Blockhouse Work Task Trial

Predicting Stock Prices and Developing a Trading Strategy Using a Transformer Model

INDEX

* INTRODUCTION
* IMPLEMENTATION DETAILS
  + Installation of required libraries
  + Data preprocessing
  + Feature Scaling
  + Dataset and DataLoader Creation
  + Model Creation (why opted that specific model)
  + Training the Model
  + Model evalutation and Trade Recommendation
  + Reinforcement learning-based trade strategy
* CONCLUSION

1. **Introduction**

Objective:

This document outlines the implementation and fine-tuning process of a transformer-based model designed to generate trade recommendations. The objective of this project is to utilize machine learning techniques to analyze market data and provide actionable insights for trading decisions. The model incorporates technical indicators, LSTM modeling, and reinforcement learning principles to enhance its predictive accuracy and suitability for real-world trading applications.

1. **Implementation Details**

* Installation of Required Libraries

To ensure compatibility and functionality, the following libraries and tools were installed:

1. TA-Lib: A library for technical analysis in financial markets, providing essential indicators.
2. Python Development Tools: Including build-essential, cmake, libffi-dev, and python3-dev for compiling and installing TA-Lib.

* Data Preprocessing

Data Loading and Initial Adjustments

1. Dataset: Market data was sourced from a CSV file containing price, volume, bid, and ask data.
2. Adjustments: Prices were normalized for easier handling and consistency across data points.

Adding Technical Indicators

1. Technical Indicators: Various technical indicators were computed using TA-Lib, including RSI, MACD, Stochastic Oscillator, Bollinger Bands, ATR, ADX, and others. These indicators capture different aspects of market behavior and provide crucial signals for trading decisions.

Handling Missing Values

Data Cleaning: Rows with missing values resulting from indicator calculations were dropped to maintain data integrity and ensure accurate model training.

* Feature Scaling

1. Normalization: Features were standardized using StandardScaler from sklearn, ensuring mean normalization and variance stabilization across the dataset.
2. Target Definition: The target variable was defined as the next day's price movement categorized into Buy, Sell, or Hold based on percentage change in closing prices.

* Dataset and DataLoader Creation

1. Sequence Length: Data was structured into sequences of 60 time steps to capture temporal dependencies.
2. Custom Dataset: A custom PyTorch Dataset class was implemented to facilitate efficient data loading, indexing, and batching during model training.

* Model Creation

For this I opted for LSTM. LSTM (Long Short-Term Memory) model for trading recommendations is beneficial because:

* Temporal Dependencies: LSTMs are designed to capture and leverage patterns over time, which is essential for financial time series data where past prices influence future prices.
* Handling Sequential Data: LSTMs can effectively manage sequential data, making them suitable for analyzing trends and making predictions based on historical data.
* Mitigating Long-Term Dependency Issues: LSTMs address the vanishing gradient problem, allowing them to learn from long-term dependencies in the data, which is critical in financial markets.
* Flexibility: LSTMs can model complex relationships in trading data, capturing non-linear patterns and interactions between different market factors.

LSTM Model Architecture

1. Long Short-Term Memory (LSTM): Chosen for its ability to model sequential data and capture long-term dependencies.
2. Architecture: The model consisted of an LSTM layer followed by a fully connected layer mapping LSTM outputs to the final prediction classes (Buy, Sell, Hold).

Loss Function and Optimization

1. Loss Function: Cross-Entropy Loss was utilized to optimize the model for multi-class classification tasks.
2. Optimizer: Adam optimizer was chosen for its adaptive learning rate capabilities, aiding in faster convergence during training.

* Training the Model
  1. Training Loop: The model was trained over multiple epochs, with each epoch involving forward and backward propagation of data batches.
  2. Loss Monitoring: Training loss was monitored and logged after each epoch to track model learning and convergence.
* Model Evaluation and Trade Recommendations

1. Evaluation: The trained model was evaluated on a separate test dataset to assess its accuracy in predicting trade recommendations (Buy, Sell, Hold).
2. Recommendations: Generated trade recommendations were mapped to corresponding dates and saved in a CSV file for further analysis and application.

* Reinforcement Learning-Based Trading Strategy

Trading Environment

1. Custom Environment: A custom trading environment was developed using OpenAI's Gym framework, simulating real-world trading scenarios.
2. State Representation: State information included market indicators and current portfolio status.

Agent Implementation

1. Neural Network Policy: A simple policy gradient-based agent was implemented to interact with the trading environment.
2. Training: The agent was trained over multiple episodes using rewards to learn optimal trading strategies and maximize cumulative profit.
3. Conclusion

* The transformer model effectively predicts stock prices, and the trading strategy shows potential for profitability. Future work includes live trading implementation and further model comparisons.