Quantum Encoding of Classical Data using PennyLane

Kavya

March 2025

1 Introduction

This document presents the implementation of quantum encoding techniques using PennyLane. The Iris dataset is preprocessed and encoded using Basis Encoding, Amplitude Encoding, and Angle Encoding.

2 Dataset Preparation

The Iris dataset is loaded, and its features are normalized between 0 and 1 using Min-Max scaling. The first row is selected for encoding.

```
import pennylane as qml
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_iris

# Load and normalize Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(df)
row_data = normalized_data[0].astype(float)
```

3 Quantum Encoding Techniques

3.1 Basis Encoding

Maps classical binary data to quantum states. A fixed number of bits is assigned per feature, and qubits are initialized accordingly.

```
def basis_encoding(data, max_bits=4):
    scaled_data = np.round((data - np.min(data)) / (np.max(data) -
    np.min(data)) * (2**max_bits - 1))
    binary_data = np.concatenate([
```

```
        np.array(list(map(int, format(int(x), f'0{max_bits}b')))) for x in scaled_data
    ])
    dev = qml.device("default.qubit", wires=len(binary_data))

    @qml.qnode(dev)
    def circuit():
        for i, bit in enumerate(binary_data):
            if bit == 1:
                qml.PauliX(wires=i)
        return qml.state()
    return circuit()
```

3.2 Amplitude Encoding

Encodes data into quantum states by normalizing it and applying state preparation.

```
def amplitude_encoding(data):
    norm_data = data / np.linalg.norm(data)
    num_qubits = int(np.ceil(np.log2(len(norm_data))))
    padded_data = np.pad(norm_data, (0, 2**num_qubits - len(norm_data)), 'constant')
    dev = qml.device("default.qubit", wires=num_qubits)

    @qml.qnode(dev)
    def circuit():
        qml.MottonenStatePreparation(padded_data, wires=list(range(num_qubits)))
        return qml.state()
    return circuit()
```

3.3 Angle Encoding

Encodes classical data as rotation angles on qubits.

```
def angle_encoding(data):
    dev = qml.device("default.qubit", wires=len(data))
    @qml.qnode(dev)
    def circuit():
        qml.AngleEmbedding(data * np.pi, wires=list(range(len(data))), rotation='Y')
        return qml.state()
    return circuit()
```

4 Results and Storage

The encoded data is stored as NumPy arrays for later use.

```
basis_encoded = basis_encoding(row_data)
amplitude_encoded = amplitude_encoding(row_data)
angle_encoded = angle_encoding(row_data)
```

```python
np.save("basis_encoded.npy", basis_encoded)
np.save("amplitude_encoded.npy", amplitude_encoded)
np.save("angle_encoded.npy", angle_encoded)

loaded_basis = np.load("basis_encoded.npy")
loaded_amplitude = np.load("amplitude_encoded.npy")
loaded_angle = np.load("angle_encoded.npy")

print("Basis Encoded:", loaded_basis)
print("Amplitude Encoded:", loaded_amplitude)
print("Angle Encoded:", loaded_angle)
```

5 Conclusion

This implementation demonstrates quantum encoding techniques applied to classical data using PennyLane. The encoded quantum states can be utilized in quantum machine learning applications.