Quantum Machine Learning with Circuit Cutting

Saasha Joshi
Dept. of Computer Science
University of Victoria
Victoria, British Columbia, Canada
saashajoshi08@gmail.com

Angadh Singh
Dept. of Computer Science
University of Victoria
Victoria, British Columbia, Canada
angadh.singh1@gmail.com

Abstract

Quantum Machine Learning (QML) techniques, including variational Quantum Tensor Networks (QTN), pose a huge implementation challenge regarding qubit requirements. An approach to circumvent this issue is to perform circuit cutting that segments large quantum circuits into multiple smaller sub-circuits that can be trained easily on a quantum device [1]. This project lays down the workflow for training a variational QTN circuit that implements circuit cutting, specifically gate cutting, to perform data classification. The workflow is built with the help of the Qiskit SDK with dependence on the Circuit Knitting Toolbox [3] for circuit cutting procedures. Additionally, a significant amount of modifications are made to algorithms like SamplerQNN, a part of the Qiskit Machine Learning package, to accommodate the training of multiple sub-circuits with Qiskit Aer's Sampler runtime primitive. The dataset used for classification is the diabetes dataset from the National Institute of Diabetes and Digestive and Kidney Diseases publicly available on Kaggle.

The project can be found on GitHub here.

Index Terms

Quantum Machine Learning, Circuit Cutting, Qiskit, Quantum Tensor Networks, Classification

I. Introduction

Quantum Machine Learning models such as Quantum Tensor Networks (QTN), pose a huge implementation challenge regarding qubit requirements. To address this challenge, one approach is to adopt circuit cutting techniques that segment large quantum circuits into multiple smaller sub-circuits that can be trained easily on a quantum device [1]. This segmentation facilitates an easier training process, as highlighted in [1].

To integrate circuit cutting techniques with QML model training, we come up with a workflow, built on the Qiskit SDK, that utilizes the Circuit Knitting Toolbox [3] to cut the larger QML circuits into multiple shorter, trainable sub-circuits. The circuits are trained according to one of the two workflows as depicted in Figure 2 and 3 using a Sampler estimator from Qiskit Aer. The results, in the form of quasi-probability distributions, are then used to reconstruct the expectation value of the original circuit.

II. Circuit Cutting

Circuit cutting is the method to partition a circuit into multiple smaller sub-circuits that can be executed independently of each other. This cutting procedure can be implemented through two distinctive approaches - gate cutting, shown via a red dotted line in Figure 1, or wire cutting, shown by a blue dashed line in Figure 1.

A gate cutting procedure involves the decomposition of a unitary gate into multiple quantum channels that can be run on a specific quantum device. This procedure helps in the reduction of the circuit depth or width, according to the problem statement.

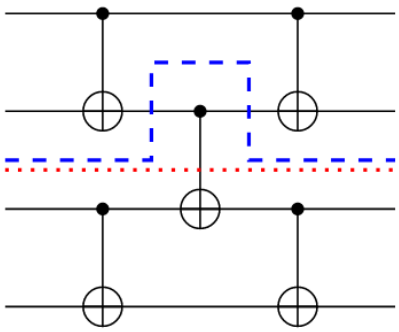

Fig. 1. Circuit Cutting with gate cuts (red) and wire cuts (blue) [4]

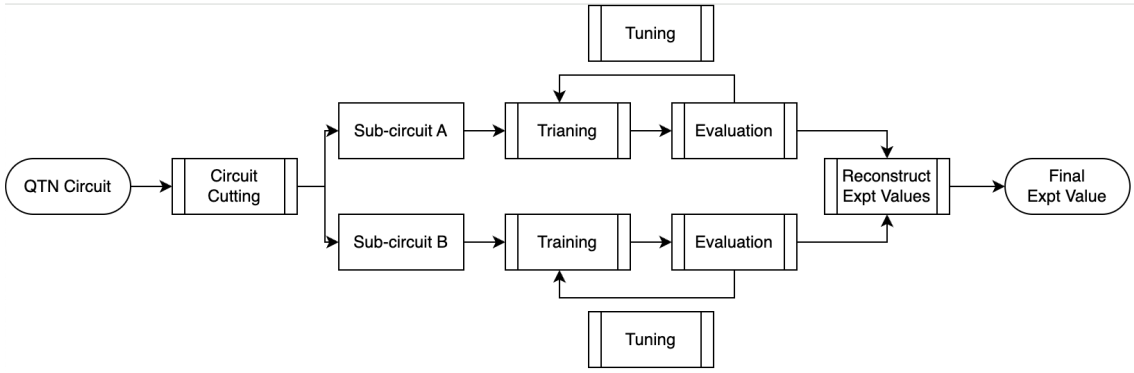

Fig. 2. QML model workflow (A) with Circuit Cutting

On the other hand, a wire cutting procedure partitions the circuit along the qubit wire in time. The resultant subcircuits of a wire cut, however, have a few extra qubits added. Both techniques reconstruct the final expectation value of the original circuit by sampling the quasi-distribution of the subcircuits and performing classical post-processing.

The Circuit Knitting Toolbox [3] by Qiskit provides implementation strategies for both gate and wire cutting procedures.

III. Methodology

The training of QML models integrated with circuit cutting can be performed using two possible workflows, as depicted in Figures 2 and 3.

A. Training Workflows

Workflow A, after performing the circuit cut, trains the sub-circuits with respect to the original train labels. This training happens before the results are combined to reconstruct the expectation value of the original circuit. This procedure helps in maintain a parallel workflow where the sub-circuits can be trained simultaneously. This simultaneous training, over multiple iterations, can be performed with the help of existing Batching techniques in the Qiskit Runtime primitives [2]. The quasi-probability distributions received at the end of the training and evaluation process are combined to retrieve the expectation values of the original circuit.

Workflow B, also proposed in [6], reconstructs the original expectation value after every training iteration. The optimization procedure is performed on the reconstructed expectation value that triggers the model tuning loop. This workflow facilitates simultaneous training over one training iteration at a time. After every training result is received, the sub-circuits go through the classical post-processing steps to reconstruct the original expectation value.

In this project, we are choosing to work using the Workflow A structure. This decision was made keeping in mind the restrictions posed by the existing Qiskit code and the time limitations of the event.

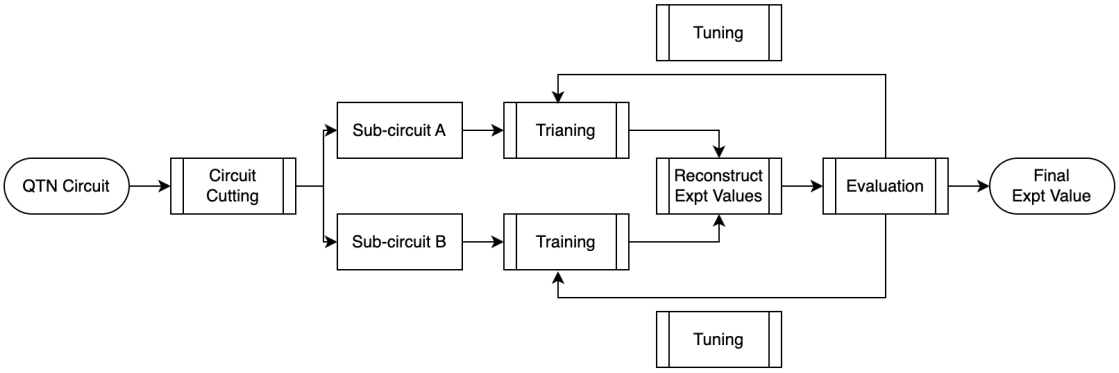

Fig. 3. QML model workflow (B) with Circuit Cutting

B. Preparing the data

In this project, we aim to perform the classification on a diabetes dataset by the National Institute of Diabetes and Digestive and Kidney Diseases publicly available on Kaggle [5]. The dataset consists of 8 feature inputs that predict the chances of diabetes in a patient.

Diabetes Dataset							
Glucose	Blood Pressure	BMI	Insulin	Diabetes Pedigree Function	Pregnancies	Skin Thickness	Age

Fig. 4. Caption

Since the dataset consists of classical data, we perform preliminary data embedding to represent this data as quantum states. We use the angle embedding technique to perform this encoding. The data is normalized in the range $[-\pi, \pi]$.

C. Constructing QTN circuit and sub-circuits

Initially, a variational tree tensor network (TTN) circuit is constructed with qubits equal to the number of data features, in our case it makes a total of 8 qubits. A circuit cutting process is initiated that partitions the original TTN circuit into sub-circuits based on a predefined partitioning scheme - gate or wire cutting. In our case, we utilize gate cutting to split the TTN at specific gate locations, creating sub-circuits that can be executed independently. In the context of diabetes data analysis, where the relationships between features (such as glucose levels, BMI, age, etc.) may involve complex, nonlinear interactions, gate cutting can allow for a nuanced manipulation of these relationships in smaller, more manageable quantum operations.

For simplicity, we cut the QTN circuit such that each of the sub-circuits consists of half of the qubits in the original circuit. This results in sub-circuits having 7 and 8 trainable parameters, respectively. The sub-circuits are then composed with the embedding circuits that contain 4 feature inputs respectively.

Since we are using the gate cutting procedure to build the sub-circuits, each sub-circuit generates multiple different sub-experiments consisting of various single-qubit basis rotation gates at the location of the cuts. For one cut, a total of 6 sub-experiments are generated per sub-circuit. Hence, we have a total of 12 sub-experiments (sub-circuits) to train at the end of the circuit cutting procedure.

D. Neural networks and sub-circuits

Next, a custom quantum neural network sampler (CustomSamplerQNN) is instantiated and acts as the core of the quantum neural network training process. The QTN sub-circuits are parameterized by input parameters (related to the data embedding) and variational weight parameters (related to the trainable aspects of the quantum circuits). The sampler provides the forward and backward pass methods for training the variational parameters in the QTN sub-circuits. This allows the network to learn from the data by adjusting the weight parameters. The neural network runs with two passes to evaluate the final output.

Forward Pass: The forward pass involves running the prepared QTN sub-circuits with the given input data and weight parameters. This step simulates the execution of the sub-circuits for the entire dataset, generating predictions or outputs based on the current state of the network's weights. The output of this step is a dictionary containing the results of the computations for each subexperiment, providing a detailed view of how the input data is transformed through the quantum circuits.

Backward Pass: Following the forward pass, a backward pass is conducted to compute gradients with respect to the input data and weight parameters. This is crucial for the training process, as it identifies how changes to the weight parameters affect the network's output. The gradients are used to adjust the weights in a way that minimizes the difference between the predicted outputs and the actual targets, thereby improving the model's accuracy over time.

This process is run for both training sets 'A' and 'B' such that we are able to evaluate the sub-circuits for each set of features encoded on each training data input.

IV. Evaluation

A. Loss and Optimization

To optimize the parameter values in the sub-circuits, an objective function is defined that governs the behavior of the optimizer. Since we are dealing with a Sampler primitive that results in quasi-probability distributions at the end of its run, a multi-class objective function is defined. A minimize function calls this objective function to optimize the variational parameters in the 12 sub-experiments. The result of this minimize function is a set of 12 lists with 7 and 8 new parameter values corresponding to the first and the second sub-circuit structures.

B. Reconstruction of Expectation Values

The parameter values received from the optimizer are used in the sub-circuits and the final quasi-probability distributions received are used to reconstruct the expectation value of the original circuit.

Our future work includes analysis of these expectation values to verify the accuracy of the QML model. These values are also to be compared to the QML training process that is completed with the original circuit.

V. Requirement for a GPU

- The application of circuit cutting significantly increases the number of sub-circuits, resulting in multiple sub-experiments for each cut performed on a circuit. Here, utilizing a GPU becomes particularly advantageous in efficiently training these short yet large number of circuits.
- Since a QML task involves multiple training procedures, the inclusion of a GPU can significantly boost the efficiency of the training process by reducing the training duration for each circuit.
- Moreover, we are curious to observe the potential outcomes when employing GPU acceleration for circuit cutting in a QML model.

References

[1] D. Guala, S. Zhang, E. Cruz, C. A. Riofrío, J. Klepsch, and J. M. Arrazola, "Practical overview of image classification with tensor-network quantum circuits," Scientific Reports, vol. 13, no. 1, p. 4427, Mar. 2023, doi: https://doi.org/10.1038/s41598-023-30258-y.

[2] Gadi Aleksandrowicz, 'Qiskit: An Open-source Framework for Quantum Computing'. Zenodo, Jan. 23, 2019. doi: 10.5281/zenodo.2562111.

[3] Jim Garrison, 'Qiskit-Extensions/circuit-knitting-toolbox: Circuit Knitting Toolbox 0.6.0'. Zenodo, Feb. 12, 2024. doi: 10.5281/zenodo.10651875.

[4] L. Brenner, C. Piveteau, and D. Sutter, "Optimal wire cutting with classical communication," arXiv.org, Feb. 07, 2023. https://arxiv.org/abs/2302.03366

[5] Mehmet Akturk, "Diabetes Dataset," Kaggle.com, 2020. https://www.kaggle.com/datasets/mathchi/diabetes-data-set

[6] M. Beisel, J. Barzen, M. Bechtold, F. Leymann, F. Truger, and B. Weder, "QuantME4VQA: Modeling and Executing Variational Quantum Algorithms Using Workflows," Proceedings of the 13th International Conference on Cloud Computing and Services Science, 2023, doi: https://doi.org/10.5220/0011997500003488.