QML project

Ludovico Venturi

Table of contents

1 Introduction

In this project work I tried to replicate some of the results of the paper "Hierarchical quantum circuit representations for neural architecture search"[1].

In this paper the authors propose a framework named HierarQcal for representing Quantum Convolutional Neural Networks (QCNN) architectures using techniques from Neural Architectural Search (NAS). This framework enables search space design and architecture search, the former being the most challenging point in applying NAS to QCNN.

In this work we generate the family of QCNN architectures resembling reverse binary trees and we evaluate this family on the GTZAN music genre classification dataset showing that it is possible to improve model performance without increasing complexity.

2 Background

2.1 Supervised Learning for classification

The goal of classification is to use some data X alongside a function f_m (model) to accurately represent a discrete categorization y:

$$f_m(X, \theta) = \hat{y} \approx y$$

The data is used by iteratively changing the model parameters θ based on the disparity between the current representation $\hat{y}$ and the actual categorization y, measured with a cost function $C(y, \hat{y})$. The cost function I use in this project is Binary Cross Entropy (BCE) from `torch.nn.BCELoss`.

Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events. In other words, if we consider a target probability distribution P and an approximation of the target distribution Q, then the cross-entropy of Q from P is the number of additional bits to represent an event using Q instead of P:

$$H(P, Q) = -\sum_{x \in X} P(x) \cdot ln(Q(x))$$

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are successful because they don't need manual feature design and can learn high-level features from raw data. With CNN there is a focus shift: from feature design to architecture design.

2.3 Quantum Machine Learning

Using quantum computers it is possible to write hybrid quantum-classical algorithms already usable in the NISQ era, where the optimization of parameters is done classically and the function f_m is built as a Variational Quantum Circuit (VQC) that acts on a quantum state $|\psi\rangle$.

A VQC is a quantum circuit with some trainable parameters, for example rotation angle $\theta : RY(\theta)$.

The point in using a VQC is that the state can move along all Hilbert space at every change of the parameters, so it is possible to sample from a classically intractable probability density function (pdf).

The state $|\psi\rangle$ must be obtained through an embedding since we're in a Classical-Quantum (CQ) setting and the data we use for training is classic. This is done with a *feature map*, as can be seen in Figure 2.1.

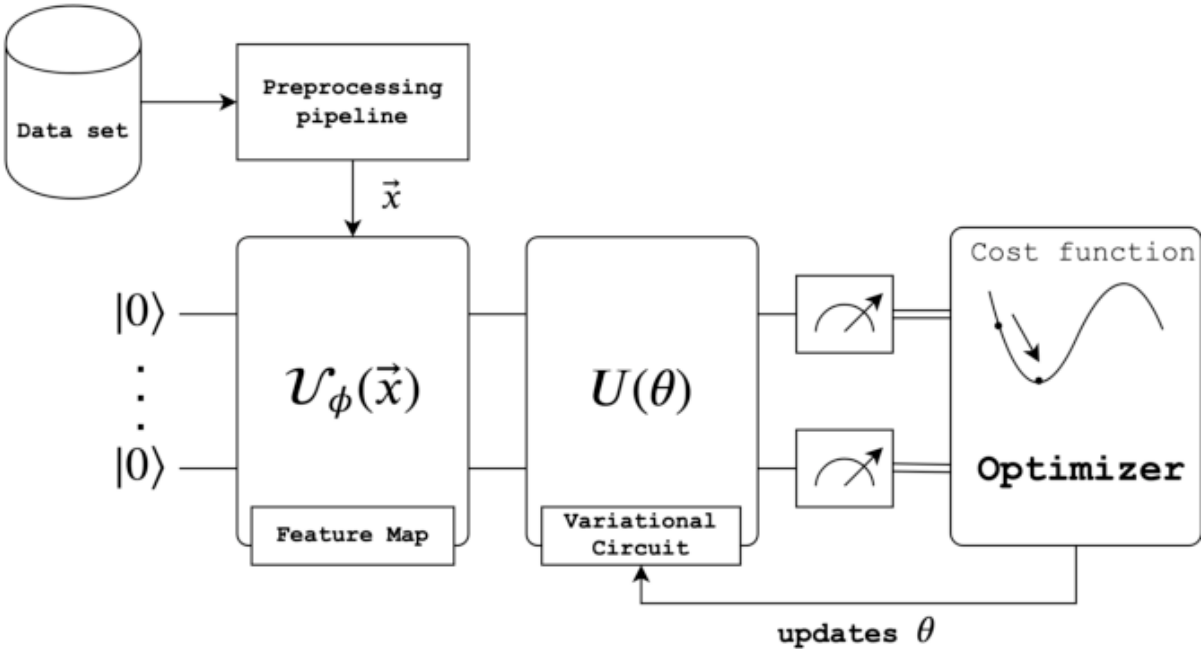

Figure 2.1: Variational Quantum Circuit[2]

2.3.1 Data encoding

A quantum embedding represents classical data as quantum states in a Hilbert space via a quantum feature map. It takes a classical data x and translates it into a set of gate parameters in a quantum circuit, creating a quantum state $|\psi_x\rangle$.

In this project I use *angle embedding* to encode classical data from dataset to the circuit (see line 10 in Figure 3.2). With angle embedding, single-qubit rotation gates encode a classical $x_i \in \mathcal{R}$.

Each element of the input determines the angle of the rotation gate (e.g. an RY rotation gate). This approach requires n qubits to encode n input variables and

can be defined as:

$$|\psi_x\rangle = \bigotimes_{i=1}^{n} cos(x_i)\,|0\rangle + sin(x_i)\,|1\rangle = \bigotimes_{i=1}^{n} R(x_i)\,|\psi_0\rangle$$

2.3.2 QCNNs

QCNN stands out among other parametrized quantum circuits (PQC) models for its shallow circuit depth, good generalisation capabilities and absence of *barren plateaus*.

A *barren plateau* happens when the gradient of a cost function vanishes exponentially with system size, rendering the architecture untrainable for large problem sizes. For PQC, random circuits are often proposed as initial guesses for exploring the space of quantum states, due to exponential dimension of Hilbert space and the gradient estimation complexity on more than a few qubits.

It is important to note that for a wide class of PQC the probability that the gradient along any reasonable direction is non-zero to some fixed precision is exponentially small as a function of the number of qubits[3]. For QCNN in particular it is guaranteed that randomly initialized QCNN are trainable unlike many other PQC architectures, since the variance of the gradient vanishes no faster than polynomially[4] so QCNNs do not exhibit *barren plateaus*.

The next step is learning network architecture, which NAS aims to achieve[5]. NAS consists of 3 main components:

- search space
- search strategy
- performance estimation strategy

The *search space* defines the set of possible architectures that a search algorithm can consider, and a carefully designed search space is important for search efficiency.

2.4 HierarQcal

HierarQcal is an open-source python package[6] that simplifies the process of creating general QCNN by enabling an hierarchical design process. It makes automatic generation of QCNN circuits easy and it facilitates QCNN search space design for NAS.

The package includes primitives such as *convolutions, pooling* and *dense layers* that can be stacked together hierarchically to form complex QCNN circuit architectures.

3 Model implementation

3.1 Dataset

I use the dataset GTZAN from Kaggle[7].

3.2 QCNN architecture

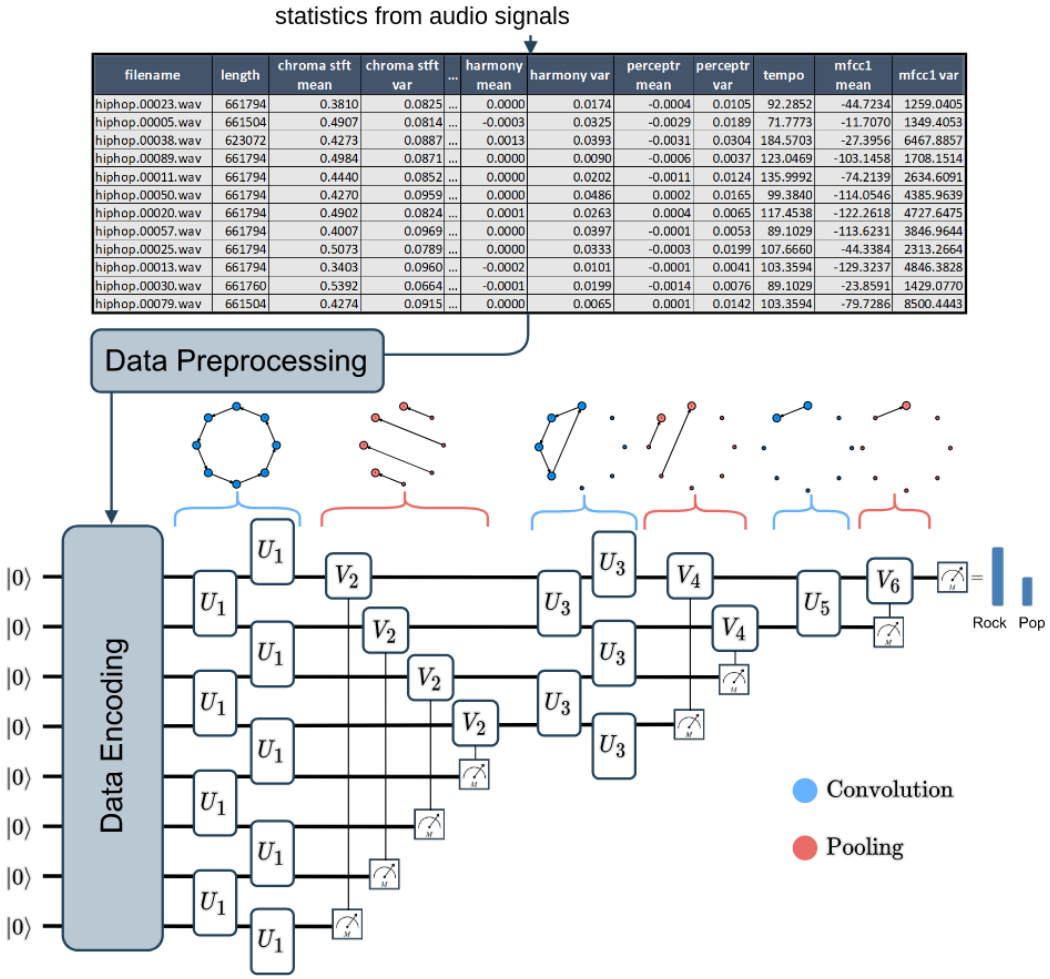

Figure 3.1: Main workflow...[1]

Circuit

The code below illustrates the motif built using HierarQcal.

```
def qcnn_motif(ansatz_c, conv_stride, conv_step, conv_offset,
 ↪  share_weights, ansatz_p, pool_filter):
    qcnn = (
        Qinit(8)
        + (
            Qcycle(
```

```
def get_circuit(hierq, x=None):
    dev = qml.device("default.qubit.torch", wires=hierq.tail.Q,
 ↪  shots=None)

    @qml.qnode(dev, interface="torch", diff_method="backprop")
    def circuit():
        if isinstance(next(hierq.get_symbols(), False), sp.Symbol):
            # Pennylane doesn't support symbolic parameters, so if
             ↪  no symbols were set (i.e. they are still symbolic),
             ↪  we initialize them randomly
            hierq.set_symbols(np.random.uniform(0, 2 * np.pi,
 ↪  hierq.n_symbols))
        if x is not None:
            AngleEmbedding(x, wires=hierq.tail.Q, rotation="Y")
        hierq(backend="pennylane")  # This executes the compute
 ↪  graph in order
        return qml.probs(wires=hierq.head.Q[0])

    return circuit
```

Figure 3.2

```
                stride=conv_stride,
                step=conv_step,
                offset=conv_offset,
                mapping=ansatz_c,
                share_weights=share_weights,
            )
            + Qmask(pool_filter, mapping=ansatz_p)
        )
        * 3
    )

    return qcnn
```

Table 3.1: One-hot vectorization example

	Conv. stride	Pool. filter	Mean validation score
0	2	!*	0.635831
1	3	!*	0.635831
2	2	!*	0.635831
3	3	!*	0.706753

The model was trained without batch for 100 epochs employing the Adam optimizer with a learning rate of $1 \times 10 - 1$ that minimizes the Cross-Entropy Loss. All experiments were performed using Pytorch and the PennyLane Quantum library[8] on an AMD Ryzen 7 PRO 5850U with 16GB of RAM.

References

1. Lourens, M., Sinayskiy, I., Park, D. K., Blank, C. & Petruccione, F. Hierarchical quantum circuit representations for neural architecture search. *npj Quantum Information* **9**, 79 (2023).
2. Building a Variational Quantum Classifier | Q-munity Tutorials.
3. McClean, J. R., Boixo, S., Smelyanskiy, V. N., Babbush, R. & Neven, H. Barren plateaus in quantum neural network training landscapes. *Nat. Commun.* **9**, 1–6 (2018).
4. Pesah, A. *et al.* Absence of Barren Plateaus in Quantum Convolutional Neural Networks. *Phys. Rev. X* **11**, 041011 (2021).
5. Elsken, T., Metzen, J. H. & Hutter, F. Neural architecture search: A survey. *The Journal of Machine Learning Research* **20**, 1997–2017 (2019).
6. matt-lourens. hierarqcal. *GitHub* (2024).
7. GTZAN Dataset - Music Genre Classification.
8. Bergholm, V. *et al.* Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968* (2018).