Predictive Trading with Transformer Models

Zohair Hashmi | University of Illinois Chicago | hashmi.zohair@gmail.com

Introduction:

Transformers, initially introduced in the field of natural language processing (NLP), are widely used for various tasks such as language translation, text summarization, and sentiment analysis. Their self-attention mechanism allows them to capture long-range dependencies and relationships within data, making them highly effective for sequence-based tasks.

In our use case, we leverage the power of transformer models to analyze financial data, specifically trade and market data for Apple's stocks. By interpreting patterns and trends, the model provides trade recommendations—buy, sell, or hold—at each step of the market data. This approach aims to enhance decision-making processes in trading by utilizing the sophisticated pattern recognition capabilities of transformers, driven by Technical Indicators derived from the available data.

Model Implementation:

1. Data Processing

The provided Reinforcement Learning (RL) model in the RL-PPO Jupyter notebook generates trade recommendations based on the trade's dataset. It evaluates each trade within a window of 60 preceding trades to recommend an action regarding the number of shares to be bought, sold, or whether no transaction should be conducted.

We use the same dataset as input for the transformer model, validating the loss incurred by the generated predictions against the actual outcomes produced by the RL model (Trade Blotter's outcomes).

Input: The ticker dataset is created using the Technical Indicators class, which derives the necessary features for making trading decisions from the initial dataset.

- **Features Set:** Price, RSI, MACD, MACD_signal, MACD_hist, Stoch_k, Stoch_d, OBV, Upper_BB, Middle_BB, Lower_BB, ATR_1, ADX, +DI, -DI, and CCI.
- **Creating Sequences:** The input data is converted to sequences of size 60, consistent with the window size used by the original Blotter. This ensures that the Blotter uses current trade information along with rolling information from the preceding 60 trades to decide the appropriate action. Input Sequences Shape: (59236, 60, 16), Output Labels Shape: (59236,)
- **Normalization:** The sequences are further normalized to ensure the data is scaled appropriately, enhancing the model's ability to learn patterns and make accurate predictions.
- **Train/Test Split:** Dataset is split into training and test sets, with the training set further split into training and validation set for model training.

Output: We use the blotter's trade recommendations as the comparable output for our transformer-based model. The output initially contains the BUY and SELL signals only for the respective steps the action was taken on. We label the rest of the steps as HOLD, assigning numerical labels to each respective action.

- **Actions:** {0: HOLD, 1: BUY, 2: SELL}

Length of Datasets: Train: 40280, Validation: 10070, Test: 8886

2. Model Architecture

The transformer model's strength lies in its scalability and ability to capture complex patterns in sequential data, making it suitable for trade and market data analysis. For our use case of generating trade recommendations based on financial data, we adapted the transformer architecture to process sequences of technical indicators. The implementation involves the following key components:

Positional Encoding: Our `PositionalEncoding` class is designed to add positional information to the input embeddings. The encoding is created using sine and cosine functions of different frequencies, ensuring that each position in the sequence is uniquely represented.

TransformerTradeModel: Our custom model `TransformerTradeModel` is designed to take sequences of technical indicators and output trade recommendations. The model architecture includes:

- **Embedding Layer**: A linear layer that projects the input features into a higher-dimensional space suitable for the transformer.
- **Positional Encoding**: Adds positional information to the embedded inputs.
- **Transformer Encoder**: A stack of transformer encoder layers that process the input sequence and capture dependencies among the features.
- **Fully Connected Layer**: A linear layer that reduces the encoded sequence to the final output, representing the trade recommendation

In this model, the input data is first embedded and enriched with positional encoding. It then passes through multiple transformer encoder layers, which apply self-attention and feedforward neural networks to capture complex patterns in the data. Finally, a fully connected layer generates the trade recommendation based on the processed sequence. This architecture allows the model to effectively learn from historical trade data based on the sequence of the last 60 trades and make informed predictions.

Hyper-Parameter Tuning: Selecting the optimal hyper-parameters is crucial for the performance of the transformer model. We experimented with various values for key hyper-parameters such as the embedding dimension, the number of attention heads, the number of transformer encoder layers, the learning rate, and the dropout rate.

```
input_dim = 16  # Number of input features from the ticker dataset
embed_dim = 64 # Embedding dimension: the input dimension is embedded into a higher dimensional space
num_heads = 8 # Number of attention heads in the multi-head attention model
num_layers = 4 # Number of transformer layers
output_dim = 3  # Number of Output Classes (HOLD, BUY, SELL)
dropout = 0.5  # Dropout rate for regularization
patience = 5 # Patience for early stopping
learning_rate = 0.0005 # Learning rate for the optimizer
num_epochs = 40 # Number of epochs for training
```

- Instead of applying a grid search approach (which would be the ideal approach for the task), we used **trial-and-error** approach to identify the best hyper-parameters. This was in accordance with the limited computing resources available.
- The **dropout rate** and **patience** threshold were specifically modified to prevent overfitting to the model starting with lower dropout and higher patience to observe resulting accuracy and F1 score of the model.

- The **learning rate** was reduced from 0.01, 0.001 to 0.005 to allow the model to capture more complex information from the sequential data.

The tuning process aimed to balance model complexity and generalization capability, ultimately leading to improved accuracy and reliability in trade recommendations.

3. Model Training, Validation & Testing

Training Data Preparation

The dataset was divided into training (70%), validation (15%), and test (15%) sets to ensure that the model was evaluated on unseen data during both the validation and testing phases. Initially, the data was split into an 85-15 ratio, where the last 15% was reserved for testing. This decision was based on the assumption that the model would be trained as more trading data became available throughout the day, allowing for better predictions for the later part of the day. This split ratio was chosen to provide a sufficient amount of data for training while maintaining robust validation and test sets. No data augmentation techniques were applied, as the sequential nature of the data required preserving its integrity.

Training Process

The training process involved iterating over the training data in mini-batches. For each batch, the forward pass computed the model's predictions, the loss was calculated using the Cross Entropy Loss function, and the backward pass updated the model parameters using the Adam optimizer with a learning rate of 0.0005 and a dropout rate of 0.5, as indicated in the hyperparameters earlier.

Validation Strategy

During training, the model's performance was monitored on the validation set after each epoch. Early stopping was implemented based on the validation loss, with training halting if the loss did not improve for 10 consecutive epochs. This strategy helped prevent overfitting and ensured the model's generalization capability.

Evaluation Metrics

The model's performance was evaluated using accuracy and the F1 score. These metrics were tracked after the training process, providing insights into the model's ability to make trade recommendations closer to the Blotter’s recommendations.

Challenges and Solutions

One challenge encountered was the model's initial tendency to overfit the training data. This was addressed by implementing early stopping and applying dropout regularization. Additionally, careful tuning of the learning rate and batch size helped stabilize the training process.

Fine-Tuning:

Hyperparameter Adjustments

1. **Learning Rate:** Different learning rates were tested from the set [0.01, 0.005, 0.001]. Optimal behavior was observed at 0.005, which provided more precise weight updates without overshooting. A learning rate of 0.01 was too high, causing unstable training, while 0.001 was too low, leading to slower convergence.
2. **Dropout:** Dropout rates were tested from the set [0.1, 0.3, 0.5, 0.7]. The model returned the best accuracy and F1 score with a dropout rate of 0.5, as it effectively captured the most complex relationships within the sequences. A higher dropout rate (e.g., 0.7) resulted in the model losing

important information, which reduced accuracy, while lower rates (e.g., 0.1 and 0.3) did not sufficiently prevent overfitting.

3. **Patience/Early Stopping:** The patience threshold for early stopping was tested with values of 5 and 10, with 10 epochs proving suitable. Early stopping monitors the model's performance on the validation set and halts training when the validation loss fails to improve for 10 consecutive epochs compared to the best recorded validation loss. This strategy prevents overfitting by preventing scenarios where the model's validation performance declines while its training performance improves. Higher patience allows more training time, potentially risking overfitting, whereas lower patience may prematurely halt training before optimal performance is achieved.
4. **Number of Epochs:** The number of epochs was increased from 10 to 40. The model often stopped before reaching 40 epochs due to early stopping, which prevented overfitting. Therefore, setting 40 epochs as the maximum allowed sufficient training time while relying on early stopping to halt training once performance plateaued.

Evaluation:

To assess the Transformer Model's performance in a trading environment, we first calculated its test accuracy and F1 score. The model achieved a test accuracy of 0.8482 and an F1 score of 0.8480 on unseen test data, indicating robust performance in classifying trade actions as either 'BUY', 'SELL', or 'HOLD'. To further compare its effectiveness with practical trading outcomes, we predicted actions for the entire dataset and compared them with those of the Blotter. The model predominantly recommended 'HOLD' actions (29,412), followed by 'BUY' (16,162) and 'SELL' (13,662) actions, aligning closely with typical trading strategies.

Model Comparison:

In a simulated trading environment with identical initial capital and transaction costs, both the Transformer Model and the Blotter were evaluated based on their final portfolio values. The Transformer Model incorporated a Momentum Strategy, making trading decisions based on recent price changes. This strategy aimed to confirm 'BUY' or 'SELL' actions by observing price momentum: 'BUY' if prices showed upward momentum and 'SELL' if downward.

Without the Momentum Strategy, the Transformer Model managed capital effectively, achieving a total portfolio value of 9,981,649.012. Implementing the Momentum Strategy increased its portfolio value to 10,069,832.658, showcasing improved performance through strategic trading decisions.

Comparing results, the Transformer Model achieved a portfolio value of 10,069,832.658 with the Momentum Strategy, while the Trading Blotter achieved 10,057,494.458. This slight outperformance by the Transformer Model, approximately 12,338.2 in total portfolio value, highlights its predictive capabilities and strategic decision-making under similar trading conditions.

Key Observations

- **Comparative Performance with Trading Blotter:** The Transformer Model demonstrates performance comparable to the Trading Blotter, evidenced by a similar distribution of actions recommended—predominantly 'HOLD', followed by 'BUY' and 'SELL'. This similarity underscores the model's ability to analyze market data and make informed trade recommendations akin to traditional trading strategies.
- **Evaluation in Trading Environment:** Evaluation metrics used for both the Transformer Model and the Blotter include profitability and effective management of transaction costs. The Transformer Model's ability to generate profits and maintain robustness against transaction costs showcases its practical utility in real-world trading scenarios.

- **Impact of Momentum Trading Strategy:** Introducing the Momentum trading strategy enhances the Transformer Model's performance by refining trading decisions based on recent price trends. This adaptive strategy reduces the number of trading actions, highlighting the model's flexibility in responding to different market conditions and improving overall portfolio outcomes.
- **Consistency in Profit Generation:** To better interpret the model's performance, executing the trading environment multiple times and averaging the portfolio values provides insights into its consistent profit-generating capabilities over time. This approach mitigates the impact of individual trading outcomes, offering a more reliable assessment of the model's effectiveness.
- **Hyperparameter Optimization:** The Transformer Model's hyperparameters, including the number of layers, hidden units, and learning rate, were optimized through iterative experimentation. This optimization process aimed to maximize model performance, ensuring it effectively learns from historical data and adapts to changing market dynamics.