the last decade, Natural Language Processing (NLP) has made giant leaps in treating textual data, by means of large deep neural networks capable of addressing complex tasks such as machine translation, language generation and text summarization. These models are based on the principle of *distributionality* — a word's meaning is defined by the context in which it appears — and learn a representation of words in a vector space. This implies training the model on a massive amount of data to learn the interdependencies among words. In addition, despite the successful achievements, the general problem of natural language understanding is still unsolved as neural network models lack of *explainability*, which raises further concerns when we entrust our decisions to their predictions in critical domains.

Another line of research, sparked by linguists such as N. Chomsky and J. Lambek and culminated in the distributional compositional categorical (DisCoCat) model of Coecke, Clark and Sadradeh tries to address natural language understanding from a different perspective, introducing a simple, yet powerful, analogy between computational linguistics and quantum theory: the grammatical structure of text and sentences connects the meaning of words in the same way that entanglement connects the states of a quantum system. This language-to-qubit analogy is mathematically formalized using category theory.

Starting from these foundations, this thesis work engineers the process of using quantum computers for natural language processing on real-world data related to the travel industry, assessing at the same time the expressiveness of the approach and the maturity of the technology it is destined to, providing experiments on and benchmarks of real world quantum hardware and showing that noisy intermediate-scale quantum (NISQ) devices are Quantum-NLP-friendly. Lastly, we provide comparisons with popular state-of-the-art models and we show interesting advantages in terms of model size.

Résumé

Au cours de la dernière décennie, le traitement du langage naturel (NLP) a fait progresser à pas de géant le traitement des données textuelles, au moyen d'immenses réseaux de neurones profonds capables de traiter des tâches complexes telles que la traduction automatique, la génération de texte et le résumé de texte. Ces modèles sont basés sur le principe de *distributionnalité* — le sens d'un mot est défini par le contexte dans lequel il apparaît — et apprennent une représentation des mots dans un espace vectoriel. Cela implique d'entraîner le modèle sur une quantité massive de données pour apprendre les interdépendances entre les mots. De plus, malgré les succès, le problème général de la compréhension du langage naturel n'est toujours pas résolu car les modèles de réseaux de neurones manquent d'*explicabilité*, ce qui est problématique lorsque leurs prédictions sont utilisées pour prendre des décisions dans des domaines critiques.

Une autre ligne de recherche, initiée par des linguistes tels que N. Chomsky et J. Lambek et culminant dans le modèle catégorique de composition distributionnelle (DisCoCat) de Coecke, Clark et Sadradeh, tente d'aborder la compréhension du langage naturel d'un point de vue différent, en introduisant une analogie simple, mais puissante, entre la linguistique computationnelle et la théorie quantique : la structure grammaticale du texte et des phrases relie le sens des mots de la même manière que l'intrication relie les états d'un système quantique. Cette analogie langage-qubit est formalisée mathématiquement en utilisant la théorie de catégories.

Partant de ces fondements, ce travail de thèse décrit le processus d'utilisation des ordinateurs quantiques pour le traitement du langage naturel, sur des données du monde réel liées à l'industrie du voyage, évaluant à la fois l'expressivité de l'approche et la maturité de la technologie à laquelle elle est destinée, fournissant des expériences et des références sur le matériel quantique du monde réel et montrant que les dispositifs quantiques à échelle intermédiaire bruyants (NISQ) sont compatibles avec le traitement quantique du langage naturel (QNLP). Enfin, nous fournissons des comparaisons avec les modèles de l'état de l'art les plus répandus et nous montrons des avantages intéressants en termes de taille de modèle.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Company overview and business sector

Amadeus IT Group is the world leading Global Distribution System (GDS) company operating in the travel and tourism industries and one of the world's leading software companies. It was established in 1987 as an alliance between Air France, Lufthansa, Iberia and Scandinavian Airline System and counts over 16.000 employees worldwide, with the main product development site located in Sophia Antipolis, France and the corporate headquarter in Madrid, Spain. At market level, Amadeus maintains customer operations through 173 local Amadeus Commercial Organisations (ACOs) covering over 190 countries, as shown in Figure 1.1.

Figure 1.1: Amadeus world presence as the leading company in the travel industry

More in details, Amadeus works within two main areas:

- as a Global Distribution System platform, offers the infrastructure for searching, princing, booking, ticketing and several other processing services to different travel providers and travel agencies through its central reservation system (CRS)
- as an Information Technology (IT) company, provides software to automate processes in different areas, such as inventory management, revenue management, data analysis and travel intelligence services for a wide variety of customers, including airlines, hotels, tour operators, etc.

Among the others, the company currently provides services to 770 airlines, 132 airport operators, 1M+ hotel properties, 53 cruises, 43 car rental companies, 128 ground handlers and 90 rail operators.

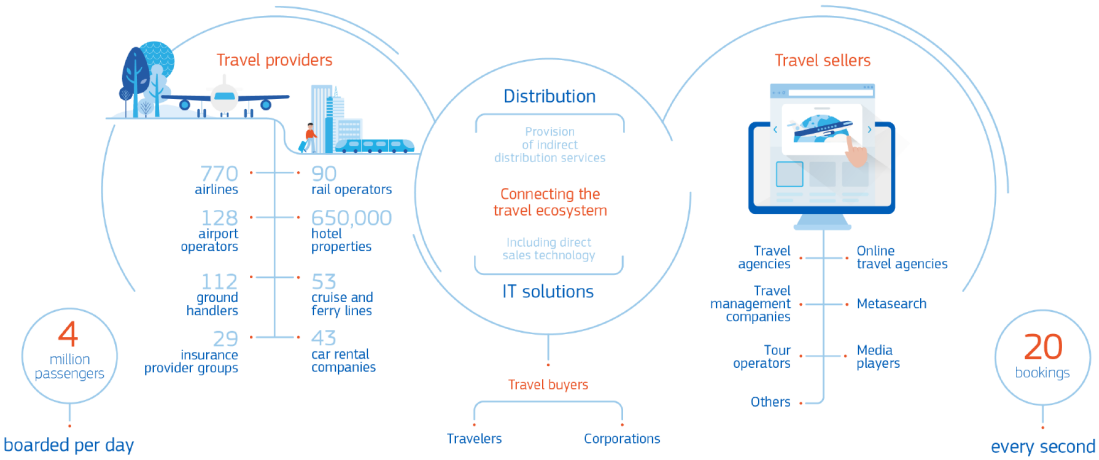


Figure 1.2: Amadeus business model

1.1.1 Team presentation

This thesis work has been carried out as part of the Applied Research and Technology Team. The team, whose acronym within the organization is ART, does applied and exploratory research to enable new applications and improve current products throughout Amadeus, specifically in the fields of Artificial Intelligence, Data Science and Emerging Technologies. This research is usually promoted by publishing in top journals and conferences and most of the research work done within the team is eventually put in production as part of the company's services pipeline or submitted as patent. Specifically, the applied side of the research work carried out within the team is use case driven and aims at solving specific problems related to company needs or customer's ones, while the exploratory branch has the

long term goal to make the company always on track on the technological novelties and consider upgrading existing solutions to more modern alternatives. This thesis work belongs to the latter branch.

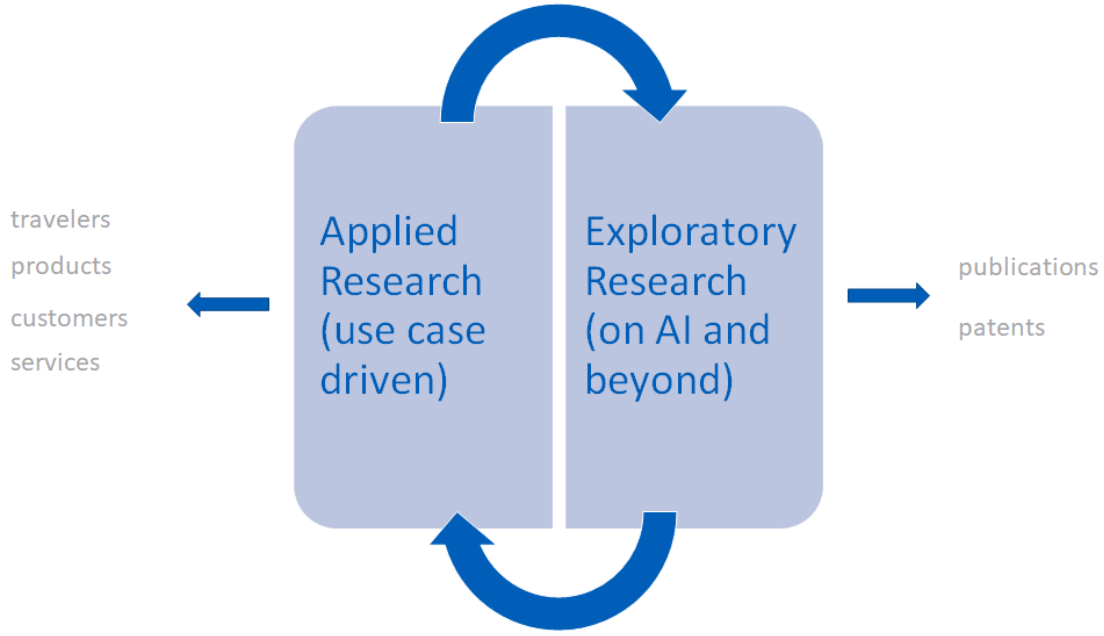


Figure 1.3: Amadeus Applied Research and Technology team activity

Moreover, the research work of the team is further boosted by active collaborations with Academia, including the Massachusetts Institute of Technology (MIT), Eurecom, the University of Côte d'Azur and Inria through internships, industrial PhDs and shared research projects.

1.2 Thesis objective

The purpose of this thesis is to explore quantum models for Natural Language Processing — with particular attention to Amadeus business and use cases — and to compare them with existing classical models such as state-of-the-art transformer-based models and recurrent neural networks approaches. In addition, this work aims at evaluating the capabilities of real quantum devices both in terms of performances and speed on tasks close to real world problems, in order to get hands-on experience on this technology and assess how far or close we are from industrial production-oriented applications. In doing so, we also want to highlight the quality of different platforms offered by quantum vendors and providers, such as IBM and Microsoft. This thesis has, thereby, a **data science** "flavour", related to the study and the applications of a different approach towards NLP and a **quantum computational** one, motivated by the advantages brought to this approach.

We will look particularly at two use cases: a customer feedback analysis task, described in Chapter 4 and an intent recognition one on Chatbot queries, object of Chapter 5. In both cases, differently from the existing applied works on this topic — which only make use of tiny, ad hoc, synthetic datasets [1, 2] —, we will use **real** textual data to further boost the meaningfulness of the comparisons.

Some of the results of this work have been presented to IBM and Quantinuum (including Cambridge Quantum) as well as at SophI.A Summit 2022 [3].

1.3 Why Quantum Computers for NLP ?

Quantum computers have, notoriously, the potential to revolutionize several branches of science and engineering, such as artificial intelligence, cryptography and communications. However, it is a common misconception that they will be only exploited to solve certain types of intractable problems for classical machines or to speedup complex optimization tasks classical machines struggle with. In fact, quantum computers implement a rather different computation paradigm than classical machines, thus there needs to be some structure that they can exploit in some unique advantageous way: their own structure in the case of physics simulation or the group-theoretic structure of cryptographic protocols in Shor's algorithm [4]. This section will argue that natural language shares a common structure with quantum theory, in the form of two linguistic principles: **compositionality** and **distributionality**.

Starting from 2011, when Apple, Amazon and Google made their vocal assistants mainstream, modern NLP models rely on deep neural networks characterized by complex architectures based on a simple statistical concept: *language models*, namely probability distributions over sequence of words. It is the case for the recurrent neural architectures, proposed by Rumelhart et al. [5] and for their more elaborated variants, i.e. LSTM [6] and its bidirectional counterpart [7], which treat a text as a sequence of words processed one at a time at each step to update their internal state.

The attention mechanism, proposed by Vaswani et al. [8] in one of the most successful papers in the history of Artificial Intelligence, provided an even more elegant solution, allowing the network to learn the interdependencies among words, instead of just using those coming before and after in the sequence. This was a truly revolutionary achievement in NLP and gave birth to *transformer models*, the first of whom was BERT [9]. Today transformers have replaced RNNs as the state-of-the-art models in natural language processing and proved capable of addressing complex tasks such as language generation and machine translation.

Despite all these successful achievements, the fundamental problem of understanding language —the iceberg lying under words and sentences— remains unsolved. As a matter of fact, we can probe the way these models map inputs to

outputs, but we have no interpretation for their weights, which also makes impossible to address problems such as fairness and biases in the data. The problem of explainability is further hindered by the models' size, characterized by a **huge amount of parameters**, in the order of magnitude of billions or even trillions, as shown in Figure 1.4.

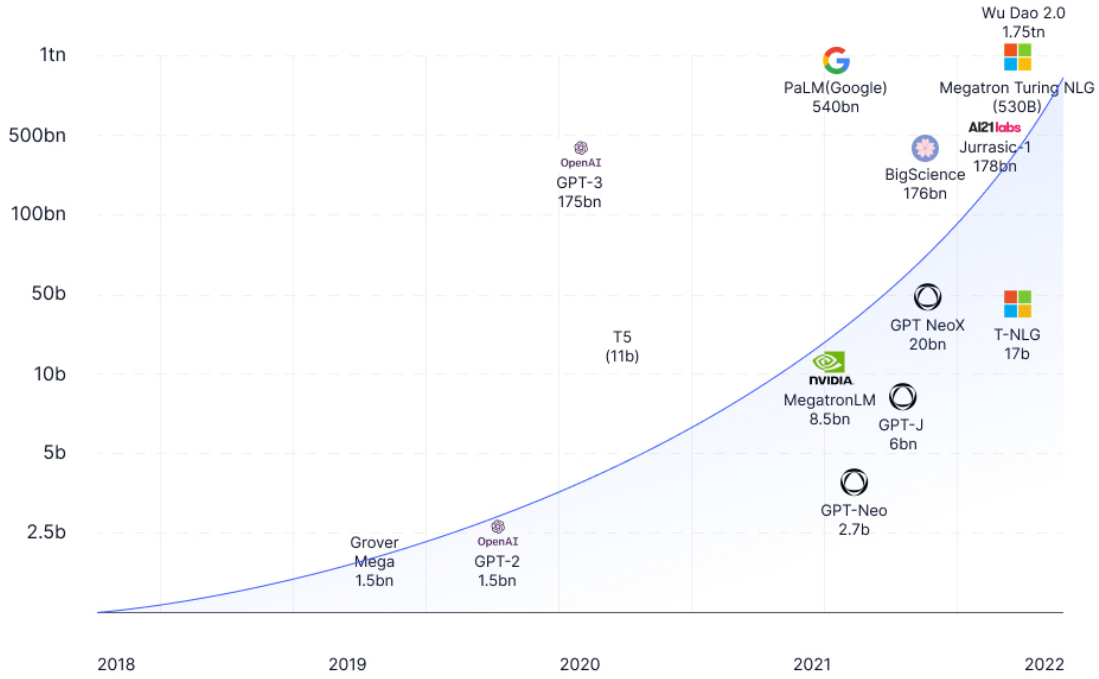


Figure 1.4: Timeline of modern language models growth in terms of size (number of parameters).

The number of parameters goes hands in hands with the amount of data needed to train the model. That's why transformers are extremely **data hungry** and massively **expensive in terms of computational power**. As a result, only big tech companies can afford to train them for scratch, while everybody else relies on pre-trained versions fine-tuned on the specific task they are addressing.

To assess the problem of explainability as well as to reduce the size of such models, we need to change the way we "learn language" and move from feeding huge amounts of data to deep networks to a different approach. The novelty is to exploit **grammar**: instead of learning the interdependencies among words, we can exploit their grammatical relations and embed them in the way we represent the language, so that we no longer have to learn it. This is what distributional compositional (DisCo) models do, assigning to words a representation in a vector space and representing their interaction with tensor products $\otimes$.

What has this to do with quantum computing ? The analogy was proposed by Clark, Coecke and Sadrzadeh [10, 11] in their distributional compositional

categorical model (DisCoCat, described more thoroughly in Chapter 3) of meaning, where they showed that modeling words as element of high dimensional vector spaces and their relations as tensor products is nothing but the way we model states in a quantum system. In addition, grammar describes how information flows in a sentence in the very same way entanglement connects quantum states and tells how information flows in a complex quantum system. This analogy allows us to borrow well-established mathematical tools from quantum theory and use them for natural language processing. Nonetheless, this would only make this approach quantum-inspired. The quantum computational side comes from the observation that representing the meaning of a sentence as a quantum process and resorting to a tensor-like representation of the interactions introduce a critical drawback: an **exponential complexity** in a classical machine. This is where quantum computers come into play: to compute the meaning of a sentence, DisCoCat requires computing in high dimensional tensor products spaces, which is something a quantum machine is "naturally" suited for, because that's how it inherently manipulates information. In other words, on a quantum computer this scales linearly in the number of qubits.

To further understand this much, we need to first describe the practical implementation of this approach. In this regard, section 3.4 will provide additional insights.

1.4 Isn't it too early ?

Given the limited capabilities of modern quantum devices, one might claim that trying to use them to tackle tasks related to one of the most challenging fields in Artificial Intelligence is premature, as stated in the following anonymous journal review:

> "I am subjectively skeptical of the core assumption that the authors are making that now is the right time to be working on applications like NLP with quantum algorithms. It sounds like many of the current challenges are at the core level of basic machine learning and dealing with the esotericities of hardware and compilers."
>
> (Anonymous reviewer of the *QNLP in practice* [1] paper)

However, as we will show in the applied chapters of this thesis, this approach can be run already on existing noisy intermediate-scale quantum (NISQ) devices. Furthermore, the development of quantum hardware is proceeding lightning-fast such that progressing in the formulation and development of frameworks capable of leveraging it became crucial. In this regard, being ambitious and trying to use a (currently) immature — but promising — technology for such effortful tasks will eventually prove to be the key.

"Despite the limited capabilities of the current quantum machines, this early work is important in helping us understand better the process, the technicalities, and the unique nature of this new computational paradigm. At this stage, getting more hands-on experience is crucial in closing the gap that exists between theory and practice, and eventually leading to a point where practical real-world QNLP applications will become a reality"

(Lambeq [12] paper)

Chapter 2

Background

This chapter aims at making the reader familiar with the theoretical concepts used throughout this work. First, an introduction to quantum computation is given, to set a common ground on the computational paradigm and, afterwards, an overview of category theory is provided, in order to establish a mathematical framework to express how quantum computers can deal with natural language.

2.1 Quantum Computing

Quantum mechanics is the most complete description of the physical properties of nature on the atomic scale. It is also at the core of quantum computation and information. This section describes the necessary background of quantum computing required for understanding quantum natural language processing.

2.1.1 Qubits

In classic information theory, the smallest unit of information is the bit. Quantum information is built upon an analogous concept: the quantum bit, or qubit. Qubits are physical objects that appear in nature on the scale of atoms and subatomic particles. A qubit can be any two-state quantum-mechanical system such as the spin of an electron, which can be spin up or down, or the polarization of a photon, which can be horizontally or vertically polarized. In this thesis, qubits will be treated as abstract mathematical objects as the physical realization of qubits is beyond the scope of this work. The state of a qubit is denoted as follows:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle . \tag{2.1}$$

Quantum states are often described using Dirac notation $|\cdot\rangle$, which describes a column vector in $\mathbb{C}^{2^n}$. The states $\{|0\rangle, |1\rangle\}$ are the computational basis states which are defined as $(1 \quad 0)^T$ and $(0 \quad 1)^T$ respectively, and form an orthonormal basis for

this vector space. The values $\alpha, \beta \in \mathbb{C}$ are the state's probability amplitudes, and cannot be examined directly. This comes as a consequence of the fact that when a qubit is measured, it collapses probabilistically to one of the basis states. Specifically, the probability of measuring 0 is given by the absolute square $|\alpha_0|^2$, and the probability of measuring 1 is given by $|\alpha_1|^2$. As these values are probabilities, they need to be normalized, i.e. it must hold that: $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Formally, a qubit can be thought of as a unit vector in a two-dimensional Hilbert space.

A qubit differs from a classical bit in the sense it can be in a linear combination, or superposition of states. While a bit can only be in the state 0 or 1, a qubit can be in one of infinitely many superpositions of states. Nevertheless, differently from classical information, the laws of quantum mechanics restrict direct access to the probability amplitudes of a state. In fact, when measuring a qubit, it collapses to basis state $|j\rangle$ with probability $|\alpha_j|^2$. For example, consider the state

$$|+\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + |1\rangle \right). \tag{2.2}$$

This state has equal probability of measuring 0 and 1, as $|1/\sqrt{2}|^2 = 1/2$. Note that measurement changes the state of a qubit: if the state from Equation (2.2) is measured as 1, the superposition is lost and the state becomes $|1\rangle$.

A helpful geometric interpretation of a qubit's state can be obtained by rewriting Equation (2.1) as

$$|\psi\rangle = e^{i\delta} \left(\cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle \right), \tag{2.3}$$

where $\delta, \theta, \varphi \in \mathbb{R}$. The global phase $e^{i\delta}$ can often be ignored, as $\forall \delta \in \mathbb{R} : |e^{i\delta}| = 1$, so it does not impact measurement outcome. Simplifying then, the state of a qubit can be written as

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle . \tag{2.4}$$

Here, θ and φ define a point on the surface of a three-dimensional sphere referred to as the Bloch sphere, shown in Figure 2.1.

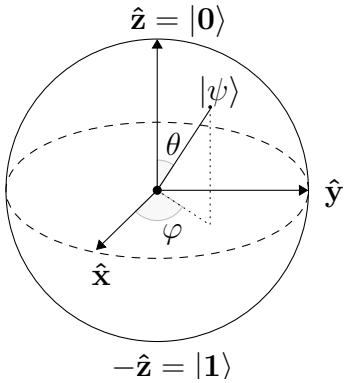


Figure 2.1: Bloch sphere representation of a qubit's state

While this visualization is limited to a single qubit, it can be a useful visual to build intuition. For example, the $|+\rangle$ state described in Equation (2.2) can be thought of as being exactly between $|0\rangle$ and $|1\rangle$ on the Bloch sphere.

The amount of probability amplitudes grows exponentially with the number of qubits: a n-qubit state has $N = 2^n$ amplitudes. Consider a two-qubit system which lives in a $2^2 = 4$-dimensional Hilbert space spanned by the computational basis states $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. This state is defined by the linear combination

$$|\psi\rangle = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle . \tag{2.5}$$

We remark again that, unlike classical bits who can only be in one state at a time, this state can be in a superposition of all four states. The normalization condition still applies for Equation (2.5): $\sum_{j=0} |\alpha_j|^2 = 1$. Single-qubit states can be combined to form multi-qubit states by taking the tensor product of the two states. Given states $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ and $|\varphi\rangle = \beta_0 |0\rangle + \beta_1 |1\rangle$:

$$\begin{aligned} |\psi\rangle \otimes |\varphi\rangle &= \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \otimes \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \\ &= \begin{pmatrix} \alpha_0 \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \\ \alpha_1 \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \end{pmatrix} \\ &= \alpha_0\beta_0 |00\rangle + \alpha_0\beta_1 |01\rangle + \alpha_1\beta_0 |10\rangle + \alpha_1\beta_1 |11\rangle . \end{aligned} \tag{2.6}$$

A multi-qubit state is mathematically formalized through the tensor product. The following notations can be employed interchangeably:

$$|0\rangle \otimes |0\rangle = |0\rangle |0\rangle = |00\rangle \tag{2.7}$$

The relative phases of α_0, α_1 and β_0, β_1 in Equation (2.6) are responsible for the quantum mechanical property of interference. When the phase of α_j and β_k is the same, they will interfere constructively and increase the probability amplitude for that state. On the other hand, if α_j and β_k have opposite phases, they will interfere destructively and decrease the probability amplitude for the considered state.

It's important to point out that not all multi-qubit systems can be expressed as a tensor product of individual states, as shown in Equation (2.6). For instance, consider the following state:

$$\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}} \left(|00\rangle + |11\rangle\right) . \tag{2.8}$$

This state cannot be expressed as a tensor product of two individual states, as that would imply $\left(\alpha_0\beta_0 = \alpha_1\beta_1 = 1/\sqrt{2}\right) \wedge \left(\alpha_0\beta_1 = \alpha_1\beta_0 = 0\right)$, which is a contradiction. States like $|\Phi^+\rangle$ are referred to as *entangled* states. Entanglement is the quantum phenomena of correlation in measurement outcomes. For example, when measuring the state $|\Phi^+\rangle$ from Equation (2.8), the only two possible measurement outcomes are 00 and 11. So by measuring one qubit, one also knows the state of the other qubit. This is by likely the most important feature of quantum computation.

2.1.2 State Evolution

The evolution of a closed quantum system is described by a unitary transformation.[1] A state $|\psi\rangle$ at time t_1 is related to state $|\psi'\rangle$ at time t_2 by a unitary operator U:

$$|\psi'\rangle = U\,|\psi\rangle\,. \tag{2.9}$$

The unitary nature of these operators implies $UU^\dagger = U^\dagger U = I$, where † is the conjugate transpose and I the identity matrix. Single-qubit operators can be represented as 2×2 complex-valued unitary matrices. A common single-qubit operator is the Pauli-X operator which transforms a state $\alpha_0\,|0\rangle + \alpha_1\,|1\rangle$ to $\alpha_1\,|0\rangle + \alpha_0\,|1\rangle$. It is part of the set of Pauli matrices:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.10}$$

These matrices are ubiquitous in the study of quantum computation and information. Another useful and common operator is the Hadamard operator:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{2.11}$$

which maps the computational basis states to an equal superposition state.

The Pauli X, Y and Z operators give rise to the rotation operators about the

[1] A perfectly closed system is assumed, even though in reality all systems interact somewhat with other systems. This is not an issue for practical quantum computers though due to quantum error correction, which can protect quantum information against noise from the external environment.

$\hat{x}$, $\hat{y}$, and $\hat{z}$ axes when exponentiated:

$$R_x(\theta) = e^{-i\frac{\theta}{2}X} = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \tag{2.12}$$

$$R_y(\theta) = e^{-i\frac{\theta}{2}Y} = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \tag{2.13}$$

$$R_z(\theta) = e^{-i\frac{\theta}{2}Z} = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}. \tag{2.14}$$

These operators can be thought of as rotating a qubit's state among its relative axis by an angle θ. Note that any single-qubit operator U can be decomposed into these operators, for example $U = R_z(\gamma)R_y(\beta)R_z(\alpha)$ where $\alpha, \beta, \gamma \in \mathbb{R}$.

Multi-qubit operators act on two or more qubits and are required for creating entangled states. Two common two-qubit operators are the CNOT and CZ operators. The CNOT operator can be thought of as a controlled-X operator, which applies a Pauli-X operation on the target qubit if the control qubit is in the $|1\rangle$ state. Equivalently, the CZ operator applies a Pauli-Z operation on the target qubit if the control qubit is $|1\rangle$. Their matrix representations are as follows:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{2.15}$$

Generally, a controlled-U operator applies U on the target qubit if the control qubit is $|1\rangle$.

2.1.3 Measurement

While the evolution of quantum states in a closed system is unitary, at some point the quantum state has to interact with the outside world. This is what measurement means: any interaction from an outside system with the quantum system. Specifically, a measurement is what brings quantum information back to the classical domain. Formally, measurement is defined by a set of measurement operators $\{M_m\}$, where the probability of measuring the state m is given by

$$p(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle. \tag{2.16}$$

With $\langle\psi| = |\psi\rangle^\dagger$, this equation can then be read as the inner product between $M_m|\psi\rangle$ and itself. This is just a generalization of the definition of measurement given in Section 2.1.1. For example, the probability of measuring 0 for an arbitrary

state $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$, using $M_0 = |0\rangle\langle 0|$ can be estimated as:

$$\begin{aligned} p(0) &= \langle\psi| M_0^\dagger M_0 |\psi\rangle \\ &= \langle\psi|0\rangle \langle 0|0\rangle \langle 0|\psi\rangle \\ &= \langle\psi|0\rangle \langle 0|\psi\rangle \\ &= \begin{pmatrix}\alpha_0^* & \alpha_1^*\end{pmatrix} \begin{pmatrix}1\\0\end{pmatrix} \begin{pmatrix}1 & 0\end{pmatrix} \begin{pmatrix}\alpha_0\\\alpha_1\end{pmatrix} \\ &= \alpha_0^*\alpha_0 \\ &= |\alpha_0|^2. \end{aligned} \tag{2.17}$$

Similarly, using $M_1 = |1\rangle\langle 1|$ gives $p(1) = |\alpha_1|^2$. Measuring with the measurement operators $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$ is referred to as measuring in the computational basis.

After measurement, the state of the system can be described as follows:

$$|\psi\rangle = \frac{M_m |\psi\rangle}{\sqrt{p(m)}}. \tag{2.18}$$

Following the example from Equation (2.17) with $M_0 = |0\rangle\langle 0|$ and $p(0) = |\alpha_0|^2$, after measuring 0 the system is in the state

$$\begin{aligned} |\psi\rangle &= \frac{|0\rangle \langle 0|\psi\rangle}{\sqrt{|\alpha_0|^2}} \\ &= \frac{|0\rangle \alpha_0}{|\alpha_0|} \\ &= \frac{\alpha_0}{|\alpha_0|} |0\rangle . \end{aligned} \tag{2.19}$$

Notice that a global phase $e^{i\delta}$ shows up as the factor $\alpha_0/|\alpha_0|$. As mentioned in Section 2.1.1, the states $e^{i\delta}|0\rangle$ and $|0\rangle$ are considered equal up to the global phase factor.

A special class of measurements called projective measurements can sometimes be used to simplify calculations. A projective measurement is defined by an observable O, which is a Hermitian operator on the state space of the system. This observable has a spectral decomposition

$$O = \sum_j \lambda_j |\lambda_j\rangle \langle\lambda_j| , \tag{2.20}$$

being λ_j the eigenvalues of O and $|\lambda_j\rangle$ the respective eigenstates. For example, the Pauli-Z operator can be thought of as an observable with the spectral decomposition

$$Z = 1 |0\rangle\langle 0| - 1 |1\rangle\langle 1| , \tag{2.21}$$

which has eigenstates $\{|0\rangle, |1\rangle\}$ with respective eigenvalues $\{1, -1\}$. As the observable Z has the computational basis states as eigenstates, a measurement of Z can also be thought of a measuring in the computational basis. With this definition of projective measurements, the expectation value of an observable O for state $|\psi\rangle$ can be defined as

$$\begin{aligned}\langle O\rangle_\psi &= \langle\psi| O |\psi\rangle \\ &= \langle\psi| \left(\sum_j \lambda_j |\lambda_j\rangle\langle\lambda_j|\right) |\psi\rangle \\ &= \sum_j \lambda_j \langle\psi|\lambda_j\rangle\langle\lambda_j|\psi\rangle \\ &= \sum_j \lambda_j |\langle\lambda_j|\psi\rangle|^2.\end{aligned} \tag{2.22}$$

The expectation value is the sum of all possible outcomes (eigenvalues of O) weighted by their probability. Calculating the expectation value experimentally means preparing and measuring the state multiple times and is a useful way of extracting a deterministic quantity from a non-deterministic model of computation. This is what operationally a quantum algorithm should be designed for.

2.1.4 Quantum Circuits

Analogous to the classical circuit model, the quantum circuit model uses gates which act on qubits. Most of the common quantum gates used in quantum computing were already described as operators in Section 2.1.2. However, to be more in line with the nomenclature of classical computing, from here onward these operators will be referred to as quantum gates in the context of quantum circuits, the abstraction usually employed to model quantum operations in quantum computer science.

In a quantum circuit, qubits are represented by wires on which gates can act, and classical bits are represented by double-lined wires. Qubits are usually assumed to be instantiated to $|0\rangle$, unless noted otherwise. Quantum gates are unitary and thus reversible, making the quantum circuit model a reversible model of computation. The inverse of a gate U is denoted as the conjugate transpose $U^\dagger$. By the definition of unitary $UU^\dagger = U^\dagger U = I$, applying the inverse $U^\dagger$ after U essentially uncomputes U and vice versa. Gates whose conjugate transpose are equal to themselves are referred to as Hermitian. For example, H is Hermitian as $H^\dagger = H$ and thus $H^2 = I$. A list of frequently used quantum gates is reported in Table 2.1.

Figure 2.2 demonstrates a simple quantum circuit which creates the maximally entangled state $|\Phi^+\rangle = (|00\rangle + |11\rangle)/\sqrt{2}$ from Equation (2.8) and measures both qubits. This state is one of four maximally entangled two-qubit states referred to as a Bell state. This circuit does the following. The system starts in state $|\psi\rangle = |00\rangle$.

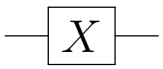
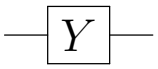
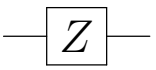
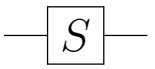
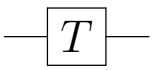
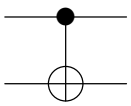
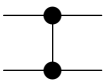
Gate name	Circuit symbol	Matrix representation
Hadamard		$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$
Pauli-X		$\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$
Pauli-Y		$\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}$
Pauli-Z		$\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$
Phase (S)		$\begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix}$
T		$\begin{pmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{pmatrix}$
CNOT		$\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}$
CZ		$\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & -1\end{pmatrix}$

Table 2.1: Frequently used quantum gates with their circuit symbol and matrix representation.

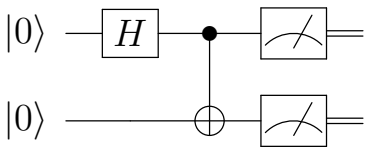


Figure 2.2: Quantum circuit for creating and measuring a Bell state $|\Phi^+\rangle$.

A Hadamard gate is applied on the first qubit:

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)|0\rangle \tag{2.23}$$

$$= \frac{1}{\sqrt{2}}\left(|00\rangle + |10\rangle\right). \tag{2.24}$$

Then, a CNOT is applied with the first qubit as control and the second qubit as target, giving the final state

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right). \tag{2.25}$$

This state is then measured, which will measure 00 or 11 with equal probability. Quantum circuits are often executed multiple times (called "shots") to get a probability distribution of all possible outcomes. This is what, operationally, makes quantum computing probabilistic.

Quantum gates can be applied in parallel on different qubits, which is why it is often more useful to talk about the depth of a quantum circuit rather than the number of gates. The depth of a quantum circuit corresponds to the number of steps required for the quantum operations making up the circuit to run. For example, the quantum circuit represented in Figure 2.3 consists of 8 quantum gates, but has a depth of 4 (measurement gates are omitted from the depth calculation). The gates in every column are considered to be one time step as they can be executed in parallel. Instead of applying the first column of gates sequentially as $(I \otimes I \otimes S)(I \otimes X \otimes I)(H \otimes I \otimes I)\,|000\rangle$, it can be thought of as applying the single operator $(H \otimes X \otimes S)\,|000\rangle$.

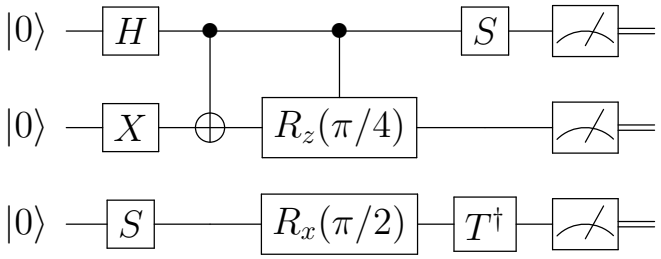


Figure 2.3: A quantum circuit consisting of 8 gates with a depth of 4.

In classical computations, copying bits is a common operation. However, in quantum computing, the data one can copy is much more restricted. If a qubit is in a computational basis state, that state can be copied by a CNOT gate (Figure 2.4).[2] However, trying to copy an arbitrary state $|\psi\rangle = \alpha_0\,|0\rangle + \alpha_1\,|1\rangle$ using the circuit in Figure 2.4 yields:

$$|\psi\rangle = \alpha_0\,|00\rangle + \alpha_1\,|11\rangle. \tag{2.26}$$

This state does not contain two copies of $|\psi\rangle$ (unless $\alpha_0\alpha_1 = 0$ as is true with computational basis states), which should have the form

$$|\psi\rangle\,|\psi\rangle = \alpha_0^2\,|00\rangle + \alpha_0\alpha_1\,|01\rangle + \alpha_1\alpha_0\,|10\rangle + \alpha_1^2\,|11\rangle. \tag{2.27}$$

[2]More formally, if a quantum state is in a basis state of a known orthogonal basis, the state can be copied.

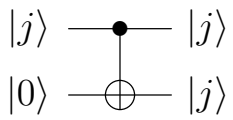


Figure 2.4: Copying a computational basis state using CNOT where $j \in \{0, 1\}$.

In fact, is it impossible to copy an unknown quantum state. There exists no solution for a unitary operator U that does the transformation

$$\alpha_0 |00\rangle + \alpha_1 |01\rangle \xrightarrow{U} \alpha_0^2 |00\rangle + \alpha_0\alpha_1 |01\rangle + \alpha_1\alpha_0 |10\rangle + \alpha_1^2 |11\rangle . \tag{2.28}$$

This property is known as the *no-cloning theorem*, and is a fundamental limit of quantum information. Note that this holds for unknown quantum states. If one has the circuit to prepare $|\psi\rangle$, you could simply create $|\psi\rangle|\psi\rangle$ by executing the circuit on a second qubit.

2.1.5 Universal Gate Sets

A universal gate set is defined as a finite set of gates that can be used to represent any other gate. In the classical circuit model, NAND is a universal gate with which all other gates can be represented. Equivalently, in the filed of quantum computing, universal gate sets are sets of quantum gates any quantum operation can be reduced to. A common universal gate set is $\{\text{CNOT}, H, S, T\}$, i.e. any quantum operation can be reduced to a combination of CNOT, H, S, and T gates. For instance, the three-qubit equivalent of the CNOT gate, the Toffoli gate, can be decomposed into these gates as shown in Figure 2.5. The Solovay-Kitaev theorem shows that any universal gate set can simulate any other universal gate set efficiently, so the choice of universal gate set does not impact the asymptotic efficiency of a quantum computer. It turns out that almost any set of one- and two-qubit gates is universal — in fact, [13] shows that just the Toffoli and Hadamard gates are universal.

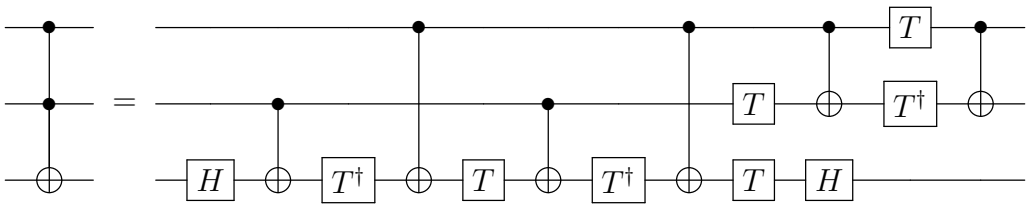


Figure 2.5: Decomposition of the Toffoli gate using CNOT, H, S and T gates.

2.1.6 Parameterized Quantum Circuits

Now that the foundations of quantum information and quantum circuits are set, we move our attention to Parameterized Quantum circuits, also known as Variational Quantum circuits, which are the key element of Quantum Machine Learning approaches (including QNLP).

Like standard quantum circuits, they consist of three elements:

- Preparation of a fixed initial state (e.g., the vacuum state or the zero state)
- A quantum circuit $U(\theta)$, parameterized by a set of free parameters θ
- Measurement of an observable $\hat{B}$ at the output. This observable may be made up from local observables for each wire in the circuit, or just a subset of wires.

The set of free parameters $\theta = (\theta_1, \theta_2, ..., \theta_n)$ of the circuit is tuned to optimize some cost function computed after measurement for some given task. This scalar cost can be defined as $f(\theta) = \langle 0|U^\dagger(\theta)\hat{B}U(\theta)|0\rangle$.

Variational circuits are trained by a **classical** optimization algorithm that makes queries to the quantum device. The optimization is usually an iterative scheme that searches out better candidates for the parameters θ with every step, analogously to classical machine and deep learning algorithms. A schematic representation is shown in Figure 2.6.

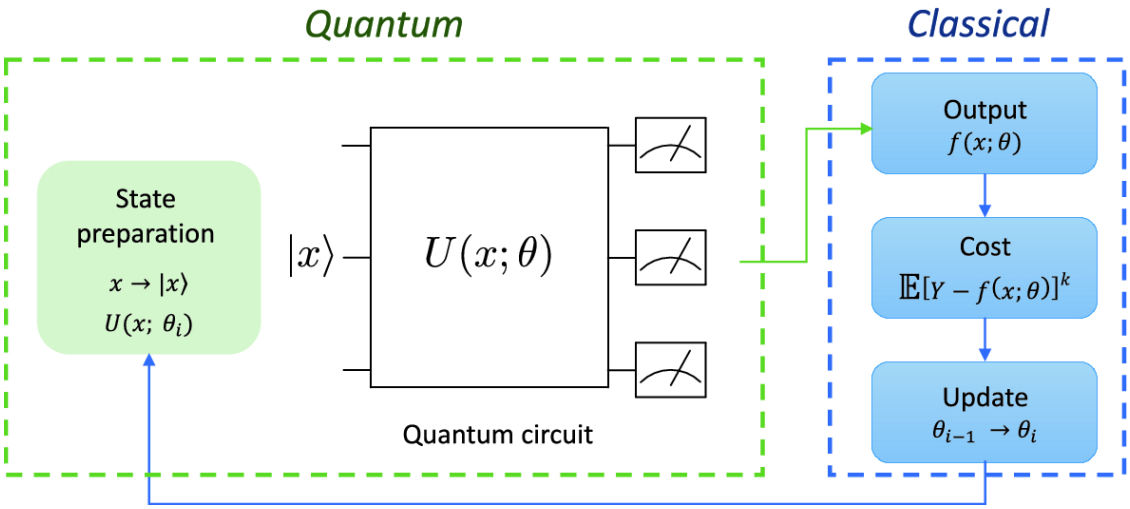


Figure 2.6: Scheme of a variational quantum circuit for supervised learning

Variational circuits have become popular as a way to think about quantum algorithms for near-term quantum devices. Such devices can only run short gate sequences, since without fault tolerance every gate increases the error in the output. Usually, a quantum algorithm is decomposed into a set of standard elementary operations, which are in turn implemented by the quantum hardware. This is the approach we will adopt in the following chapters of this thesis.

2.2 Category Theory

This section aims at providing a **gentle** introduction to category theory and, especially, to make the reader familiar with pregroup grammars, which is the algebraic formalism the quantum model for natural language — presented in section 3.1 — is built upon. Category theory is an abstract framework which deals with mathematical structures and relations between them. It thus allow to formalize, through the categorical concept of functors, the mapping between the different representation of the language, from pregroups to vector spaces.

2.2.1 Categories

A *category* consists of a collection of objects $A, B, C, ...$ and a collection of morphisms between objects of the form $f : A \to B$, $g : B \to C$, $h : C \to D, ...$ such that

- morphism with matching type compose. For instance, given two morphisms $f : A \to B$ and $g : B \to C$, they can compose to make $g \circ f : A \to C$, but not the opposite.
- morphism compose in an associative way: $(h \circ g) \circ f = h \circ (g \circ f)$
- each object has an identity arrow $1_B \circ f = f = f \circ 1_A$

This formalism can be intuitively grasped from the cumulative diagram showed in Figure 2.7.

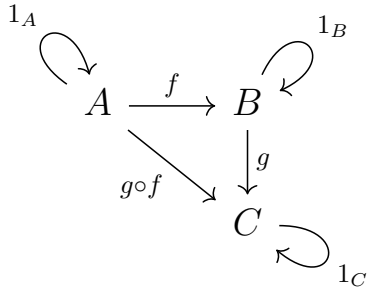


Figure 2.7: Cumulative diagram for 3 morphism

2.2.2 Monoidal categories

A *monoidal category* is a category equipped with the monoidal product $\otimes$ and monoidal unit $\mathcal{I}$ and has the following properties

- morphism can be combined to get a new morphism whose domain and codomain are given applying the monoidal product

$$(f : A \to B) \otimes (g : C \to D) = f \otimes g : A \otimes C \to B \otimes D$$

- $\otimes$ is associative on objects: $(A \otimes B) \otimes C = A \otimes (B \otimes C)$
- $\otimes$ is associative on morphisms: $(f \otimes g) \otimes h = f \otimes (g \otimes h)$
- $\mathcal{I}$ is the identity operator for objects: $A \otimes \mathcal{I} = A = \mathcal{I} \otimes A$
- $1_{\mathcal{I}}$ is the identity operator for morphisms: $f \otimes 1_{\mathcal{I}} = f = 1_{\mathcal{I}} \otimes f$

2.2.3 Rigid Monoidal Categories

A *rigid category* is a monoidal category where every object A has a left adjoint A^l and a right adjoin A^r. The left adjoint and the right adjoint of the object are nothing but the object itself, i.e. $(A^r)^l = (A^l)^r = A$.

The key property of a rigid category is the existence of cups and caps between an object and its adjoint: these are special morphisms that are drawn as bent wires in diagrammatic notation and they satisfy the snake equations (Figure 2.8)

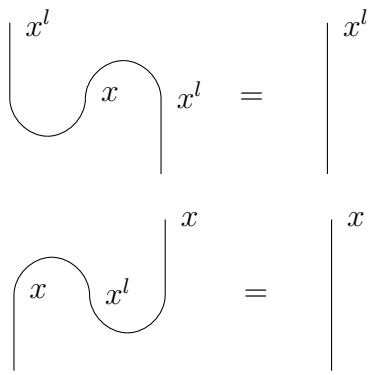


Figure 2.8: Caps, Cups and snake equations

In other words, exploiting the diagrammatic calculus of monoidal algebra, one can visually "yank" a cup or a cup, i.e. given an object A, it holds that

$$A^l \cdot A = 1 = A \cdot A^r \tag{2.29}$$

which are also referred to as the contraction rules of rigid monoidal categories.

2.2.4 Pregroup Grammars

Given the introductory concepts of monoidal categories, presented above, we look now at the main formalism we will extensively use in the following chapters: **pregroup grammar**, devised by Joachim Lambek in 1999 [14].

A *pregroup grammar* is a partially-ordered monoid where each element is a type p with a left adjoint p^l and a right adjoint p^r such that the following conditions (also known as Ajdukiewicz laws) hold:

$$p \cdot p^r \leq 1 \leq p^r \cdot p \qquad p^l \cdot p \leq 1 \leq p \cdot p^l \tag{2.30}$$

To treat this partially-ordered monoid as a grammar, the two axioms $p \cdot p^r \leq 1$ and $p^l \cdot p^l \leq 1$ serve as reductions in the grammar and are represented as:

$$p \cdot p^r \to 1 \qquad p^l \cdot p \to 1 \tag{2.31}$$

An example grammar might be composed using the following pregroup assignments:

- a noun is given the base type n
- an adjective consumes a noun on the noun's left to return another noun, so it is given the type $n \cdot n^l$
- a transitive verb consumes a noun on its left and another noun to its right to give a sentence, so its type is $n^r \cdot s \cdot n^l$.

In the context of pregroups, the left and right adjoints n^l and n^r can be thought of as, respectively, the left and right inverse of the atomic type n. Words are concatenated using the monoidal product $\otimes$ and linked using cups, as explained in the previous section. A sentence is grammatically sound if its derivation has a single uncontracted s wire in the diagrammatic representation or if, applying the contraction rules (2.31), we are only left with an s type. For instance, a sentence containing the subject with its adjective, a transitive verb and an objective complement, using the above assignments, will yield

$$\begin{aligned}(n \cdot n^l) \cdot n \cdot (n^r \cdot s \cdot n^l) \cdot n &\to (n \cdot 1 \cdot n^r \cdot s \cdot n^l \cdot n) \\ &\to (1 \cdot s \cdot n^l \cdot n) \\ &\to (s \cdot 1) \\ &\to s\end{aligned}$$

We provide more extensive examples both of this analytical calculus and its much more powerful graphical variant in Chapter 3.

2.2.5 Monoidal Functors

The last introductory concepts we need to define before diving into quantum natural language processing are monoidal functors.

Given two monoidal categories C and D, a *monoidal functor* $F : C \to D$ is a structure-preserving transformation from one category to another which satisfies the following properties: given 2 objects A, B, the functor F

- preserves the monoidal structure: $F(A \otimes B) = F(A) \otimes F(B)$
- preserves the adjoints: $F(A^l) = F(A)^l \qquad F(A^r) = F(A)^r$
- preserves the monoidal structure of morphism: $F(g \otimes f) = F(g) \otimes F(f)$
- preserves the compositional structure of morphisms: $F(g \circ f) = F(g) \circ F(f)$

In a free monoidal category, applying a monoidal functor to a diagram amounts to simply providing a mapping for each generating object and morphism.

Functors are one of the most powerful concepts in category theory. In fact, the encoding, rewriting and parameterization steps of the QNLP pipeline we will present in the next chapter are implemented individually as monoidal functors, resulting in an overall functorial transformation from parse trees to tensor networks and circuits.

Chapter 3

Quantum Natural Language Processing

3.1 A Quantum model for Natural Language

The approach towards Quantum Natural Language Processing followed throughout this work is based on the DisCoCat model, proposed by Coecke et al. as a **Dis**tributional, **Co**mpositional, **Cat**egorical model of natural language [10, 11]. This mathematical framework is based on a pregroup grammar to keep track of the types and the interactions between words in a sentence through a composition of their semantic representation. The model is said to be distributional because the semantic representations of the words are vectors in some finite-dimensional vector space.

As introduced in Section 2.2, in a pregroup grammar each type p has a left p^l and a right p^r adjoint, for which the contraction equations 2.31 hold:

$$p^l \cdot p \to 1 \qquad p \cdot p^r \to 1$$

Adjoints are, thus, used to express the expected input or output of a word. Considering, for instance, the sentence "service is very good", whose diagrammatic representation is showed in Figure 3.4 and will be further illustrated in the next section. n corresponds to a noun or a noun phrase and s to a sentence. The adjoints n^r and n^l indicate that a noun is expected on the left or on the right, respectively, so that the adjective "good" of type $n^r s$ requires a noun to its right to return a sentence type. This makes sense intuitively: an adjective and a noun alone form sentence. Applying the reduction shown above we end up with a properly formatted sentence:

$$\begin{aligned} n \cdot (n^r \cdot s \cdot s^l \cdot n) \cdot (n^r \cdot s \cdot s^l \cdot n) \cdot (n^r \cdot s) &\to n \cdot (n^r \cdot 1 \cdot n) \cdot (n^r \cdot 1 \cdot n) \cdot (n^r \cdot s) \\ &\to n \cdot 1 \cdot 1 \cdot (n^r \cdot s) \\ &\to 1 \cdot s \\ &\to s \end{aligned}$$

The transition from pregroups to vector space semantics, which is what DisCoCat adds to pregroup grammar, is achieved by a mapping that sends atomic types to vector space ($n \to \mathcal{N}$ and $s \to \mathcal{S}$) and composite types to tensor product spaces, for instance "is" in the previous example

$$n^r \cdot s \cdot s^l \cdot n \to \mathcal{N} \otimes \mathcal{S} \otimes \mathcal{S} \otimes \mathcal{N}$$

Therefore, each word can be seen as a specific state in the corresponding space defined by its grammatical type, i.e. a tensor, the order of which is determined by the number of wires emanating from the corresponding box. A concrete instantiation of the diagram requires the assignment of dimensions (which in the quantum case amounts to fixing the number of qubits) for each vector space corresponding to an atomic type as further discussed in section 3.2.3.

This theory, as hinted in Section 1.3, can be "naturally" recasted in quantum computational terms, as showed by Coecke et al. [15]. Given a string of words $w_1, ..., w_N$ we can associate to them a quantum state $|\psi_{w_i}\rangle \in \mathbb{C}^{k_i 2}$ where the dimension of this space depends on its grammatical type. We then tensor these states together:

$$|\psi_{w_i}\rangle \otimes ... \otimes |\psi_{w_N}\rangle$$

In this scenario, all the words are still disentangled. Let's consider the sentence "Passenger books flight". In the diagrammatic representation, that we will present more extensively afterwards, it corresponds to the following scenario:

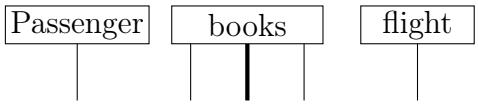


Figure 3.1: Diagrammatic representation of three unentangled states associated to the sentence "passenger books flight"

In other words, it doesn't exist any interaction between them. To introduce the semantics of the sentence, we exploit the fact that cups and caps, introduced in the

previous chapter, can be modeled in Dirac notation as Bell states

$$\langle Bell| = \langle 00| + \langle 11| \qquad |Bell\rangle = |00\rangle + |11\rangle$$

which are, in diagrammatic formalism

$\langle Bell| =$  $|Bell\rangle =$ 

The final state is mathematically obtained applying a map $f_{\mathcal{G}}$ corresponding to a pregroup grammar as follows

$$f_{\mathcal{G}}(|\psi_{w_i}\rangle \otimes ... \otimes |\psi_{w_N}\rangle)$$

that we can construct as follows

$$\langle Bell| \otimes \mathbb{I} \otimes |Bell\rangle$$

The resulting state, in quantum computational formalism, is thereby

$$|\psi_{n_s\ t_v\ n_o}\rangle = (\langle Bell| \otimes \mathbb{I} \otimes |Bell\rangle) \circ (|\psi_{n_s}\rangle \otimes |\psi_{t_v}\rangle \otimes |\psi_{n_o}\rangle) \tag{3.1}$$

which produces the following string diagram:

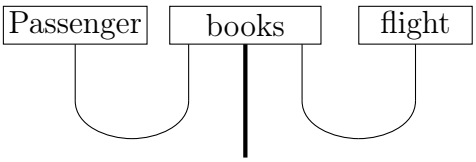


Figure 3.2: Non-annotated string diagram associated to the sentence "passenger books flight"

Equation 3.1 shows how the complexity of the Dirac notation grows quickly with the length of the sentence. The diagrammatic notation, on the other hand, proves more expressive and effective, as we will show in the following sections.

3.2 The QNLP pipeline

This section describes the sequence of steps involved in the training of a QNLP model using the DisCoCat framework. The proposed pipeline has been described

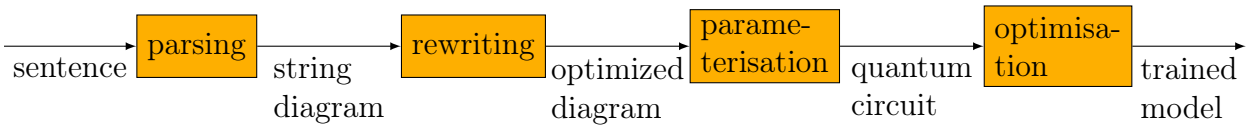


Figure 3.3: The general QNLP pipeline

in the lambeq paper [12], extensively used in this work and further presented in Section 3.3. An high-level overview is showed in Figure 3.3.

The core idea is to produce a diagrammatic representation of a sentence (called a string diagram), map it to a parameterized quantum circuit and embed this in a model architecture to be trained for some machine learning task and eventually used to make inference on unseen data.

3.2.1 Parsing

The DisCoCat model of natural language is based on string diagrams, i.e. a diagrammatic representations of parts of text based on category theory and rooted on the idea to build connections between the words according to their grammatical role in the sentence [16].

Figure 3.4 shows the diagram generated for the sentence "Service is very good" – taken from one of the datasets we will describe in the next chapter – using Bobcat, a grammar-aware parser.

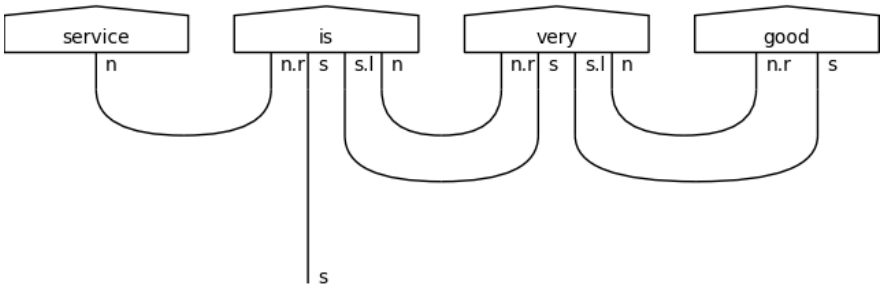


Figure 3.4: DisCoCat string diagram for the sentence *service is very good* using Bobcat parser

This is however not the only possibility, as multiple approaches exist towards parsing [17, 18]. If the text data we are dealing with is not grammatically correct (e.g. social media language), a grammar-aware approach might introduce a lot of noise, resulting in a degradation of the performances. In such a scenario, resorting to a linear reader – which only uses the *s* category – proves more beneficial. Figure 3.5 shows the parsing using a *spiders* reader, a formalism derived from Frobenius algebra.

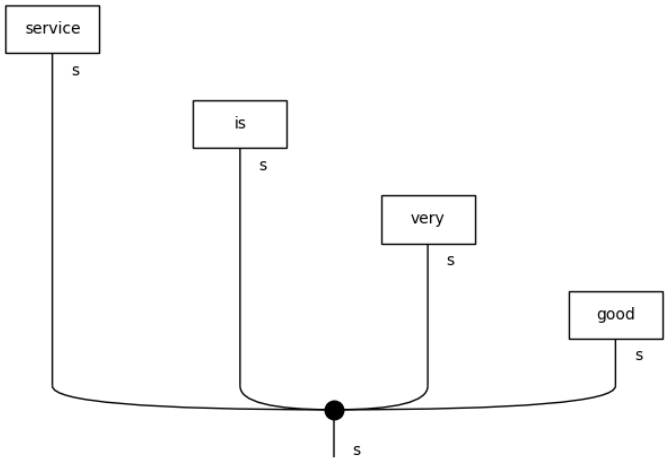
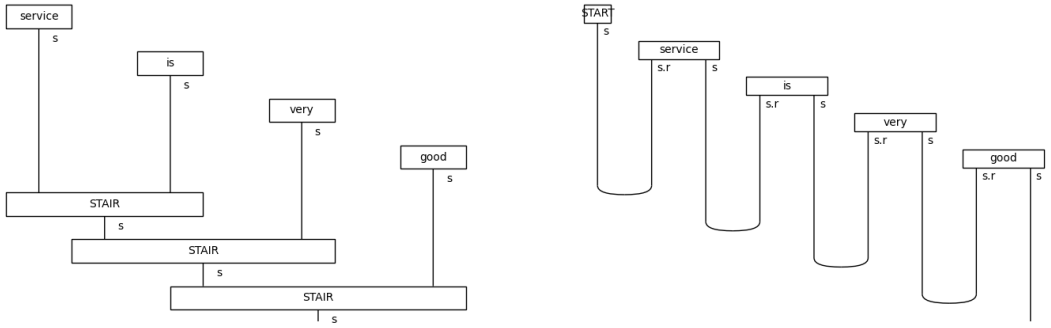


Figure 3.5: String diagram for the sentence "service is very good" using Spiders Reader

Figure 3.6 shows the parsing using two other parsing approaches: a stairs reader, which encodes a sentence in a recurrent stairs-like fashion and a cups reader, which encodes the sentence as a sequence of cups.



(a) stairs reader (b) cups reader

Figure 3.6: Stairs and Cups parsing for the sentence "service is very good"

3.2.2 Rewriting

Rewriting is an optional step of the pipeline, where the string diagram can be simplified and optimized for resource constraints, without losing its expressiveness. This is especially relevant since, as we will explain in Section 3.2.3, the diagram will be associated to a quantum representation and quantum computers are nowadays quite limited in terms of number qubits.

Figure 3.7 shows how to simplify the string diagram showed in 3.4 using the *auxiliary* rewriting rule, which replaces the auxiliary verbs with caps.

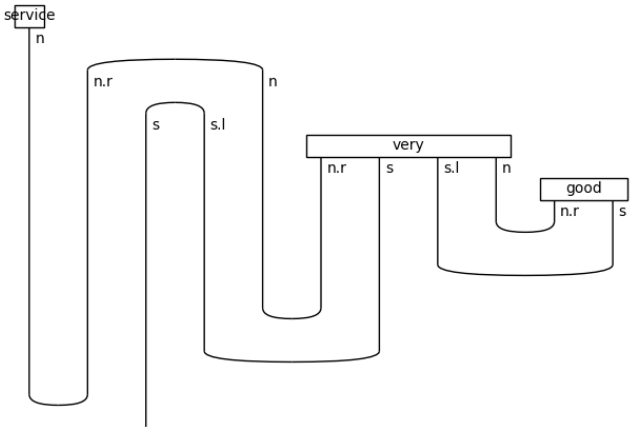


Figure 3.7: Rewritten string diagram using the *auxiliary* rewriting rule

The diagram can be further simplified applying the snake equations to yank the wires, as shown in Figure 3.8.

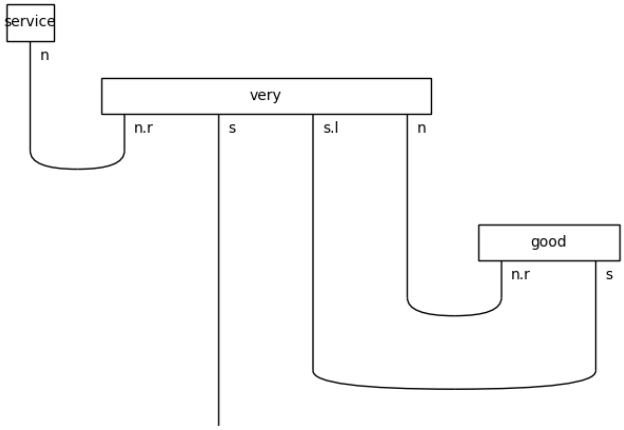


Figure 3.8: Normalized rewritten string diagram using the *auxiliary* rewriting rule

From the mathematical perspective, this step of the pipeline is performed applying a monoidal functor.

3.2.3 Parameterization

Up to this stage of the pipeline, we only dealt with classical computing. Parameterization is where quantum comes into play. In fact, the string diagram is now converted into a concrete parameterized quantum circuit via an ansatz. Ansätze

are maps defining concrete quantum structures for each box and wire in the string diagram, defining hyperparameters such as

- the number of qubits associated to each type
- the number of single qubit rotations
- the number of layers

The output of the ansatz is thereby a parameterized quantum circuit. Figure 3.9 shows the quantum circuit obtained parameterizing the string diagram showed in Figure 3.4 via a IQP (Instantaneous Quantum Polynomial) ansatz [19] using the following configuration

- 1 qubit for the n and s types
- 2 single qubit rotations
- 2 layers

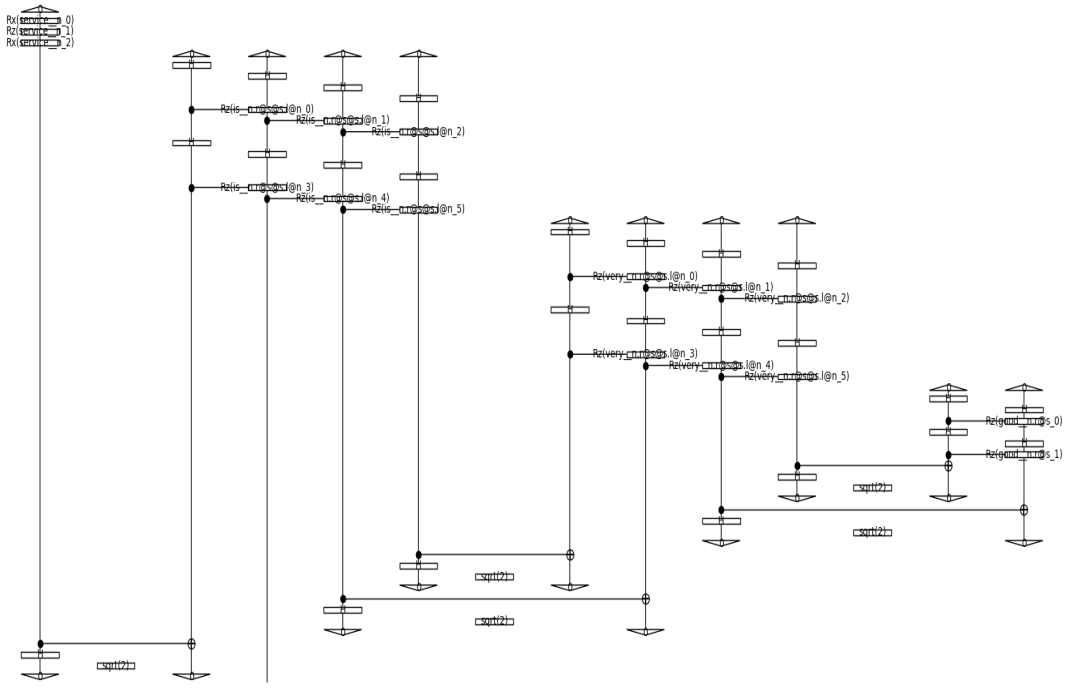


Figure 3.9: Discopy circuit produced after parameterization via IQP Ansatz

In DisCoPy [20, 21], each up-pointing arrow (state) represents an input qubit, whereas each down-pointing arrow (effect) a measurement; each box corresponds to a quantum gate while the bridges between each qubit line are controlled negations (cNOTs). In general, for m qubits, this consists of all m qubits initialized in $|0\rangle$ followed by d many IQP layers, where each layer consists of an H gate on each qubit composed with $m-1$ controlled R_z gates connecting adjacent qubits.

Figure 3.10 shows the same parameterized quantum circuit converted to Qiskit. It's important to notice that each qubit is eventually measured but the one associated to the floating wire(s), as it carries the output given to the user for its task, thus allowing the usage as a subroutine.

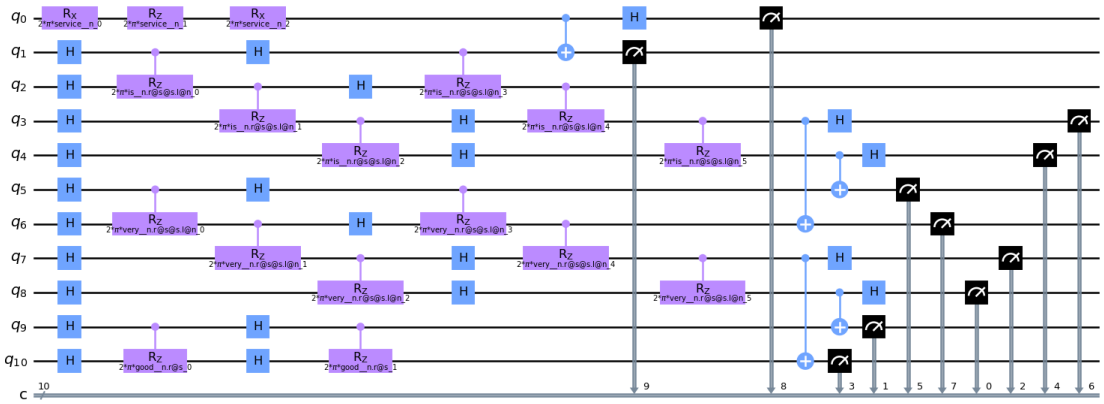


Figure 3.10: Qiskit circuit produced after parameterization via IQP Ansatz

Eventually, Figure 3.11 highlights how the parameterized quantum circuit relates to the original string diagram.
We can notice how each box contains the parameters associated to each symbol of the string diagram. The 1:1 mapping of the string diagram to the parameterized quantum circuit is not exclusive of the full grammatical parsing. For instance, Figure 3.12 shows how the string diagram obtained using a spiders reader (Figure 3.5) relates to its parameterized quantum circuit obtained using the IQP ansatz with 3 single qubit rotations and 1 layer.

3.2.4 Optimization

Optimization is where machine learning actually comes in play. As for classical models, the procedure boils up to finding a suitable parameter assignment θ such that the loss landscape described by the function $\mathcal{L}(\theta)$ is minimized.

In classical Machine Learning, the prevalent methods to find such an optimal assignment are gradient-descent and its derivations (Adam, Nesterov, etc.), which evaluate the parameter vector at iteration t as follows

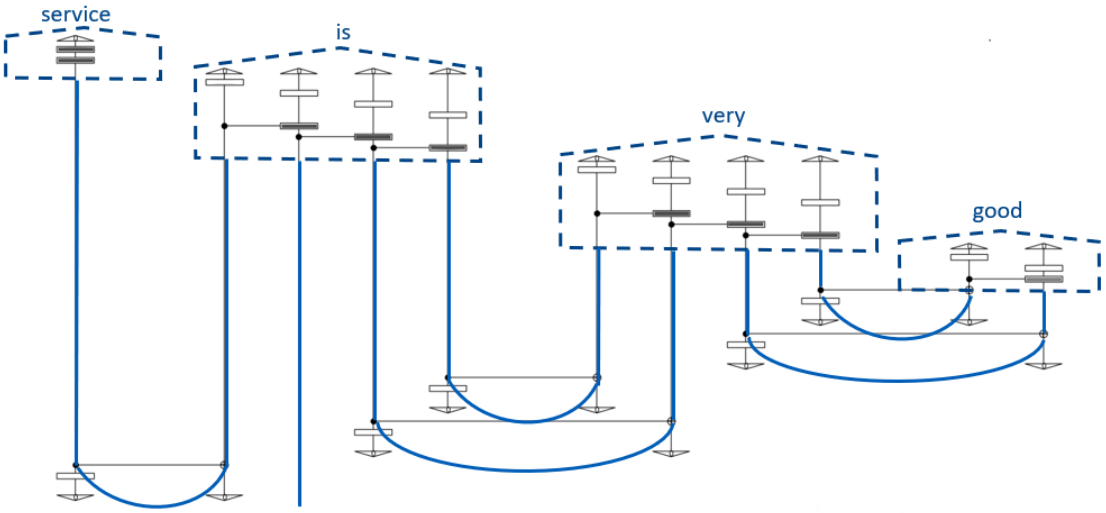


Figure 3.11: Relationship between the String diagram and the parameterized quantum circuit

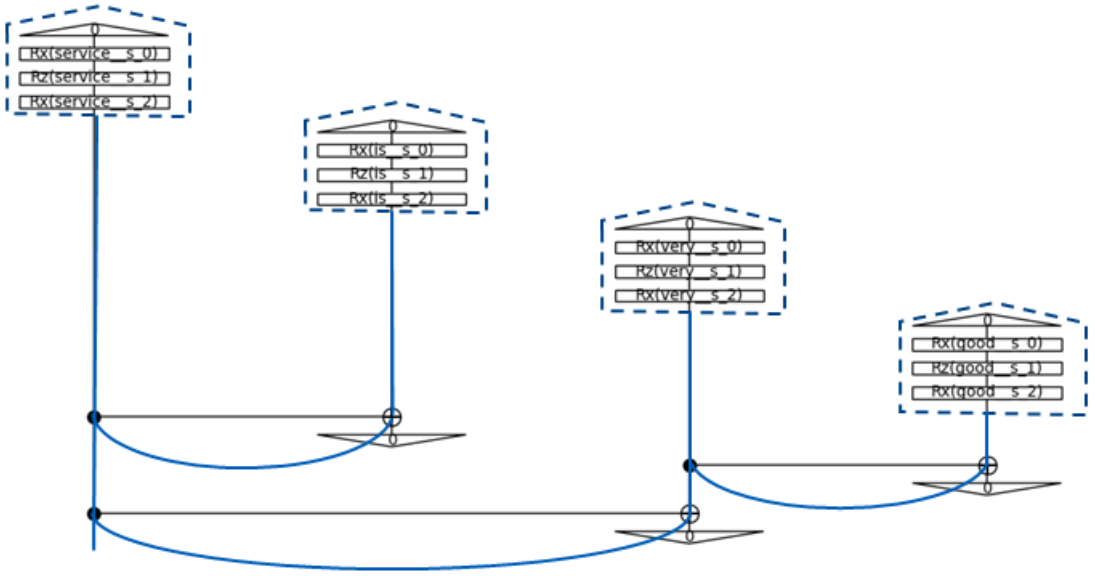


Figure 3.12: Relationship between the String diagram and the parameterized quantum circuit for spiders reader

$$\theta_{t+1} = \theta_t - \alpha_t \nabla \mathcal{L}(\theta)|_{\theta=\theta_t} \tag{3.2}$$

being α the **learning rate** or **step** of the optimizer and $\nabla\mathcal{L}(\theta)$ is the gradient of the loss function. This process is usually repeated for a fixed number of iterations or until some constraint is met.

Nevertheless, in Quantum Machine Learning, this approach usually proves computationally expensive as it would require computing the gradient of each parameter for each circuit at each step [12, 1]. More common approaches in quantum machine learning involve using gradient-based **classical** optimizer which computes an approximation of the gradient through some numerical procedure without requiring us to evaluate analytically the gradient circuit. The approach we are going to use in this work is called **SPSA** (Simultaneous Perturbation Stochastic Approximation) [22]. Just like with gradient-based methods, SPSA starts with an initial parameter vector θ_0, iteratively updated as follows

$$\theta_{t+1} = \theta_t - \alpha_t \hat{g}_t(\theta_t) \tag{3.3}$$

where, differently from 3.2, $\hat{g}_t$ is an **estimate** of the gradient at iteration t based on a prior measurement of the cost function.

Another relevant feature of this algorithm is its robustness to any noise that may occur when measuring the function $\mathcal{L}$. Let's consider a function $y(\theta) = \mathcal{L}(\theta) + \epsilon$, where ϵ is a perturbation we take into account in the output. The gradient estimation produced by SPSA at each iteration is expressed as

$$\hat{g}_{ti}(\theta_t) = \frac{y(\theta_t + c_t\Delta_t) - y(\theta_t - c_t\Delta_t)}{2c_t\Delta_{ti}} \tag{3.4}$$

being c_t a positive number and $\Delta_t = (\Delta_{k_1}, ..., \Delta_{k_p})^T$ a perturbation vector whose components are randomly generated at each iteration using a zero-mean distribution. It is this perturbation that makes SPSA robust to noise — since every parameter is already being shifted, additional shifts due to noise are less likely to hinder the optimization process. In a sense, noise gets "absorbed" into the already-stochastic process.

3.2.5 Model-level view

The above steps refer to the process each data point of the dataset goes through, from a sentence to a parameterized quantum circuit. In this section we aim at providing a model-level view, highlighting how we actually train a QNLP model and predict on unseen data.

Algorithm 1 describes the training process. Each model embeds the parameterized quantum circuits associated to the training set, obtained following the steps described above. In addition, for each symbol in the vocabulary V_{train} the model contains a parameter vector θ_i.

The optimization process aims at finding the final assignments for the parameters, similarly to what we would do for a classical Deep Learning model.

Algorithm 1 Quantum Model Training

procedure TRAIN(X_{train}, y_{train})
  $raw\_diagrams \leftarrow [parser(x_i)$ **for each** x_i *in* $X_{train}]$
  $diagrams \leftarrow [rewriter(x_i)$ **for each** x_i *in* $X_{train}]$
  $circuits \leftarrow [ansatz(d_i)$ **for each** d_i *in* $diagrams]$
  for w_i *in* V_{train} **do**
    $\theta_i^0 \leftarrow$ initialize_params()
  end for
  $\theta^0 \leftarrow (\theta_1^0, ..., \theta_{|V_{train}|}^0)$
  for $t \leftarrow 1$ ... N_{epochs} **do**
    $\theta^t \leftarrow \text{SPSA}(\mathcal{L},\ \theta^{t-1},\ circuits,\ y_{train})$
  end for
  $weights \leftarrow \{w_i : \theta_i\ for\ w_i\ in\ V_{train}\}$
  return weights
end procedure

Algorithm 2 describes the process to evaluate a trained model on unseen data. The test set is again converted into a string diagram and then into a quantum circuit. Now circuits are fed with the weights learned during the training phase and their measured outputs used to predict labels matched against some evaluation metric depending on the specific problem we are tackling, e.g. accuracy.

Algorithm 2 Quantum Model Evaluation

procedure EVALUATE($weights$, X_{test}, y_{test})
  $test\_diagrams \leftarrow [parser(x_i)$ **for each** x_i *in* $X_{train}]$
  $diagrams \leftarrow [rewriter(x_i)$ **for each** x_i *in* $X_{train}]$
  $circuits \leftarrow [ansatz(d_i)$ **for each** d_i *in* $diagrams]$
  for w_i *in* V_{test} **do**
    $\theta_i \leftarrow$ weights$[w_i]$
  end for
  $\theta_{test} \leftarrow (\theta_1, ..., \theta_{|V_{test}|})$
  $\hat{y} \leftarrow model(circuits, \theta_{test})$
  score $\leftarrow eval\_metric(y_{test}, \hat{y})$
  return score
end procedure

3.3 Lambeq

Lambeq [12] is an open-source high-level Python toolkit for quantum natural language processing built on top of DisCoPy [20, 21] and specifically designed to allow

people run QNLP experiments without dealing with the low level details of the pregroup grammar modeling.

It covers a wide range of experimental setups in three broad categories, summarized in Figure 3.13:

- quantum simulations on classical hardware
- run on real quantum hardware
- evaluation of tensor networks on classical hardware

In this work we never make use the third setup as, despite tensor network "emulate" the way quantum hardware computes, it is a fully classical approach. We use simulation approaches both to validate the model and to test them beyond the limitations of current quantum hardware and eventually assess the performances yielded when run on real devices.

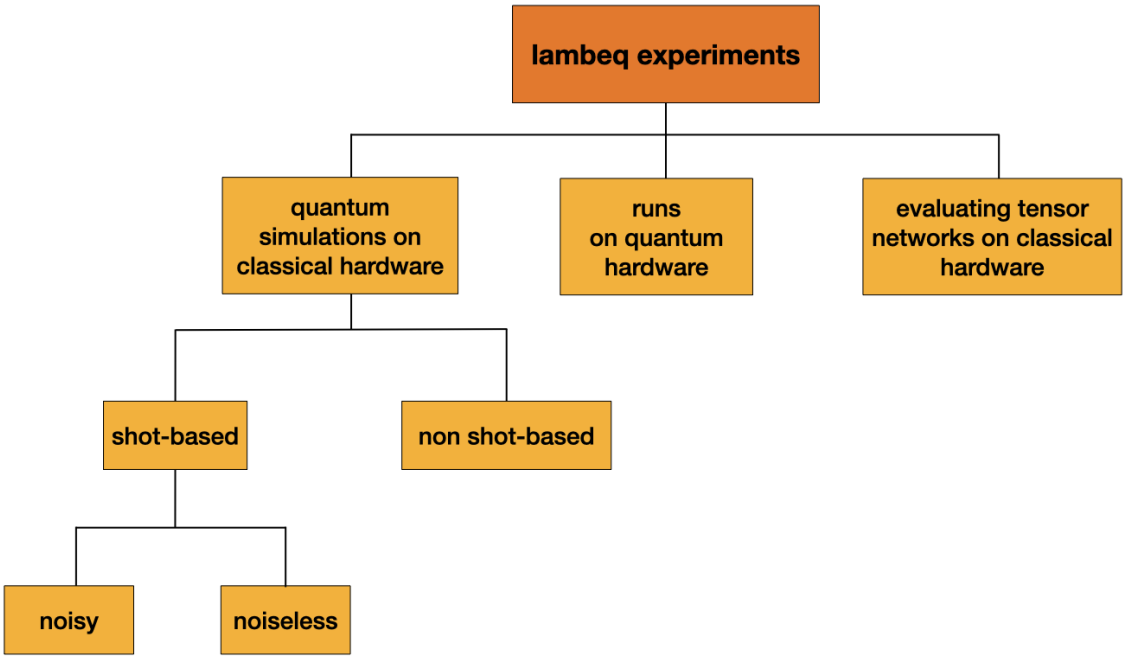


Figure 3.13: Hierarchy of experimental setups in lambeq

3.3.1 Quantum Simulations on Classical Hardware

Computing with NISQ (Noisy intermediate-scale Quantum) devices is nowadays constrained by the several limitations that real hardware suffers from, especially the limited number of qubits, the noise and the systematic need of error correction protocols. In addition, being the various quantum devices available on the market

quite heterogeneous, only relying on the results yielded by a particular platform might be deceiving. For such reasons, simulation is always the best option during the first stages of modeling. To this regards, lambeq supports two different options:

- exact **noiseless** simulation using Jax [23]
- **noisy** shot-based simulation

The former one is a non shot-based approach where quantum gates are associated to their algebraic representation — meaning complex-valued tensors — and the measured probability distribution comes out of tensor contraction. This approach allows to run QNLP experiments in a scenario where a shot-based simulation would be impossible because of the required number of qubits whose simulation on classical hardware would make it run out of memory.

The latter one is the usual probabilistic approach used in quantum computation: the same circuit is ran many times (or shots), exploiting statistical aggregation, using a noisy quantum simulator, e.g. Qiskit Aer.

3.3.2 Run on Real Quantum Hardware

Once the results of the simulation are consistent, the model can be tested on real quantum hardware, taking however into account that the size of the quantum computers offered for free by the various providers is quite limited. Lambeq exploits **tket** [24] as a quantum compiler, which is integrated with all the major providers (IBM Quantum, Google Cloud, Microsoft Azure, Amazon Braket, etc.). In this work, we use IBM and Microsoft Azure's quantum cloud platforms.

3.3.3 Lambeq compared to quantum machine learning

In this last section we further highlight how lambeq – and, in general , this approach towards quantum language processing – inherently differs from "traditional" quantum machine learning. The latter usually relies on (classical) neural network which learned some kind of vector representation of the vocabulary that is the encoded in some qubit representation and used to carry the computation inside a quantum machine. The output is afterwards measured and optimized via a classical optimizer. In QNLP and specifically in lambeq, we focus on the sentence representation and we directly output a quantum representation that matches the string diagram and we learn the parameters we eventually use to make inference on unseen text for which we will produce a representation to be evaluated. An high-level view of this comparison is shown in Figure 3.14.

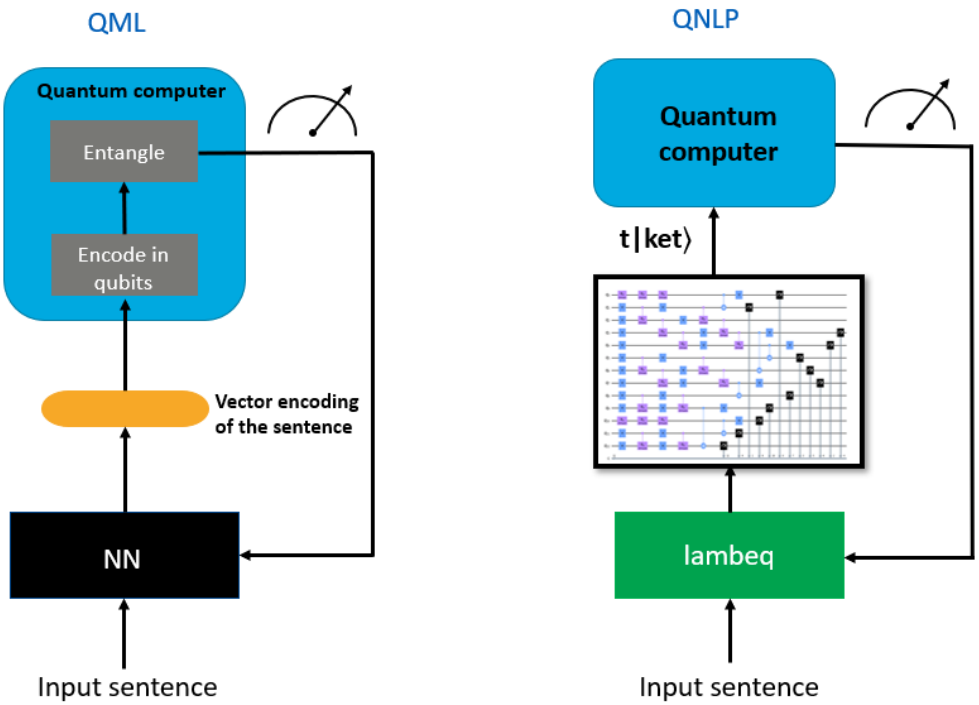


Figure 3.14: High-level comparison of "traditional" Quantum Machine Learning vs Lambeq

3.4 Quantum native and quantum advantage

In the introductory part of this thesis, we hinted at the fact that a quantum machine is naturally suited for the distributional compositional categorical approach towards natural language, alluding at the similarity between the way it carries the computation and the framework's mathematical formulation — most notably in terms of diagrams — as well as the exponential complexity of the classical implementation. Given the concepts presented in this chapter, we can now describe two additional values:

- **NISQ devices are QNLP-friendly**: in the light of the variational quantum circuits paradigm, when used to encode classical data on quantum hardware, accounting for linguistic structure comes as a **free lunch** while on classical hardware it looks to be exponentially expensive. We showed this visually in Figure 3.11.
- **QNLP is meaning-aware**: regardless of the performances to solve some kind of tasks, which is often a matter of engineering, the flows of meanings in QNLP are clearly exposed and understood, due to the linguistic structure. This is true for all the parsers presented above.

In light of these considerations, it emerges that QNLP is not just another quantum counterpart to some classical method. Instead, already at a foundational level, it's a brand new approach inspired by computational linguistics and specifically tailored to quantum hardware.

Chapter 4

Customer Feedback Analysis

In this chapter we will explore the suitability of the quantum natural language processing models described in the previous chapter on two sentiment analysis datasets related to customers' feedback in the travel domain. The first one is the popular Twitter US Airlines dataset[1], containing data scraped from February of 2015 on six major US airline companies. The second one is an internal dataset of hotel reviews, enjoying a good grammar. The purpose of this use case is to first apply a modeling approach where the grammatical constraints are only loosely enforced, as tweets are in general very far from plain English, and then to try the full grammatical parsing approach on the second dataset. In doing so, we want to also assess the performance yielded on real quantum hardware and the requirements in terms of size of the problem.

4.1 Sentiment Analysis on Twitter Airline Dataset

This original dataset consists of 3 classes corresponding to the passenger's sentiment on their experience traveling with one of the 6 major American Airlines (Virgin America, United, US Airways, American, Southwest, Delta). To ensure a fair comparison with the hotel reviews use case, which includes applying the very same approach at all the stages of the pipeline after the parsing, the dataset has been binarized. A wordcloud visualization is shown in Figure 4.1

4.1.1 Modeling the problem

In this section we describe how the problem has been modeled practically. Firstly, we preprocess the dataset removing tags and nicknames and converting slangs and

[1] https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment

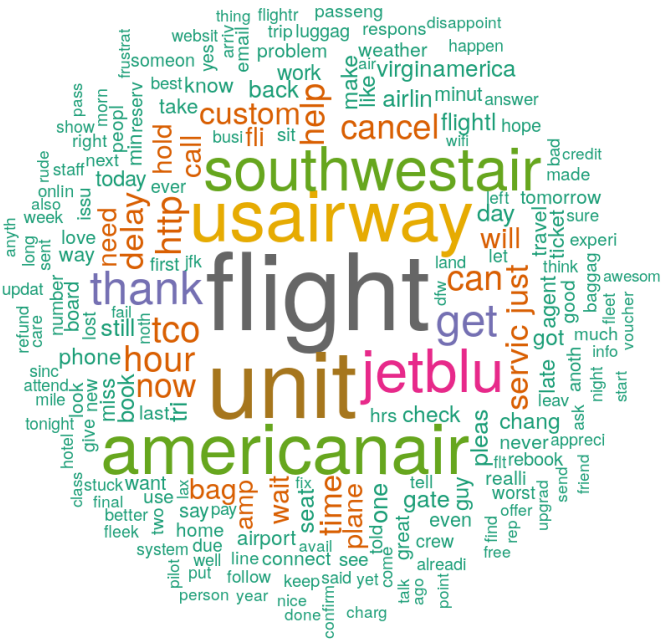


Figure 4.1: Wordcloud visualization of the dataset

abbreviation into plain English using the TextBlob library[2]. This step aims at slightly improving the quality of the text we are dealing with. In addition, emoticons and emojis are converted into the corresponding word associated to the sentiment they're trying to convey. Eventually, to reduce the noise introduced by infrequent words — which is often the case in the context of tweets — we apply a functor to replace those with an *UNK* token

```
from discopy import Word
from discopy.rigid import Functor

def replace(box):
        if isinstance(box, Word) and dataset.count(box.name) < 5:
                return Word('unk', box.cod, box.dom)
        return box

replace_functor = Functor(ob=lambda x: x, ar=replace)
```

Differently from standard practices in NLP, we don't apply any lemmatization or stemming nor remove the stopwords, as they might prove useful in our parsing

[2]https://github.com/sloria/TextBlob

approach. We randomly split the training set into a training and a validation set (70%:30%). To convert each tweet in a string diagram, we employ spiders, a linear reader:

```
from lambeq import SpidersReader

reader = SpidersReader()

train_diagrams = reader.sentences2diagrams(train_data.text)
val_diagrams = reader.sentences2diagrams(val_data.text)
```

As regards the targets, we will exploit a one-hot encoding. This is a rather convenient representation for our models because the output of the quantum measurement is nothing but a vector of probabilities for the measured qubit (remember that $|0\rangle = [1, 0]$, $|1\rangle = [0, 1]$).

```
import numpy as np
from sklearn.preprocessing import OneHotEncoder

_sentiment_categories = np.unique(train_data.target).reshape((-1, 1))
enc = OneHotEncoder().fit(_sentiment_categories)

def encode_label(label: str):
        """
        Encodes sentiment to a one-hot label
        given an input string
        """
        try:
            # wrap the string label into a 2d array
            # to comply with sklearn requirements
            label = [[label]]
            # transform the label to a one-hot vector
            return enc.transform(label).toarray().tolist()[0]
        except ValueError:
            print(f"Label {label} is not valid."
                  f"Value must be one of {_sentiment_categories}")
```

Each diagram is then converted into a parameterized quantum circuit using an IQP Ansatz and the diagram optimized using the `remove_cups` functor. Since we need to measure a bidimensional output of probabilities $[p_0, p_1]$ we need to allocate one qubit to the measured wire. This is due to the fact that the vector spaces in which the noun and sentence embeddings live have dimension $N = 2^{q_n}$ and $S = 2^{q_s}$, where

q_n and q_s are the number of qubits associated those embeddings. In the general case, when using a linear reader, to represent a sentence of w words we need:

$$n_{qubits} = w \cdot q_s \tag{4.1}$$

since the measured wire needs to be d_{out} dimensional, where d_{out} is the desired output size, we need to enforce that

$$q_s = \min_{q \in \mathbb{N}^+} q \quad \text{s.t.} \quad d_{out} \leq 2^q \tag{4.2}$$

which implies that:

$$n_{qubits} = w \cdot \lceil log_2(d_{out}) \rceil \tag{4.3}$$

where the ceiling is applied to take into account the constraints of Equation 4.2. To keep the depth of the circuit as small as possible, we set the number of layers $k = 2$.

```
from lambeq import IQPAnsatz, AtomicType, remove_cups

S = AtomicType.SENTENCE
N = AtomicType.NOUN

ob_map = {
        S: 1,
        N: 1
}

ansatz = IQPAnsatz(ob_map, n_layers=2, n_single_qubit_rotations=3)

train_circuits = [ansatz(remove_cups(diag)) for diag in train_diagrams]
val_circuits = [ansatz(remove_cups(diag)) for diag in test_diagrams]
```

We use accuracy as evaluation metric and a binary cross-entropy as loss function.

```
from sklearn.metrics import accuracy_score

def accuracy(y_hat, y):
        return accuracy_score(y_true=y, y_pred=np.round(y_hat))

def cross_entropy(y_hat, y, epsilon=1e-12):
        y_hat = np.clip(y_hat, epsilon, 1. - epsilon)
        N = y_hat.shape[0]
        return -np.sum(y * np.log(y_hat)) / N
```

Lastly, as explained in Section 3.2.3, Simultaneous Perturbation Stochastic Approximation is the technique we use to optimize the loss landscape.

4.1.2 Exact simulation

Since the full dataset is relatively big and each tweet might be long several words, using a shot-based simulation would be impossible on a classical machine. For this reason, as first proof of concept, we resort to an exact non shot-based one, using `NumpyModel`. We use a gridsearch to tune the batch size B as well as the optimizer hyperparameters, i.e. the learning rate lr, the decay factor c while leaving the stability factor A to the recommended value $0.01 \cdot n_{epochs}$. We train the classifier for 10 epochs. The most meaningful results are summarized in Table 4.1.

learning rate	batch size	decay factor	training accuracy	validation accuracy
0.02	16	0.06	77.7%	71.4%
0.02	32	0.03	89.5 %	77.3 %
0.02	32	0.06	93.0 %	89.7 %
0.05	32	0.06	91.0 %	86.4%
0.1	32	0.06	88.4 %	85.4 %

Table 4.1: Hyperparameter tuning of the exact simulation on the Twitter US Airlines dataset

After choosing the best configuration, we compare the performances yielded with classical models, including a state-of-the art BERT uncased and recurrent-based neural networks.

Model	number of parameters	best score
Quantum model	3207	88.6 %
Pretrained BERT uncased	110M	99 %
distil BERT	66M	94%
bi-LSTM	1.4M	76.4 %
Attention LSTM	1.4M	81.4 %
Vanilla RNN	180k	57.8 %

Table 4.2: Comparison of classical models with the Quantum architecture, in terms of accuracy and number of parameters (averaging 20 runs).

We can notice that, despite losing to BERT, the quantum model outperforms all the other architectures with a significant smaller number of parameters. Furthermore, it should be noted that the uncased BERT-base model we used for this comparison has been pretrained for 1M steps on the entire English Wikipedia corpus (2500M words) and on BookCorpus [25] (800M words), for a total of 3.3B words and eventually fine-tuned on this task for 5 epochs. In this sense, it's understandable that it

outperforms any other model trained from scratch on much less data and for much less iteration steps. To further support this claim, we reported the same experiment, with the very same preprocessing, using distilBERT [26], a distilled version of BERT obtained from the bert-base-uncased checkpoint, having half of the layers and 60% of the original parameters. As we can see, it again outperforms the quantum model, however the accuracy dropped of 5% and the number of parameters is still 18000 times higher.

4.1.3 Run on IBM Oslo and Nairobi via IBM Cloud

After analyzing the results yielded by the exact non shot-based simulation, useful to understand the potential of this approach and its limitations, we take one step further into our analysis targeting real quantum hardware. Specifically, we will work on IBM Oslo and IBM Nairobi, the two biggest superconducting quantum computers available for free on IBM Cloud, with 7 qubits connected as shown in Figure 4.3 and quantum volume 32[3].

In the context of this experiment, apart from measuring the accuracy, we want to benchmark training and inference time of the platforms in order to provide further insights on the status of quantum hardware today.

Figure 4.2 shows an high-level view of how quantum hardware is accessed in the cloud. When a user submits a job to a quantum system, it enters the scheduler for the specific system, joining the pool of jobs (from all users) that are waiting to be executed on that system. The order in which these jobs are executed is, by default, determined by a fair-share formula attempting to balance the workload between different plans according to the allocated system access amount over a given time window. As a consequence, running an experiment on IBM Quantum might take several hours or days, even if the actual running time of the experiment might be in the ballpark of seconds. In the following benchmarks we will only take into account the actual running time.
In our QNLP use cases, a job is a batch of samples. Thereby we need to run

$$n_{jobs} = n_{epochs} \left\lceil \frac{|D_{train}|}{|B|} \right\rceil$$

being $|B|$ the batch size.

Given this much and the limitations imposed by Equation 4.3 and that we only have access to 7 qubits, we need to restrict our problem. Specifically, we extracted

[3]Quantum volume is a metric which allows quantum computers of different architectures to be compared in terms of overall performance. It quantifies the largest circuit of equal width and depth that the device can successfully implement

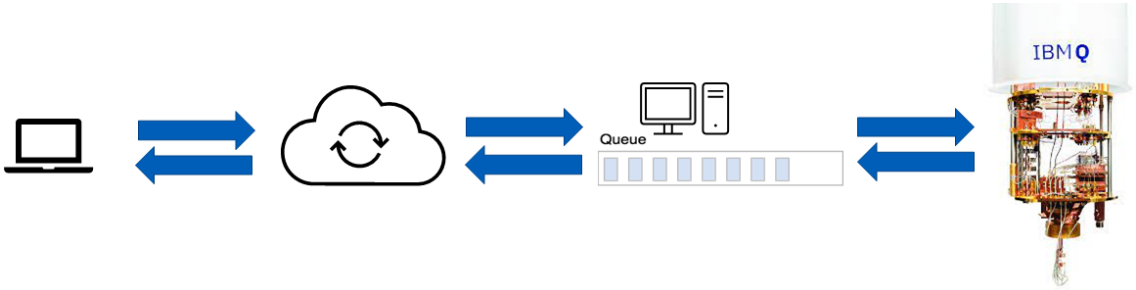
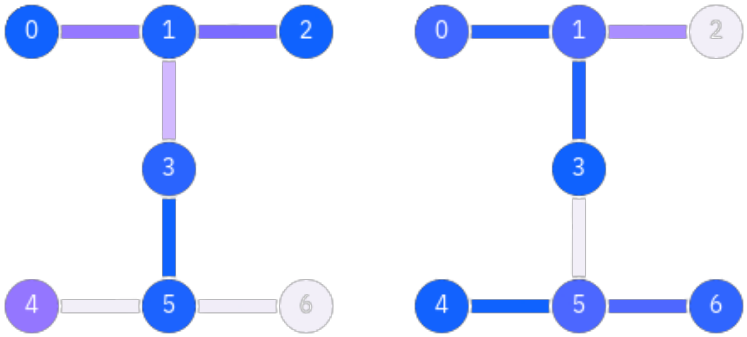


Figure 4.2: High-level model for accessing a quantum processing unit (QPU) on IBM cloud



(a) IBM Oslo (b) IBM Nairobi

Figure 4.3: IBM Oslo and Nairobi map views

from the original dataset a **balanced** subset of 250 tweets of, at most, 7 words, 50 of which are used as test set. The implementation only requires to change the backend to properly interface with IBM's platform. For instance, for IBM Oslo:

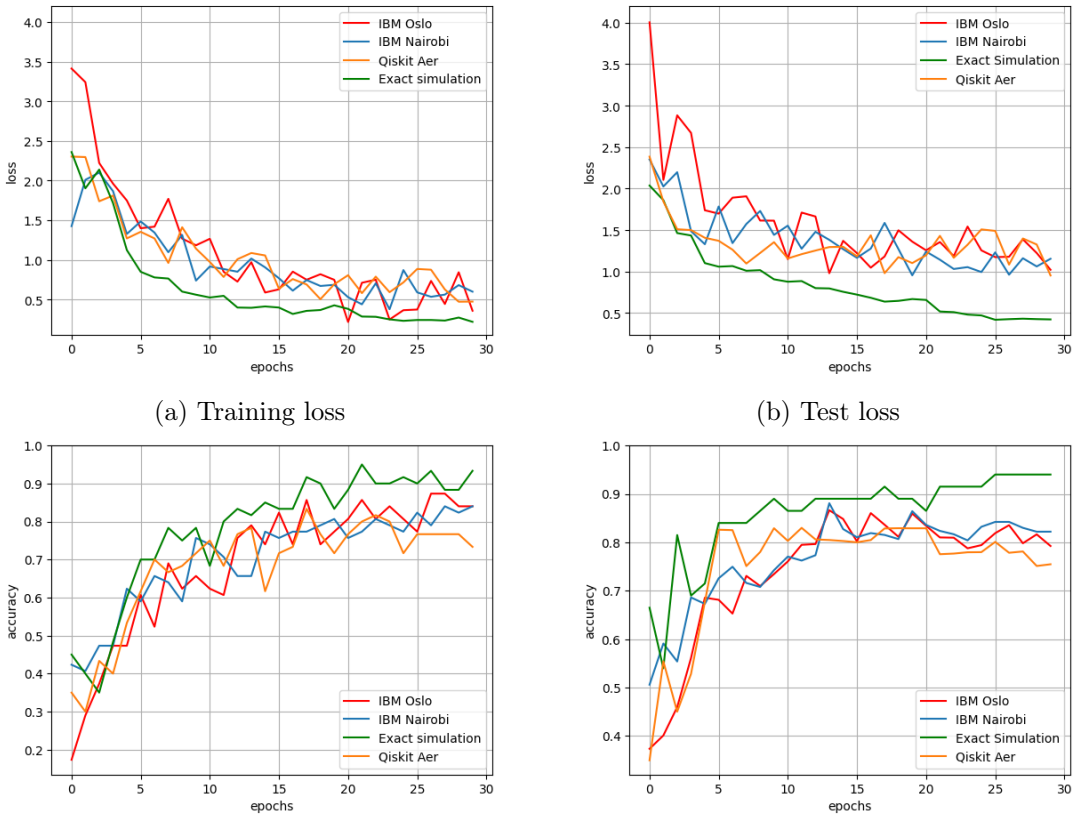
```
from qiskit import IBMQ
from lambeq import TketModel
from pytket.extensions.qiskit import IBMQBackend

# assuming the IBM API token has been cached
IBMQ.load_account()

backend = IBMQBackend(backend_name='ibm_oslo', hub='ibm-q', group='open')
backend_config = {
        'backend': backend,
        'compilation': backend.default_compilation_pass(2),
        'shots': 8192
}

```

```
15 model = TketModel.from_diagrams(circuits, backend_config=backend_config)
```

We also provide a comparison with the results yielded by a shot-based simulation using qiskit's Aer simulator.

```
1 from pytket.extensions.qiskit import AerBackend
2
3 backend = AerBackend()
```

Lastly, we want to assess how optimistic the exact simulation is. Figure 4.4 shows a comparison of the results obtained in the four scenarios averaging 5 runs[4].



(a) Training loss

(b) Test loss

(c) Training accuracy

(d) Test accuracy

Figure 4.4: Results of the quantum computation on IBM Oslo and Nairobi compared to the exact and shot-based simulations in terms of convergence and accuracy(averaging 5 runs).

[4]running experiments on quantum hardware is very time expensive because of the queuing time to access the platform, which is the reason why we only repeat the experiment 5 times

We notice that the exact simulation has a smoother trend both in terms of loss decrease and accuracy growth, whereas the other three platforms are overall aligned, with Oslo and Nairobi reaching a slightly better accuracy than the simulator both in the training and validation set. We remark that, differently from the trend one would observe on classical models, these plots are quite noisy, which is peculiar of today's quantum hardware. In all the four analyzed cases, the models are appropriately fitting the data.

Table 4.3 compares the training and inference time of the above experiments. It should be noted however that 7 qubits are not many which makes the simulation on a classical CPU competitive with real hardware. This is even more evident if one considers that bringing data from the control server to the quantum device and eventually reading the results might introduce a significant component of latency, definitely not negligible for such a small quantum device. We expect an experiment on a bigger system, e.g. 27 qubit's IBM Toronto, to show some advantages with respect to the simulated version.

System	training time	inference time (100 sentences)
Aer Simulator	32.9 min	15.0 s
IBM Oslo	32.4 min	26.8 s
IBM Nairobi	34.3 min	28.3 s
Exact Simulation	3.7 min	1.53 s

Table 4.3: Training and inference time of IBM Oslo and IBM Nairobi compared to the exact and the shot-based simulations on the Twitter US Airlines use case (averaging 5 runs).

Despite the last consideration, we never mentioned speedup in this work. In fact, the purpose of this benchmark is to provide further insights on the status of the technology today, for industrial interest and as a glimpse on its current capabilities and its future prospects on that side, conscious of the fact that today's quantum technology is still far from being able to bring the speedup theoretically provable.

4.2 Hotel Reviews

The previous use case showed how this approach is valuable and versatile even when the quality of text doesn't allow to enforce strict grammatical constraints. In the second use case we aim at probing the capabilities of a full-grammatical approach. To accomplish this goal we make use of an internal Amadeus dataset of labeled single-sentence hotel reviews consisting of 1200 binary samples. The goal is again to use classical, neural network-based models as a mean of comparison to evaluate the performances of the quantum approaches. For this task we exploit Bobcat, a

statistical CCG (combinatory categorial grammar) parser[5] [17].

```
from lambeq import BobcatParser

parser = BobcatParser(text='verbose')
```

Since most of the reviews are relatively long in terms of the number of words and considering that `BobcatParser` is slightly more demanding in terms of qubit — as it has to learn for each word its possible grammatical role in the sentence — for this use case we only resort to the exact simulation. This will also allow us to repeat the experiments more times. The modeling approach described in the previous section still applies to this case. We compare this approach with the spiders reader we used before as well as classical models. Table 4.4 shows the obtained results averaging 20 runs. A visual representation of the comparison is provided in Figure 4.5

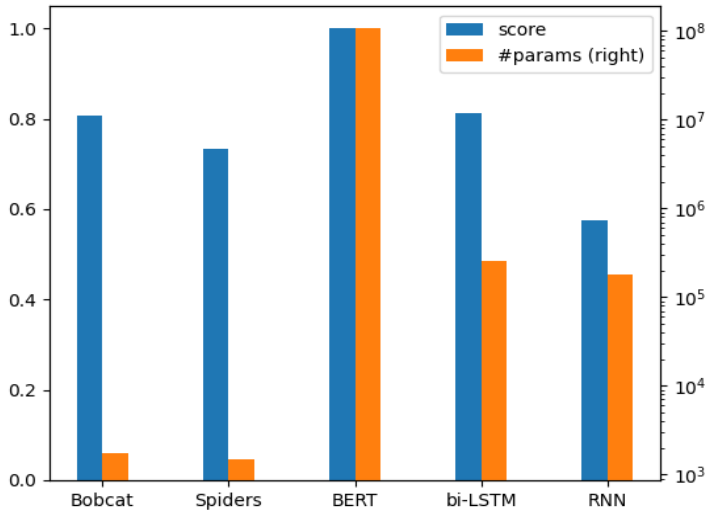


Figure 4.5: Quantum and classical models in terms of performance and number of parameters

The full grammar-aware approach outperforms spiders reader on a comparable number of parameters, which shows that grammar was an added value in this scenario. In addition, the performances yielded are comparable to the bi-LSTM with a much smaller number of parameters. On the other hand, BERT outperforms

[5]In computational linguistics, a parser is said to be *statistical* if it makes use of a procedure to search over a space of all candidate parses, and the computation of each candidate's probability, to derive the most probable parse of a sentence

Model	number of parameters	best score
Bobcat parser	1718	80.7 %
Spiders parser	1506	73.5 %
Pretrained BERT uncased	110M	100.0 %
bi-LSTM	256k	81.4 %
Vanilla RNN	180k	57.5 %

Table 4.4: Comparison of classical models with the Quantum one using 2 parsing approaches, in terms of accuracy and number of parameters (averaging 20 runs)

again all the other models we compared it, which is understandable considering that the employed dataset is quite small with respect to the size of the vocabulary, thus a pretrained model has a lot of advantage. Overall, we conclude that the grammatical approach has a lot of expressive potential, but requires to be applied to the proper use case, making sure that the structure of the text is appropriate and that the task we want to solve can leverage syntax, which is not necessarily the case for a sentiment analysis task.

Chapter 5

Passenger Intent Recognition on Chatbot Queries

Starting from the experiments and the results of the previous chapter, we will now take a significant step forward tackling a natural language understanding (NLU) task dealing with recognizing the intent of a user query, one of the main steps of a Chatbot platform pipeline. To address this task, we will use the popular ATIS dataset [1], a benchmark dataset widely used as an intent classification. ATIS Stands for Airline Travel Information System.

Within a chatbot, intent refers to the goal the customer has in mind when typing in a question or comment. While entity refers to the modifier the customer uses to describe their issue, the intent is what they really mean. For example, if a user says, "I need new shoes.", the intent behind the message is to browse the footwear on offer. Understanding the intent of the customer is key to implementing a successful chatbot experience for end-user.

It should be noted that, differently from the previous two use cases, this is a multiclass task (Table 5.1), which will require a relatively different design in the architecture as well as taking care of the depth of the architecture to allow the model to effectively learn the pattern. As for the previous use cases, we will run a reduced version of this problem on IBM superconducting quantum computers and, in addition, we will explore trapped ions quantum computers using Microsoft Azure.

5.1 Modeling the problem

As explained in the previous chapter, the output size of the measurement depends on the way we parameterize the string diagram, specifically it is equal to the number

[1] https://www.kaggle.com/datasets/hassanamin/atis-airlinetravelinformationsystem

class	example entry	percentage
atis_abbreviation	what is fare code h	3%
atis_aircraft	what kind of aircraft is used on a flight from cleveland to dallas	1%
atis_airfare	round trip fares from pittsburgh to philadelphia under 1000 dollars	9%
atis_airline	could you please tell me the airlines that fly from toronto to san diego	3%
atis_flight	i need to know information for flights leaving dallas on tuesday evening and returning to atlanta	74%
atis_flight_time	what is the arrival time in san francisco for the 755 am flight leaving washington	2%
atis_ground_service	show me the car rentals in baltimore	5%
atis_quantity	please tell me how many nonstop flights there are from boston to atlanta	2%

Table 5.1: ATIS dataset overview with example entries and distribution

of qubits allocated to the s wire. In this scenario, we need to be able to discriminate among 8 classes. Recalling again that the size of of the vector space in which the sentence embedding lives is $S = 2^{q_s}$, we need 3 qubits, i.e. $q_s = 3$. In this fashion, the possible measurement outcomes are $|000\rangle$, $|001\rangle$, ..., $|111\rangle$.

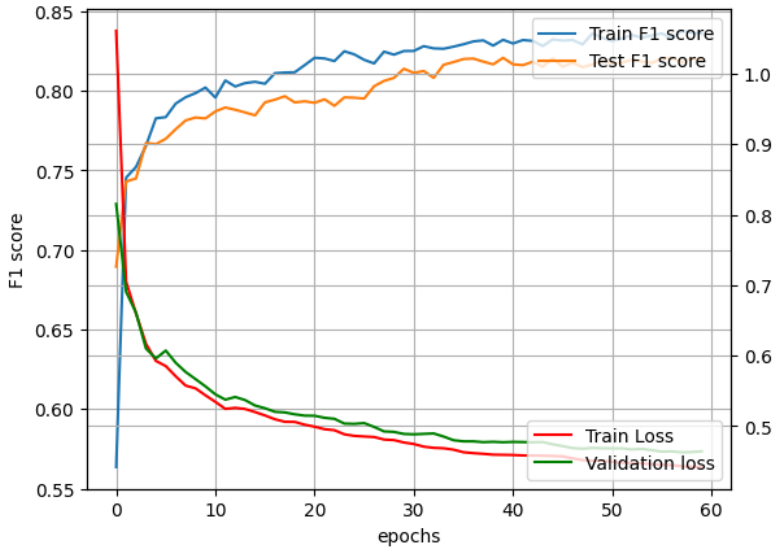
```
ansatz = IQPAnsatz({S: 3}, ...)
```

Since each measurement accounts for one qubit, the final output size of a measurement is going to be a tuple of q_s vectors of size 2, which additionally requires to squash the probability vector before computing the loss and the evaluation metrics. Despite the problem grew significantly in terms of size and complexity (number of classes) with respect to the previous one, we notice empirically that 2 layers and 3 parameters per gate suffices for learning.

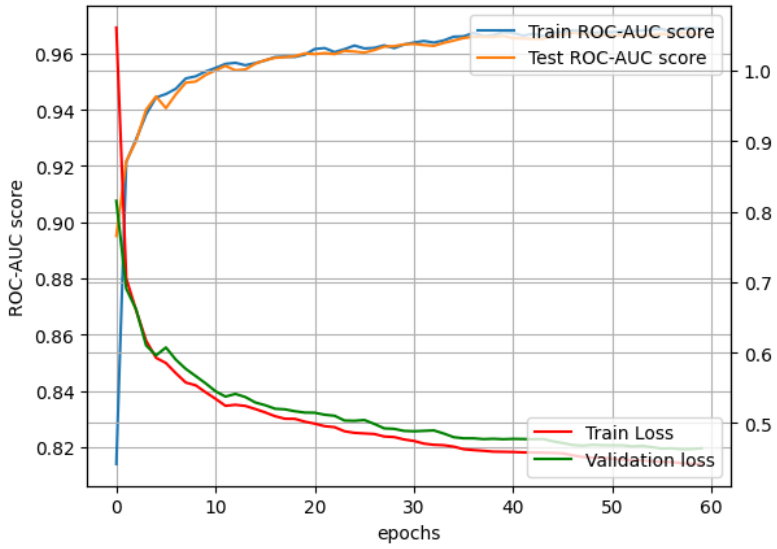
Lastly, being this dataset extremely unbalanced, we will use F1-score and ROC-AUC score as metrics and a (categorical) cross-entropy as loss function, resorting again to SPSA for the optimization procedure.

5.2 Exact simulation

As observed in the previous chapter, a shot-based simulation wouldn't be feasible on a classical CPU when working with this kind of datasets. This is even more true in this scenario, as we have deeper circuits and a much higher requirement in terms of qubit (recall that simulating q qubits requires probing 2^q combinations). For such reasons, in the first part of this use case, we resort to exact simulation using `NumpyModel`. The results yielded in terms of F1 score and ROC AUC score are reported in figure 5.1.



(a) F1 score



(b) ROC AUC score

Figure 5.1: F1 score, ROC AUC score and cost functions produced by the exact simulation

5.3 Run on IBM Oslo and IBM Nairobi on IBM Cloud

As done in Chapter 4, we want again to assess the performances of real hardware on this task, with a novelty: we will compare the results obtained on superconducting qubits technology with trapped ions topological qubits, object of the next section.

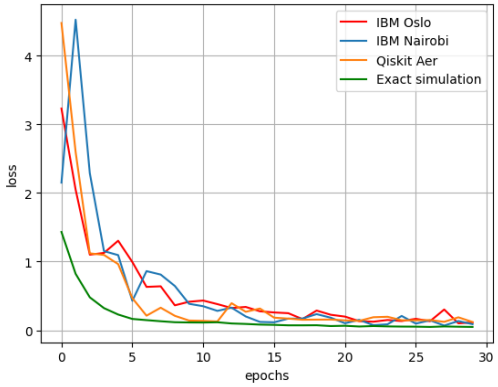
To achieve this much, we need again to shrink the dataset. Looking again at

the constraints imposed by Equation 4.3, we are forced to regress to a binary task. In fact, with only 7 qubits, a 3 or 4 classes classification task would allow at most 3 words per query, which is not enough for any of those available in the ATIS dataset. We gather 200 queries from the training set respecting this constraints and 82 from the test set, trying to keep the original proportion of the classes and we resort to Qiskit Aer simulator to tune the model.

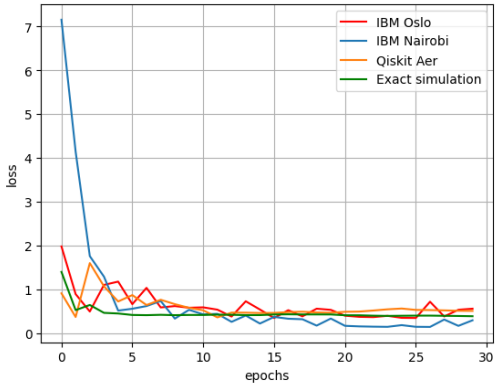
learning rate	batch size	f1 score	roc-auc score	precision	accuracy
0.001	32	0.63	0.67	0.66	0.60
0.01	32	0.88	0.93	0.91	0.90
0.01	16	0.91	0.97	0.99	0.93
0.05	32	0.72	0.81	0.75	0.72
0.05	16	0.58	0.61	0.54	0.61

Table 5.2: Hyperparameter tuning of the shot-based simulation using t|ket⟩ compiler and Qiskit Aer

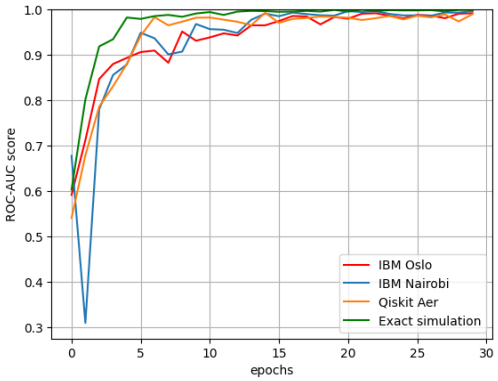
Eventually, we use the best hyperparameters to run the experiment on IBM Oslo and IBM Nairobi, as shown in Figure 5.2.



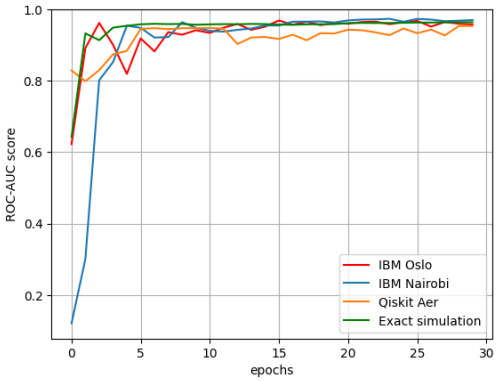
(a) Training loss



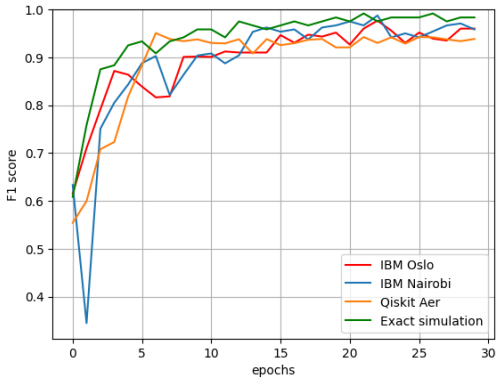
(b) Test loss



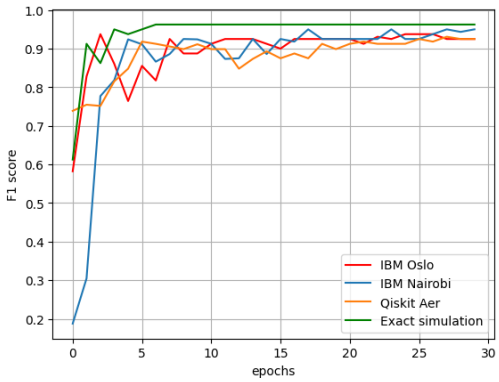
(c) Training ROC AUC score



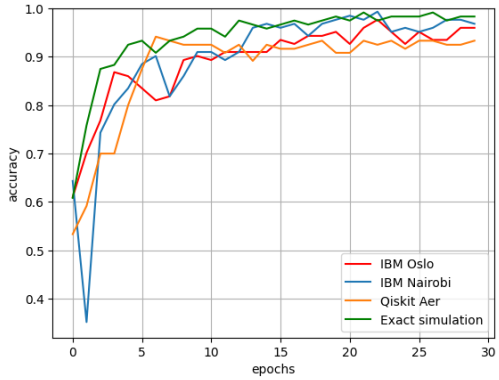
(d) Test ROC AUC score



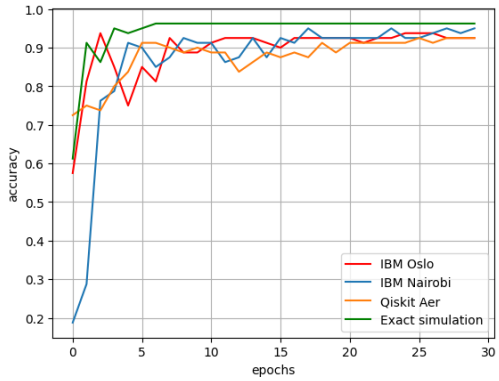
(e) Training F1 score



(f) Test F1 score



(g) Training Accuracy



(h) Test Accuracy

Figure 5.2: Results of the quantum computation on IBM Oslo and Nairobi compared to the exact and shot-based simulations in terms of convergence, accuracy, ROC-AUC and F1 score (averaging 5 runs).

Specifically, we follow the procedure explained in Section 4.1.3 and we average over 5 runs (remind that these experiments are very expensive). The models are trained for 30 epochs and we show the training and test losses and metrics against the noisy shot-based simulation using Qiskit Aer and the exact non shot-based one.

We notice that the curves obtained on the two platforms are overall aligned between each other and with the simulator, while the exact simulation yields overall slightly better results and smoother trends both in the minimization of the loss and in the learning process. Going more into the details, IBM Nairobi's results are closer to the noisy shot-based simulation while IBM Oslo is slower in convergence and eventually underperforms the other platforms in terms of accuracy and F1 score. Moreover, IBM Nairobi seems to be initially slightly underfitting (until epoch 12) despite the loss minimization being globally consistent. Overall, as stated in the previous chapter, we conclude that the exact simulation is in fact optimistic with respect to NISQ technology and that despite having the same quantum volume, the same number of qubits and the same connectivity, two existing quantum

devices can yield results not entirely aligned, as a lot of other factors impact the overall computation. We remark again that the plots are very noisy because of technological limitations of today's quantum devices.

5.4 Run on IonQ QPU via Microsoft Azure

At this stage, it most certainly matters which technology one uses, and in particular which circuit one wishes to execute. Although all of existing technologies implement universal gates, the topology of superconducting devices is quite different and potentially more limiting from those of trapped ions. For instance, the latter one can have (almost) all-to-all connectivity, meaning any qubit within a line can act on (almost) any other qubit on the line. On the other hand, superconducting qubits have a particular topology where one qubit can only act on some nearest neighbor qubits through particularly fabricated microwave couplers, as we already hinted in the previous chapter, specifically Figure 4.3. Accordingly, superconducting devices may use a lot of SWAP gates to move qubits around; this can impact overall gate fidelity. Consider, for instance, a quantum circuit consisting of 24 qubits where qubit 1 acts as a control for the negation of qubit 23. A trapped-ion system may be able to have qubit 1 act directly on qubit 23 with a CNOT gate. However, a superconducting system may require that qubit 23 swaps with qubit 22, which swaps with qubit 21 and so on before qubit 3 is swapped with qubit 2, and then qubits 1 and 2 can finally apply the CNOT gate. In such a dramatic example all of these swap gates really impact the overall fidelity of the superconducting system. Of course you could always attempt to redesign a circuit so that qubits 1 and 23 don't have to be doing a CNOT gate therebetween, which is what transpiler are there for. It's not always guaranteed this is possible though and here comes our exploration of a different technology.

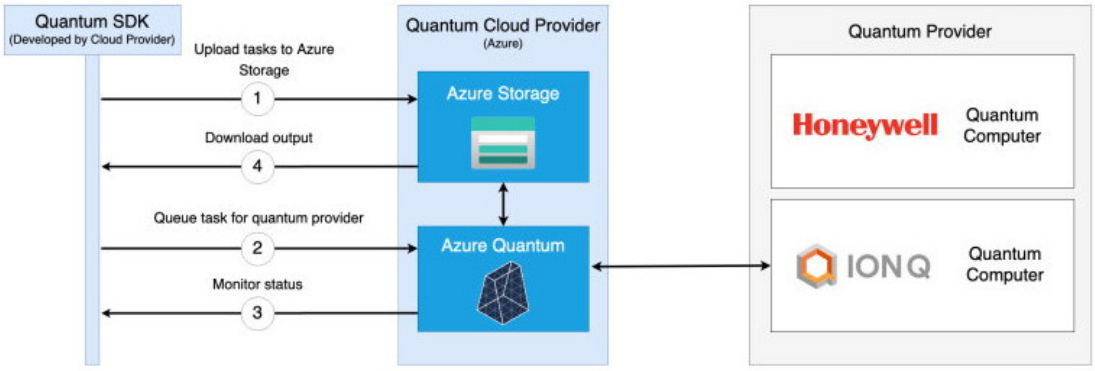


Figure 5.3: High-level model for accessing a quantum processing unit (QPU) on Microsoft Azure

The access model of a quantum computing unit on Microsoft Azure cloud is

shown in Figure 5.3. Differently from IBM Cloud, Azure is a middle layer between the user and the real quantum providers (IonQ and Honeywell, including Quantinuum) offering, at infrastructure level, the storage system and the communication means. However, running a quantum program follows the same job submission paradigm: before execution, each task is appended to a queue, which can have a waiting time of 6-12 hours. This significantly impacts the overall processing time, as the quantum SDK does not currently support Azure Durable Functions, which would prove paramount to perform asynchronous executions.

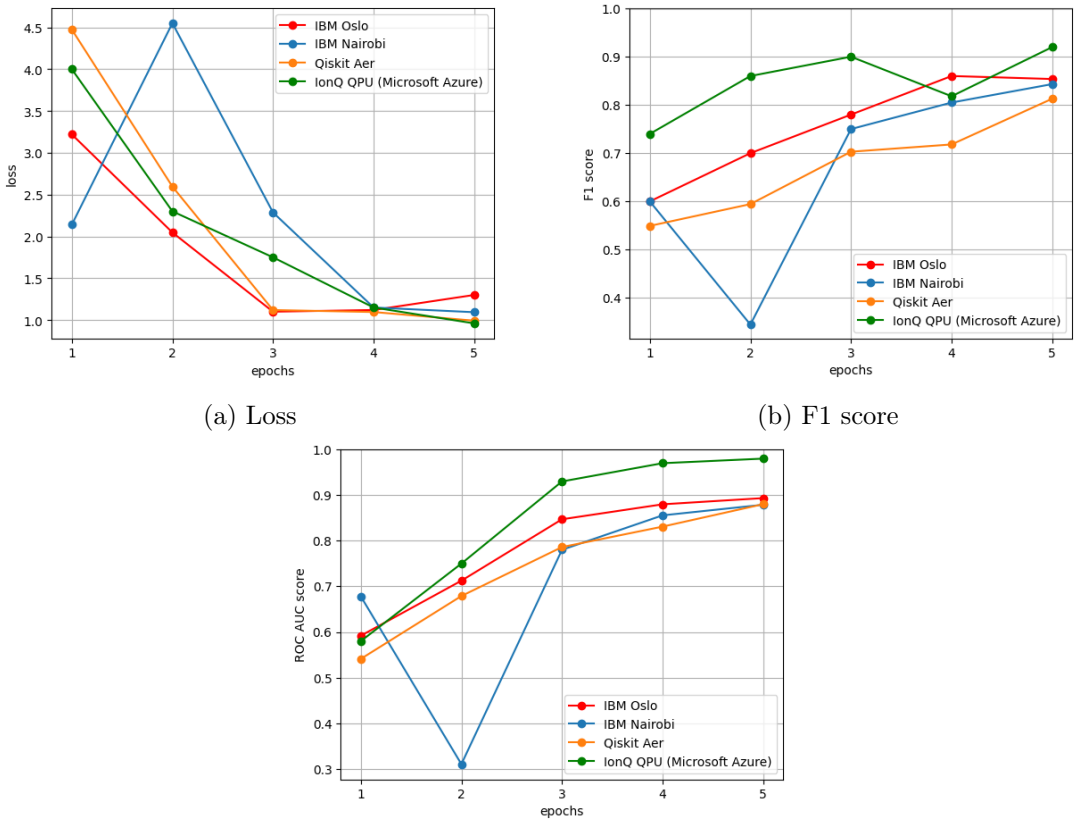
In this section, we explore the 11 qubits trapped ions quantum processing unit (QPU), offered by IonQ, to run the same experiment described in the previous section. From the software perspective, we only need to provide an appropriate backend able to interface with platform:

```
from azure.quantum import Workspace
from pytket.extensions.qsharp import AzureBackend

workspace = Workspace(
        subscription_id='YOUR_SUBSCRIPTION_ID',
        resource_group='YOUR_RESOURCE_GROUP',
        name='ENV_NAME',
        location='westeurope' # might change depending on where you are
)

backend = AzureBackend(
        target_name='ionq.qpu',
        resourceId='YOUR_RESOURCE_ID',
        location='westeurope' # might change depending on where you are
)
```

Since a training experiment on this platform on the described setup is quite expensive (in the order of $50 per epoch), we only let it run for 5 epochs. This will unfortunately not allow us to make a fair comparison with any of the above experiments run on IBM devices as we're not iterating on the same number of steps and we're not statistically assessing the experiment several times with multiple runs. However, in the context of our quantum hardware exploration, it is useful to give further insights on the different existing technological branches in the quantum computing industry, without making any strong statement on the comparison. The obtained results are shown in Figure 5.4.

We notice a faster convergence and, in general, a smoother trend of the curves, with the model yielding better results both in terms of F1 score and ROC-AUC score compared to the noisy shot-based simulation and to IBM Oslo and Nairobi. This might be due to the several factors, including relevant differences in the way

the computation is carried out on this hardware. Nonetheless, in the light of the conditions in which this last experiment was performed, these observations are not very conclusive, but help us understanding how the transition from a quantum program developed and tested on some quantum platform to a different one might require reconsidering the approach used or even the design of the algorithm (for instance, in our case, the choice of the ansatz and its hyperparameters).



(a) Loss

(b) F1 score

(c) ROC AUC score

Figure 5.4: Results of the Ionq QPU on Microsoft Azure after 5 epochs (on a single run) on the training set, compared with IBM Oslo, IBM Nairobi and Qiskit Aer.

Chapter 6

Conclusion and Future Work

In this thesis we explored Quantum models for Natural Language Processing based on the DisCoCat framework derived from the field of computational linguistics. We assessed the performances of such models on problems close to real world applications related to the travel industry and evaluated the status of real quantum devices taking into considerations two different technological branches: superconducting qubits and trapped ions. Despite the limited capabilities of the technology as well as the limitation of our budget — which foreclosed us from trying existing devices with more qubits — we obtained overall promising results and significant insights on the usefulness of this technology for one of the most challenging field in Artificial Intelligence.

Future work includes evaluating this approach on quantum hardware with more qubits as well as extending it to deal with entire texts, developing on the new circuital textual approach proposed by Hamza Waseem, Coecke et al. [27]. In addition, we believe this approach has a lot of potential to deal with tasks like language generation: state-of-the-art models to produce text are trained in a self-supervised fashion to predict the following word using the attention mechanism. Nevertheless, exploiting a mathematical framework capable of determining whether the next word is supposed to be an adjective, a noun or a verb based on grammar — which is not supposed to be learned but embedded in the representation, as we explained — might prove more successful. Eventually, applying this approach to other kind of languages, e.g. programming languages since they have a fixed and limited grammar, might result effective.

Acknowledgments

My first words of gratitude go to my company supervisors Alix Lhéritier, Nicolas Bondoux and Mourad Boudia for believing in me, for their constant help and constructive feedback, for their rigor and attention to details, for the magnificent opportunity to present this work at SophI.A Summit 2022 and to interact with world leading companies in the quantum industry like IBM and Quantinuum; for the social and friendly environment they created within the team, for treating me as an experienced professional (which I'm not) and a friend, for always taking into consideration my proposals and ideas. I'm also much indebted to them for reviewing this work.

I'm grateful to Rodrigo Acuna Agost, who played a crucial role in steering this thesis in an industrial direction, with his pragmatism and attention to business, and to all the members of the Amadeus Applied Research and Technology team, who will recognize themselves.

I would like to also thank Professor Marios Kountouris and Professor Bartolomeo Montrucchio for accepting me as master thesis student and for embracing my project, being always available for help, suggestions and support.

On a more personal side, my most special thanks to my family: my parents Maria and Pantaleone, my siblings Sonia and Dario and my girlfriend Martina, the girl I've been growing up with, my safe haven, my adventure partner and by far the person who made me the man I am today. Thanks for all your unconditional love, support and appreciation. Without you all, I wouldn't be here.

Bibliography

[1] Robin Lorenz, Anna Pearson, Konstantinos Meichanetzidis, Dimitri Kartsaklis, and Bob Coecke. QNLP in Practice: Running Compositional Models of Meaning on a Quantum Computer.

[2] Victor Martinez and Guilhaume Leroy-Meline. A multiclass q-nlp sentiment analysis experiment using discocat, 2022.

[3] SophI.A Summit. https://univ-cotedazur.eu/events/sophia-summit.

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

[5] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. 323(6088):533–536.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. 9:1735–80.

[7] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. 45(11):2673–2681.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

[10] Stephen Clark, Bob Coecke, and Mehrnoosh Sadrzadeh. A Compositional Distributional Model of Meaning. In *Proceedings of the Second Symposium on Quantum Interaction (QI-2008)*, pages 133–140.

[11] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical foundations for a compositional distributional model of meaning. *arXiv preprint arXiv:1003.4394*, 2010.

[12] Dimitri Kartsaklis, Ian Fan, Richie Yeung, Anna Pearson, Robin Lorenz, Alexis

Toumi, Giovanni de Felice, Konstantinos Meichanetzidis, Stephen Clark, and Bob Coecke. Lambeq: An Efficient High-Level Python Library for Quantum NLP. abs/2110.04236.

[13] Yaoyun Shi. Both toffoli and controlled-not need little help to do universal quantum computation. 2002.

[14] J Lambek. Type grammars revisited. a. lecomte, f. lamarche and g. perrier (eds.): Logical aspects of computational linguistics. lnai 1582, 1999.

[15] Bob Coecke, Giovanni de Felice, Konstantinos Meichanetzidis, and Alexis Toumi. Foundations for Near-Term Quantum Natural Language Processing.

[16] Richie Yeung and Dimitri Kartsaklis. A CCG-Based Version of the DisCoCat Framework.

[17] Stephen Clark. Something old, something new: Grammar-based CCG parsing with transformer models. *CoRR*, abs/2109.10044, 2021.

[18] Masashi Yoshikawa, Hiroshi Noji, and Yuji Matsumoto. A* CCG parsing with a supertag and dependency factored model. *CoRR*, abs/1704.06936, 2017.

[19] V Havlíček, AD Córcoles, K Temme, AW Harrow, A Kandala, JM Chow, and JM Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019.

[20] Giovanni de Felice, Alexis Toumi, and Bob Coecke. DisCoPy: Monoidal Categories in Python. In *Proceedings of the 3rd Annual International Applied Category Theory Conference, ACT*, volume 333. EPTCS.

[21] Alexis Toumi, Giovanni de Felice, and Richie Yeung. Discopy for the quantum computer scientist, 2022.

[22] J.C. Spall. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronic Systems*, 34(3):817–823, 1998.

[23] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: composable transformations of python+ numpy programs. *Version 0.2*, 5:14–24, 2018.

[24] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. t| ket>: a retargetable compiler for nisq devices. *Quantum Science and Technology*, 6(1):014003, 2020.

[25] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[26] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[27] Muhammad Hamza Waseem, Jonathon Liu, Vincent Wang-Maścianica, and Bob Coecke. Language-independence of discocirc's text circuits: English and

urdu. *arXiv e-prints*, pages arXiv–2208, 2022.