Financial Prediction Quantum Neural Network (QNN) App

Your Name

January 2, 2025

Abstract

This report presents the design and implementation of a Quantum Neural Network (QNN) application for financial prediction. The app leverages quantum computing principles combined with machine learning to analyze financial data, predict stock prices, and evaluate financial trends. The workflow, model development, backend and frontend integration, and deployment processes are discussed.

1 Introduction

Quantum computing has gained attention for its potential to solve problems that are computationally infeasible for classical computers. In the domain of financial prediction, quantum machine learning techniques can potentially enhance prediction accuracy. This project develops a Quantum Neural Network (QNN) application that predicts financial data trends, such as stock prices and cryptocurrency movements, by utilizing quantum algorithms alongside classical machine learning techniques.

2 Project Overview

The goal of the app is to combine quantum computing with machine learning to analyze historical financial data and make predictions about future stock prices and trends. The workflow consists of several stages: project planning, data collection and preprocessing, quantum model development, backend and frontend integration, deployment, and testing.

3 Features

The key features of the app include:

- **Quantum Neural Network Model**: Utilizes quantum computing to enhance prediction accuracy in financial markets.
- **Financial Data Analysis**: Capable of processing historical stock data, currency exchange rates, and other financial time series.
- **Prediction Output**: Provides probability predictions for stock prices, trends, or other financial indicators.
- **Jupyter Notebooks**: Interactive notebooks for model training, evaluation, and testing on various financial datasets.
- **Modular Architecture**: Allows for easy extension to other data types and prediction models.

4 Workflow

The project is structured into multiple stages, as described below.

4.1 1. Project Planning & Setup

- **Define Scope**: Identify the core features, including data input, prediction functionality, and visualizations.

- **Select Tools & Tech Stack**: The tech stack includes quantum libraries (PennyLane, Qiskit), backend frameworks (Flask or Django), frontend frameworks (React or Vue.js), and visualization tools (Plotly, D3.js).

4.2 2. Data Collection & Preprocessing

The data is fetched from Alpha Vantage API, which provides real-time and historical stock data. The stock data includes open, high, low, close, and volume values for a given symbol. Below is the Python code used to fetch and preprocess stock data:

```
import pandas as pd
import numpy as np
import requests
from sklearn.preprocessing import MinMaxScaler
from datetime import datetime

# Alpha Vantage API key (get one from https://www.alphavantage.co/support/#api-key)
API_KEY = 'H1C6A8TG86YEY64G'

# Function to fetch stock data from Alpha Vantage API
def fetch_stock_data(symbol, interval='daily', outputsize='full'):
    """Fetch stock data from Alpha Vantage API."""
    url = f"https://www.alphavantage.co/query"
    params = {
        'function': 'TIME_SERIES_DAILY' if interval == 'daily' else 'TIME_SERIES_INTRADAY',
        'symbol': symbol,
        'interval': '1d',  # 1 minute, 5 minutes, etc. for intraday
        'apikey': API_KEY,
        'outputsize': outputsize  # full or compact
    }
    response = requests.get(url, params=params)
    data = response.json()

    # Check if the request was successful and data is present
    if 'Time Series (Daily)' not in data:
        raise Exception(f"Error fetching data for {symbol}: {data.get('Note', 'Unknown error')}")

    # Convert the fetched data into a pandas DataFrame
    time_series = data[f'Time Series ({interval.capitalize()})']
    df = pd.DataFrame.from_dict(time_series, orient='index')

    # Convert the index to datetime format
    df.index = pd.to_datetime(df.index)

    # Rename columns to more convenient names
    df.columns = ['open', 'high', 'low', 'close', 'volume']

    # Convert column data to numeric
    df = df.astype({
        'open': 'float64',
        'high': 'float64',
        'low': 'float64',
        'close': 'float64',
        'volume': 'int64'
    })

    # Sort data by date (ascending)
    df = df.sort_index()

    return df

# Function to preprocess data (handling missing values, scaling, etc.)
def preprocess_data(df, scale=True):
    """Preprocess stock data (handle missing values and scale)."""

    # Handle missing data: drop rows with any missing values
    df = df.dropna()

```

```
    # Feature engineering: Extracting additional features if needed
    df['daily_return'] = df['close'].pct_change()  # daily percentage change in closing price
    df['5_day_moving_avg'] = df['close'].rolling(window=5).mean()
    df['30_day_moving_avg'] = df['close'].rolling(window=30).mean()

    # Drop rows with NaN created by rolling calculations
    df = df.dropna()

    # Scaling data: Use MinMaxScaler to scale data (feature normalization)
    if scale:
        scaler = MinMaxScaler(feature_range=(0, 1))
        scaled_columns = ['open', 'high', 'low', 'close', 'volume', 'daily_return', '5_day_moving_avg', '30_day_moving_avg']
        df[scaled_columns] = scaler.fit_transform(df[scaled_columns])

    return df
```

Listing 1: Fetching and Preprocessing Stock Data

The fetched data is then preprocessed to handle missing values and feature-engineered by adding new features like daily returns and moving averages. The data is normalized using MinMaxScaler to ensure proper scaling before training the model.

4.3 3. Quantum Neural Network Model Development

The QNN is developed using quantum computing frameworks like Qiskit or PennyLane. The model consists of:

- **Quantum Circuits**: Used to process financial data using quantum gates.
- **Variational Parameters**: Optimized during training to improve prediction accuracy.
- **Hybrid Approach**: Combines quantum circuits with classical optimization techniques.

5 Backend Integration

The backend server is built using Flask. It consists of several routes to handle stock data retrieval, data preprocessing, and generating predictions from the trained QNN model. Below is an overview of the backend code implementation:

```
from flask import Flask, request, jsonify
import pandas as pd
import torch
from qiskit import Aer
from qiskit.utils import QuantumInstance
from qiskit_machine_learning.connectors import TorchConnector
from qiskit_machine_learning.neural_networks import TwoLayerQNN
from qiskit.circuit.library import RealAmplitudes
from qiskit.circuit import ParameterVector
import yfinance as yf
import os

app = Flask(__name__)

# Load the trained QNN model
def create_qnn(num_qubits):
    feature_map = RealAmplitudes(num_qubits, reps=1)
    feature_map_params = ParameterVector('fm_theta', feature_map.num_parameters)
    feature_map.assign_parameters(feature_map_params, inplace=True)

    ansatz = RealAmplitudes(num_qubits, reps=1)
    ansatz_params = ParameterVector('ansatz_theta', ansatz.num_parameters)
    ansatz.assign_parameters(ansatz_params, inplace=True)

    qnn = TwoLayerQNN(
        num_qubits=num_qubits,
        feature_map=feature_map,
        ansatz=ansatz,
        quantum_instance=QuantumInstance(
```

```
            Aer.get_backend("qasm_simulator"),
            shots=1024,
            optimization_level=1,
            backend_options={"max_parallel_threads": 4}  # Enable parallelization
        )
    )
    return qnn

@app.route('/predict', methods=['POST'])
def predict():
    # Get data from the POST request
    data = request.get_json()
    df = pd.DataFrame(data)
    model, losses = train_qnn(df['X_train'], df['y_train'], num_qubits=4, epochs=10, learning_rate=0.01)
    prediction = model(torch.tensor(df['X_test']))
    return jsonify(prediction.tolist())

if __name__ == "__main__":
    app.run(debug=True)
```

Listing 2: Flask Backend for QNN Prediction

5.1 Frontend Integration

The frontend of the app is built with Vue.js. The main page allows users to either upload a CSV file or fetch stock data by entering a ticker symbol. The page includes sections for uploading files, fetching stock data, and displaying prediction results. Below is the Vue.js code for the frontend:

```
<template>
  <div id="app">
    <header>
      <h1>Quantum Stock Predictor</h1>
    </header>

    <section class="input-section">
      <h2>Upload Stock Data or Fetch by Ticker</h2>
      <div class="file-upload">
        <input type="file" @change="handleFileUpload" />
        <div class="ticker-input">
          <input v-model="ticker" type="text" placeholder="Enter stock ticker (e.g., AAPL)" />
          <button @click="fetchStockData">Fetch Data</button>
        </div>
      </div>
    </section>

    <section class="prediction-section" v-if="stockData.length">
      <h2>Stock Data</h2>
      <plotly-chart :data="stockGraphData" :layout="graphLayout" />

      <button class="predict-btn" @click="predictStockPrices">Predict Future Prices</button>

      <div v-if="predictions.length">
        <h2>Predicted Prices</h2>
        <plotly-chart :data="predictionGraphData" :layout="graphLayout" />
      </div>
    </section>

    <!-- Floating Animation in the Bottom Right -->
    <div class="animated-graph-icon">
      <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" width="50" height="50">
        <path fill="none" d="M0 0h24v24H0z"/>
        <path fill="#00aaff" d="M5 3h14c1.1 0 1.99.9 1.99 2L21 19c0 1.1-.89 2-1.99 2H5c-1.1 0-1.99-.9-1.99-2L3 5c0-1.1.89-2 1.99-2zm0 2v14h14V5H5zm7 6h5v2h-5v-2zm0-4h5v2h-5V7zm0 8h5v2h-5v-2z"/>
      </svg>
    </div>
  </div>
</template>
```

Listing 3: Vue.js Frontend for User Interaction

6 Conclusion

The use of quantum computing in financial prediction shows promising potential for improving prediction accuracy. The optimized QNN training process, including reducing dataset size, using shallow quantum circuits, and enabling parallelization, enhances the speed and efficiency of model training, making it feasible to deploy the model in a production environment.