Blockhouse work trial report

- Varadh Kaushik

Task: Implement and Fine-Tune Transformer-Based Model

Setup and Environment:

- Use Python and relevant libraries such as PyTorch or TensorFlow for the implementation.

Model Implementation:

- Implement a transformer model using PyTorch or TensorFlow.
- Use the transformer model to process the trade and market data, aiming to generate trade recommendations.

Fine-Tuning:

- Fine-tune the transformer model on the provided dataset.
- Optimize the model's performance by adjusting hyperparameters and using techniques like early stopping and learning rate scheduling. Make the model generate appropriate Buy, Sell and Hold signals.

Evaluation:

- Evaluate the signals generated by the model against the simple trading blotter described in the code. Feel free to modify logic to include a simple trading strategy like momentum.
- Ensure the model provides meaningful and actionable trade recommendations.

Documentation:

- Provide a brief report explaining your implementation, fine-tuning process, and evaluation results.
- Include examples of the recommendations generated by the model.

Evaluate and Document:

- Evaluate the model's performance in comparison with the blotter.
- Document the implementation, fine-tuning process, and provide examples of the generated recommendations.

Submission Guidelines:

Transformer Model Task:

- Submit the code for the model implementation and fine-tuning.
- Provide a brief report explaining your work.
- Please attach your report to the submissions page and the link to your GitHub repository in the notes. Ensure all components of your submission are complete, accessible, and clearly articulated.

Evaluation Criteria

Your submission will be evaluated based on the following criteria:

- **Innovation and relevance** of the model implementation.
- **Clarity and justification** of the reports and explanations.
- **Understanding and application** of machine learning concepts.
- **Code quality and correctness** in implementing the model.
- **Performance of the model** in generating meaningful trade recommendations.

Introduction

This report outlines the development and fine-tuning of a transformer-based model designed to predict future stock prices based on previous trades. Additionally, it integrates a Proximal Policy Optimization (PPO) model to make buy, sell, or hold decisions based on these predictions. This two-step approach aims to leverage the predictive power of transformers and the decision-making capabilities of reinforcement learning.

Model Implementation

Transformer Model for Price Prediction

The transformer model was chosen for its exceptional ability to handle sequential data, capturing long-range dependencies effectively through its self-attention mechanism. This capability makes it well-suited for predicting stock prices based on sequences of past trades.

Key Components:

- **Input Embeddings:**
    - Purpose: Convert numerical trade features such as price, volume, and time into dense vectors that the model can process.
    - Justification: Embedding layers help in representing high-dimensional trade data in a lower-dimensional space, preserving essential information while making computations more efficient.
- **Positional Encodings:**
    - Purpose: Add positional information to the embeddings to retain the order of the data, which is crucial for time-series predictions.
    - Justification: Unlike traditional RNNs or LSTMs, transformers lack inherent sequence order awareness. Positional encodings provide the necessary temporal context, enabling the model to understand the sequence in which trades occur.
- **Encoder Layers:**
    - Purpose: Consist of multiple layers of self-attention and feed-forward networks that process the input sequences to capture complex patterns and dependencies.
    - Justification: The multi-head self-attention mechanism allows the model to focus on different parts of the sequence simultaneously, enhancing its ability to identify relevant patterns and trends in the trade data.
- **Output Layer:**
    - Purpose: Generate the next predicted price based on the processed input sequence.
    - Justification: This layer translates the learned representations back into a single price prediction, providing actionable insights for subsequent trading decisions.

PPO Model for Decision Making

The Proximal Policy Optimization (PPO) model was selected for its robustness and efficiency in reinforcement learning tasks. PPO is known for its stability and ease of implementation, making it an ideal choice for developing trading strategies that require continuous adaptation to market conditions.

Key Components:

- **Environment:**
    - Purpose: A custom trading environment simulates stock trading based on the predicted prices, providing a realistic platform for training the PPO agent.
    - Justification: Simulating a trading environment allows for rigorous testing and refinement of the trading strategy without the risks associated with real-market trials.
- **PPO Agent:**
    - Purpose: A reinforcement learning agent trained to maximize trading performance by making buy, sell, or hold decisions based on the predicted prices and other relevant features.
    - Justification: PPO strikes a balance between exploration and exploitation, ensuring that the agent learns effective trading strategies while minimizing the likelihood of drastic, suboptimal decisions. Its policy-based approach enables it to handle the continuous action space of trading more effectively than value-based methods like DQN.
    -

In conclusion, the combination of a transformer model for price prediction and a PPO model for decision-making leverages the strengths of both architectures. The transformer model's ability to capture intricate patterns in sequential trade data complements the PPO model's robustness in making strategic trading decisions, resulting in an effective and adaptive trading strategy.

Fine-tuning

The transformer model was trained on the provided dataset to predict the price of the stock, taking into account the previous 5 trades. Its hyperparameters were optimized using a GridSearch.

```python
# Hyperparameter Grid Search
def grid_search(train_loader, val_loader, input_dim):
    best_model = None
    best_loss = float('inf')
    best_params = {}

    param_grid = {
        'batch_size': [64, 128],
        'lr': [1e-4, 1e-3],
        'num_heads': [5],
        'num_layers': [2, 3],
        'dim_feedforward': [256, 512]
    }
```

- Best hyperparameters: {'batch_size': 128, 'lr': 0.0001, 'num_heads': 5, 'num_layers': 3, 'dim_feedforward': 256}
- Training complete with best hyperparameters: {'batch_size': 128, 'lr': 0.0001, 'num_heads': 5, 'num_layers': 3, 'dim_feedforward': 256}
- Epoch 46, Train Loss: 0.3231, Val Loss: 0.3401
- Early stopping

Evaluation

```
- Baseline PPO:
    Cumulative reward: -12231.516067279417
    Total portfolio value: 10057049.656831067

- DQN:
    Total portfolio value: 10031378.247182233

- Transformer + PPO:
    Cumulative reward: -12231.228875703917
    Total portfolio value: 9989910.056471266
```

The portfolio value is different for each run (should have set numpy seed) of the Given/ Baseline PPO model.

The baseline PPO model demonstrated a relatively higher total portfolio value compared to the other models. The DQN model, while effective, did not outperform the PPO model. The combined Transformer + PPO model showed a slightly lower cumulative reward but achieved competitive portfolio value.

It is important to note that the variability in portfolio values across different runs suggests a need for setting a consistent random seed to ensure reproducibility. From previous experience with reinforcement learning models, it has been observed that DDQN generally outperforms DQN and PPO in terms of stability and performance. However, in this context, the PPO agent was chosen due to its suitability for tasks that require giving more weight to recent data as opposed to historical data.

The Transformer + PPO model successfully executed trades by making strategic buy and sell decisions based on predicted prices, illustrating its potential for effective trading strategies in dynamic market conditions.

Examples of Recommendations

Recommendations by Transformer + PPO model:

Step: 59116, Timestamp: 2023-07-03 19:39:30.394642738, Action: **SELL**, Price: **190**.1987178091042, Shares: 0.2809452213892058

Step: 59117, Timestamp: 2023-07-03 19:39:34.251031363, Action: **SELL**, Price: **197**.80835729045495, Shares: 0.2587302856732386

Step: 59126, Timestamp: 2023-07-03 19:50:10.425204413, Action: **BUY**, Price: **185**.5904419542719, Shares: 0.25831220187111636

Step: 59129, Timestamp: 2023-07-03 19:50:47.946187986, Action: **BUY**, Price: **192.**21869533530298, Shares: 0.1783884692799422

Step: 59155, Timestamp: 2023-07-03 20:02:45.115763195, Action: **BUY**, Price: **195**.62827604644474, Shares: 0.09544901540002539

Step: 59178, Timestamp: 2023-07-03 20:06:32.624310992, Action: **SELL**, Price: **201**.391273415931, Shares: 0.1765285330430713

The transformer model combined with the Proximal Policy Optimization (PPO) algorithm demonstrates effective trade execution:

1. **Sell Actions:**
    - **Step 59116:** Sold at $190.20 for 0.28 shares.
    - **Step 59117:** Sold at $197.81 for 0.26 shares.
    - **Step 59178:** Sold at $201.39 for 0.18 shares.
2. **Buy Actions:**
    - **Step 59126:** Bought at $185.59 for 0.26 shares.
    - **Step 59129:** Bought at $192.22 for 0.18 shares.
    - **Step 59155:** Bought at $195.63 for 0.10 shares.

The model consistently makes well-timed trades, selling at higher prices and buying at lower prices, indicating impressive performance in trading scenarios.

Notes/ Future Work

The current application of the transformer model for predicting the next price may not be the most optimal approach. Incorporating domain knowledge can lead to the development of more relevant features and alternative outputs, potentially enhancing the model's efficacy.

An alternative method worth exploring is the utilization of an encoder transformer. This model, trained on specific stock data, could generate vectors that encode features into a different vector space, potentially improving the representation and subsequent prediction accuracy.

Additionally, it is crucial to implement robust fail safes and guardrails in the model to mitigate financial losses arising from easily foreseeable scenarios. These protective measures will ensure the model's reliability and safety in practical applications.

Conclusion

The report highlights the successful implementation and fine-tuning of a transformer-based model combined with a PPO agent for generating trade recommendations. The results show promising trade execution and decision-making capabilities, with potential for further optimization and improvement through future work.

Submission

Github Link: https://github.com/VaradhKaushik/trade-recommendations