Quantum Natural Language Processing with lambeq

Thomas Hoffmann, Charles London,
Richie Yeung

July 2022

Introduction

Quantum Natural Language Processing (QNLP) is a very young area of research, aimed at the design and implementation of NLP models that exploit certain quantum phenomena such as superposition, entanglement, and interference to perform language-related tasks on quantum hardware. The advent of the first quantum machines, known as noisy intermediate-scale quantum (NISQ) computers, has already allowed researchers to make the first small steps towards exploring practical QNLP, by training models and running simple NLP experiments on quantum hardware ([9], [8]). 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 [6].

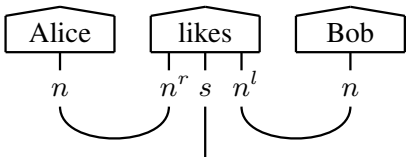
With that in mind, the objective for the participants of this hackathon is to get some familiarity with compositional models of natural language, running on quantum hardware. There exists a rich literature on formal grammar theory and philosophy of language for analysing linguistic structures, of *how words compose* in sentences, and how meaning interacts with context. In parallel, in the last decade, machine learning methods have made great progress by using *vector representations* of the meanings of words. Combining, both grammar and vector-space models of language, the DisCoCat (Distributional Compositional Categorical) model, introduced 10 years ago [4], attempts to create vector embeddings for sentence meanings by composing word tensors according to grammatical structure.

Now, by an algebraic analogy between the mathematics of natural language and of quantum theory, one can choose to use the quantum Hilbert space as the vector space, or "feature space", in which meanings are embedded. The particular "pregroup" grammar model that we use was introduced by Joachim Lambek (for a quick review see [7]). Quantum DisCoCat

then invokes categorical quantum mechanics to interpret grammatical reductions, or syntax trees, as quantum processes that prepare quantum states carrying the meaning of a sentence. Equipped with a pipeline that maps sentences to quantum processes, we are ready to define NLP tasks and tackle them with readily available NISQ processors.

How does DisCoCat work?

First, we begin with a pregroup grammatical derivation of the sentences. This grammatical derivation is then represented as a monoidal diagram (see below). The central idea is that such diagrams represent processes, inspired by functional programming. Diagrams are read from top to bottom, and they are composed of elementary processes depicted as boxes with input and output wires which carry types. States are boxes with no input, hence, they are inputs themselves. Effects are boxes with no output, and so they are tests for outputs. Importantly, when boxes are composed by joining outputs to inputs, only compositions that type-check are allowed.



Coming back to pregroup grammars, a parser assigns some types to words. Looking at the diagram above, we can see that the two nouns "*Alice*" and "*Bob*" are assigned with the atomic noun type n. However, nouns alone don't form a sentence. To get a valid sentence, we need to combine a noun with at least one verb. In the above example, we observe the transitive verb "*likes*". A transitive verb expects a noun to it's right- and left-hand side in order to create a sentence. In other words, algebraically a transitive verb is assigned with the pregroup type $n^r \cdot s \cdot n^l$. The superscripts $\cdot^r$ and $\cdot^l$ indicate so-called right- and left-adjoints. A right adjoint cancels out with a noun on its left-hand side, i.e. $n \cdot n^r \to 1$, and respectively a left-adjoint cancels a noun on its right-hand side: $n^l \cdot n \to 1$. Hence, the complete type reduction for the sentence "*Alice likes Bob*" looks like this:

$$n \cdot n^r s\ n^l \cdot n \to s\ n^l \cdot n \to s.$$

As we can see, the type reduction yields the atomic sentence type s, indicating that the sentence is grammatically sound. Diagrammatically, words are depicted as boxes (⏢) that initialise an object of their type, and type reductions as so-called "cups" (◡).

This was the compositional part. In order to endow this diagram with distributional meaning, all we need to do is reinterpret it as a quantum process, where word-states are (pure) quantum states and cups are "Bell" (maximally entangled) effects. Hence, every type gets mapped to a vector space dimension, or equivalently to a number of qubits. These are seen

as hyperparameters that specify the model. Regarding the quantum word embeddings, consider a set of classical parameters for each word, describing the preparation of the state. These parameters could be angles (phases) of an ansatz circuit defined on a number of qubits specified by the types of the word.

Finally, after both the compositional and the distributional parts of the setup have been specified, one needs to just run this quantum process by preparing the quantum word-states, and then apply the "grammar-aware" Bell effects. Thus, one has prepared a quantum state which encodes the meaning of the whole sentence in a Hilbert space of dimension 2^{q_s}.

Recommended reading

For a high-level informal exposition of quantum NLP with focus on NISQ devices, see [3]. Specifically, for sentence classification experiments, see [9, 8], the former of which was written with a physicist audience in mind and the latter one was targeted at the NLP engineer and practitioner.

Tasks for this hackathon

The following tasks will guide you towards running your own natural language experiments on a quantum computer. This tutorial assumes that you have experience with programming in Python.

Task 1: Make yourself familiar with lambeq

LAMBEQ is a high-level python library that facilitates the development of QNLP models. It is modular, easily extensible and supports a wide variety of backends (PyTorch, TensorNetwork, qiskit, ...). Read the lambeq white-paper [6] and make yourself familiar with the interface by browsing through the online documentation [2]. LAMBEQ is built on DisCoPy, a toolbox for computing with monoidal categories. DisCoPy makes it easy to represent and manipulate string diagrams. It might be helpful, but not required, to go through this tutorial notebook [1].

Task 2: Get an overview of the dataset

In this task, you will familiarise yourself with the dataset we will use in the experiments and the learning objective of the model. The dataset consists of 100 sentence pairs. Each short sentence falls into one of two categories: Either IT or food. Two examples are given here:

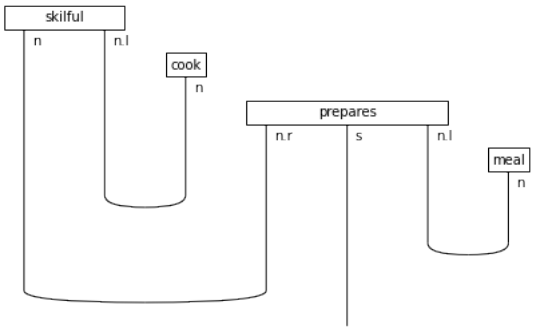
- food: "skilful cook prepares meal"
- IT: "programmer writes complicated code"

The goal of the project is to write, train and test a simple QNLP model that, given two sentences, can detect whether they belong to the same category or not. In order to achieve the goal you will need to find a suitable quantum representation of each sentence and a way to compare their similarity. The next steps will guide you to run the experiment.

Task 3: Parse sentences to diagrams

As a first step towards a quantum representation, we want to create a monoidal diagram from a sentence. Hence, we need to decipher its grammatical structure. To do so, LAMBEQ comes with a state-of-the-art CCG parser, called `BobcatParser`. Make yourself familiar with the workflow described under Step 1. Sentence input of the lambeq documentation [2].

Using the `BobcatParser`, the example sentence "skilful cook prepares meal" parses to the following diagram:



Be careful: Some sentences might not be parsed correctly, i.e. the output wire of the diagram is not of type s (sentence). You will need to take care of these cases. You might want to have a look at the `root_cats` attribute of the `BobCatParser`.

Besides creating monoidal diagrams, LAMBEQ provides the interface for other compositional language models through the `Reader` class. Some concrete implementations are the `cups_reader`, `spider_reader` or the `tree_reader`. You can experiment with these models, or you might even come up with your own `Reader`.

Task 4: From diagram to circuit

In order to create a sentence circuit from a diagram, we have to use an ansatz that maps each wire to a qubit system and each box to a variational quantum circuit. LAMBEQ provides an interface to create such "ansätze" and ships with a concrete implementation, namely the IQPANSATZ. Besides quantum ansätze, LAMBEQ also comes with a hand-full of classical tensor ansätze, which should be omitted here.

The goal of this task is to prepare the quantum circuits for each sentence in your dataset. Try to experiment with different numbers of qubits and ansatz layers. These hyperparameters have a large impact on the model's performance and should be chosen wisely.

Task 5: Define the learning objective

So far, we've seen how to map a sentence to a quantum circuit. However, to tackle the task, you will need to establish a comparison score that measures the similarity of two sentences. Try to take the numeric output of the two sentence circuits you want to compare, and define an output layer, that returns a binary score.

To facilitate the training process you should have a look at the `Models` that LAMBEQ provides in the training module. In the beginning, you may want to use LAMBEQ's NUMPYMODEL with JAX's just-in-time compilation enabled. This is significantly faster than simulating on qiskit's AerBackend. Once you have a working experiment, then you can use the AerBackend to import a noise model to simulate a particular device of your choosing.

Hint: You need to modify the default forward pass such that the model performs a sentence comparison.

Task 6: Run your first QNLP experiment

Now that you have a encoded your sentences in variational quantum circuits and a custom model, you are ready to perform your first QNLP experiment. To facilitate the training, you can use LAMBEQ's `Trainer`. The pipeline for running an experiment is described here. Make sure that you split your dataset into a train, validation and test set to ensure the proper evaluation of your model. Furthermore, you will need to make sure that you use a suitable loss function and that the hyperparameters of the optimiser are set to reasonable values.

Changing the backend configuration and using the `TketModel`, you can now run your model using qiskit's AerBackend (with a noise model) or even on an actual quantum computer.

Finally, submit a short report where you include the following:

- A short discussion explaining any design choices in the model, e.g. choice of loss function, use of regularisation or not and so on.
- A short discussion on the results.
- Optimisation plots showing the convergence of the model.

Possible extensions

In the following paragraphs, you'll find some possible extensions to improve the QNLP experiment. You can either follow the suggestions here to squeeze the last bits out of

your model and/or you might get creative by finding a real-life dataset and try to make a QNLP model perform the desired task. Since lambeq models normally returns vectors that embed the meaning of sentences, it is very well suited for classification tasks like sentence classification, spam detection, sentiment analysis, entailment, or many others.

Diagrammatic rewriting

Explore LAMBEQ's `remove_cups` method, and explain what it does. Perform your experiments with the modified diagrams and discuss why the model performs better or worse. What is the benefit of having diagrams with fewer cups?

Different ansätze

Try converting the word states into different parameterised ansätze and see how it affects performance. There is a wide selection of ansätze available in the quantum machine learning literature (see Section 4 of [10] for example). Certain words exhibit special linguistic properties, so you might consider using different ansätze for different words, within the same experiment. You can for example implement the following circuit or a variation of it [5]:

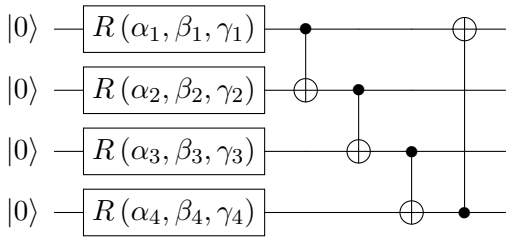


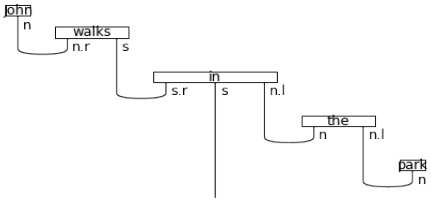
Figure 1: Example of a strongly entangled layer using four qubits.

Implement a new reader

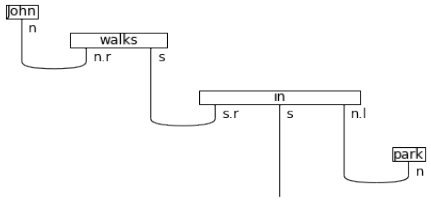
So far you've seen the DisCoCat model based on grammar parsing, the `cups_reader`, the `spider_reader` and the `tree_reader`. Can you come up with your own (simple) compositional model? Maybe you can come up with a variation of the CUPS_READER that simplifies the computational graph and uses fewer qubits.

Implement a rewrite rule

Part of a successful QNLP experiment is to simplify the diagrams as much as possible and cut out words that may not contribute in the meaning of a sentence. A simple example is the article "the". Articles are generally of type $(n, n.l)$, i.e expect a noun to their right-hand side. However, in most cases, articles do not come with a specific distributional meaning (since they appear in almost every possible context) and can be left out. Hence, the following diagram

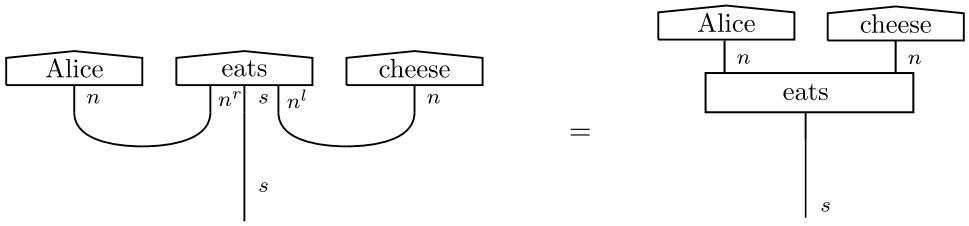


simplifies to



Investigate LAMBEQ's rewrite rules and try to come up with a novel rewrite.

Tip: you can make use of the map-state duality, which states



References

[1] discopy: Qnlp tutorial. https://discopy.readthedocs.io/en/main/notebooks/qnlp-tutorial.html. Accessed: 2021-06-30.

[2] lambeq: Online documentation. https://cqcl.github.io/lambeq/. Accessed: 2022-04-04.

[3] Bob Coecke, Giovanni de Felice, Konstantinos Meichanetzidis, and Alexis Toumi. Foundations for near-term quantum natural language processing, 2020.

[4] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical foundations for a compositional distributional model of meaning, 2010.

[5] Thomas Hoffmann. Quantum models for word-sense disambiguation. https://hdl.handle.net/20.500.12380/302687. Accessed: 2022-04-04.

[6] 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. https://arxiv.org/abs/2110.04236, 2021.

[7] Joachim Lambek. Pregroups and NLP. https://www.math.mcgill.ca/rags/JAC/124/Lambek-Pregroups-s.pdf.

[8] Robin Lorenz, Anna Pearson, Konstantinos Meichanetzidis, Dimitri Kartsaklis, and Bob Coecke. QNLP in practice: Running compositional models of meaning on a quantum computer. *CoRR*, abs/2102.12846, 2021.

[9] Konstantinos Meichanetzidis, Alexis Toumi, Giovanni de Felice, and Bob Coecke. Grammar-aware question-answering on quantum computers, 2020.

[10] Richie Yeung. Diagrammatic design and study of ans\"{a} tze for quantum machine learning. *arXiv preprint arXiv:2011.11073*, 2020.