**Weather Forecasting with Classical vs Quantum Machine Learning # team 13 **

Introduction

Weather prediction is one of the most data-driven and computationally intensive tasks in modern science. Accurate forecasts impact everything from agriculture and energy systems to disaster preparedness and aviation. Traditionally, meteorologists relied on physical simulations using numerical weather prediction (NWP) models, but in recent decades, machine learning has emerged as a complementary approach, allowing models to learn directly from data.

Recently, a new frontier has opened: **Quantum Machine Learning (QML)**. It promises exponential improvements in computational capabilities by leveraging the unique properties of quantum mechanics—superposition, entanglement, and interference. But while theoretical quantum algorithms often outperform their classical counterparts in specific scenarios, **real-world implementation is still in its infancy**.

In this project, we aim to test the practicality of QML by applying it to a real-world problem: **predicting average daily temperature** based on weather features. We compare the results of a quantum model (EstimatorQNN from Qiskit) with a classical multilayer perceptron (MLPRegressor). The results not only highlight current limitations of quantum models but also emphasize their future potential.

Motivation

Quantum computing is still in its Noisy Intermediate-Scale Quantum (NISQ) era, where quantum computers have limited qubit counts and are prone to noise. Despite that, many researchers are actively exploring the possibility of using quantum circuits as neural networks. EstimatorQNN, for instance, is a quantum neural network architecture that maps classical data into quantum states using parameterized circuits, then trains the circuit using classical optimizers.

If quantum models can even approximate classical model performance, that would be a major proof-of-concept. The long-term goal is to build hybrid quantum-classical systems where the quantum part does the heavy lifting for high-dimensional feature spaces.

Environment Setup

We use Google Colab for ease of use and availability of required libraries. The key packages include Qiskit for quantum models, Scikit-learn for classical models, and common Python tools for data processing.

```
!pip install qiskit qiskit-machine-learning qiskit-aer qiskit-algorithms
matplotlib scikit-learn pandas numpy pylatexenc
!pip install --upgrade qiskit qiskit-machine-learning
```

Data Preparation

The dataset contains weather measurements such as rainfall, wind speed, humidity, cloud cover, and more, along with the target: average temperature (`temperature_2m_mean (°C)`).

```
from google.colab import files
uploaded = files.upload()

import io
import pandas as pd
filename = next(iter(uploaded))
data = pd.read_csv(io.BytesIO(uploaded[filename]))
```

Outlier Removal

We use Z-score to remove outliers, ensuring the model learns general patterns, not noise.

```
import numpy as np
from scipy.stats import zscore

features = [...]
target = 'temperature_2m_mean (°C)'

data = data.dropna(subset=features + [target])
z_scores = np.abs(zscore(data[features + [target]]))
data_clean = data[(z_scores < 3).all(axis=1)]

X = data_clean[features].values
y = data_clean[target].values
```

Feature Selection

We applied F-regression to choose the top 4 features most correlated with the target variable:

```
from sklearn.feature_selection import SelectKBest, f_regression

selector = SelectKBest(score_func=f_regression, k=4)
X_selected = selector.fit_transform(X, y)
important_features = np.array(features)[selector.get_support()]
```

This reduces the dimensionality for the quantum circuit, which suffers from exponential cost with increasing input size.

Data Scaling and Subsampling

Quantum simulators cannot handle large datasets efficiently. We used subsets of the data:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X_selected, y,
test_size=0.2, random_state=42)
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1)).flatten()
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1)).flatten()

X_train_subset = X_train_scaled[:500]
y_train_subset = y_train_scaled[:500]
X_test_subset = X_test_scaled[:100]
y_test_subset = y_test_scaled[:100]
```

Classical Model: Multilayer Perceptron (MLP)

```
from sklearn.neural_network import MLPRegressor

classical_model = MLPRegressor(hidden_layer_sizes=(64,), max_iter=500,
random_state=42)
classical_model.fit(X_train_scaled, y_train_scaled)
y_pred_classical_scaled = classical_model.predict(X_test_scaled)
y_pred_classical =
scaler_y.inverse_transform(y_pred_classical_scaled.reshape(-1, 1)).flatten()
```

Quantum Model: EstimatorQNN

```
from qiskit.circuit.library import ZZFeatureMap, RealAmplitudes
from qiskit.primitives import Estimator
from qiskit_machine_learning.neural_networks import EstimatorQNN
from qiskit_machine_learning.algorithms import NeuralNetworkRegressor
from qiskit_algorithms.optimizers import COBYLA

feature_map = ZZFeatureMap(feature_dimension=4, reps=3)
ansatz = RealAmplitudes(num_qubits=4, reps=5)
qnn = EstimatorQNN(
    circuit=feature_map.compose(ansatz),
    input_params=feature_map.parameters,
    weight_params=ansatz.parameters,
    estimator=Estimator()
)

optimizer = COBYLA(maxiter=500)
regressor = NeuralNetworkRegressor(neural_network=qnn, optimizer=optimizer)
regressor.fit(X_train_subset, y_train_subset)
```

Model Evaluation

We compare model performance using MAE, RMSE, and R^2 score:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

y_pred_qnn = regressor.predict(X_test_subset)
y_pred_qnn_original = scaler_y.inverse_transform(y_pred_qnn.reshape(-1,
1)).flatten()

mae_classical = mean_absolute_error(y_test, y_pred_classical)
rmse_classical = mean_squared_error(y_test, y_pred_classical, squared=False)
r2_classical = r2_score(y_test, y_pred_classical)

mae_quantum = mean_absolute_error(y_test[:100], y_pred_qnn_original)
rmse_quantum = mean_squared_error(y_test[:100], y_pred_qnn_original,
squared=False)
r2_quantum = r2_score(y_test[:100], y_pred_qnn_original)
```

Results:

- **Classical Model:** MAE = \~1.5°C, R^2 = \~0.85

- **Quantum Model:** MAE = \~3.0°C, R^2 = \~0.42

Visualization

We use Matplotlib to show predicted vs actual values:

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plt.plot(y_test[:50], label='Actual', marker='o')
plt.plot(y_pred_classical[:50], label='Classical', marker='x')
plt.title("Classical Model Predictions")

plt.subplot(1, 2, 2)
plt.plot(y_test[:50], label='Actual', marker='o')
plt.plot(y_pred_qnn_original[:50], label='Quantum", marker='x')
plt.title("Quantum Model Predictions")
plt.tight_layout()
plt.show()
```

Limitations and Challenges

- **Simulator Overhead:** Qiskit simulators are slow beyond a few hundred samples.
- **Noise and Scaling:** Quantum circuits are sensitive to noise, and scaling up requires better qubit quality and error correction.
- **Feature Limit:** We had to limit input features to 4 due to quantum circuit depth limitations.
- **Training Time:** QNN training is significantly slower even on classical hardware.

Conclusion

In this project, we demonstrated how both classical and quantum models can be used to forecast weather. While the **classical model clearly outperformed** the quantum one in terms of accuracy and reliability, the quantum model still showed promise. It managed to learn a useful mapping from limited data using a real quantum-inspired neural network.

These findings align with current global research: quantum machine learning is promising, but not yet superior to classical techniques in real-world regression tasks. However, as quantum hardware and software ecosystems mature, we expect this gap to close.

Bottom Line: Classical ML remains dominant for now, but quantum ML has entered the race—and it's running.