Recurrent Returns: How Recurrent Neural Networks can Improve Momentum Strategies

Peter McMullan[*]
Department of Computer Science
MSc Computational Finance
COMP0162: Advanced Machine Learning in Finance
ucabpmc@ucl.ac.uk

March 25, 2024

Abstract

A recurrent adage in financial markets posits that "markets can remain irrational longer than an investor can remain solvent" accompanied by the observation that during bear markets, asset correlations tend towards one. These maxims underpin the profitability of the momentum premium, an anomaly present across asset classes and markets. This paper extends current momentum research by constructing long-short, cross-sectional momentum portfolios from the constituents of the S&P 500. Distinct from traditional momentum strategies which develop signals based on historical returns, this paper harnesses the predictive power of Recurrent Neural Networks (RNNs) to forecast next-week returns. Results show that such models can generate superior out-of-sample performance in both bull and bear markets compared to traditional momentum strategies.

1 Introduction

1.1 Motivation

Accurate return prediction is the holy grail of trading. This paper explores the use of Recurrent Neural Network (RNN) and Long-Short Term Memory (LSTM) models in order to predict next-week returns for sectors in the S&P500. This task is notoriously hard given the low signal-to-noise ratio in financial markets, as well as the consistent competition among agents in the market, where an increasing amount of capital is chasing a finite 'alpha pie'. Alongside this, return predictions at a large scale can be extremely difficult due to the high-dimensionality and non-linearity within financial time series.

Given this backdrop, traditional modelling techniques have proven inadequate in generating consistent out-of-sample results, where models must consciously adapt to new information while still incorporating longer-term relationships. In the face of such challenges, Neural Networks, and specifically RNNs and LSTMs, are superior in their ability to predict returns. Due to the 'black box' nature of deep learning techniques like these, return predictions are considered within the framework of constructing a long-short momentum portfolio. This helps with both simplicity and interpretation of results. Each portfolio is tested against a traditional momentum-ranking technique used extensively within the hedge fund industry today to assess the ability of RNNs and LSTMs to outperform.

[*]Thanks to Dr. Paolo Barucca, Prof. Tomaso Aste, Shan Conrad-Wolf, and Antonio Briola

1.2 Literature Review

Momentum The momentum anomaly/premium, in which stock returns tend to be positively autocorrelated over longer time horizons, has been well researched since the seminal work by Jegadeesh and Titman [1993] [23]. Their research demonstrated that stocks that performed well (poorly) in the past three to twelve months tend to continue performing well (poorly). This anomaly contradicts the efficient market hypothesis, which posits that past price movements should not predict future returns. Work by Asness et al. [2013] [6] further confirms the persistence of this anomaly across asset classes and markets.

Momentum has been attributed to delayed information propagation and behavioural effects include herding, under reaction to news, and overreaction to recent returns (extrapolation of past returns) [8]. Arnott et al. [4] found that momentum is more common in group forecasts, such as for industry or factor returns, where the oscillatory residual noise tends to cancel out. Exploiting this market premium has shown to perform well during times of crisis, with a straddle-like payoff, but a more reserved pay-off in slow, rising markets [5].

Regime Modelling Botte & Bao [2021] at Two Sigma [7] present Gaussian Mixture Models as a method for Regime modelling of financial markets, where several normal distributions can be utilised to estimate returns in different regimes. Furthermore, Yuan & Mitra [2019] [20] used Hidden Markov Models (HMM) for a similar task using data on the FTSE 100 and Euro Stoxx 50, discovering that a 2-state HMM performs well in capturing several stylized factors within financial markets, as well as finding the market signal to forecast future market conditions.

Neural Networks for Return Prediction Rather than absolute returns, Andres [2018] [3] used Neural Networks (NNs) to predict the percentile a group of stock's returns will fall into for the next month using the past 11 months of returns. Andres found that Feed Forward NNs were able to generate a portfolio that beats traditional momentum strategies. A comprehensive review of recent literature on the topic of time series forecasting with deep learning conducted by Sezera et al. [2020] [22] has also informed several aspects of this research.

In this paper, individual stocks are split by sector and combined to create equal-weight sector index portfolios. RNN and LSTM models are then deployed to predict 1-week forward returns for each index. This makes the prediction task more robust out-of-sample and aids implementation [4]. The regime identifier outputs [0,1] of a Gaussian Mixture HMM (GMHMM) are then included as an additional layer in the network to assess whether predictive power improves. Finally, portfolios based on such predictions are constructed in an identical manner to the null strategy and performance is compared.

The remaining sections of this paper are structured as follows; Section 2 outlines the methodology followed, including a discussion of the data and models developed. Section 3 describes the results obtained, and Section 4 provides a discussion of the significance and financial reasoning behind these results. Finally, the salient conclusions drawn from the research are highlighted in Section 5

2 Methodology

2.1 Features

Despite the ability of neural network to cope well with large datasets, to ensure comparability of results across the methods employed, features were kept consistent throughout this paper. The features included were;

1. Rolling weekly 12-month return for each stock (or sector index for NN models)
2. Rolling weekly 1-month return for each stock (or sector index for NN models)
3. Regime indicator: Binary [0,1] classification, with 0 representing a bear market and 1 representing a bull market.

A Gaussian Mixture Hidden Markov Model (GMHMM) was used to develop the regime indicator. This model is based on both HMMs and Gaussian Mixture models. Stock returns are often assumed to be normally-distributed, but the tails of this distribution understates actual returns. Therefore, a Gaussian Mixture can improve this, but using multiple normal-distributions for different market environments (regimes). This is illustrated in Figure 1, where the conditional and unconditional Gaussian distributions[2] are observed based on the S&P500 Index over this papers sample period, based on the GMHMM trained.

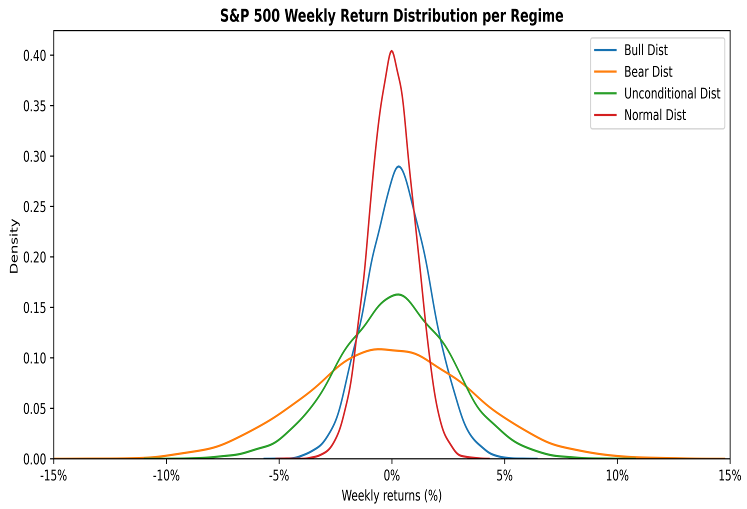

Figure 1: Conditional and unconditional PDFs for S&P500 returns based on regime

2.2 Gaussian Mixture Hidden Markov Model

Hidden Markov Models (HMMs) are an unsupervised machine learning technique developed from Markov Chain's. The number of states are defined prior to training, and the transition/emission probabilities are learned during training. Transitions define the probability of moving from one state to another, and emission probabilities are the likelihood of an observation belonging to a particular state. For this paper, a two-state process was defined, for both explainability and simplicity. Financial markets are often defined as being in bull or bear markets, so a two-state regime-switching model makes intuitive sense, and research by Yuan et al. [2019] [20] confirms the efficacy of such an approach.

For each state the GMHMM generates observations according to a Gaussian Mixture model. Unlike HMMs where each state is associated with a single emission probability distribution, in GMHMMs the emission probability of observing a particular value is determined by a mixture of Gaussian distributions. Each distribution is characterized by its mean and variance, and each component of the mixture has its own weight indicating its contribution to the overall distribution. The mathematical notation to develop HMMs is omitted for brevity but can be found in [1] and [20]. From Figure 1 it's clear that a GMHMM can be effective in learning the distributional properties of each regime, and assign a higher probability to an asset's return being part of a bull or bear market. Initial probabilities for each state were set to 50% to ensure equal likelihood of learning either regime at the beginning of the training period. Since the portfolios are constructed from S&P500 constituents, the GMHMM model was trained on S&P500 Index returns.

2.3 Null Strategy

Alongside the economic rationale for momentum, there must be a statistical backing of the anomaly, which comes in the form of autocorrelation of returns (trending prices). Returns for a subset of the selected stocks were tested for autocorrelation using the Durbin-Watson test. For the subset of 100 stocks analysed, all rejected the null hypothesis, confirming autocorrelation.

The null momentum strategy was developed as a benchmark for which the performance of RNN and LSTM models would be compared, considering the trade-off between complexity and performance. The null strategy is a 12mth-1mth, long-short momentum strategy. This is an industry-standard strategy, where stocks are ranked based the long-term trend excluding the last month because assets are often mean-reverting in the short-term. The steps to build the strategy are as follows;

- Signal: Score each stock based on its $\frac{\text{price}_{i,t-1\text{mth}}}{\text{price}_{i,t-12\text{mth}}}$ at each time step t
- Relative Ranking: Cross-sectionally rank the signal within each sector (i.e. versus all other stocks in the sector at t)
- Selection: Long the top 20% of stocks in each sector and short the bottom 10% when in a bull market regime, and vice versa when in a bear market regime
- Portfolio Construction: Equal-weight within and across sectors.

All subsequent neural network based strategies are constructed in the same manner, where the signal step is replaced with the return prediction. Due to research mentioned in Section 1.2 [4], the NNs were used to predict weekly returns for a *sector index*, rather than for individual stock. Each sector index is an equal-weight portfolio of all the underlying stocks in the sector. Each stock's return was then calculated by multiplying the return prediction for the sector index by each stocks rolling 3-month beta to that index. 3-months was selected as a sufficient time frame to ensure a reduction in the 'noise' of the estimate, while allowing for shorter-term variation due to regime-specific volatility. Beta was calculated as;

$$\text{Rolling Beta} = \frac{\text{Covariance (Stock, Benchmark) over 3 months}}{\text{Variance (Benchmark) over 3 months}} \tag{1}$$

This formula provides a measure of the stock's correlation relative to its sector index and is therefore suffice in calculating each stocks predicted movement based on the index prediction. The predicted returns for each stock were then passed into the same relative ranking procedure. Therefore, stocks are ranked on the neural network's 1-week return prediction, so the predicted best performers over the next week are bought and the worst are shorted. The portfolio construction procedure is the same as above, ensuring direct comparability between all the approaches described. The portfolio construction technique chosen (1/N) reduces the chances that performance is attributed to the success of an optimiser rather than the task at hand, return prediction. [1]

[1]Each model was also ran to predict individual stock returns, but performed poorly so were disregarded. This is likely a mix of data issues and even lower signal-to-noise at the single stock level.

2.4 Neural Network

2.4.1 Architecture[2]

Unlike traditional feed-forward neural networks, RNNs have connections that form directed cycles, allowing them to maintain a 'memory' of previous inputs in their internal state. The model recursively applies a transition function to its internal hidden states using the current input x_t and the previous hidden state h_{t-1}:

$$h_t = \phi(W x_t + U h_{t-1}) \tag{2}$$

where W is the input-to-hidden weight matrix, U is the state-to-state recurrent weight matrix, and ϕ is the hyperbolic tangent function (tanh) for this papers implementation (see Section 2.4.2). An RNN can then predict the probability of the next step x_{t+1} given the current hidden state h_t, which is a function of all previous steps and the current one x_t:

$$p(x_{t+1}|x_1, ..., x_t) = g(h_t) \tag{3}$$

While RNNs can theoretically handle long-range dependencies, in practice they often fail because backpropagation can lead gradients to converge or diverge exponentially, making learning unstable [16] - the infamous vanishing and exploding gradient problem. LSTMs help mitigate against this.

LSTMs effectively retain long-term dependencies meaning the gradients can flow over many time steps, helping to mitigate against vanishing gradients. This is done through gates that regulate cell states. LSTM cells consist of input i_t, forget f_t and output gates o_t, as well as a memory cell. These gates control the flow and modification of information from the memory content, enabling LSTMs to maintain stable error gradients. This memory exposure/updating is the benefit of LSTM versus standard RNNs, and is critical for learning from lengthy data sequences. The memory cell carries the information about previous and current content of an LSTM unit. The gated activation function is designed to have more persistent memory so that it can capture long-term dependencies more easily, improving upon the architecture of RNNs. The content of the memory cell is updated as the weighted sum of the previous and current memory content, controlled by the input and forget gates[3]:

$$\begin{aligned} c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t & &\text{(Cell State Update)} \\ \text{where} \quad \tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) & &\text{(Cell State Candidates)} \end{aligned} \tag{4}$$

This is depicted in Figure 2.

The input and forget gates control how much information is memorised and how much is forgotten at each time step. Once again, these gates are calculated using the weighted sum of the current and previous states:

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \qquad \text{(Input Gate)} \tag{5}$$

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \qquad \text{(Forget Gate)} \tag{6}$$

$$\tag{7}$$

where $\sigma(.)$ is a logistic sigmoid function and b_i, b_f, b_c, and b_o represent the bias associated with a particular gate. The bias plays an important role in adjusting the output of the gates, effectively shifting the activation function to the left or right, which can help model the data better.

[2]Methods, information, and parameter selection throughout this section are adapted from Geron [19], Hochreiter et al. [15] and Chung et al. [13].

[3]Equations taken from [11]

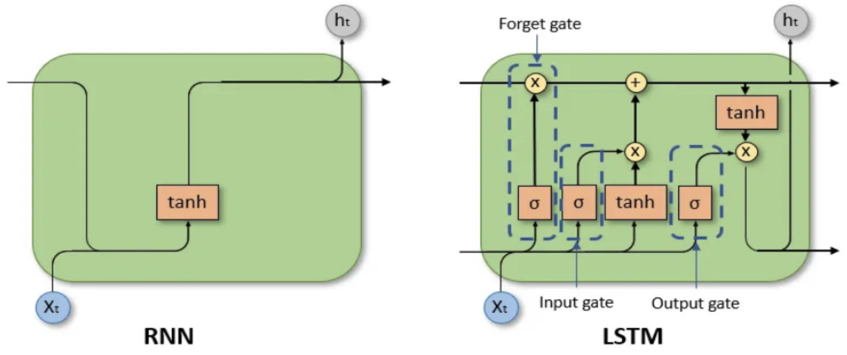

Figure 2: RNN and LSTM cell architecture [9]

The hidden state is then updated after the memory content:

$$h_t = o_t \odot \sigma_h(c_t) \qquad \text{(Hidden State)} \qquad (8)$$

The output gate is once again a weighted sum of previous and current variables. This controls how much memory is used at each time step:

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \qquad \text{(Output Gate)} \qquad (9)$$

Therefore, LSTM calls can incorporate, discard, and reveal information at each time step, capturing long and short-term dependencies in time series. These processes are controlled by the gates, and in multi-neuron networks these processes can happen simultaneously so both long and short-term stock movements can be incorporated into the model[13]. This is why LSTMs are beneficial to include in momentum strategies, where exposure to long-term trends are key, while mitigating against the risk of short-term mean-reversion.

For this paper, two-layer deep RNN and LSTM models were built. The RNN model was used as the base prediction model to beat given its simplicity versus LSTM. Each layer includes 32 neurons, with the final fully-connected layer contained 11 neurons, since 11 predictions (11 sector index returns) per unit time t (1-week) were needed. The number of neurons was selected to be sufficiently large to capture meaningful relationships while not being too large versus the number of outputs as this could hinder the model's generalization capabilities. Cross-validation of the base prediction model (RNN) confirmed that 32 neurons was approapriate. A 20% recurrent dropout within each LSTM layer, and between the two layers was also deployed, which is standard practice as described in Geron [2023] [19]. Dropout randomly sets a fraction of input units to 0 at each update, meaning the network becomes less sensitive to the specific weights of neurons. This reduces the chance of overfitting.

Layer normalisation (LN) was also included to standardise the activation's of each layer [17]. This technique calculates the first and second moments of the activation's across features for each instance in a batch. Similar to Batch Normalization, LN also adjusts each input by learning unique scale and shift parameters. It is applied to each sample *independently*, which makes it less dependent on the batch size. Consequently, it maintains consistent behavior between training and testing phases, unlike Batch Normalization. Implementing Batch Normalization in RNNs presents challenges due to varying sequence lengths and the requirement to compute statistics across full sequences [19]. Thus, LN proves to be a more fitting option for this papers implementation. The structure of the LSTM network used in this paper is conveyed in Figure 3. This paper's RNN model has an identical set up, replacing the LSTM layers/cells with RNN cells. For the advanced models developed in this paper (e.g. 'RNN with Regime'), a fully-connected feed-forward layer composed of the regime indicator is added prior to the RNN/LSTM layer.

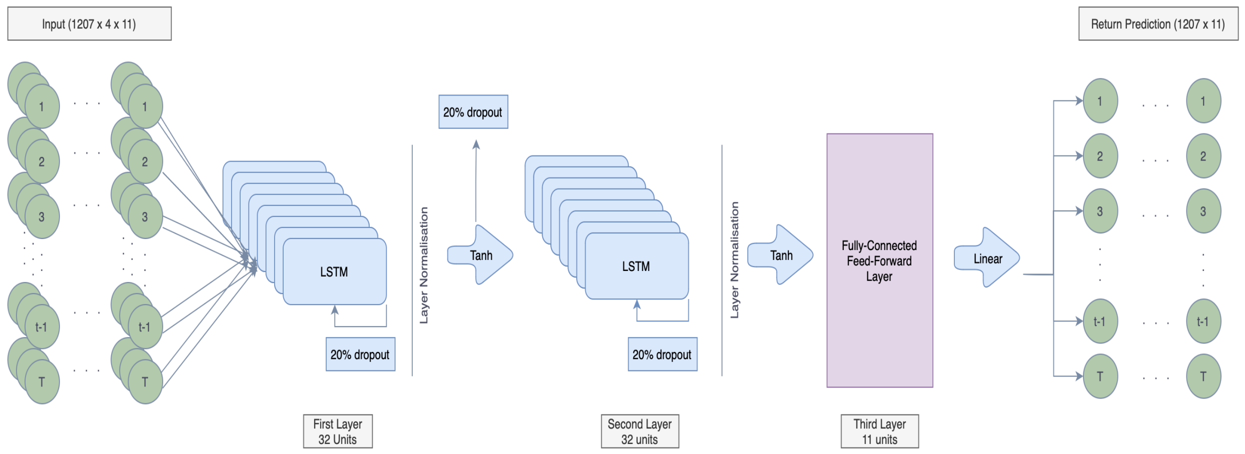

Figure 3: LSTM Model Architecture

Finally, the time frames (batches) passed into the models were split into rolling 4 period (week) windows. 4, 8, and 12-period windows were tested and scored in-sample via the performance metrics discussed in Section 3, and the 4-week period performance was significantly better. This also directly aligns with the 1-month price used in the null strategy, aiding comparability. Passing this rolling window into the RNN/LSTM cell helps to make clearer the temporal relationship between a sequence of inputs and the corresponding output. Therefore, the model learns from fixed-size windows and predicts the next time step based on the most recent fixed window of history.

A number of parameters were also specified in the models to aid stability during training.

2.4.2 Parameter Selection

The hyperbolic tangent (Tanh) activation function was chosen for all models as it is the preferred function for RNNs [18]. This function scales outputs in the range -1 to 1 and is a common activation function for hidden layers because it can help manage the gradient flow during back-propagation and centres the output, making it easier for the next layer to learn. Gradient clipping and the `he_normal` kernel initializer could also be used to mitigate against unstable training, but this papers implementation was relatively stable so there was no need for this added complexity. The final layer has a linear activation function. This is a regression problem, so using a linear activation function allows the model to output values in the range that the returns can take, which may be beyond the [-1, 1] range of Tanh.

L1 regularization was also applied to the kernel weights with a regularization strength $(\lambda) = 0.01$. L1 regularization penalizes large weights in the network by adding a term to the loss function proportional to the absolute value of the weights. This creates an optimisation landscape that can sets weights to zero, helping reduce over-fitting through feature selection. This was chosen to improve the likelihood of out-of-sample success for the models. $\lambda = 0.01$ was confirmed through cross-validation.

The Adaptive Moment Estimation (ADAM) optimizer excels in managing sparse gradients and customizing the learning rate for individual parameters. It dynamically modifies the learning rate for each parameter, drawing on the estimated first and second moments of the gradients. This makes it effective when working with noisy data or when model's learning characteristics change over time. Both are true for financial time series. However, pairing ADAM with L1 regularization tends to produce models which generalise poorly, so AdamW was used to overcome this.

In order to speed up training, prevent overfitting, and mitigate against instability, both `EarlyStopping` and `ReduceLROnPlateau` were employed. The former stops training epochs when the validation loss rises from its minimum on 40 subsequent epochs. Once it has risen 40 times, the best weights are restored and the model training terminates. This is essential in return predictions as epochs will continue to learn the exact path of each stock and generalise very poorly. Therefore, the choice of 150 epochs had little impact on the models. `ReduceLROnPlateau` reduces the learning rate for the AdamW optimiser when the validation loss plateaus/rises on 10 subsequent epochs with a `factor`=0.5 and the minimum learning rate was set to `min_lr` = 0.0001. The loss function was chosen to be Mean Squared Error (MSE), a standard measure for regression tasks. MSE is discussed further in Section 3.

2.5 Data & Pre-processing

This paper is based on stock data for the constituents of the S&P500 Index from 1st January 1990 - 1st March 2024. Company/Sector information was collated using web-scraping from Wiki [24], and price data was obtained from *Yahoo Finance* using its API. Information on the Federal Funds rate was collected from the *FRED* website using its API.

Due to issues gathering survivorship-bias free data, -3% was taken from performance of the portfolios as a correction term to reflect fairer performance measurement versus benchmark indices [12]. Because of such data issues, stocks with missing values were ignored during the scoring/portfolio construction process, and stock returns were clipped in a range of -20% to 20% to ensure small-cap stocks in the sample period did not positively-bias results.

All features were shifted to ensure no look-ahead bias, and scaled to ensure equal importance when training the RNN and LSTM models. Features were also resampled to a weekly period, which becomes the portfolio rebalance frequency. Although a monthly rebalance would be more intuitive given the monthly momentum signal measurement, weekly periods were necessary to obtain statistically robust results out-of-sample for the deep learning techniques applied, so the trade-off is justified. Finally, transaction costs were also accounted for, with an annual -3% taken from performance to account for bid-ask spread, commissions, and market impact from trading.

3 Results

3.1 Gaussian Mixture Hidden Markov Model

The regime model performed strongly in learning the bull/bear regimes of S&P500 Index returns. One drawback is that the method is slow to change between states, as can be seen following the 2008 Financial Crisis and the 2020 COVID-crash in Figure 4.

3.2 Model performance

Root Mean-Squared Error, RMSE $= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$, was used as the scoring metric for quality of prediction, a standard metric for regression tasks. This metric maintains the properties of MSE but is more interpretable in terms of error magnitude. It is also sensitive to large errors, so it penalizes large deviations more severely. MSE was used as the loss function when training each model, so each cell state and gate updates its weights based on minimising this loss function.

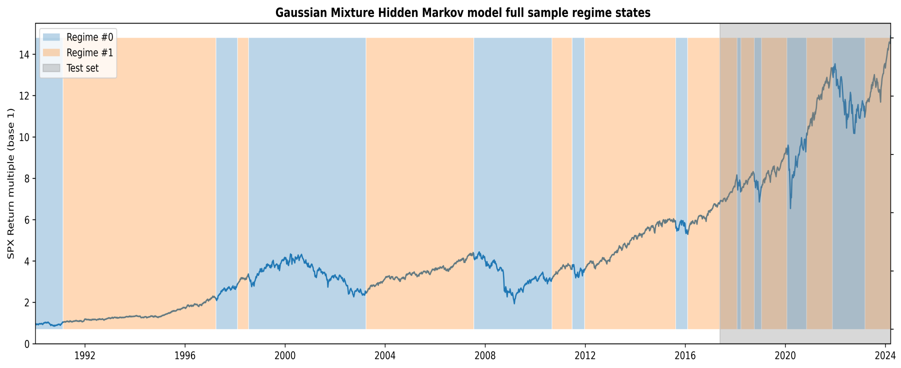

Figure 4: Gaussian Mixture HMM model performance: full-sample

K-fold cross-validation fails in financial times series because *'observations cannot be assumed to be drawn from an IID process... and the testing set is used multiple times in the process of developing a model, leading to multiple testing and selection bias'* [10]. Therefore, to ensure statistically robust results, a `tensorflow.random_seed(42)` was set and each neural network was ran 100 times to obtain an average performance. A number of trading related metrics were included to assess the quality of each strategy, including;

$$\text{Sharpe Ratio (SR)} = \frac{R_p - R_f}{\sigma_p} \qquad \text{where } R_p = \text{portfolio return}, \quad R_f = \text{risk-free rate}, \quad \sigma_p = \text{standard deviation of portfolio return} \tag{10}$$

$$\text{Max Drawdown (MDD)} = \max_{t\in[0,T]}\left(\max_{\tau\in[0,t]}(V_\tau) - V_t\right) \qquad \text{where } V_t = \text{portfolio value at time } t, \quad T = \text{total time period} \tag{11}$$

$$\text{Total Return (TR)} = \left(\frac{V_T - V_0}{V_0}\right) \times 100 \qquad \text{where } V_T = \text{final portfolio value}, \quad V_0 = \text{initial portfolio value} \tag{12}$$

In order to accurately assess the significance of the results, the Sharpe ratio can be converted to a t-test using methods developed by Harvey et al.[14]. The Sharpe ratio was multiplied by $\sqrt{6.66}$ as the out-of-sample period was 6yrs 8 months. This becomes the test statistic. A Bonferroni correction is used (n=100) to account for the 100 test runs completed. Statistics are highlighted in Table 3.2 and one out-of-sample performance comparison across all strategies, including the S&P500 Equal-Weight Index (RSP), is included in Figure 5.

4 Discussion

Table 1 shows all four models perform strongly on average, outperforming both the null strategy and the S&P500 Equal-weight Index on a Sharpe Ratio and Total Return basis, with all SRs statistically significant at the 1% level. The RMSE is identical across the models, and this score actually conveys a large difference between the actual and predicted values of returns for the sector indices. However, this doesn't impact performance due to the percentile-ranking conducted following prediction. This

Model	Av. SR*	Av. MDD	Av. TR	Av. RMSE	Significance
S&P500 Equal Weight	1.06	-36.8%	75%	-	-
Null Model	1.27	-8.4%	44%	-	-
RNN	1.95	-11.9%	96%	0.04	1%
LSTM	1.81	-13.2%	81%	0.04	1%
RNN w regime	1.78	-13.0%	82%	0.04	1%
LSTM w regime	1.77	-13.5%	81%	0.04	1%

Table 1: Average performance metrics out-of-sample based on 100 runs per model
*including a 3% risk-free rate over the period.

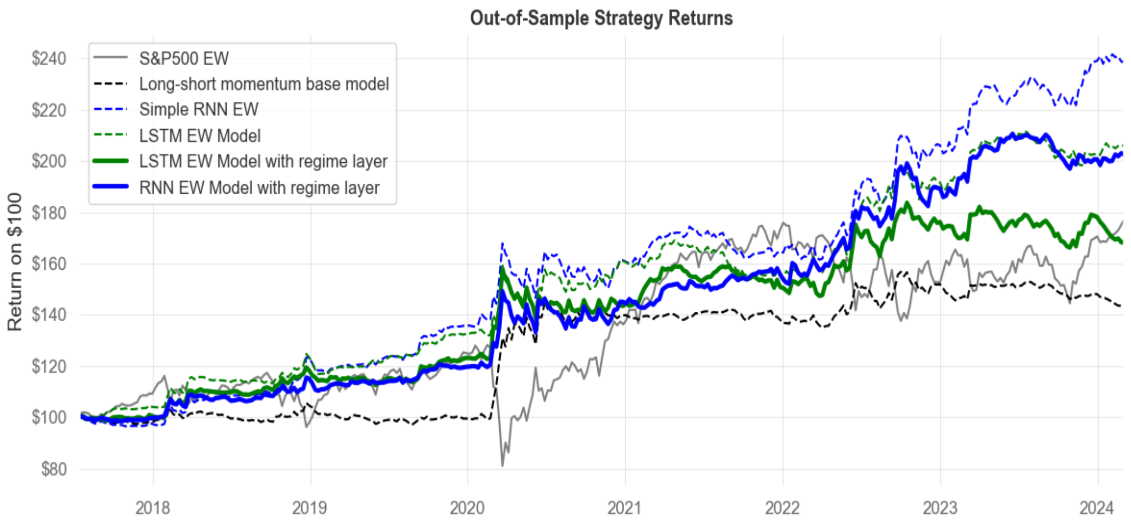

Figure 5: Out-of-sample returns for all strategies with one model run

means the neural network only needs to get the *relative* performance correct at the next time step t in order to be successful in trading, which it evidently does. It should be noted that all strategies (including the null) exhibit a negative correlation to the S%P500 Equal-Weight Index (RSP) (average of -0.64), and all the momentum strategies are highly correlated to each other, with values ranging from 0.72-0.86.

Out-of-sample, strong performance was exhibited during the COVID-19 crash of March 2020 across all strategies/models. Over the four weekly rebalances from 01/03/2020 - 22/03/2020, all models made strong returns, with short positions in the Energy and Financial sectors paying off handsomely. It should be noted that all four deep-learning based strategies produce more consistent returns while cutting the number of drawdown days in half versus the null strategy *with similar volatility* ($\sim$ 23% annualized). This suggests that RNN models can improve both risk and return for momentum strategies.

One potential explanation for the outperformance of the simplest model (RNN without regime layer) is that the LSTM models and the RNN with regime layer model were ran with the same number of neurons as the RNN model. This was done in order to gain a direct comparison between approaches. Rerunning the LSTM model with 64 neurons (vs 32 neurons for the published results), leads to a slightly higher average SR of 1.88, MDD of -13.7%, and TR of 85%. This shows that doubling the neurons for the LSTM still doesn't beat the simplest model. This is actually the best result that could have been obtained, as there is no complexity/performance trade-off to consider — the simplest model is the best performer across all metrics!

Given that the construction of all models architecture and portfolios is identical, another explanation for poorer performance of the LSTM models could be due to the difference in cell architecture. Analysing in and out-of-sample allocations across sectors helps to understand this. The base RNN model has more stable returns versus all other models. The LSTM models seems to be 'over-confident' in-sample, making larger returns but also larger losses, which may be the reason for its under-performance out-of-sample. This may relate to overfitting in the short-term memory part of the LSTM cell. The Simple RNN model seems to avoid large losses while still capturing adequate gains compared to LSTM. For example, during the in-sample period 2008-10 (Financial Crisis), the RNN model made 22% returns with 7% ann. volatility compared to the LSTM which made 43% returns with 13% ann. volatility. The underperformance of the regime-layered models is surprising, but may be because such models can learn regime-specific features themselves without the need for added complexity.

5 Conclusion

This paper has demonstrated that Recurrent Neural Networks can be successful in improving momentum strategies out-of-sample (including transaction costs). Performance suggests such methods could be incorporated into existing momentum strategies to provide an aggregate signal which can be used to influence trading decisions. This method can improve both drawdown statistics and partaking in the upside of beta strategies, therefore providing upside unification and downside diversification. To extend this analysis, a number of techniques could be employed.

Firstly, rerunning the models with a larger number of neurons/layers may lead to different results. This paper produces relatively 'shallow' networks to avoid overfitting out-of-sample, and due to computational constraints. Extending this analysis to deep networks with more features may improve results further.

The next challenge comes at the single-stock level. Cross-sectional momentum exhibits 0 correlation to carry strategies and 0.3 correlation to Time-Series Momentum (TSMOM). This paper's research could be extended by building a neural network to predict at the single stock level, therefore incorporating TSMOM in order to build a portfolio based on both top-down and bottom-up momentum. Additional features could be added to predict stock returns, such as volume data. Furthermore, since carry strategies traditionally perform better during bull markets while momentum is preferred in bear markets, carry could also be incorporated to build an 'all-weather' portfolio. The regime indicator could allocate between momentum and carry based on such regimes. The methods described in this paper could also be extended to include multiple markets/asset classes.

Neural Networks could also be used in portfolio construction at the intra/inter-sector level, using features including correlation, and volatility. Ramamohan et al. [2020] [21] used LSTMs to predict both the direction and strength of a stock trend using Indian SENSEX stock data with deep learning, and proposes a portfolio construction framework which could be used to extend the methods described in this paper.

References

[1] Andreas A. Aigner. A gaussian mixture hidden markov model for the vix. *TradeFlags, SSRN*, 2022.

[2] Wifey Alpha. Models, regimes and trend-following. *wifeyalpha.com*, 2023.

[3] Andres. Beating momentum: Using neural networks to uncover additional information in past returns. *SSRN*, 2018.

[4] Kalesnik Linnainmaa† Arnott, Clements. Factor momentum. *SSRN*, 2017.

[5] Kalesnik Linnainmaa† Arnott, Clements. The best of strategies for the worst of times: Can portfolios be crisis proofed? *SSRN/Strategic Risk Management*, 2019.

[6] Pedersen Asness, Moskowitz. Value and momentum everywhere. *The Journal of Finance*, 2013.

[7] Botte Bao. A machine learning approach to regime modeling. *Two Sigma*, 2021.

[8] Fan Bianchi, Drew. Commodities momentum: A behavioral perspective. *The Journal of Banking and Finance*, 2016.

[9] Jonte Dancker. A brief introduction to recurrent neural networks. *https://towardsdatascience.com/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4*, 2022.

[10] Marcos Lopez de Prado. Advances in financial machine learning. *Wiley*, 2018.

[11] Dr. Aste Dr. Barucca. Comp0162 advanced machine learning course powerpoint. *UCL*, 2024.

[12] Blake Elton, Gruber. Survivorship bias and mutual fund performance. *NYU Stern School of Business*, 1994.

[13] Chung et al. Gated feedback recurrent neural networks. *arXiv, year=2015,*.

[14] Harvey et al. Evaluating trading strategies. *SSRN*, 2014.

[15] Hochreiter et al. Long short-term memory. *Neural Computation*, 1997.

[16] Hochreiter et al. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 1998.

[17] Jimmy Lei Ba et al. Layer normalization. *arXiv*, 2016.

[18] Vu Pham et al. Dropout improves recurrent neural networks for handwriting recognition. *arXiv*, 2013.

[19] Geron. Hands-on machine learning with scikit-learn, keras, and tensorflow. *O'Reilly.com*, 2023.

[20] Yuan Mitra. Market regime identification using hidden markov model. *UCL/OptiRisk Systems*, 2019.

[21] Parth Mahlawat & Prabhakar Ramamohan, Goyal. A framework for utilizing stock trend prediction outputs in stock selection and portfolio optimization. *SSRN*, 2020.

[22] Ozbayoglua Sezera, Gudeleka. Financial time series forecasting with deep learning : A systematic literature review: 2005-2019. *SSRN*, 2020.

[23] Jegadeesh Titman. Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, 1993.

[24] Wiki. List of s&p 500 companies. *https://en.wikipedia.org/wiki/List_ of_ S%26P_ 500_ companies*, 2024.