Systematic trading : predicting cross-sectional returns of US Equities using Recurrent Neural Networks

Martin Boutier
martin.boutier@ensae.fr

1 Introduction

As financial markets continue to evolve and generate vast amounts of data, the integration of advanced technologies such as Deep Learning has become pivotal in the realm of quantitative trading. Deep Learning, renowned for its capacity to model intricate and non-linear relationships between various features and a target variable, has transcended its application domains, extending its influence into quantitative finance. In this study, we discuss the use of recurrent neural networks (RNNs) to forecast next-day stock returns specifically for SP500-listed stocks.

Recurrent Neural Networks (RNNs) stand as a distinctive class of artificial neural networks designed to address temporal dependencies in sequential data, making them particularly suitable for time-series forecasting tasks. Unlike traditional feedforward neural networks, RNNs possess internal memory that enables them to capture and utilize information from previous time steps. This memory retention mechanism allows RNNs to discern intricate patterns and dependencies within sequential data, making them well-suited for modeling the dynamic and time-varying nature of financial markets.

The primary objective of this project is to leverage the power of Recurrent Neural Networks to discern and exploit momentum and mean-reversion patterns in US stocks. Momentum refers to the tendency of an asset's price to persist in its current direction, and mean reversion suggests that prices tend to revert to their historical average over time. Using RNNs, we aim to develop a predictive model that can effectively identify and capitalize on these market dynamics. The significance lies in the potential enhancement of trading strategies, leading to more informed and profitable decision-making in the dynamic landscape of financial markets.

The subsequent sections will delve into the methodology, data preprocessing, and model architecture, providing a comprehensive understanding of the intricacies involved in implementing several RNN based trading models for SP500 stocks.

2 Momentum vs. Mean Reversion

The concept of momentum risk premia stands as a well-documented and established phenomenon within the field of finance. Equally recognized is the principle of Mean-Reversion. Both of these concepts are categorized as stylized facts, observable across various asset classes, with particular prominence in stocks and futures contracts. Momentum risk premia refers to the tendency of assets to continue moving in their current direction, whether upward or downward, over time. The persistence of this trend has been extensively observed and quantified in numerous studies. On the other hand, Mean-Reversion is predicated on the assumption that asset prices and returns eventually revert to their long-term mean or average level, following periods of deviation.

These phenomena are not only crucial in theoretical finance, but also serve as foundational elements in the development of trading strategies and financial models. Their prevalence across different asset classes, especially in stocks and futures, highlights the deep-rooted patterns inherent in financial markets and underscores the importance of these concepts in understanding market dynamics. Time-series momentum (TSMOM) strategies, also known in financial literature as trend-following or

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

'follow the winner' approaches, operate on a straightforward principle: they adopt a long position in assets exhibiting positive past returns and a short position in those with negative returns over a specified look back period. This strategy represents a notable anomaly in the context of asset pricing, particularly deviating from the predictions of the Capital Asset Pricing Model (CAPM) [1]. Empirical evidence suggests that stocks demonstrating higher relative returns in the past year are likely to produce above-average returns in the following year. This observation stands in contrast to the Efficient Market Hypothesis (EMH), challenging its assertion of market efficiency. The momentum effect, therefore, has garnered considerable attention and has been extensively analyzed in quantitative finance research.

We delve deeper into the concept of mean reversion in financial stock returns by discussing the Ornstein-Uhlenbeck process in appendix. Additionally, we dissect the long-term and short-term variance across stock returns to identify momentum trends (see [5]).

3 Data

Our dataset comprises the 206 biggest stocks (in Market Capitalization) of the SP500, spanning from January 1990 to June 2023. This is denoted as our investment universe. The daily closing prices of these stocks constitute our primary data, sourced from EodHistoricalData. For more detailed insights into the dataset, you can explore the information on EodHistoricalData. From these 206 time series, we calculate various metrics across multiple time lags, essential for a model which aims at discerning patterns related to both mean reversion and momentum. A complete list of the features we use is detailed in ppendix].

3.1 Stock Daily Returns

In this subsection, we detail the computation of daily returns and volatility-adjusted daily returns. Volatility-adjusted returns are the target feature we aim to predict each day for each stock, primarily because adjusting for volatility offers a more stable and consistent basis for forecasting. Unlike direct price predictions, which can be misleading due to non-stationarity and price fluctuations, volatility adjustment accounts for the inherent risk and variability in stock prices. This approach provides a more normalized and reliable metric for prediction, reducing the impact of extreme market movements and ensuring a focus on underlying trends rather than short-term price volatility.

Computation of Daily Returns

For each element p_t in a pandas time-series $price$, the returns over the past N-days specified by the offset day_offset are calculated using the following formula:

$$\text{Returns}(p_t, N) = \frac{p_t - p_{t-N}}{p_{t-N}}$$

This formula represents the percentage change in the stock price over a specified number of days.

Computation of Daily Volatility

The daily volatility is calculated using the exponentially weighted moving average (EWMA) method with a look back period $vol\_lookback$. The formula is given by:

$$\text{Volatility}(r_t, N) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(r_{t-i} - \text{Mean}(r_t, N))^2}$$

Here, r_t represents the daily returns, and $\text{Mean}(r_t, N)$ is the mean of the past N daily returns.

Computation of normalized Returns

For a given day offset N, the normalized returns are calculated using the following formula:

$$\text{Normalised Returns}(N) = \frac{\text{Returns}(p_t, N)}{\text{Volatility}(r_t)}$$

where $\text{Returns}(p_t, N)$ represents the returns over the past N days, and $\text{Volatility}(r_t)$ is the daily volatility. The target returns are computed as volatility scaled returns with an annualized target volatility of 15%. The formula is given by:

$$\text{Target Returns}(r_t, N) = \frac{\text{Returns}(r_t)}{\text{Volatility}(r_t)} \times \frac{\text{VOL\_TARGET}}{\text{Volatility}(r_t)} \tag{1}$$

The choice of various time lags in computing normalized returns serves to capture and analyze different aspects of the stock's performance. Each time interval corresponds to a specific timeframe, providing insights into distinct market dynamics. By incorporating various time lags, the model can capture trends at different scales, enhancing its ability to discern patterns related to mean reversion and momentum across multiple timeframes. A detailed representation of the autocorrelation of Apple's stock returns can be found in appendix.

3.2 Beta to the Market

The beta coefficient (β) represents the sensitivity of a stock's returns to market returns. It is calculated as the covariance of the stock's returns with market returns divided by the variance of the market returns. In this context, we compute beta on a rolling basis for different time lags.

For a given stock with returns (R_s), market returns (R_m), and risk-free rate (R_f), the expected return ($E(R_s)$) according to the Capital Asset Pricing Model (CAPM) [1] is expressed as:

$$E(R_s) = R_f + \beta_s \cdot (R_m - R_f) + \alpha_s$$

Here:

- $E(R_s)$ is the expected return of the stock.
- R_f is the risk-free rate.
- β_s is the beta coefficient, previously calculated.
- R_m is the market return.
- α_s is the stock's idiosyncratic return, representing the excess return not explained by the market.

The CAPM model is valuable for understanding the relationship between a stock's expected returns and market factors. In the context of this model, the alpha term represents the stock's ability to generate returns beyond what is predicted by its exposure to systematic market risk (captured by beta). Positive alpha suggests the stock outperforms the market, while negative alpha suggests underperformance. The role and benefits of using Beta in the model is detailed in the appendix.

3.3 Autocorrelation of Stock Returns

In this section, we explore the autocorrelation of stock returns as a measure of momentum and trend persistence over different time lags. The essence of market momentum is encapsulated in the contrast between long-term and short-term realized variances. As detailed in [4], Momentum in stock returns may arise from autocorrelation, where a stock's past outperformance predicts future returns, or from its negative correlation with lagged returns of other stocks, indicating that past underperformance leads to future gains, or simply from a consistently higher average return compared to peers. Therefore, we introduce the autocorrelation of the stock returns, denoted as ACR, which is computed using the Correlation Trend Indicator (CTI). For a given stock with returns R_t, the CTI is calculated as the slope of the linear regression of these returns over a specified period N:

$$\text{CTI}_N = \beta_N$$

where β_N is the slope of the linear regression line fitted to the stock returns R_t over the interval N. This slope indicates the strength or persistence of the trend in returns. A detailed representation of the autocorrelation of Apple stock returns can be found in the appendix.

3.4 MACD Signal

The Moving Average Convergence Divergence (MACD) is a trend-following momentum indicator used in our RNN models to capture market trends and momentum. The MACD is calculated by subtracting the long-term moving average from the short-term moving average. Mathematically, it is represented as :

$$\text{MACD} = EMA_{\text{short}}(P) - EMA_{\text{long}}(P) \tag{2}$$

where EMA_{short} and EMA_{long} are the exponential moving averages over short and long periods, respectively, and P represents the price at close. In our model, we compute the MACD signal over various lags to capture different aspects of market behavior. Lagging is defined by a combination of short and long periods. Specifically, we use the following pairs of short and long periods:

- (8, 24)
- (16, 48)
- (32, 96)

These combinations are chosen to analyze the stock data at different time scales. This approach allows the RNN model to consider various temporal dynamics in the data, thereby enhancing its ability to predict future trends based on historical price movements. A representation of MACD signal for Apple (APPL) is given in appendix.

3.5 Volatility of past returns

Incorporating historical volatility data in an RNN model, especially in the context of the leverage effect, significantly enhances the model's predictive power. As shown in [5], the negative correlation between past returns and future volatility (e.g, the leverage effect) is a moderate and slowly decreasing trend over 50 days for individual stocks. By acknowledging the leverage effect, where equity volatility increases as stock prices fall, the model can more accurately capture the asymmetric response of markets to different financial stimuli. This integration leads to a deeper understanding of risk dynamics and market sentiment, crucial for predicting stock price movements with greater precision, particularly in volatile or bearish market conditions. To enhance our model's predictive capabilities, we will incorporate historical volatility, calculated using various rolling window periods, as a key feature. This approach ensures a more nuanced and comprehensive understanding of market trends and dynamics. The Leverage effect is further illustrated in the appendix , by introducing the Merton Equity model and Black-Scholes formula.

4 Recurrent Neural Network architecture

This paper is clearly a similar approach as in [6], but with different features as their investment universe is CTAs, whereas this work seeks to find if there is a similar alpha in the US equity space. In [3], the authors also leverage the RNN architecture in the systematic trading space. They notably found that the LSTM model, refined to maximize the Sharpe ratio, exhibited a performance doubling that of conventional approaches without transaction fees and maintained its superior results with fees up to 2-3 basis points, therefore being able to capture the several stylized facts in time series momentum for equity, fixed income and commodities index [7].

Recurrent Neural Networks (RNNs) are a class of neural networks designed for processing sequential data, making them well-suited for time series analysis like stock prices. RNNs incorporate feedback loops, allowing them to maintain a memory of past information, enabling the capture of temporal dependencies. This memory capability makes RNNs particularly effective in modeling the dynamic and sequential nature of financial markets. They excel in capturing patterns, trends, and dependencies within historical stock price data. In our case, we will try several RNN architecture (RNN classic,

DeepRNN, GRU, LSTM and DeepLSTM) as well as different length of input time sequence (T = 21, 63, 126, 252 days)

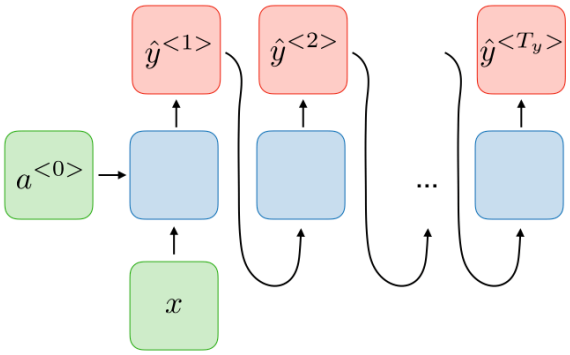

Figure 1: Visualization of a Recurrent Neural Network (RNN) architecture with a Many-to-Many architecture.

Vanishing gradient and alternatives

Vanishing gradients are a problem to look after when using RNN. Vanishing gradients generally mean that contributions from faraway steps vanish and don't affect the training, therefore making it difficult to learn long-range dependencies.

The Vanishing Gradient Problem is a challenge often encountered in the training of Recurrent Neural Networks (RNNs) through backpropagation. Backpropagation is a fundamental process in neural networks, used to update the model's parameters in response to the loss, or error, the model incurs in its predictions. In RNNs, backpropagation works by computing derivatives, which are essential for determining how changes in the weights and biases of the network will affect the overall loss. The goal is to adjust these parameters in a way that minimizes the loss. However, during this process, the gradients (which are the values of these derivatives) can become very small, a phenomenon known as the Vanishing Gradient Problem.

This issue arises because, in RNNs, the gradients are calculated by repeated application of the chain rule over several time steps. Since these gradients are often multiplied by small numbers (the weights of the network), they can exponentially shrink as they propagate backward through the network. As a result, the weights in the earlier layers of the network receive very tiny updates, making the training process extremely slow and sometimes ineffective, especially for long sequences. This problem hampers the network's ability to learn from data points that are far apart in time, thus limiting the performance of the RNN. The derivatives to be used for backpropagation can be computed as follows :

$$\frac{\partial \mathcal{L}}{\partial W_{aa}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{(t)}}{\partial W_{aa}}$$

$$\frac{\partial \mathcal{L}^{(t)}}{\partial W_{aa}} = \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial a^{(i)}}{\partial a^{(i-1)}} \right) \frac{\partial a^{(k)}}{\partial W_{aa}} \cdot a^{(t)} \cdot \frac{\partial a^{(t)}}{\partial W_{aa}}$$

Suppose only one hidden unit, then it is a scalar, and consequently, the product becomes a scalar. If $|\mathbf{W}_{aa}| < 1$, then the product goes to 0 exponentially fast (vanishing gradient).

Gated Recurrent Units (GRU) and Long Short-Term Memory (LSTM) are advanced variants of Recurrent Neural Networks designed to address the vanishing gradient problem. They achieve this by incorporating sophisticated gating mechanisms that allow them to selectively retain and update information over time. The key components, such as forget gates and input gates, enable GRU and LSTM networks to capture long-term dependencies more effectively than traditional RNNs. The figure below illustrates the architecture of an LSTM network.

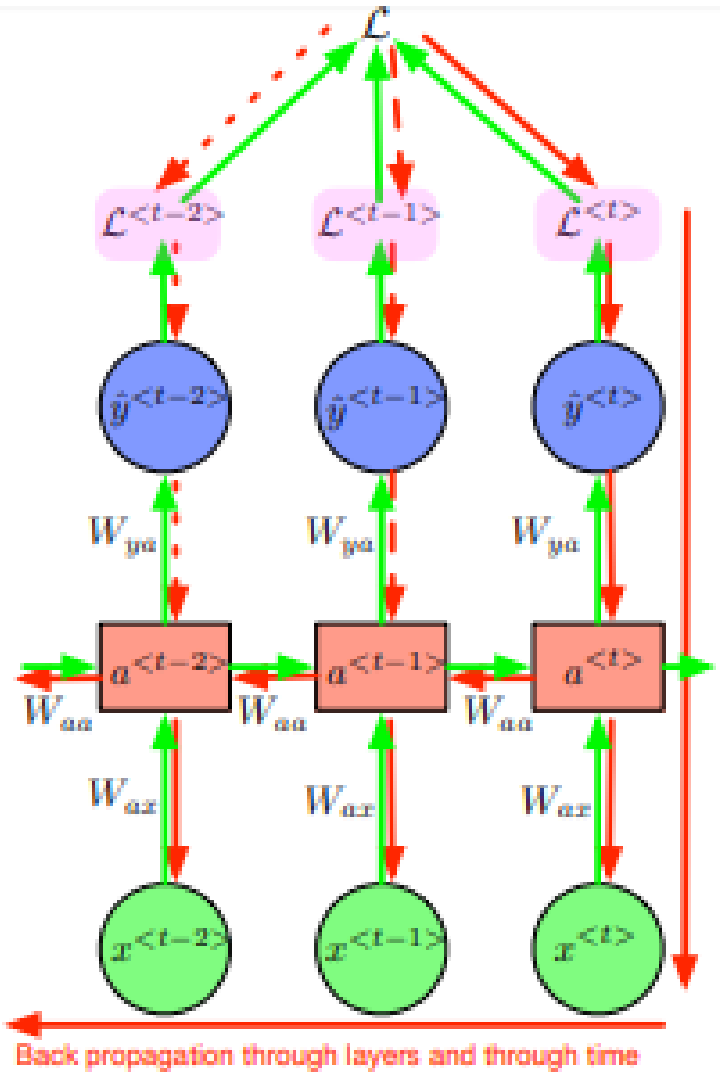

Figure 2: Visualization of Backpropagation in a Deep RNN network

Fitting procedure

We conduct an expanding window backtest for our model. Initially, we train the model using data from 1990 to 1997 and use it to predict positions for the year 1998. Following this, we incrementally add one year of data at a time to fit again the model, subsequently predicting the positions for the following year. This process continues until we get the data up to June 2023.

To optimize our model, we aim to minimize the Sharpe Ratio Loss. The Sharpe Ratio ([8]) is a measure used to assess the performance of an investment relative to its risk (e.g. risk adjusted returns performance). The mathematical formulation for the Sharpe ratio is as follows:

$$\text{Sharpe Ratio} = \frac{E[\text{perf\_backtest}]}{\sqrt{\text{Var}[\text{perf\_backtest}]}} \times \sqrt{252}$$

Here, $E[\text{perf\_backtest}]$ is the expected (mean) return of the backtest performance, and $\text{Var}[\text{perf\_backtest}]$ is the variance of the backtest performance. The term $\sqrt{252}$ annualized the Sharpe Ratio, assuming 252 trading days in a year.

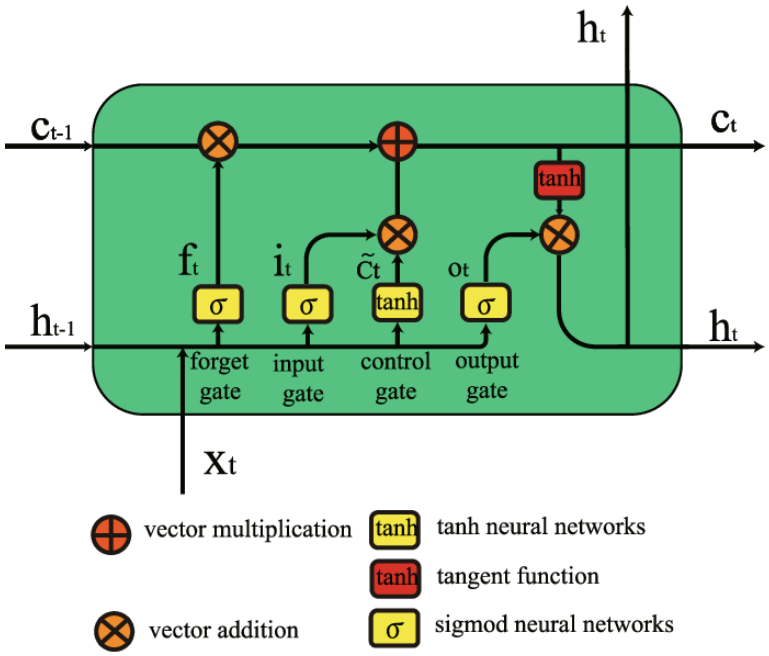

Figure 3: Visualization of the LSTM architecture with forget gates, input gates, and memory cell.

The Sharpe Ratio Loss is then defined as the negative of the Sharpe Ratio, since our goal is to maximize the Sharpe Ratio, which corresponds to minimizing its negative value. Optimizing the Sharpe ratio loss allows the strategy to have a better volatility scaling and risk adjusted performance. This is particularly relevant as raw momentum strategies without adequate risk adjustments, such as volatility scaling (see [9]), are susceptible to large crashes during periods of market panic. Furthermore, even with volatility scaling, which leads to positively skewed returns distributions and long-option-like behavior, trend following strategies can place more losing trades than winning ones (see [10]) and still be profitable on the whole – as they size up only into large but infrequent directional moves. In short, a custom Sharpe ratio loss function can achieve a better position sizing, accounting for tradeoffs between risk and reward.

Prior to calculating the Sharpe Ratio Loss, we apply a normalization layer to our predictions. This layer centers our predictions (e.g. our position weights) around 0 and ensures that they have an absolute sum of 1. This normalization is crucial as it facilitates the next step : construction of a well-balanced and optimized portfolio.

In our training process, we train the model for a total of 250 epochs each time. We employ a callback mechanism based on the validation loss, which plays a crucial role in monitoring and improving model performance during training. The validation data set constitutes 20% of the training sample at each training step. This means that, during the training phase, the model's performance is evaluated against this separate validation set, which is not used for the actual training. The callback is configured with a patience parameter set to 25 epochs. This parameter defines the number of epochs to wait before early stopping if no progress is being made on the validation loss. This means that if the validation loss does not improve for 25 consecutive epochs, the training process will be halted prematurely to prevent overfitting and to ensure efficient use of computational resources. In our model's training process, we employ the Adam optimizer, a sophisticated variant of stochastic gradient descent, specifically tailored for deep learning applications. The instantiation of this optimizer is as follows:

```
adam = keras.optimizers.Adam(lr=learning_rate, clipnorm=10)
```

Overview of Adam Optimizer:
Adam, an acronym for Adaptive Moment Estimation, effectively combines the strengths of two other prominent optimization algorithms. AdaGrad and RMSProp. Adam maintains two moving averages for each parameter, the first moment (the mean) and the second moment (the unentered variance) of

the gradients. This dual approach enables Adam to adjust the learning rates for individual parameters based on their previous gradients.

Adaptive Learning Rate (`lr`):
The learning rate in Adam, set as a hyperparameter, is crucial for controlling the step size during the optimization process. Unlike traditional fixed learning rates, Adam's adaptive learning rate mechanism adjusts each parameter's learning rate based on the historical gradient information. This feature allows Adam to be highly effective in scenarios with nonstationary objectives and high gradient noise.

Gradient Clipping (`clipnorm`):
In our configuration, the gradient norm is clipped at 10. Gradient clipping is a technique to prevent exploding gradients in deep neural networks, which can lead to numerical instability and poor model performance. By limiting the maximum value of the gradient norm, gradient clipping ensures that updates to the model's parameters remain manageable and within a defined range.

Training and Validation Split:
For the model training, we employ a split of 80% for training and 20% for validation. This split allows the model to learn from a significant portion of the data while also providing a separate, unbiased dataset for validating the model's performance. This approach helps in detecting overfitting and ensures that the model generalizes well to new, unseen data.

Learning Rate as a Hyperparameter:
Notably, the learning rate in our model is not a fixed value, but rather treated as a hyperparameter. This decision underscores the learning rate's pivotal role in model convergence and performance. By optimizing the learning rate alongside other hyperparameters, we aim to find the most effective combination that drives the best performance of our model.

4.1 Batch Normalization

All our RNNs models tested include a Batch Normalization layer. Batch normalization is a technique to improve the stability and performance of neural networks. It involves the following steps:

- **Given some intermediate values in a given layer l in the network:** $Z^{[l](1)}, Z^{[l](2)}, \ldots, Z^{[l](m)}$
- **Normalization:**

$$\mu^{[l]} = \frac{1}{m} \sum_i Z^{[l](i)}$$

$$\sigma^{2[l]} = \frac{1}{m} \sum_i \left(Z^{[l](i)} - \mu^{[l]} \right)^2$$

$$Z^{[l](i)}_{\text{norm}} = \frac{Z^{[l](i)} - \mu^{[l]}}{\sqrt{\sigma^{2[l]} + \epsilon}}$$

- **Re-scaling:**
  - New parameters to be trained: γ and β.
  - $\tilde{Z}^{[l](i)} = \gamma \cdot Z^{[l](i)}_{\text{norm}} + \beta$
- **Non-linearity:**

$$a^{[l](i)} = g^{[l]}(\tilde{Z}^{[l](i)})$$

Note: If $\gamma = \sqrt{\sigma^{2[l]} + \epsilon}$ and $\beta = \mu$ then $\tilde{Z}^{[l](i)} = Z^{[l](i)}$.

Batch normalization can enhance backpropagation by reducing the internal covariate shift, which refers to the change in the distribution of network activations due to the update of weights during training. This helps in stabilizing the learning process and accelerating convergence. By normalizing the inputs to each layer, batch normalization allows each layer to learn on a more stable distribution of inputs, which makes the training process faster and more efficient.

Furthermore, batch normalization has a regularizing effect, which can mitigate overfitting. During training, the use of mini-batches introduces noise through the estimation of the mean and variance for

normalization. This noise can be beneficial, as it adds a form of regularization to the model. The network is less likely to overfit to the training data, as the noise prevents it from mapping the training data too closely.

Finally, the schema of the LSTM architecture we use, one of the many model tests in this work, can be seen in the appendix.

Hyperparameter Random Search Grid

We will conduct a comprehensive hyperparameter search using the following grid for our recurrent neural network architectures:

Hyperparameter	Values
Time sequence (days)	21, 63, 126
Dropout Rate	0.1, 0.25, 0.5
Hidden Layer Size	60, 80, 100, 120
Minibatch Size	32, 64, 128
Learning Rate	10^{-3}, 10^{-2}, 10^{-1}

Table 1: Hyperparameter Random Search Grid

We will explore these hyperparameters across different recurrent neural network architectures, including a classic RNN, Deep RNN with two layers, Gated Recurrent Unit (GRU), and Long Short-Term Memory (LSTM).

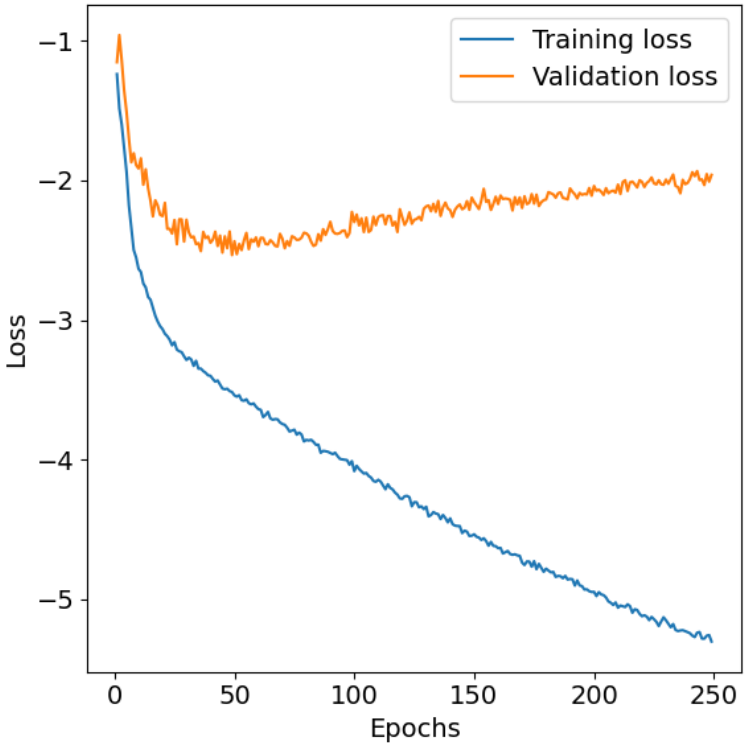

Figure 4: LSTM 63 - Visualization of training and validation loss for one backtest period with learning rate of 0.0001, batch size of 32 and without callbacks

The loss graph ([**??**]) graph illustrates the manifestation of overfitting, as evidenced by the divergence between the training and validation loss. The training loss demonstrates a consistent decline, indicative

of the model's increasing proficiency on the training dataset. Conversely, the validation loss reaches a nadir and subsequently plateaus, an indication that further learning iterations cease to generalize to the validation dataset. This divergence is symptomatic of overfitting, wherein the model begins to memorize the training data rather than learning to generalize from it.

To mitigate such overfitting, it is imperative to implement control mechanisms such as callback functions. Callbacks can be programmed to monitor the validation loss and initiate appropriate actions, such as model check pointing and early stopping, when the loss fails to decrease. Such interventions prevent the model from excessive training on the noise within the dataset, thereby preserving its ability to generalize to new data. Without the incorporation of these control measures, the model is susceptible to learning idiosyncrasies of the training data, which can severely impede its predictive performance on independent datasets.

Layer (type)	**Output Shape**	**Param #**
input_18 (InputLayer)	(None, 63, 21)	0
simple_rnn_8 (SimpleRNN)	(None, 63, 120)	17,040
batch_normalization_29 (BatchNormalization)	(None, 63, 120)	480
dropout_17 (Dropout)	(None, 63, 120)	0
tf.__operators__.getitem_17 (SlicingOpLambda)	(None, 63, 120)	0
time_distributed_17 (TimeDistributed)	(None, 63, 1)	121
zero_mean_layer_17 (ZeroMeanLayer)	(None, 63, 1)	0
Total params		17,641 (68.91 KB)
Trainable params		17,401 (67.97 KB)
Non-trainable params		240 (960.00 Byte)

Table 2: LSTM 63 Model Summary

5 Results

Conversion of model Predictions to Long-Short Positions

Post model training, we initiate an optimization process to convert the model's raw outputs into actionable long-short positions, a key step in transforming predictive insights into practical trading strategies. This process includes maintaining a balanced position across all GICS (General Industry Classification Standards) sectors, with the sum of positions being zero, and targeting an annual volatility of 15%. Further details on this mean-variance optimization, applied following model predictions, can be found in the appendix.

In Figure 6, we observe a significant correlation among the various models, which is an expected outcome given their shared objective of capturing market momentum and reversion. However, each model employs distinct time sequence input lengths, leading to variations in their performance. Figure 5 illustrates that all models generate positive and cumulative upward trends in their respective PnLs. Notably, some models, specifically LSTM 63, LSTM 21, and GRU 63, exhibit superior performance. This enhanced performance can be attributed to the inherent architectural advantages of GRU (Gated Recurrent Unit) and LSTM (Long Short-Term Memory) models. These architectures are particularly adept at processing and selectively assimilating information from past data, which is crucial for effective prediction in time-series analysis.

GRU and LSTM models are designed to capture long-range dependencies and contextual nuances in sequential data, enabling them to discern and leverage patterns that simpler models (RNN and DeepRNN) might overlook. This ability to differentiate and prioritize relevant historical information is likely a key factor in their superior performance observed in our analysis.

The calculation and detailed explanation of each metric can be found in appendix.

Optimal model architecture - LSTM 63

Overall, the LSTM with 63 days as time sequence inputs performs better, reaching a Sharpe Ratio around 2.17 over the backtest period, not taking into account transaction costs. Our empirical

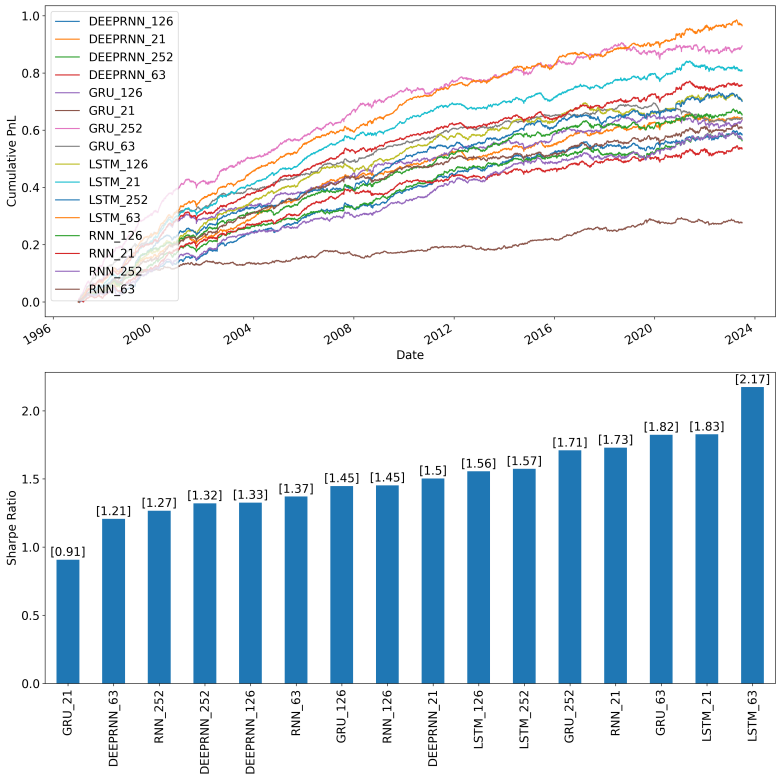

Figure 5: PnL and Sharpe ratios of our trained models after weights optimization

analysis reveals that the Long Short-Term Memory (LSTM) network, configured for a 63-day input sequence, yields the most promising results. Utilizing the RandomSearch technique for hyperparameter optimization, we identified the following optimal settings:

- **Dropout Rate:** 0.25
- **Learning Rate:** 0.005
- **Number of Neurons in LSTM Cell:** 120
- **Minibatch Size:** 32

The choice of a 63-day time sequence aligns well with financial reporting cycles, encompassing a typical earnings period. This duration is significant for capturing the inherent seasonality in stock market data. Moreover, we observe that the incorporation of Batch Normalization layers and the application of L1 regularization notably enhance the model's performance and contribute to mitigating overfitting. We can see from the table 3, that the LSTM 63 has the best Sharpe Ratio over the backtest period, as well as the best bias statistic and T-stat. We can clearly see the dominance of the model from the PnL graph in 5. For additional evaluation, we conduct stress tests on our predictions by introducing a delay in executing the trading signals. Specifically, we shift the signals and trade them at various delayed intervals 1, 5, 10, or 21 days later. This analysis reveals a significant decline in performance, a result that aligns with our expectations considering the high turnover of our portfolio observed from close to close.

We observe a significant impact on the PnL when the trading signal is delayed by one day(see [7]). Most notably, a substantial portion of the PnL is forfeited if the signal execution is lagged by just a single day compared to its intended trading schedule. This observation aligns well with the high turnover rate of the signal, which stands at approximately 47%. High turnover implies that the signal's positions change frequently on a close to close basis, thereby accentuating the importance of timely trade execution. Conversely, in scenarios where the turnover is relatively low (e.g., 10%), the immediacy of executing a trade becomes less critical. This is because the positions held are more

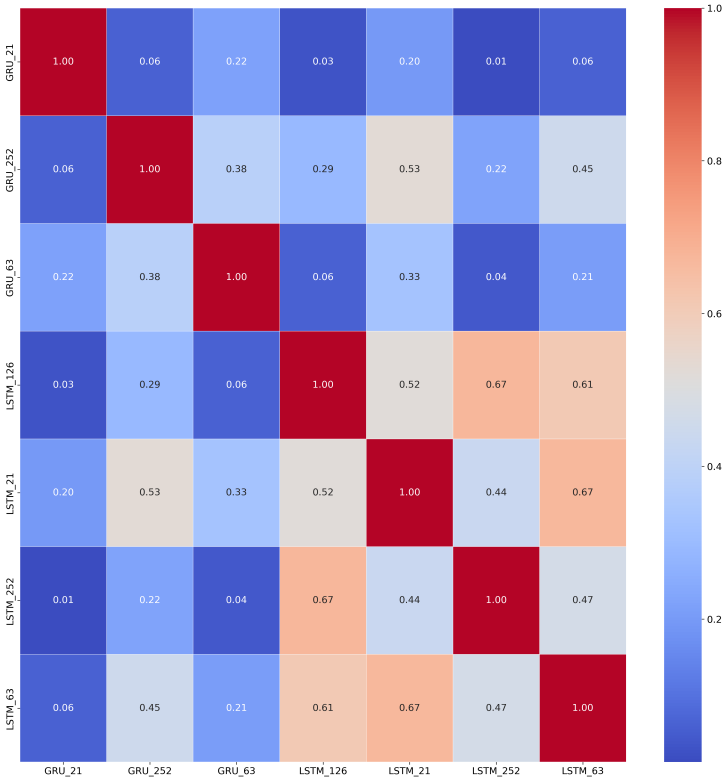

Figure 6: PnL correlation matrix of GRU and LSTM models

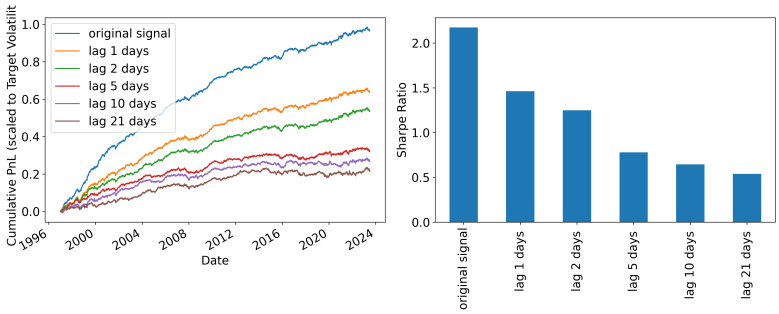

Figure 7: LSTM 63 - Stress test with lagging signal

static, and thus, the impact of a day's delay is mitigated. Nevertheless, it is noteworthy that we still observe a persistent, albeit modest, positive performance for lagged signals, extending up to a 21-day delay. This persistence suggests that the model is not only capturing short-term market movements but is also attuned to longer-term trends. These longer-term trends exhibit a lower sensitivity to precise trading timing, offering some resilience to the performance of lagged signals.

Upon the advent of financial perturbations, we observe an immediate adjustment in model allocations, precipitating elevated portfolio turnover. The resilience of our LSTM 63 model to financial tumult is not absolute; however, the integration of lagged features, alongside market beta, historical volatility, and measures of autocorrelation, provides a robust framework for loss attenuation. Consequently, the model demonstrates commendable performance notwithstanding the vicissitudes of market crises. This suggests a degree of efficacy in the predictive capabilities and risk management protocols of our models, reflecting the diversification of temporal inputs and market sensitivity parameters. Thus, our strategic incorporation of multifaceted financial indicators serves to enhance the stability and adaptability of our investment strategies during periods of economic stress.

Model	Sharpe	Calmar	Sortino	annual vol	tail ratio	alpha	beta	max drawdown	Turnover	Bias	R2	T-stat
DEEPRNN 126	1.33	0.70	2.03	0.02	1.14	0.02	-0.00	-0.03	21.41	0.88	0.00	7.65
DEEPRNN 21	1.50	0.71	2.31	0.02	1.13	0.02	0.00	-0.03	70.26	0.96	0.01	9.09
DEEPRNN 252	1.32	0.58	2.04	0.02	1.11	0.02	-0.00	-0.04	30.90	0.84	0.01	8.47
DEEPRNN 63	1.21	0.79	1.82	0.02	1.11	0.02	-0.00	-0.03	20.81	0.80	0.00	7.89
GRU 126	1.45	0.51	2.13	0.02	1.11	0.02	-0.00	-0.05	55.30	0.95	0.00	7.75
GRU 21	0.91	0.39	1.34	0.01	1.08	0.01	-0.00	-0.03	50.00	0.41	0.00	7.28
GRU 252	1.71	0.60	2.58	0.02	1.14	0.03	0.00	-0.06	58.78	1.34	0.01	9.58
GRU 63	1.82	0.36	2.86	0.01	1.17	0.02	-0.00	-0.07	53.50	0.96	0.01	12.03
LSTM 126	1.56	0.52	2.46	0.02	1.17	0.03	0.00	-0.05	41.24	1.06	0.01	10.24
LSTM 21	1.83	0.85	2.85	0.02	1.16	0.02	-0.00	-0.04	53.01	1.21	0.01	13.45
LSTM 252	1.57	0.78	2.54	0.02	1.17	0.03	-0.00	-0.03	30.05	1.05	0.01	11.90
LSTM 63	2.17	1.45	3.51	0.02	1.23	0.04	-0.00	-0.03	47.00	1.45	0.03	19.44
RNN 126	1.45	0.76	2.25	0.02	1.16	0.02	-0.00	-0.03	21.85	0.98	0.01	9.15
RNN 21	1.73	0.80	2.70	0.02	1.14	0.02	-0.00	-0.04	63.14	1.13	0.01	12.27
RNN 252	1.27	0.74	1.98	0.02	1.15	0.02	-0.00	-0.03	27.53	0.85	0.00	7.66
RNN 63	1.37	0.79	2.09	0.02	1.13	0.02	-0.00	-0.03	20.57	0.91	0.01	10.87

Table 3: Backtest statistics (1997-2023) for trained models

As a next step, we conducted a comprehensive evaluation of our best LSTM model (LSTM 63), comparing it against several basic benchmark strategies: - A naive equi-weighted, long-only portfolio. - 3 momentum portfolio strategies that buy stocks with a winning record over the past 21, 63 or 126 days and sells those with a losing record during the same period.

All these benchmark portfolios were adjusted to match the target volatility of our LSTM model. Additionally, the momentum portfolios were designed to be market-neutral, maintaining a constant nominal value through weight optimization similar to that of the LSTM model.

A further analysis of the importance of the features can be found in appendix.

Pushing the LSTM to its limit

In appendix, we tested a novel LSTM architecture that incorporates a depth of three LSTM cells. We observed earlier that we achieved the best results with an LSTM and input's time sequence length of 63 days. To further explore the capacity of this model, we have stacked three LSTM cells on top of each other. We did observe worst performance when increasing the model's complexity by stacking three LSTM to create a "DeepLSTM" model

Conclusion

In this study, we primarily adapted the methodology from [6] to the context of the US equity market. Our findings align closely, highlighting the Long-Short-Term Memory (LSTM) cell as the most effective model, particularly when used with a 63-day time sequence. Enhancing the LSTM model with a custom Sharpe Loss objective function, specifically tailored for business applications, improved results in terms of volatility scaling and investment sizing. We also integrated L1 regularization and batch normalization layers, which further minimized result variance and reduced overfitting risk.

However, our model has limitations. Despite observing a positive bias, its magnitude may be insufficient for practical trading, given our portfolio's turnover. Our assumption of a frictionless backtest, without transaction costs, does not reflect realistic trading conditions. Incorporating such costs and liquidity constraints, as done by [11] or [12], is likely to yield less favorable results.

There are avenues for improving our model. Currently, it does not utilize fundamental data (e.g., earnings announcements, accounting statements, and 13-F filings), which, although having quarterly release dates, could be valuable for predicting stock returns and detect anomalies in stock pricing. Additionally, more advanced techniques for enhancing time series momentum could be employed. In this context, the recent work of Wood et al. [13] is relevant, as they developed a novel time

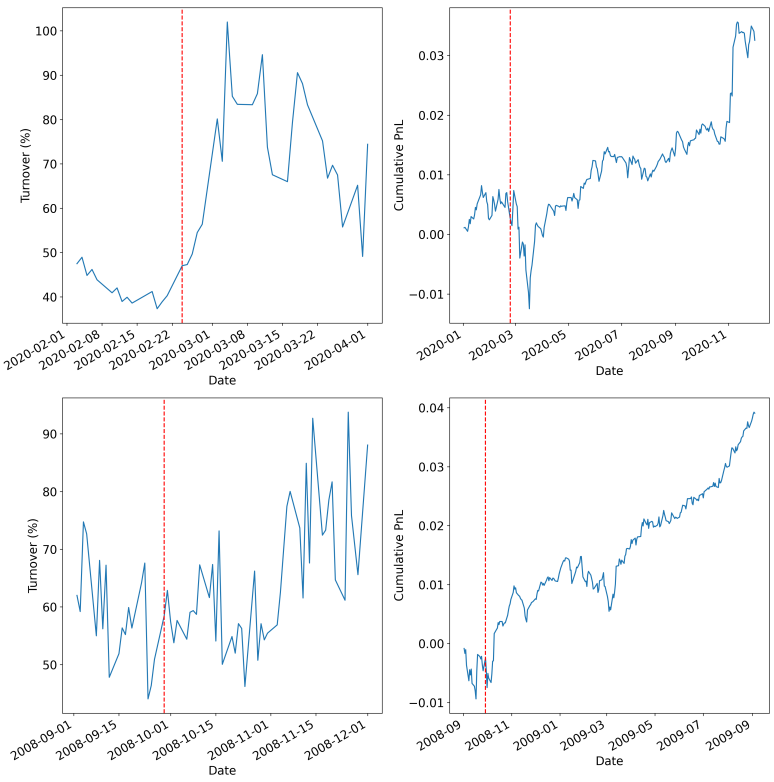

Figure 8: LSTM 63 - Portfolio turnover and PnL around global financial crisis (2008 subprimes and 2020 Covid-19)

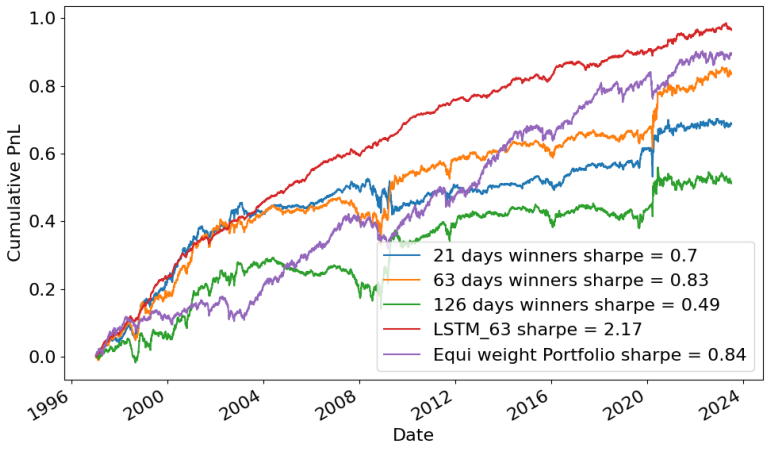

Figure 9: LSTM 63 vs benchmarks

series trend-following forecaster, the Cross-Attentive Time Series Trend Network (X-Trend), which demonstrates the ability to adapt rapidly to new market conditions or regimes. Leveraging few-shot learning, X-Trend has shown significant improvements in adapting to financial market changes, like those during the COVID-19 pandemic, with a notable increase in the Sharpe ratio compared to traditional methods and a quicker recovery from market downturns.

References

1. E. F. Fama and K. R. French, *The Capital Asset Pricing Model: Theory and Evidence*, 2004.
2. P. Bergault, F. Drissi and O. Guéant, *Multi-asset optimal execution and statistical arbitrage strategies under Ornstein-Uhlenbeck dynamics*, 2021.
3. T.-L. Dao, T.-T. Nguyen, C. Deremble, Y. Lempérière, J.-P. Bouchaud and M. Potters, *Tail Protection for Long Investors: Trend Convexity at Work*, 2016.
4. J. Lewellen, *Momentum and Autocorrelation in Stock Returns*, 2002.
5. J.-P. Bouchaud, A. Matacz and M. Potters *The leverage effect in financial markets:retarded volatility and market panic*, 2001.
6. B. Lim, S. Zohren, and S. Roberts, *Enhancing time-series momentum strategies using deep neural networks*, The Journal of Financial Data Science, vol. 1, no. 4, pp. 19–38, 2019.
7. T. J. Moskowitz, Y. H. Ooi, and L. H. Pedersen, *Time series momentum*, Journal of Financial Economics, vol. 104, no. 2,pp. 228 – 250, 2012.
8. W. F. Sharpe,*The Sharpe ratio*, The Journal of Portfolio Management, vol. 21, no. 1, pp. 49–58, 1994.
9. A. Y. Kim, Y. Tse, and J. K. Wald, *Time series momentum and volatility scaling*, Journal of Financial Markets, vol. 30, pp. 103 – 124, 2016.
10. M. Potters and J.-P. Bouchaud, *Trend followers lose more than they gain*, Wilmott Magazine, 2016.
11. J.de Lataillade, C. Deremble, M. Potters and J.-P. Bouchaud, *Optimal Trading with Linear Costs*, 2012.
12. X. Brokmann, D. Itkin, J. Muhle-Karbe and P. Schmidt, *Tackling Nonlinear Price Impact with Linear Strategies*, 2023.
13. K. Wood, S. Kessler, S. J. Roberts, S. Zohren, *Few-Shot Learning Patterns in Financial Time-Series for Trend-Following Strategies*, 2023

Appendix

Understand Mean reversion with Ornstein-Uhlenbeck process

The concept of mean reversion in financial markets, particularly in stock returns, can be efficiently explained by the Ornstein-Uhlenbeck process. Ornstein-Uhlenbeck process is particularly suited to modeling stock prices; due to their ability to capture mean reversion, they often lead to closed form formulas as well as they are able to capture several multivariate dynamics, notably in terms of cointegration. OU processes are therefore often used in Market Finance to modeling stock returns. In 2, the authors explore the Ornstein-Uhlenbeck dynamics in the context of multi-asset optimal execution and statistical arbitrage strategies. They emphasize that this process not only assumes the mean reversion property in stock returns but also allows for varying strengths of this reversion, alongside the potential for jumps. This makes the Ornstein-Uhlenbeck model a robust framework for understanding and modeling the dynamic behavior of stock returns, acknowledging their tendency to revert to a long-term meanwhile accounting for sudden, discrete changes in value.

The Ornstein-Uhlenbeck process is a common choice for such modeling, expressed as

$$dR_t = \theta(\mu - R_t)dt + \sigma dW_t \tag{3}$$

Here, R_t is the stock return at time t, θ represents the speed of mean reversion, μ is the mean return, dt is the differential time element, and dW_t is a Weiner process representing the stochastic shock. Additionally, the volatility parameter σ influences the spread of potential returns. A higher σ results in more significant fluctuations, affecting the time it takes for returns to revert to the mean.

The discrete-time equivalent, suitable for daily stock returns, can be expressed using an autoregressive (AR) process:

$$R_t = \rho R_{t-1} + \epsilon_t \tag{4}$$

In this equation, ρ is the autoregressive coefficient capturing the mean-reverting behavior, and ϵ_t is a white noise term that represents stochastic shock.

Understand Momentum with Long Term Variance Property

Momentum in financial markets can be understood through the lens of the disparity between long-term and short-term realized variance. In [3], the authors highlight that the efficacy of trend following strategies is largely attributable to this variance differential. They further elucidate the pronounced convexity in the performance of Commodity Trading Advisors (CTAs), linking it to the inherent relationship between trend strategies and market volatility. This perspective not only deepens the understanding of momentum, but also reveals intriguing parallels with risk-parity portfolios and options-based strategies emphasizing long-term market variance. The model assumes that stock returns can be conceptualized as a sequence of uncorrelated realizations of a random walk, in accordance with the principles laid out by Black and Scholes. Let us start by recalling the work in [3].The authors suppose that the price at time t, S_t, is written as the sum over past price changes $D_{t'}$ for $t' < t$:

$$S_t = S_0 + \sum_{t'=1}^{t} D_{t'}$$

We assume that the sequence of price changes $D_{t'}$ for $t' < t$ are stationary random variables with zero mean and covariance given by :

$$\langle D_u D_v \rangle = C(|u - v|)$$

Here, and henceforth, $\langle ... \rangle$ denotes averaging. The scenario of uncorrelated random walks corresponds to $C(u) = \sigma^2 \delta_{u,0}$, where σ is the volatility. Trending random walks are characterized by $C(u > 0) > 0$, while mean-reverting random walks exhibit $C(u > 0) < 0$.

The volatility of scale τ is defined conventionally as:

$$\sigma^2(\tau) := \frac{1}{\tau}\langle (S_{t+\tau} - S_t)^2 \rangle$$

such that $\sigma^2(1) = \sigma^2$. The explicit formula for $\sigma^2(\tau)$ in terms of $C(u)$ is straightforward to derive and is given by:

$$\sigma^2(\tau) = \sigma^2 + \frac{2}{\tau}\sum_{u=1}^{\tau}(\tau - u)C(u)$$

The resulting time-dependent volatility, often referred to as the "signature plot," is illustrated for an exponentially decaying $C(u)$ in Figure 3. In particular, positive correlations (indicating trends) result in an elevation of long-term volatility compared to short-term volatility, while negative correlations (indicating mean reversion) lead to the opposite effect. As expected, uncorrelated random walks give rise to a strictly constant volatility $\sigma(\tau)$.

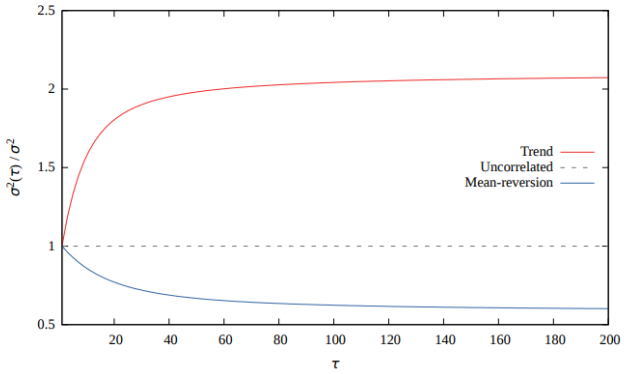

Figure 10: Volatility plotted against scale τ. The red line represents positive autocorrelation (indicating trends) with $C(u > 0) = 0.1e^{1-u/5}$. The dashed line corresponds to an uncorrelated random walk. The blue line illustrates negative autocorrelation (indicating mean-reversion) with $C(u) = -0.02e^{1-u/10}$.

Exhaustive list of features use in the model

For each stock, we calculated these features daily, utilizing them as integral components in our RNN models.

- **Target Returns:** The primary feature to predict, adjusted for daily stock volatility.
- **Normalized Daily Return:** Known input reflecting normalized returns on a daily basis.
- **Normalized Weekly Return:** Weekly return data, normalized for consistency.
- **Normalized Monthly Return:** Monthly return figures, adjusted for normalization.
- **Normalized Quarterly Return:** Quarterly return data, provided in a normalized format.
- **Normalized Biannual Return:** Biannual return information, adjusted for normalization.
- **Normalized Annual Return:** Annual return figures, presented in normalized form.
- **MACD Long/Short (8/24, 16/48, 32/96):** Moving Average Convergence Divergence signals for various durations, indicating trend reversals.
- **Daily Volatility (5, 21):** Known input reflecting the stock's volatility over short periods.
- **Beta (126, 252):** Measures the stock's volatility relative to the overall market.
- **Alpha Daily/Weekly/Monthly Returns:** Alpha returns for various time frames, indicating excess returns over a benchmark.

- **Annual Alpha Variance Ratio:** Ratio of alpha variance to total variance of the stock.
- **CTI (21, 63, 126, 252):** Autocorrelation of stock returns for various periods, indicating momentum or mean reversion.

Role of Beta in the Model

In mathematical terms, beta is a measure of systematic risk, indicating how much the returns of a stock move in relation to the returns of the market. By computing beta on a rolling basis for different time lags (e.g., $w = 252$ and $w = 126$), we capture changing relationships over time. In the context of this model, beta serves as a key factor in modeling the interaction between individual stock returns and market returns. A beta greater than 1 indicates that the stock is more volatile than the market, while a beta less than 1 suggests lower volatility. This information is crucial to understanding the risk and trend dynamics of individual stocks within the broader market. Rolling the beta computation helps to adapt the model to changing market conditions and provides insight into the evolving relationship between stock and market returns.

In addition to the dynamic beta measures, our model incorporates alpha returns derived from the Capital Asset Pricing Model (CAPM) across various time lags (21, 63, 126 days), enriching the predictive strength of the model by accounting for time-sensitive risk premiums. Moreover, we integrate the alpha variance ratio, which juxtaposes the variance of alpha returns against the total returns' variance, providing a nuanced view of risk-adjusted performance.

Including these metrics enhances the model's depth, although it is recognized that they do not offer the comprehensive breadth found in multifactor models such as those by Fama-French or Barra, which encompass a broader array of style and industry factors. Nevertheless, the inclusion of alpha returns and their variance ratio contributes significantly to the model's ability to navigate diverse market conditions and to pinpoint idiosyncratic risk elements that may not be captured by beta alone.

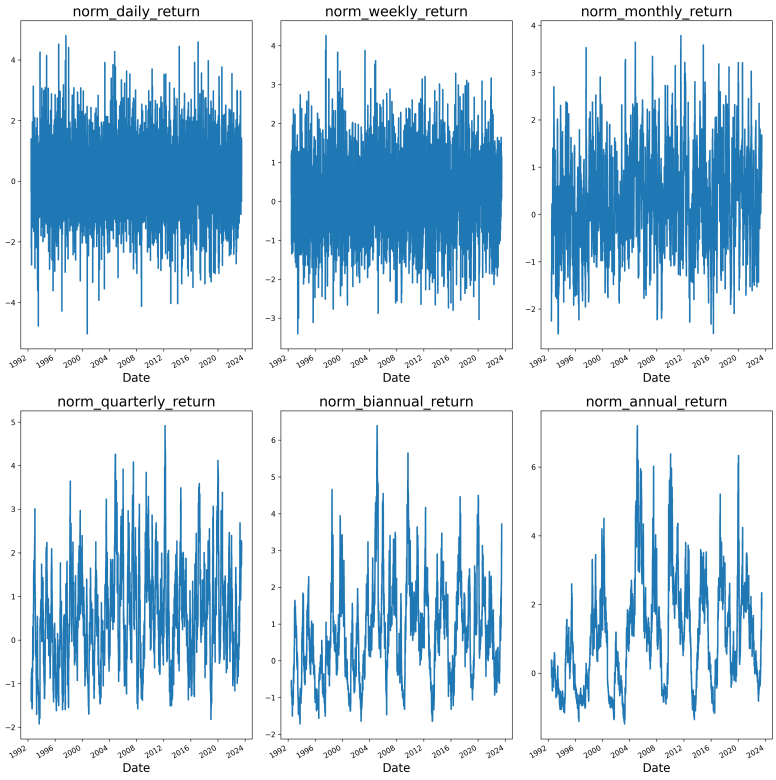

Figure 11: features for AAPL returns

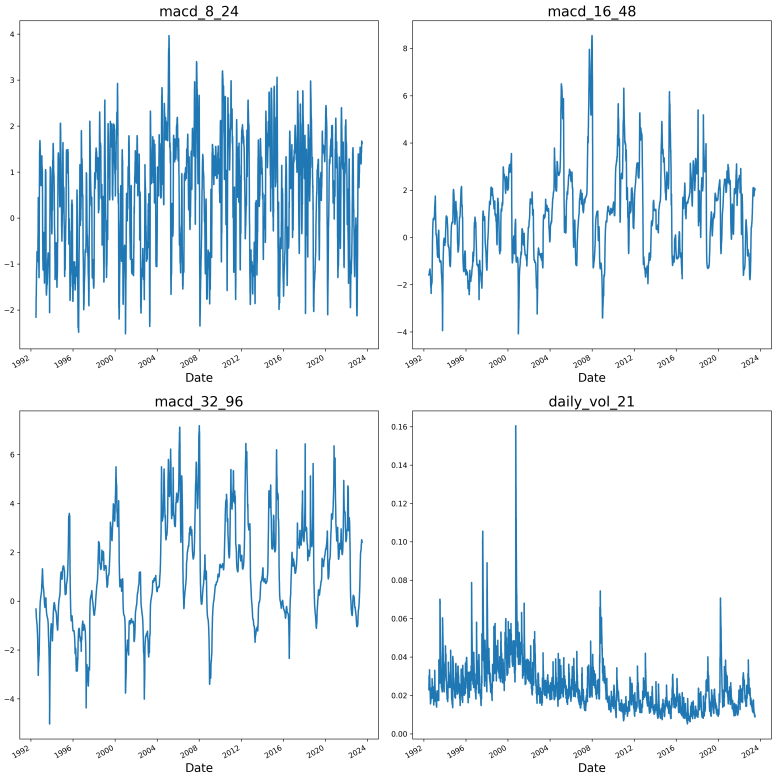

Figure 12: MACD signal for AAPL and 21 days volatility

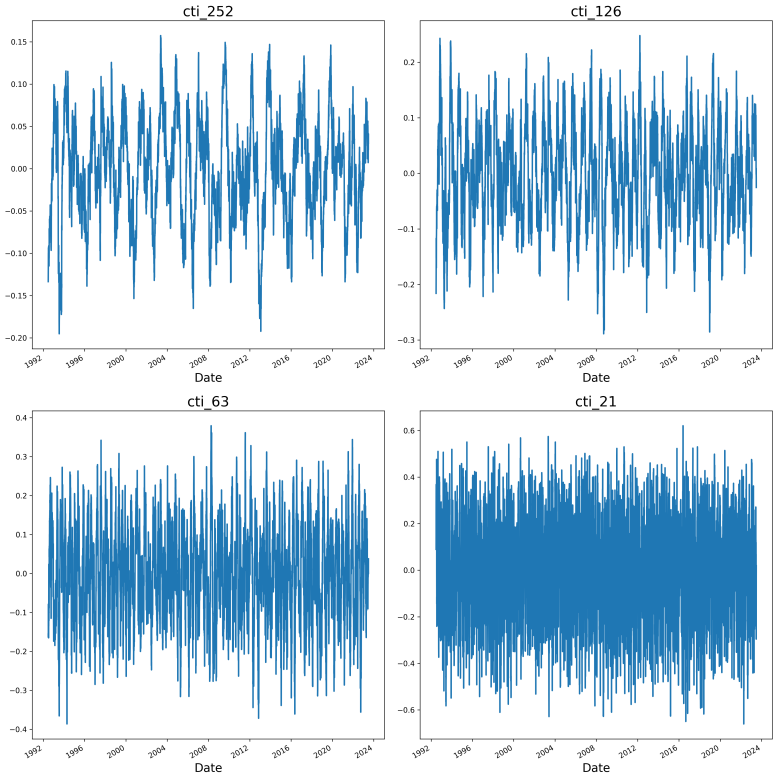

Figure 13: CTI features for AAPL

Leverage effect and Merton model

To further explain the leverage effect, let's consider a firm with equity value E and debt value D. The total value of the firm, V, is the sum of equity and debt:

$$V = E + D \tag{5}$$

The leverage ratio, L, is defined as:

$$L = \frac{D}{E} \tag{6}$$

According to the Merton Model, equity can be viewed as a call option on the firm's assets. Let's denote S as the firm's assets, K as the debt level (strike price), and σ_S as the volatility of the firm's assets. The Black-Scholes model gives the price of this call option (equity) as:

$$E = S\Phi(d_1) - Ke^{-rT}\Phi(d_2) \tag{7}$$

where

$$d_1 = \frac{\ln(\frac{S}{K}) + (r + \frac{\sigma_S^2}{2})T}{\sigma_S\sqrt{T}}$$

$$d_2 = d_1 - \sigma_S\sqrt{T}$$

Φ is the cumulative distribution function of the standard normal distribution, r is the risk-free rate, and T is the time to maturity. As the equity value E decreases, typically due to a decrease in S, the firm becomes more leveraged (higher L). This increase in leverage leads to a higher equity volatility (σ_E). The relationship can be approximated as:

$$\sigma_E \approx (1 + L)\,\sigma_S \tag{8}$$

Thus, a decline in equity value increases the leverage, which in turn increases the equity volatility. The leverage effect highlights the increased risk (volatility) associated with equity as its price declines, primarily due to the increasing leverage ratio. This effect is a crucial consideration for investors and risk managers in the financial markets.

Architecture of the LSTM 63 model

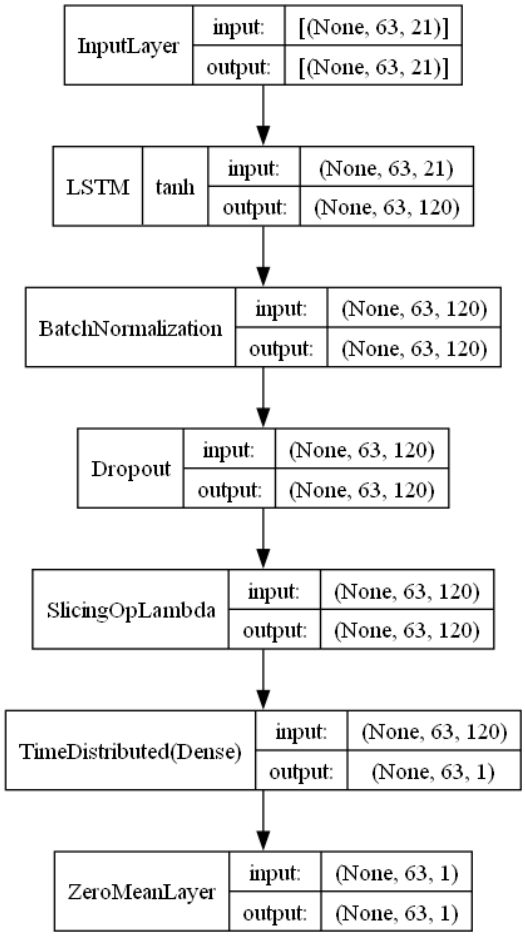

Figure 14: LSTM G3 model architecture

Mean variance framework to transform raw model predictions into optimized long-short positions

Our investment strategy is focused on translating raw model predictions into daily long-short signals across a portfolio universe of N assets:

- p represents a vector of model-predicted returns for each asset, of size N.
- μ denotes the forecast of asset returns: $\mu = E(p)$.
- w is a vector indicating our positions or holdings in each asset

For any vector v, its transpose is denoted by v^T.

We employ a mean-variance objective, balancing the portfolio's expected return (as $w^T\mu$) against its expected risk (as w^TCw), formalized as:

$$U = w^T\mu - \frac{w^TCw}{2\gamma},$$

where γ signifies the risk tolerance.

The optimal allocation w that maximizes the mean variance objective is given by

$$w = \gamma C^{-1}\mu.$$

We extend the mean-variance formula to incorporate linear constraints

$$Bw = d.$$

Here, B is a matrix consisting of rows with binary digits, representing the sector allocation, such that $B_{i,j} = 0$ if asset j is not in sector i.

Introducing the Lagrangian $\mathcal{L}$ (with Lagrange multiplier η),

$$\mathcal{L} = w^T\mu - \frac{w^T C w}{2\gamma} - (w^T B^T - d^T)\eta,$$

the Lagrange multiplier η is a 'tuning parameter' ensuring that the constraint is satisfied. At its optimal value, the constrained problem simplifies to an unconstrained problem with adjusted return forecast $\mu - B^T\eta$. The optimal allocation maximizing $w^T\mu - \frac{w^T C w}{2\gamma}$ under the linear constraint $Bw = d$ is

$$w = C^{-1}B^T\left(BC^{-1}B^T\right)^{-1}d + \gamma C^{-1}\left[\mu - B^T\left(BC^{-1}B^T\right)^{-1}BC^{-1}\mu\right].$$

Additionally, we introduce a constraint to maintain a constant nominal sum of absolute weights equal to 1, with some degree of relaxation for non smoothness. This constraint ensures that each day, our portfolio invests 1 dollar (as long or short positions) in the market.

Metrics to assess the performance of a long short strategy

Here we explain the metrics that we use to compare our results.

Calmar Ratio
The Calmar Ratio is a performance metric that compares the expected annual return of an investment to its maximum drawdown during the same period. It is calculated as:

$$\text{Calmar Ratio} = \frac{\text{Annual Return}}{\text{Maximum Drawdown}} \tag{9}$$

This ratio is particularly useful for evaluating the risk of strategies during periods of significant losses.

Sortino Ratio
The Sortino Ratio modifies the Sharpe Ratio by considering only the downside deviation instead of the total standard deviation of returns. It is defined as:

$$\text{Sortino Ratio} = \frac{\text{Expected Return} - \text{Risk-Free Rate}}{\text{Downside Deviation}} \tag{10}$$

This ratio provides a better tool for investors to assess risk-adjusted returns, especially in asymmetric return distributions.

Alpha and Beta
Alpha and Beta are parameters derived from the regression of the portfolio's excess returns over the benchmark (like S&P 500) returns. Alpha represents the portfolio's performance independent of market movements, while Beta indicates the portfolio's volatility relative to the market.

$$\text{Portfolio Return} = \alpha + \beta \times \text{Market Return} + \text{Error} \tag{11}$$

Tail Ratio
Determines the ratio between the right (95%) and left tail (5%). For example, a ratio of 0.25 means that losses are four times as bad as profits.

Maximum Drawdown of a PnL Strategy
Maximum Drawdown (MDD) is a measure of the largest drop from a peak to a trough in the value of a portfolio, before a new peak is achieved. It is an indicator of downside risk over a specified time period. It is defined as:

$$\text{Maximum Drawdown} = \max\left(\max_{t\in[0,T]}\left(\max_{\tau\in[0,t]} P(\tau) - P(t)\right)\right) \tag{12}$$

where $P(t)$ is the portfolio value at time t and T is the total period.

Turnover of the Signal
Turnover measures the frequency and volume with which assets within a portfolio are bought and sold. In the context of a trading signal, it quantifies the change in the signal's positions over time, reflecting the level of trading activity. The turnover of the signal, when considering a GICS (Global Industry Classification Standard) hedge, is calculated as:

$$\text{Daily Turnover (in \%)} = \sum_{i=1}^{N} |\Delta \text{Signal}_i| \times 100 \tag{13}$$

where ΔSignal_i is the change in the signal from one day to the next for each stock i in the portfolio, and N is the number of stocks. The turnover is expressed as a percentage of the portfolio value. Turnover is a very important metrics for a systematic trading signal. The lower the turnover, the less rebalance there is from one day to another. This means fewer transactions costs and less execution. There is no "better" turnover, but it is very important to maintain a high bias while enhancing turnover, otherwise trading the signal is too costly for very little reward on average.

Bias of a PnL
The bias of a Profit and Loss (PnL) strategy reflects the average performance of the strategy adjusted for the risk-free rate. In the context of financial strategies where the risk-free rate is assumed to be zero, the bias is simply the mean of the daily PnL. This is calculated as follows:

$$\text{Bias} = (\text{Mean of Daily PnL} - \text{Risk-Free Rate}) \times 10^4 \tag{14}$$

Given that the risk-free rate is assumed to be zero for simplification, the formula reduces to:

$$\text{Bias} = (\text{Mean of Daily PnL}) \times 10^4 \tag{15}$$

The result is presented in basis points (bips), hence the multiplication by 10^4, where one basis point is equal to 1×10^{-4} or 0.01%.

R^2 (Coefficient of Determination)
R^2 in our context is the coefficient of determination of the regression between the model's predicted signals for each stock and their subsequent returns. It indicates how well the model's predictions explain the variation in the actual stock returns.

T-Statistic
The T-Statistic associated with the signal in the regression is a measure of the significance of the predictive power of the model. It tests the null hypothesis that the signal has no effect on the returns, with a higher absolute value indicating stronger evidence against the null hypothesis.

Feature Importance

The given code segment is designed to evaluate the importance of individual features in a predictive model. The process is outlined as follows:

1. Iterate over each feature (denoted by the elements in `COLS`) using a `for` loop.
2. In each iteration, select a random sample of 500 data points from the validation dataset (`val_data`).
3. For the currently selected feature (indexed by `k`), perform the following steps:
   (a) Temporarily store the original values of the feature in `save_col`.
   (b) Randomly shuffle the values of this feature across the sample. This shuffling disrupts the relationship between the feature and the target, allowing us to assess the feature's impact.
4. With the shuffled feature, make predictions using the LSTM 63 fitted model (`fitted_model`) and calculate the Mean Squared Error (MSE) against the actual labels (`val_labels`).
5. Record the MSE associated with the shuffled feature. A higher MSE implies that the shuffled feature was important for the model's predictions.

6. Restore the original values of the feature to maintain the integrity of the validation dataset for subsequent iterations.
7. Append the results, including the feature name and its associated MSE, to a list for further analysis.

This method of feature importance assessment is particularly useful as it quantitatively measures the impact of each feature on the model's predictive accuracy. By shuffling one feature at a time and observing the change in MSE, we can infer the relative importance of each feature in the model. Here are the results with our best LSTM 63 model

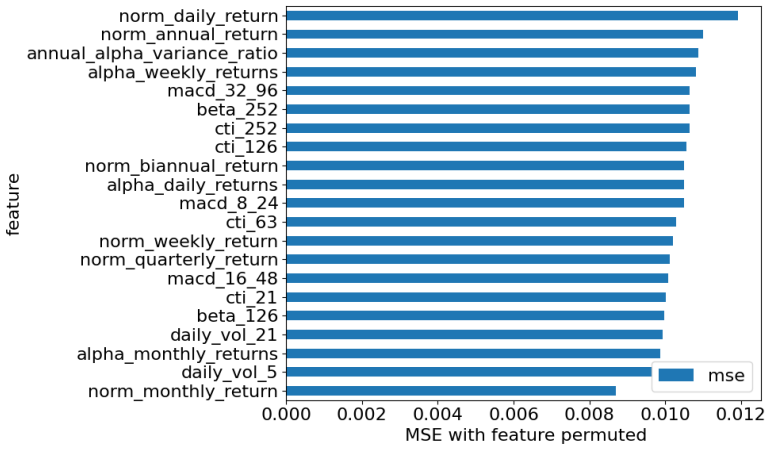

Figure 15: LSTM 63 feature importance

Increasing the model's complexity : a DeepLSTM architecture

The mathematical formulation of an LSTM cell is given by the following equations: This deep LSTM structure is designed to capture more complex patterns and longer dependencies within the data. The configuration is as follows:

Table 4: Deep LSTM Model Summary

Layer (type)	**Output Shape**	**Param #**
input_20 (InputLayer)	(None, 63, 21)	0
lstm_22 (LSTM)	(None, 63, 120)	68,160
batch_normalization_33 (BatchNormalization)	(None, 63, 120)	480
lstm_23 (LSTM)	(None, 63, 120)	115,680
batch_normalization_34 (BatchNormalization)	(None, 63, 120)	480
lstm_24 (LSTM)	(None, 63, 120)	115,680
batch_normalization_35 (BatchNormalization)	(None, 63, 120)	480
dropout_19 (Dropout)	(None, 63, 120)	0

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

Where f_t, i_t, and o_t are the forget, input, and output gates, respectively; $\tilde{C}_t$ is the cell input activation vector; C_t is the cell state; and h_t is the output vector of the LSTM cell. Stacking multiple LSTM cells amplifies the model's ability to learn from data with even more complex patterns, but it also substantially increases the number of parameters, which is evident from the model summary. This can lead to more powerful feature extraction at the cost of a higher risk of overfitting. In terms of backpropagation, this architecture requires more computational resources due to the increased number of weights, and gradients must be calculated through a deeper network, which can be challenging and may require techniques like gradient clipping to manage exploding gradients.

Overall, we observe worst results when stacking three LSTM cells together. In the context of quantitative trading, where market data often exhibits non-linear and complex patterns, adding excessive complexity with multiple LSTM layers might not always yield better results. It's crucial to balance the model's capacity to capture intricate relationships in financial data against the risk of overfitting, which can lead to poor generalization and unreliable trading strategies when applied to real-world market scenarios.

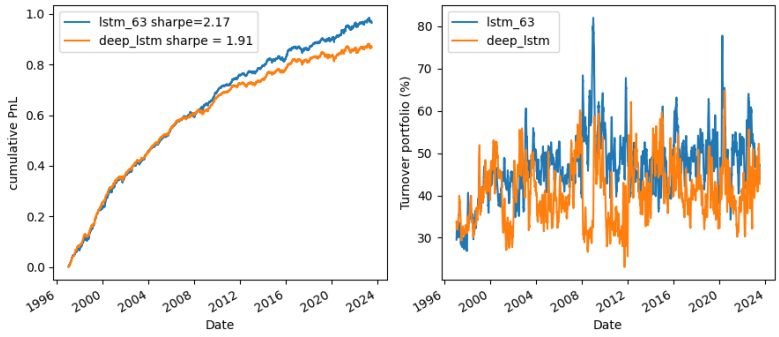

Figure 16: LSTM 63 vs Deep LSTM 63