1. **Introduction**

Quantum natural language processing (QNLP) is a recent application of quantum computing that aims at representing sentences' meaning as vectors encoded into quantum computers [1]. Natural Language Processing (NLP) is often used to perform tasks such as machine translation, sentiment analysis, relationship extraction, word sense disambiguation and automatic summary generation.

Cambridge Quantum today announced the release of the world’s first toolkit and library for Quantum Natural Language Processing (QNLP). The toolkit is called lambeq, named after the late mathematician and linguist Joachim Lambeq.

lambeq is capable of converting sentences into a quantum circuit. It is designed to accelerate the development of practical, real-world QNLP applications, such as automated dialogue, text mining, language translation, text-to-speech, language generation and bioinformatics.

1. **The problem**
2. **Methodology**

We start with importing NumPy and specifying some training hyperparameters. We read the dataset that was provided. Then, we start building the model. To begin with, the data had to be split into a training and validation set. We also print some examples. Labels are represented in 2-dimensional arrays. In order to obtain a DisCoCat-like output, we first use the BobcatParser class from text2diagram package, which, in turn, calls the parser, obtains a CCG derivation for the sentence, and converts it into a string diagram.This step is needed to produce string diagrams for our sentences.

We import Rewriter to create a rewriting functor. Afterwards, We simplify the diagrams by calling normal\_form() and filter out any diagrams that could not be parsed.

Creating circuits comes next. In order to run the experiments on a quantum computer, we need to apply to string diagrams a quantum ansatz. For this experiment, we will use an IQPAnsatz, where noun wires (n) are represented by a one-qubit system, and sentence wires (s) are discarded (since we deal with noun phrases). we remove the cups before parameterising the diagrams. By doing so, we reduce the number of post-selections, which makes the model computationally more efficient.

To instantiate the model, We will use a TketModel, which we initialise by passing all diagrams to the class method TketModel.from\_diagrams(). The TketModel needs a backend configuration dictionary passed as a keyword argument to the initialisation method. This dictionary must contain entries for backend, compilation and shots. We keep track of the loss as well, where we use standard binary cross-entropy as the loss.

Next, In lambeq, quantum pipelines are based on the QuantumTrainer class. Furthermore, we will use the standard lambeq SPSA optimizer, implemented in the SPSAOptimizer class. This needs three hyperameters:

a: The initial learning rate (decays over time),

c: The initial parameter shift scaling factor (decays over time),

A: A stability constant, best choice is approx. 0.01 \* number of training steps.

To facilitate data shuffling and batching, lambeq provides a native Dataset class. Shuffling is enabled by default, and if not specified, the batch size is set to the length of the dataset.

Once this is complete, we pass the datasets to the fit() method of the trainer to start the training.

Finally, we can now visualise the results and evaluate the model on the test data.

1. **Results and Analyses**

**References:**

1. Du, S. L., Santana, S. H., & Scarpa, G. (2022). A gentle introduction to Quantum Natural Language Processing. *arXiv preprint arXiv:2202.11766*.