# Investments: Machine Learning for Finance

Alexander Jean-Marc Fuchser, Daniel Eymann, and Stefan Richard Saxer

***Abstract*—Stock price forecasting is a fundamental challenge in the field of finance. This report presents a survey of various machine learning architectures and examines how different input signals influence prediction performance across multiple future time horizons. The study aims to provide insights into the effectiveness of diverse modeling approaches and signal combinations for S&P 500 market forecasting.**

## I. Introduction

Financial market forecasting is a high-noise, non-stationary time-series problem. Traditionally, econometric techniques, most notably autoregressive integrated moving-average (ARIMA) and generalized autoregressive conditional heteroskedasticity (GARCH) models, have been employed to predict the evolution of individual stocks and broad indices, often through carefully engineered factor specifications that capture trends, seasonality, and volatility clustering. While such models provide interpretable parameterisations of market dynamics, their linear structure limits their ability to approximate the more complex relations observed in practice. In recent years, however, it has been shown that modern machine learning architectures, from shallow multilayer perceptrons to attention-based Transformers, can uncover more complex patterns in financial time series. This paradigm shift motivates a systematic evaluation of deep-learning approaches under an experimental framework.

A recent meta-analysis covering 380 deep learning papers underscores how quickly architectural diversity has expanded, from multilayer perceptrons (MLPs) to graph and attention-based models, yet also highlights persistent reproducibility gaps and data-leakage pitfalls in the literature [1]. This paper reviews four neural network architectures: the standard MLP, CNN, an LSTM, and a Transformer under a unified experimental protocol for S&P 500 index prediction and trading.

*a) Multilayer perceptrons (MLP):* Early evidence that shallow nets can extract exploitable non-linearities in U.S. equity returns dates back many years [2]. Subsequent studies refined feature sets with technical indicators and macro factors, but performance improvements often fade out-of-sample owing to the models' limited temporal inductive bias [3]. Recent work revisiting MLPs with modern regularisation and learning rate schedules reports solid baselines for index-level regression tasks [4].

*b) Convolutional Neural Networks (CNN):* Although multilayer perceptrons (MLPs) are universal function approximators in theory, they are not necessarily the most effective architecture for every practical task. Rather than relying exclusively on standard MLPs, many researchers employ convolutional neural networks (CNNs) for time-series analysis. Originally designed for images, CNNs slide learnable kernels across an input to detect local features; the same idea applies to temporal data, where a kernel moves along the time axis to isolate recurring patterns or anomalies. In quantitative finance, this analogy has motivated a series of studies that deploy CNNs for stock-market prediction [5], [6].

*c) Long short–term memory (LSTM):* While CNNs are architecturally well-suited for learning local dependencies in time-series data, their finite receptive fields make it difficult to capture long-horizon effects. Long-short-term memory (LSTM) networks overcome this limitation by augmenting recurrent layers with a memory cell and three gates: input, forget, and output, which regulate the flow of information and gradients. This design enables the network to retain or discard signals over many timesteps, allowing the modeling of short-lived market shocks and slow macroeconomic drifts within a single framework. Leveraging these properties, Pilla and Mekonen [7] applied an LSTM to S&P 500 forecasting. Earlier, Fischer *et al.* [8] benchmarked LSTMs against traditional econometric models on the same index to underline the advantages of this architecture for financial time-series analysis.

*d) Transformer:* Transformers, introduced by "Attention Is All You Need" [9], rely on a self-attention mechanism in which each timestep dynamically weights all others, allowing the model to capture both short and long-range patterns without recurrence. The success of large language models raised interest in applying the same principle to various tasks. Recent studies have shown that sparse attention Transformers can predict daily S&P 500 movements more accurately than LSTM and CNN baselines while maintaining fast inference [10]. These findings position Transformers as a promising next-step architecture in quantitative finance.

## II. Objective

This paper aims to investigate and leverage the potential of the deep learning architectures previously outlined for predicting the S&P 500 index.

The art of investing lies in identifying an asset's future price movement. Thus, investors need to make assumptions about the future state of an asset to turn a profit. Depending on the position prediction, an investor will either go long or short on the asset. In the space of asset investing, there exists an uncountable number of methodologies designed to predict the future performance of stocks. This paper will focus on developing ML-based trading algorithms that predict the future

values of the S&P 500 market index (the feature "_MKT" in the dataset).

The ML-based trading algorithms are trained based on a test sample dataset. Afterwards, the financial performance as well as the predictive power of the trading algorithms are compared in order to identify algorithms that could be applied to real portfolios.

## III. Signal Selection

An essential step before training any model is acquiring and selecting data. Dr. Feiler provided us with a dataset containing 28 features. This necessitates a feature selection protocol, as it would be computationally inefficient to train a model on such a large feature space and probably lead to severe overfitting.

Several well-known models provide economic rationale for the development of asset prices. Based on such models, it would be possible to pick features. Two relevant ones are the classic dividend discount model and the lesser-known short-selling model. Both economic models are based on financial theory and use a maximum of four signals to avoid overfitting issues.

Additionally, there are well-established feature selection methods in data science, such as filter and wrapper methods. These approaches have the advantage of finding a feature selection on their own without economic rationale; however, they sometimes have huge computational requirements and suffer from explainability. This paper will focus on heuristic feature searches, a specific type of wrapper method.

### A. Classic

The dividend discount model (DDM) [11] proposes that the current price is the sum of all future dividends discounted by the difference between the discount rate and the projected growth rate. All future dividends are approximated in this model by the signal of corporate earnings (ED). The discount rate is approximated with the real interest rate (RR) while the growth rate is mimicked with the GDP growth rate (GDP).

### B. Short Selling

In brief, the Short Selling model is designed to generate a trade signal that anticipates crashes. Thus, when the model predicts a crash, the market is shorted, and when the model does not predict a crash, the algorithm takes a long position in the market.

Historically, during boom periods, signs of exuberance can be observed preemptively. Such signs could be enormous deviations between book and market values, hyper-exponential growth in stock prices, etc. The signals most supporting this theory would be the NYF index, which indicates the probability of a recession, the YSS index, which measures systemic risk, a consumer confidence index (CF), and the price of gold (AU), as the price of gold typically increases in a crisis.

### C. Heuristic Feature Search

Heuristic feature search leverages heuristic search algorithms to efficiently search the entire feature space for an optimal feature selection based on the performance of a simplified proxy model on a validation set (and not the test set). Exhaustive search of the entire feature space is infeasible as there are $2^{28}$ possible feature combinations. Theoretically, such a systematic feature selection protocol should significantly outperform the signal selection based on economic theories, albeit at the cost of computational power requirements. However, economic rationale can be used to validate and explain the findings of the algorithmic feature selection.

The heuristic feature selection algorithm implemented for this paper is outlined in detail in the experiments section later in this paper.

## IV. Network Architectures

For this paper, we explored various network architectures for time series forecasting. Popular architectures include recurrent neural networks, such as LSTMs, convolutional neural networks, and transformers. In addition to these three network types, we also decided to implement a standard multi-layer perceptron network as a baseline. Given the absence of an inherent mechanism to model sequences, the pure multi-layer perception network should be expected to perform the worst among the four.

A particular challenge with time-series forecasting is limited data availability. Our dataset spans from 1988 to 2024 and is spaced by weekly intervals, which implies ca. 36 * 52 = 1'872 datapoints. This is nothing in the context of deep learning and should result in severe overfitting. Therefore, strong regularization is necessary.

### A. Multi Layer Perceptron (MLP)

The multi-layer perception network we implemented has at least an input layer, a hidden layer, and an output layer, i.e., follows a standard two-layer architecture, as the universal function approximation theorem suggests that this network should be able to approximate any function to any reasonable degree given a sufficient number of hidden neurons. The number of hidden layers and their size can be increased if necessary. All the PReLU activation functions of the network are preceded by a batch-norm layer and followed by a dropout layer. If the target sequence length is larger than one, the output layer consists of the number of neurons matching the target sequence length.

### B. Convolutional Neural Network (CNN)

Our convolutional neural network consists of several convolution blocks followed by a flattening layer and a multi-layer perception network as outlined above. For a given network, all convolution blocks follow the same architecture and, in addition, use a common shared kernel size, padding size, and stride. Each convolution block starts with a 1D kernel

convolution of odd size. The stride of this convolution is always one, and the padding is either zero or half the kernel size to ensure a constant size for input and output if desired. Typically, the number of output channels is double that of input channels. This convolution is then followed by a max-pooling layer, a batch-normalization layer, and a PReLU activation function.

Multiple of these blocks are chained, and finally the output is flattened and fed into the multi-layer perception part of the network.

### C. Long short-term memory (LSTM)

The LSTM network consists of multiple chained LSTM blocks arranged in the typical structure of an encoder, where the cell-state and hidden-layer state are fed from one block to another. We did not use a decoder structure for the LSTM network, as this would have a) required teacher-forced training and autoregressive inference, resulting in an exposure bias, and b) been a little overkill for a target sequence length of one, which was the case for all our experiments. Instead, the final LSTM block's final hidden state is fed into a multi-layer perception network as outlined above.

### D. Transformer

We also implemented a Transformer model following the vanilla Transformer architecture from the "Attention Is All You Need" paper [9]. Attention is potentially advantageous over convolutions as it has a larger receptive field that spans the whole input. However, transformers suffer from similar limitations as LSTMs with an encoder-decoder structure. Additionally, transformers have millions of parameters, which could lead to overfitting. Therefore, training a transformer only makes sense if the other models fail to converge and suffer from underfitting. As it turned out, this was not the case, and we ultimately did not train a transformer model.

### E. Regularization techniques

Where applicable, we applied the following regularization techniques:

- **Activation Functions**: We always used the PReLU activation function. The PReLU activation function avoids the vanishing gradient problem of other activation functions, such as sigmoid, but induces less sparsity than ReLU.
- **Dropout**: All activation functions in fully connected layers are followed by a dropout layer to prevent the creation of master neurons and strengthen the network's ability to generalize to unseen data.
- **Batch Normalization**: All activations are preceded by a batch-norm layer. This helps against exploding and vanishing gradients and tackles the internal covariate shift problem. Given the nature of batch normalization, we do not need bias neurons in our networks.
- **Normalization of Inputs**: All network inputs are normalized from zero to one to prevent scaling issues and the creation of master-features / master-neurons. Note that the min and max values from the training dataset must be applied to the validation and test sets to prevent data leakage.
- **Early Stopping**: We applied a variant of early stopping where we selected the model from the epoch with the lowest loss on the validation set instead of the model after the final epoch.
- **Target Scaling**: If necessary, we scaled the targets by a constant factor to reduce the expected partial derivatives of the loss values, which results in larger learning rates and allows for a more granular tuning of convergence.

## V. Experiments and Results

### A. Experiment Setup

Three main factors determine time-series prediction experiments.

- **The Look-back Window**: The number of time steps into the past available to the model for its prediction. This is often also referred to as input sequence length. Given a similar architecture, the longer the input sequence, the more parameters the networks require. Therefore, we chose a constant window size of 52 time steps, i.e., one year, for all our experiments.
- **The Look-Ahead**: The number of time-steps between now and the start of the prediction window. The larger the look-ahead, the further in the future lies the predicted value. And the further into the future a prediction has to be made, the more difficult the task becomes, but also the more valuable a prediction becomes economically. We evaluated various look-aheads of one, four, 13, and 52 time-steps to account for this trade-off.
- **The Prediction-Length**: This is the length of the predicted target sequence. In our experiments, this was always one. For a basic long vs short trading strategy, the final value of an asset is much more important than the exact path of the value over time.

In addition, for asset price prediction, it is intriguing to predict not only the absolute index value but also the relative change, i.e., the return, because the return is invariant to scaling. Index values can grow over time, leading to a strong train-test discrepancy where the model has to predict values of a magnitude it never saw during training. Therefore, we evaluated both the prediction of indexed targets, where the target value is the index value, as well as non-indexed targets, where the target is the cumulative relative change, i.e., the cumulative return, for the entire look-ahead time frame.

The underlying dataset for our experiments is the XLSX dataset provided by Dr. Feiler. We converted the file to CSV for simpler handling. For all our experiments, we split the dataset into three sub-datasets. A training dataset containing the first 60 % of data, a validation dataset containing the next 20 %, and a test dataset containing the last 20 % of data. We split the three datasets in a way where the three datasets overlap slightly, e.g., the input sequences for the first targets

in the test set lie within the range of the last targets of the validation dataset. This was done to avoid losing too much data, as we only have roughly 1'800 data points available. Note that the targets of the individual datasets do not overlap at all, so there is no data leakage.

### *B. Hypotheses*

Based on the previous remarks and considerations, our selected model architectures and experiment setup, we formulated the following hypotheses:

1) The longer the look-ahead, the harder accurate prediction becomes.
2) The longer the look-ahead, the larger the impact feature-selection has.
3) The choice of model and its architecture has a meaningful influence on the prediction result. The MLP performs worse than the other two as it lacks awareness of sequentiality.
4) When predicting returns, i.e., non-indexed targets, the models tend to predict a return of zero, i.e., expect the last known value of the asset.
5) The weak efficient market hypothesis holds, and it is impossible to beat the market with our trading strategy.

### *C. Heuristic Feature Search*

Heuristic feature search aims to find the features that allow a model to approximate the targets best and generalize to unseen data. Our implemented algorithm is a variant of forward feature selection inspired by the heuristic greedy best-first search algorithm (that is, A* search with a path cost weight of zero). The main deviation from greedy best-first search is the node generation process (and therefore exploration order) of child nodes and estimating the heuristic value.

The core idea of the algorithm is to start with a single feature (in our case, the target variable MKT) and then try to add other features iteratively to the previous best feature combination until the algorithm runs out of resources, in our case, a set time limit.

The "quality" of a feature selection is given by its heuristic value. For this, a small convolutional neural network following the architecture outlined above is trained on the training dataset using the given feature selection. This CNN uses three convolution layers with a kernel size of 9, followed by two fully connected layers with 50 neurons each, and further implements all the regularization tricks outlined above. We took a CNN as initial experiments showed that a CNN generalizes well for various look-ahead values. This small CNN essentially serves as a proxy for the much larger models used later on. This CNN is then instantiated 16 times (each time with random weights following the Xavier method) and trained on the training set for 30 epochs using SGD. After each epoch, the validation dataset is fed through the network to compute the mean squared error.

After the final epoch, the elbow point of the loss curve of the training losses is computed. The elbow point's epoch typically indicates the onset of overfitting. The elbow point epoch was roughly at epoch eight most of the time. This training run's score value is the mean squared error on the validation dataset at the elbow point epoch + six epochs. Ultimately, the heuristic value for a feature selection is the median value of the score values from the 16 trials (one for each training run with a given feature selection). Computing the heuristic value based on the median of 16 runs ensures that the obtained heuristic value is representative of the actual expected performance of the model and is not too sensitive to training outliers. The random weight initialization can lead to widely varying results, as the model might get stuck on a saddle point of the loss manifold, start to oscillate due to the aggressive learning rate, or, worst case, even rapidly diverge.

Additionally, for each feature selection, the mean and standard deviation of the 16 score values are also computed and stored together with the heuristic value and the feature selection in a pickle file.

The second main deviation from greedy best-first search, apart from the estimation of the heuristic value, is the process by which new child nodes are generated and consequently their exploration order. Due to the computational complexity of estimating the heuristic value as outlined above, it is not feasible to simply generate all child nodes for a given parent node and compute the heuristic value for each of them, as there are 28 features in total, and each computation of the heuristic value estimate took approximately 20 seconds on our GPU.

Therefore, we had to generate nodes one by one instead of generating all at once, as usual for best-first search algorithms. The order of generation is crucial, as one wants to explore the most promising modes/feature selections first. To ensure this, the order of generation of child nodes needs to be selected carefully. We took an approach that is heavily inspired by two-stage least squares regression. First, we take the feature selection of the parent node (including the target variable feature), for which a child needs to be generated. Then, the target is regressed against the feature selection, and the residuals of this regression are computed. These residuals represent the remaining variance in the target that is unexplained by the feature selection. Then, the correlation between the remaining features that are not in the selection and the residuals is computed. Finally, the remaining features are ordered in descending absolute correlation. This determines the generation order of features for a parent node. To ensure an adequate exploration depth, all child nodes after the 10th child are pruned. These are often unpromising features.

This process of generating new children and computing their heuristic values is performed for as long as the algorithm has resources or all nodes have been generated. To prevent the algorithm from exploring excessive depths, we limited the depth to three or four, depending on the run configuration. In other words, the algorithm never selects more than the max depth number of features plus the target variable feature. The max depth and max number of generated children are also tunable parameters.

We ran the feature selection for each of our selected look-ahead values, i.e., one, four, 13, and 52 time steps, with indexed targets only. The resource time limit was set to 3'600 seconds (i.e., one hour), and the exploration depth was set to 4, i.e., a max of four selected features plus the target variable feature. The CNN network was instantiated and trained for 30 epochs and 16 trials as outlined above. The algorithm explored roughly 130 nodes within the resource time limit for each run.

Additionally, we ran the feature selection for a look-ahead of 52 again, while limiting the depth to three to encourage a broader exploration of the search space and increasing the resource time limit to three hours. The larger the look-ahead, the greater the benefit of feature selection presumably becomes, according to our hypothesis. Limiting the depth and increasing the time limit led to a much broader search space exploration, and over 300 generated nodes.

For the evaluation and ultimate feature selection, we stored all nodes for each run in a pickle file. For brevity, we will discuss only the most important findings here.

Generally, there does not seem to be a single optimal feature selection for any given look-ahead. There always seem to be quite a few reasonably good choices whose heuristic values are close together. Conversely, the heuristic value can become arbitrarily bad for the worst feature selections. Some features rarely appear in any feature selection because they are never generated during the node generation process. An example would be the growth and value index (_GR and _VA). This is reasonable, as those features are almost perfectly correlated with the target variable and fail to explain any remaining variance when the target variable is also a feature. Within the range of good choices, some feature selections experience larger variance than others. Overall, the standard deviations seem rather large given the computed means. Therefore, the standard deviation should be a strong tie-breaking criterion for the feature selection.

Unsurprisingly, the impact of feature selection seems to grow with an increasing look-ahead. For a look-ahead of one, selecting any feature besides the target variable seems to have little effect on the result, except for sometimes lowering the standard deviation. The same applies to a look-ahead of four.

The result of the feature selection is most interesting for a look-ahead of 52. This can be best shown by the graph below, which depicts the relative occurrence of the eight most occurring features within the 20 feature selections with the lowest heuristic value. The features "unemployment" and "systemic risk" (UN and YSS) are always picked for the best 20 feature selections. This heavily implies that these two features explain much of the remaining variance in the target variable. In addition, the "probability of recession" (NYF) is also present in almost half of the feature selections.

When looking at the table with the five best feature selections for a look-ahead, all selections in the top five use at least two of these three important features, and one selection even makes use of all three of them. Conversely, this is also the combination with the lowest standard deviation.

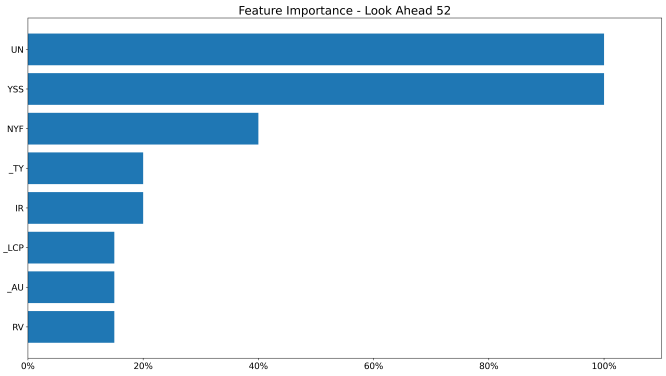


Fig. 1. Feature Importance - Look Ahead 52

TABLE I
FEATURE SELECTION LOOK AHEAD 52

| Feature Selection | Mean | Std. | Heuristic / Median |
|---|---|---|---|
| UN, YSS, DXY, _TY | 3.18 | 1.60 | 2.45 |
| UN, YSS, _AU, _TY | 3.47 | 2.49 | 2.63 |
| NYF, Rho, UN, YSS | 3.44 | 1.39 | 3.07 |
| UN, YSS, _LCP, _TY | 4.19 | 3.36 | 3.12 |
| NYF, UN, Y10, YSS | 4.00 | 2.76 | 3.19 |

Note that the chart and the table ignore the target variable feature MKT, which is always used for any feature selection as the root node feature is MKT.

Two interesting differences arise when comparing the feature selection with depth four to the broader and longer depth-limited search. Firstly, although the depth-limited search has similar features occurring within the top 20 feature selections, their relative occurrence with the top 20 is much more evenly distributed. Secondly, the heuristics and, especially, the standard deviations, are much higher. All the low standard deviation and low heuristic selections from the non-depth-limited search have four features (i.e., are at max depth), which suggests that fitting a model with a look-ahead of 52 is a rather complex endeavor, and models benefit from many features before significant overfitting occurs. For the look-aheads of one and four, we could not identify such behavior, implying the onset of overfitting at low depths (i.e., a low feature count). For a look-ahead of 13, we saw a mix of the two behaviors, with some feature selections of non-max-depth occurring in the best-scoring selections. These findings prove our hypothesis number two, which stated that the benefit of feature selection increased with the look-ahead.

Based on the above discussion and observations, we picked the following features in addition to the target variable feature MKT:

- **Look-ahead 1 Week**: CF
- **Look-ahead 4 Weeks**: CF, PE, RV, Rho
- **Look-ahead 13 Weeks**: CF, MOV, RR, _TY
- **Look-ahead 52 Weeks**: NYF, UN, YSS, Rho

Based on our observations, there seems to be little to no economic rationale behind the feature selections for the look-

ahead of one week and four weeks. The selected features are unable to decrease the heuristic value; conversely, they do not affect the model's performance on the unseen validation set meaningfully, and only manage to reduce the run-to-run variance (i.e., the standard deviation). For a look-ahead of 13 weeks, we can at least partially explain the feature selection with economic rationale. For instance, the total 10-year treasury return (_TY) and the implied bond-volatility (MOV) reflect the risk-free rate and its volatility, which should serve as a lower bound for an asset's expected return and volatility.

For a look-ahead of 52 weeks, all four picked features typically only have a long-term economic impact. The probability of a recession (NYF), the unemployment rate (UN), the systemic risk (YSS), and equity-bond correlation are all factors that have substantial long-term economic implications for the development of asset prices. Features with questionable long-term prediction power do not appear in the best-ranked feature selections for a look-ahead of 52 weeks. This shows that algorithmic automated feature selection can extract knowledge from features and implicitly apply economic rationale, in our case, the previously outlined short-selling theory.

### *D. Training*

When training the models, we explored various combinations of model types, look-ahead, and target types. We also evaluated three model architectures as outlined above: CNN, LSTM, and MLP.

As previously elaborated, we investigated the following look-aheads: one, four, 13, and 52 time-steps. Additionally, we looked at both indexed targets, where the model is supposed to predict the value of the index, and non-indexed targets, where the model is supposed to predict the cumulative relative return over the look-ahead time frame. This yields a total of 24 trained models (3 model types * 4 look-aheads * 2 target types). Note that the look-back window size was always fixed at 52 weeks for all 24 runs.

Our general training setup was identical for all 24 runs. First, a model following the architecture outlined before is instantiated with Xavier weight initialization. Next, the dataset is prepared according to the target type, and the input data is normalized from zero to one. Additionally, we select the features based on the results from the heuristic feature selection and the look-ahead value. Then, the dataset is split into a test, validation, and training set, and finally, the PyTorch dataloaders are instantiated. This process did not vary from run to run, except that the two target types differed in how they were instantiated in the datasets. Therefore, we created a custom PyTorch dataset class, the TimeSeriesDataset class, to cater to our needs for this project.

The model is then trained using stochastic gradient descent with a fixed and rather large batch size of 250 (to stabilize learning) and a learning rate that depends on the model and target type for adequate convergence. Additionally, we used momentum learning with a momentum of 0.9 to speed up convergence. We exclusively used the mean squared error loss function (MSE) for training. After each epoch, we computed the loss for both the training and validation sets and kept track of these values in an array. We typically trained the network for 100 epochs, as this was sufficient to run into overfitting territory where the network stops generalizing appropriately to unseen data. This is often evident by an increasing validation dataset loss. We applied a variant of early stopping for regularization purposes, which means that after the final epoch, the model with the lowest validation dataset loss is returned, as this is the one presumed to be the model generalizing best to unseen data out of all models trained over the epochs. This allows for higher learning rates and faster convergence.

However, this early stopping regularization can only be applied when the targets are indexed. For non-indexed targets, this is infeasible. Due to the Xavier weight initialization and the PReLU activation function, the output of the network before training, i.e., after epoch zero, is, no matter the input, always close to zero. But because a return of zero is often a good naive prediction (especially for short look-aheads) of the next target, the model with the best validation loss can sometimes be the one after epoch one, even though the network cannot have learned something meaningful yet. Therefore, we had to deactivate early stopping for these cases and adjust the learning rate accordingly.

When training the 24 models, we had to adjust the training setup based on the model type, target type, and look-ahead to ensure adequate convergence and prevent extreme overfitting. Some of these were:

- **Model Complexity**: With an increasing look-ahead, the prediction task becomes more difficult and more prone to underfitting; therefore, the model complexity needs to be increased. For instance, we increased the kernel size for the CNN from seven (for a look-ahead of one) all the way up to 21 for a look-ahead of 52. Other adjustments included increasing the number of channels for the CNN, increasing the size and number of hidden layers, or adding more LSTM blocks to the encoder structure of the LSTM network.
- **Adjusting Learning Rates**: Learning rates needed to be adjusted based on model type and target type. The LSTM network required much larger learning rates, presumably because back propagation through time led to vanishing gradients. Non-indexed targets also required much larger learning rates on average across all network types, as the derivative loss magnitude is much smaller than that for indexed targets, because the values themselves are much smaller.
- **Number of Epochs**: For the LSTM network and a look-ahead of 52, we had to increase the number from our default to 150 to achieve sufficient convergence.

Finally, when the model training is complete, the test and training datasets are fed through the model to compute the predictions for both datasets. We then evaluated the predictions for both datasets with MAE and MSE. Additionally, we

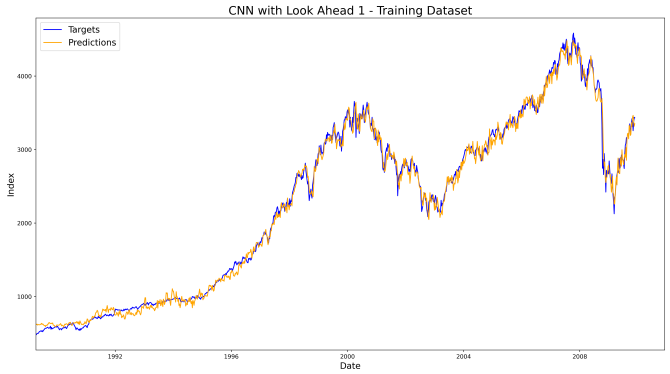


Fig. 2. CNN with Look Ahead 1 - Training Dataset

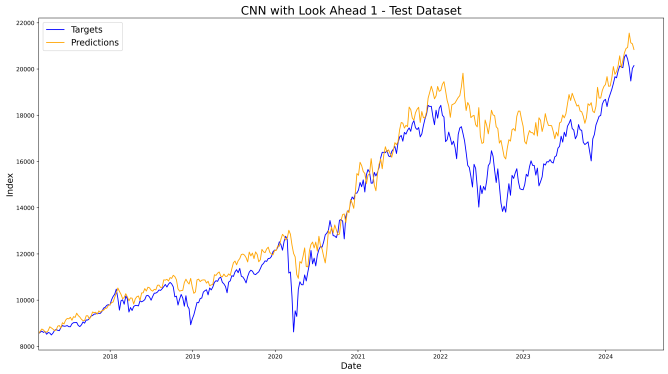


Fig. 3. CNN with Look Ahead 1 - Test Dataset

simulated a simple trading strategy. If the prediction was above the current price, we went long in the asset, and when the prediction was below, we went short in the asset. We then computed the expectation and standard deviation for trades given this strategy and our trained model for both the training and test dataset predictions. Based on these two values, we also computed the Sharpe ratio of our trading strategies under the assumption of a risk-free rate of zero percent. Finally, we computed the in-sample and out-of-sample information coefficient. All these computed values are then stored in a CSV file. The training PyTorch model's weights are also stored for potential future use.

### *E. Results*

We generated over 100 plots to evaluate the remaining four hypotheses that we formulated systematically. For every one of the 24 training runs, we have a plot of the loss curve during training and validation, a plot of the prediction on the training and test sets, and a probability distribution of the projected returns for non-indexed targets. For brevity, we only show a few hand-picked examples to convey the most important findings and answer our hypotheses. All the other visualizations can be found in the accompanying code repository under the output folder.

First, we look at the predictions of the CNN on the train and test sets for a look-ahead of one and indexed targets. It becomes immediately evident that we suffer from overfitting as the fit on the training set is near perfect and subpar on the test set. The model struggles with the index reaching values on the test set that it never saw during training, i.e., a strong train-test discrepancy. It would have been much better for the model to simply learn to predict the last seen value in the input target sequence, i.e., the value just preceding the predicted target. Generally speaking, the indexed target models perform worse than their non-indexed counterparts due to the strong train-test discrepancy.

When looking at the performance of the LSTM on the test set for a look-ahead of four and non-indexed targets, it becomes apparent that the model more or less learned to predict a return of zero. Therefore, the orange prediction line

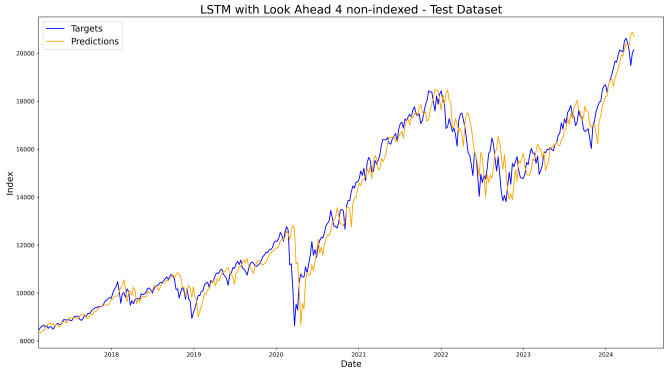


Fig. 4. LSTM with Look Ahead 4 non-indexed - Test Dataset

is effectively a right shift of the blue target line. The fit is near perfect on the training set, i.e., overfitting occurred. This behaviour of right-shifting and strong overfitting can be observed across all training runs that we performed for non-indexed targets, except for the ones with a look-ahead of 52. However, the fit on the test dataset is still much better than with indexed targets. Ultimately, this more or less confirms hypothesis number four.

Finally, we investigate the behavior of the MLP for a look-ahead of 52 and non-indexed targets on the training and test datasets. As with the previous two examples, the fit on the training is rather good, but fell apart completely on the test set, even though we were using non-indexed targets. Even though we applied many regularization techniques, the model overfitted on the training dataset to an extreme degree. This behavior is similar for the LSTM and the CNN for a look-ahead of 52 and non-indexed targets. For smaller look-aheads, such behavior is not observed. This confirms hypothesis one, that a larger look-ahead decreases the prediction quality.

Finally, when looking at all 24 runs, there seems to be no single best architecture. CNN, LSTM, and MLP all perform comparably well, with potentially a slight advantage for CNN and LSTM. Despite these slight differences, we cannot confirm hypothesis three as the model architecture does not improve model performance in a statistically significant way.

However, we have to emphasize that we suffer from over-

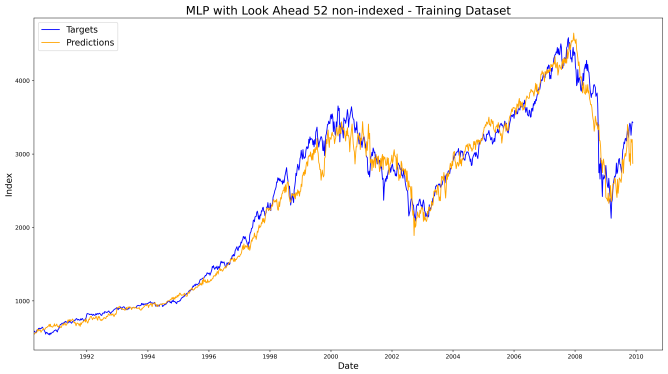


Fig. 5. MLP with Look Ahead 52 non-indexed - Training Dataset

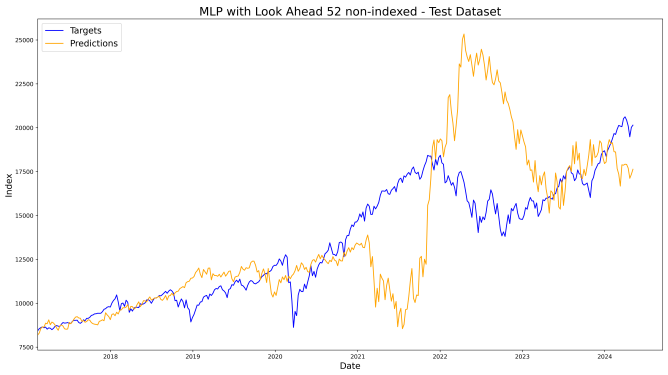


Fig. 6. MLP with Look Ahead 52 non-indexed - Test Dataset

fitting across almost all models. This is most evident when looking at the information coefficients for the training and test datasets. Often, the information coefficient for the training dataset is much larger than for the test dataset, implying poor generalization to unseen data.

Finally, we compare the Sharpe ratios of our trading strategies with the Sharpe ratio of the market (the S&P 500 index) to determine whether the models can beat the market. To do so, we picked the model with the best Sharpe ratio for each look-ahead and compared it to the market performance.

TABLE II
COMPARISON OF SHARPE RATIOS

| Look Ahead | Model Architecture | Indexed Targets | Sharpe Ratio Model | Sharpe Ratio Market |
|---|---|---|---|---|
| 1 | CNN | True | 0.12 | 0.10 |
| 4 | LSTM | False | 0.33 | 0.22 |
| 13 | MLP | False | 0.23 | 0.45 |
| 52 | CNN | False | 0.86 | 0.86 |

For a look-ahead of one and 52, even the best model performs just at the market level. For a look-ahead of 13, the model even performs significantly below the market level. Our best model can only beat the market by a small margin for a look-ahead of four. Additionally, all Sharpe ratios of our model are below any reasonable level for an active trading strategy. This confirms our fifth hypothesis. It is impossible to beat the market easily, and the weak market efficiency hypothesis seems to hold in our experiments.

## VI. Conclusion

We can conclude that it is, in fact, possible to somewhat accurately predict the S&P 500 index with machine learning models. However, these models often suffer from strong overfitting, even when strong regularization is applied. The trained models can also not beat the market by any meaningful degree. However, we also saw one instance where this was the case, suggesting that improved models or larger datasets that suffer from less overfitting and generalize better to unseen data might be able to beat the market after all.

## VII. Appendix

All our artifacts of work for this paper, including the data, code, and all plots and tables, can be found on the accompanying GitHub repository for this paper under the following URL: https://github.com/stsaxe/Investments-Machine-Learning-for-Finance-2025

## References

[1] J. Kim, H. Kim, H. Kim, D. Lee, and S. Yoon, "A comprehensive survey of deep learning for time series forecasting: architectural diversity and open challenges," *Artificial Intelligence Review*, vol. 58, no. 7, p. 216, 2025.

[2] White, "Economic prediction using neural networks: the case of ibm daily stock returns," in *IEEE 1988 International Conference on Neural Networks*, 1988, pp. 451–458 vol.2.

[3] G. S. Atsalakis and K. P. Valavanis, "Surveying stock market forecasting techniques – part ii: Soft computing methods," *Expert Systems with Applications*, vol. 36, no. 3, Part 2, pp. 5932–5941, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417408004417

[4] J. Fan and Y. Shen, "Stockmixer: A simple yet strong mlp-based architecture for stock price forecasting," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, pp. 8389–8397, 03 2024.

[5] O. B. Sezer and A. M. Ozbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Applied Soft Computing*, vol. 70, pp. 525–538, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494618302151

[6] W. Ye, J. Yang, and P. Chen, "Short-term stock price trend prediction with imaging high frequency limit order book data," *International Journal of Forecasting*, vol. 40, no. 3, pp. 1189–1205, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207023001073

[7] P. R. Pilla and R. Mekonen, "Forecasting sp 500 using lstm models," 2025. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.14759118

[8] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, vol. 270, no. 2, pp. 654–669, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221717310652

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[10] P. Brugière and G. Turinici, *Transformer for Time Series: An Application to the Samp;P500*. Springer Nature Switzerland, 2025, p. 511–528. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-84460-7_33

[11] M. Gordon, "Dividends, earnings, and stock prices." *The Review of Economics and Statistics*, vol. 41, no. 2, pp. 99–105, 1959.