Library Proposal to Data Encoding in Quantum Machine Learning

Luccas Lanfranchi Marim, Fernando Augusto Caletti de Barros
Instituto de Pesquisas Eldorado, Campinas/SP – Brasil; Instituto de Pesquisas Eldorado, Porto Alegre/RS- Brasil

Abstract — One of the primary stages in machine learning algorithms involves translating input data from the system into numerical information that can be manipulated by the model. In Quantum Machine Learning (QML) context, once the information has been transformed into numerical data a further transformation is required to convert them into quantum data, which can be interpreted by the model or the quantum algorithm. These methods are commonly referred in literature as data encoding or data embedding. One challenge is implementing these methods using currently provided Software Development Kits (SDKs) such as Qiskit (IBM), Pennylane (Xanadu) and Paddle Quantum (Baidu), which in some cases provides ready-made functions to perform data encoding. One way to do this is presented in this poster through the three most common methods: basis encoding, amplitude encoding and angle encoding. Therefore, a brief conceptual overview of the subject is presented and a python library is created and explained implementing a solution to the data encoding problem.

I. INTRODUCTION

In machine learning domain the transformation of the input information into numerical data is an intrinsic step in any algorithm to enable its interpretation, manipulation, and learning process. Similarly in QML context, the data encoding process proceeds this step where input information is pre-processed [1], [2] to enable interaction in quantum devices.

The QML algorithm performance is directly impacted by the manner in which the data encoding process is carried out [1] as shown in [3], [4], and [5]. Therefore, given each problem an appropriate data encoding method needs to be chosen to ensure efficiency. Formally the data encoding process takes a set of values X_i as input and returns a set of quantum states $|X_i\rangle$, thereby creating a mapping that is at times a bijection between the data sets.

The library proposed in this article is based in three different quantum SDKs: Paddle Quantum from Baidu [6], Qiskit developed by IBM [7] and Pennylane developed by Xanadu [8]. All of the mentioned SDKs have useful features to implement and test solutions for QML [9].

This study is directly related to practical applications inherited from classical machine learning encompassing from supervised learning tasks such as classification and regression to optimization and reinforcement learning. These have relevance in a wide range of science and engineering fields including, but not limited to climate, pharmaceuticals, data analysis, material science, logistics and cryptography [10].

Although most applications are still limited by noise and number of qubits (quantum bits) available in NISQ (Noisy Intermediate Scale Quantum) era, roadmaps from big techs such as IBM [11] have projections that corroborate the quantum systems evolution.

II. DATA ENCODING OVERVIEW

Within the quantum computing scope, each data encoding method consists of a set of quantum operations applied sequentially in a quantum circuit, in order to generate an output equivalent to the encoded input [1], [4], [5] and [2]. In fact, usually one can start with a state in the form $|0\rangle^{\otimes n}$ and apply gates which will act on the qubits in order to create the quantum state corresponding to the transformation given a data encoding method. Each method has different types and amounts of quantum gates which will be used in the process. However, this transformation goes deeper than that. Indeed, applying a data encoding method to a quantum system one can transform classical data into quantum data which has a completely different nature in data type terms, but can represent the same value for the quantum system. After applying the transformation, the user become able to execute QML models, quantum operations and algorithms based on the transformed classical value.

In this section, each data encoding method will be briefly explained and simple examples will be provided for reader concept fixation.

A. Basis Encoding

The basis encoding is the most well known method among the currently studied methods and its name comes from the fact that binary input string is transformed into a quantum state vector preserving the digits as superficially elucidated in introduction section. Repeating the process for all input strings so that for each input in classic string format, the output will be a vector of the corresponding space basis. Indeed, the encoding is pretty intuitive for instance: 010011101 → $|010011101\rangle$.

B. Amplitude Encoding

Another method also very intuitive is amplitude encoding. The idea comes from inserting the input information into qubits amplitude factors, that is given each input, it is constructed a classical vector of size N and transformed the classical vector components into the quantum state components N with respect to the computational basis. Firstly, it is necessary to normalize the data coordinates so that it results in a quantum state that satisfy the quantum mechanics normalization condition.

To illustrate, let x = [½ ½ -½ -½] be the input data. Hence, the associated quantum state will be $|x\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle - \frac{1}{2}|10\rangle - \frac{1}{2}|11\rangle]$ which in this case is already normalized.

C. Angle Encoding

In the angle encoding process, each input vector coordinate is transformed into a parameter that represents the qubit rotation angle chosen a certain axis of the bloch sphere (e.g. X, Y or Z).

Indeed let x = [3 2 0 1] be the input data. If the angle encoding method is used and the axis of rotation X is chosen, the associated quantum state will be: $|x\rangle = R_x(3)|0\rangle \otimes R_x(2)|0\rangle \otimes R_x(0)\,|0\rangle \otimes R_x(1)\,|0\rangle$ which corresponds to the application of rotation gates in terms of X axis in each system qubit and the R_x gate is a rotation quantum gate example. In this case, the initial state considered was $|0\rangle^{\otimes n}$.

III. PROPOSED LIBRARY OVERVIEW

The proposed library in this work is intended to create an abstraction layer between the implementation of the three presented data encoding methods and the quantum SDK chosen by the user. For that purpose, the library supports the three mentioned SDKs. This section presents an usage example of the proposed library with a well-known dataset called digits, from sklearn.

```
for i in range(5): c.append(d[i])

qc,result1=QiskitDataEncoding().amplitude_encoding(c)
result1.draw("latex")
```

Fig. 1. Using Qiskit-based amplitude encoding from proposed library for data encoding a MNIST samples subset

Whose output is:

$$\frac{\sqrt{11}}{11}|010\rangle + 0.784|011\rangle + 0.542|100\rangle$$

In particular, the research identified that Qiskit does not have built-in functions for data encoding. Due to the described scenario, the Qiskit SDK will be used to demonstrate the usage of the developed library to create these data encoding functions in order to extend its range of applications.

```
def basis_encoding(self,x):
    x = ["{0:b}".format(i) for i in x]
    x=''.join(x)
    n=len(x)

    qc = QuantumCircuit(n)
    x=x[::-1]
    for i in range(len(x)):
        if x[i] == '1':
            qc.x(i)

    backend = Aer.get_backend('statevector_simulator')
    result = execute(qc,backend).result().get_statevector()
    return qc,result
```

```
qc1, result1 = QiskitDataEncoding().basis_encoding(x)
print("Qiskit:", result1)
```

Fig. 2. Comparative example of proposed library application. On the left, a function that performs basis encoding using Qiskit. On the right, the code that represents the optimization proposed by the library. Both implementations have the same output when called.

```
def angle_encoding(self,x):
    n=len(x)
    qc = QuantumCircuit(n)
    for b in range(0,n):
        qc.ry(x[b], b)

    backend = Aer.get_backend('statevector_simulator')
    result = execute(qc,backend).result().get_statevector()


    return qc,result
```

```
qc1, result1 = QiskitDataEncoding().angle_encoding(x)
print("Qiskit:", result1)
```

Fig. 3. Comparative example of proposed library application. On the left, a function that performs angle encoding using Qiskit. On the right, the code that represents the optimization proposed by the library. Both implementations have the same output when called.

Another useful function of the proposed library is dealing with the input problem when it is used a number of coordinates which is not a power of 2. An example situation is shown on figure 4.

```
def amplitude_encoding(self,x):
    n=int(math.log2(len(x)))

    if(math.log2(len(x)) / int(math.log2(len(x))) > 1):
        n += 1
        desired_state_size = 2**n
        for i in range(len(x), desired_state_size):
            x.append(0)

    desired_state = x / (np.linalg.norm(x))

    qc = QuantumCircuit(n)
    qc.initialize(desired_state,[*range(0,n)] )
    backend = Aer.get_backend('statevector_simulator')
    result = execute(qc,backend).result().get_statevector()
    return qc,result
```

```
qc1,result1=QiskitDataEncoding().amplitude_encoding(x)
result1.draw('latex')
```

Fig. 4. Comparative example of proposed library application. On the left, a function that performs amplitude encoding using Qiskit. On the right, the code that represents the optimization proposed by the library. Both implementations have the same output when called.

IV. CONCLUSION

A data encoding overview was presented with a simple application case using the proposed library which uses the three mentioned quantum SDKs. Qiskit does not have built-in functions that perform data encoding. Has been shown a way to create it using a function from the proposed library in order to extend its framework. On the other hand, Paddle Quantum and Pennylane have native functions for data encoding. Though, the proposed library can be useful to perform comparative simulations between frameworks, which is relevant in an era where quantum supremacy has not been achieved yet.

In the future, the team will go through a stage of design and optimization of the library's backend where the routines that perform each encoding method will be reformulated, so that the user-programmer would be more comfortable in creating code that involve data encoding. These tasks will be for example, creating auxiliary classes that resize the data, so that the programmer can have the option of working with different types of pre-processing stages before the encoding. As it is possible to see, there are still a lot of opportunities to develop software abstractions that facilitate the usage of QML solutions, and the team believes that each contribution can help quantum computing in its path to become useful for real world applications.

REFERENCES

1. B. Roy, “All about data encoding for quantum machine learning,” 2021.
2. JF. Shah, “Quantum encoding: An overview,” 2021.
3. R. LaRose and B. Coyle, “Robust data encodings for quantum classifiers,” Physical Review A, vol. 102, no. 3, pp. 1–2, 2020.
4. M. Weigold, J. Barzen, F. Leymann, and M. Salm, “Data encoding patterns for quantum computing,” in Proceedings of the 27th Conference on Pattern Languages of Programs, 2020, pp. 5–7.
5. M. Schuld, R. Sweke, and J. J. Meyer, “Effect of data encoding on the expressive power of variational quantum-machine-learning models,”
6. “Paddle Quantum,” 2020. [Online]. Available: https://github.com/PaddlePaddle/Quantum.
7. Qiskit contributors, “Qiskit: An open-source framework for quantum computing,” 2023.
8. V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi et al., “Pennylane: Automatic differentiation of hybrid quantum-classical computations,” arXiv preprint arXiv:1811.04968, 2018.: Automatic differentiation of hybrid quantum-classical computations,” arXiv preprint arXiv:1811.04968, 2018.
9. R. Ramouthar and H. Seker, “Hybrid quantum algorithms and quantum software development frameworks,” pp. 9–10, 2023.
10. A. Bayerstadler, G. Becquin, J. Binder, T. Botter, H. Ehm, T. Ehmer, M. Erdmann, N. Gaus, P. Harbach, M. Hess et al., “Industry quantum computing applications,” EPJ Quantum Technology, vol. 8, no. 1, pp. 5–8, 2021.
11. J. Gambetta, “Ibm’s roadmap for scaling quantum technology,” IBM Research Blog (September 2020), 2020.