Online Non-linear Prediction of Financial Time Series Patterns

Joel da Costa

Supervisor: A/Prof. T. Gebbie

A dissertation submitted in partial fulfilment of the requirement

for the Master of Science degree in Advanced Analytics,

School of Statistical Sciences,

University of Cape Town

2019

Abstract

We consider a mechanistic non-linear machine learning approach to learning signals in financial time series data. A modularised and decoupled algorithm framework is established and is proven on daily sampled closing time-series data for JSE equity markets. The input patterns are based on input data vectors of data windows pre-processed into a sequence of daily, weekly and monthly or quarterly sampled feature measurement changes (log feature fluctuations). The data processing is split into a batch processed step where features are learnt using a Stacked AutoEncoder (SAE) via unsupervised learning, and then both batch and online supervised learning are carried out on Feedforward Neural Networks (FNN) using these features. The FNN output is

a point prediction of measured time-series feature fluctuations (log differenced data) in the future (ex-post). Weight initializations for these networks are implemented with restricted Boltzmann machine pretraining, and variance based initializations. The validity of the FNN backtest results are shown under a rigorous assessment of backtest overfitting using both Combinatorially Symmetrical Cross Validation and Probabilistic and Deflated Sharpe Ratios. Results are further used to develop a view on the phenomenology of financial markets and the value of complex historical data under unstable dynamics.

Keywords: online learning, feedforward neural network, restricted Boltzmann machine, variance weight initialization, stacked autoencoder, pattern prediction, JSE, non-linear, financial time series, combinatorially symmetrical cross validation, backtest overfitting, deflated Sharpe ratio, probabilistic Sharpe ratio.

Dedicated to Emilee, for whom time passes anyway.

Declaration

I declare that: "Online Non-linear Prediction of Financial Time Series Patterns" is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.
Joel da Costa

Acknowledgements

First and foremost, I would like to thank Professor Gebbie for facilitating this project, and whose advice and guidance was invaluable in seeing this through. I would also like to thank Riaz Arbi and Nic Murphy for their efforts in data capturing and collection. Lastly, I would like to thank Arno, for his tireless patience.

Contents

List of Figures

List of Tables

List of Acronyms

I

BP Backpropagation
CD1 Contrastive Divergence 1
CLR Cyclical Learning Rate
CSCV Combinatorially Symmetric Cross-Validation
DBN Deep Belief Network
DSR Deflated Sharpe Ratio
FDR False Discovery Rate
FNN Feedforward Neural Network
FST False Strategy Theorem
FWER Family Wise Error Rate
GBM Geometric Brownian Motion
IID Independently and Identically Distributed
IS In Sample
LSTM Long Short Term Memory
MCS Model Confidence Set
MMS Money Management Strategy
MSE Mean Squared Error
NN Neural Network
OHLC Open High Low Close
OGD Online Gradient Descent
ONC Optimal Number Clusters
OOS Out of Sample
PL Profit and Loss
PBO Probability of Backtest Overfitting
PCA Principal Components Analysis
PSR Probabilistic Sharpe Ratio
RBM Restricted Boltzmann Machine
RNN Recurrent Neural Network
SAE Stacked AutoEncoder
(S)DAE (Stacked) Denoising AutoEncoder
SGD Stochastic Gradient Descent
SL Supervised Learning

SR Sharpe Ratio
SVM Support Vector Machine
SVR Support Vector Regression
UL Unsupervised Learning

1 Introduction

In this paper, we present a novel framework with a non-linear and mechanistic approach to asset price fluctuation prediction in financial markets. We set out to test two primary hypotheses: whether such an approach can be profitable while incurring a low chance of backtest overfitting, and whether feature selection on financial time series is possible and effective. In doing so, we also explore the complexity and unstable dynamics of financial time series.

We found compelling results for both our hypotheses. A large set of neural networks were trained, and had profitable results verified through the use of sophisticated financial strategy assessments. We also found that feature selection was effective, despite the dynamic nature of financial data. Further, we saw the results from numerous experiments emphasise that while historical financial data has value, the real benefit to predictive modelling lies in recent cross sectional data.

The framework developed combined several concepts to achieve this: the training of deep neural networks, feature selection through the use of Stacked AutoEncoders and unsupervised learning, supervised learning through both batch and online methods for asset price fluctuation prediction and current assessment techniques such as Combinatorially Symmetric Cross-Validation (CSCV) and the Deflated Sharpe Ratio (DSR). In doing so, a modular process was constructed, where individual components are decoupled and can be configured as necessary. This delivers a simplest high complexity framework which allows further adaptation for the exploration of a high dimensional solution space. The framework was developed under a scientific process and data implementation, such that all results presented are fully reproducible using the configurations noted in Appendix 11.5.

As Bailey et al. have noted, and which is discussed more fully in Section 2.6, backtest overfitting for trading strategies has become problematically widespread in financial literature [8]. Neural networks, in their capacity as universal function approximators with few model limitations, offer an effective and appropriate methodology to generate predictions in an environment as complex as financial markets. However, the increased complexity and nature of backtest overfitting leaves traditional validation methods such as hold-out or cross validation falling short. The framework presented investigates how more rigorous validation techniques can be applied to deep learning models in order to avoid such overfitting. Further validation takes place in assessing the potential profitability of the model in a live market.

The literature review in Section 2 has a fuller discussion of work that precedes the various techniques which have been implemented. A brief introduction to technical analysis in the financial sector is discussed and forms the basis

for proposing and using the technical analysis methods throughout this paper (Section 2.1). The usage and history of Neural Networks is discussed, covering the key areas of development which led to widespread use of deep learning models (Section 2.2). The efficacy of Stacked AutoEncoders in data reduction for complex systems has led them to be pivotal tools in deep learning models, which is discussed in Section 2.3. Online learning methods are discussed in Section 2.4 with a coverage of both the historical basis as well as the developments which have resulted in algorithmic improvements. The section finishes off with discussing the impact of backtest overfitting and results validation. The Probability of Backtest Overfitting (PBO) and Deflated Sharpe Ratio (DSR), as developed by Bailey et al. [8] and Lopez de Prado and Lewis [90] respectively, are presented as validation techniques for the framework as a whole (Section 2.6).

Section 3 provides the details of data processing that takes place prior to training, discussing both the scaling and fluctuation aggregation techniques used. Geometric Brownian motion has been used for the generation of synthetic data, and the section includes a discussion of its characteristics and usage in financial prediction models.

Section 4 provides more in-depth details on the algorithms and structures used to implement the framework. The structure of feedforward neural networks is discussed, including how they are trained using the backpropagation algorithm, and how that can be applied in a stochastic descent framework (Section 4.2). This section also provides details for how network weight initializations can impact performance, and how this can be affected through RBM pretraining (as per Section 4.3) or through variance based techniques (Section 4.5). This includes the structures and training techniques used for the Stacked AutoEncoders (Section 4.4). The Money Management Strategy (MMS) constitutes the implementation of trading decisions and actions based on the network model predictions and is detailed in Section 4.6. The CSCV techniques suggested by Bailey et al. [8] were implemented and used to derive a PBO figure for the full training and testing processes, as detailed in Section 4.7. Finally, the implementation of the Optimal Number of Clusters (ONC) and DSR calculations are covered in Section 4.8.

Where Section 4 discusses the functional implementations of each module of the framework, Section 5 is concerned with how these modules fit together to form an end to end system. The framework detailed considers all steps from the data preparation to the output of a PBO or DSR figure. This includes discussions of various considerations for individual aspects of the framework, as well as decisions around the combinations of these parts with their justifications and relevant advantages or disadvantages. In light of the configurable and modular system developed, potential alternatives and how they might be used are also discussed here.

Section 6 discusses the implementation tools chosen and the libraries created in the process of developing the software for the framework. The libraries offer a generic set of tools with highly configurable parameters in order to train and assess neural networks, including an extensive collection of diagnostic visualizations.

Section 7 provides a brief overview of the datasets used for the trials run, including both the synthetic and actual datasets. Visualisations for the individual asset prices are also included here.

Finally, Section 8 discusses the full set of results we found from the experiments, and offers several key takeaways. In Section 8.2, we discuss the nature of prediction strategies learnt, and show they were influenced heavily by the data horizon choices. Further, we show that SAE based feature selection is both possible and effective, though with results that were also subject to data horizons and recency. In Section 8.3, we show that historical financial data is of limited use in financial model training, and predictions are served best by a recent cross sectional view of information. Online learning, and consequently network initialization, is shown to be of high importance, the results for which are covered in Section 8.4. The complexity of financial time series, and how it impacts both supervised and unsupervised learning, is discussed in Section 8.6, with comparisons to synthetic data being made in Sections 8.5 and 8.11. The MMS returns are presented in Section 8.8, and validated through the PBO and DSR methods in Sections 8.9 and 8.10, showing that the framework was able to deliver a profitable and verifiable set of strategies.

2 Literature Review

2.1 Technical Analysis

Technical analysis is a financial analytical practice that makes use of past price data in order to identify market structures, as well as forecast future price movements. The techniques are typically objective methodologies which rely solely on past market data (price and volume). They stand in contrast to fundamental analysis, where experts will consider a companies operations, management and future prospects in order to arrive at an evaluation. The basis of much technical analysis, originally developed through Dow Theory, is the belief that stock market prices will move directionally (upwards, downwards or sideways), and that past movements can be used to determine these trends [97].

One of the primary methods in technical analysis is the use of charts in order to identify price patterns. These charts will be produced using the available market data and a known design, such as the popular candle-bar plot, which can then be compared to historical data to match it to a particular pattern. These patterns are thus indicative that the stock is likely to take on a particular price trend, or is in a particular state [97, 2]. There is a certain amount of controversy around technical analysis, where many argue that it is contradictory to the random walk and weak form efficient market hypotheses, and as such is not valuable or useful [62]. The argument against this, is that technical analysis does not rely on past action to predict the the future, but is rather a measure of current trading, and how the market has reacted after similar patterns have occurred in the past [78]. Further, even if the analysis is unable to effectively forecast future price trends, it can still be useful to exploit trading opportunities in the market [112].

With the advent of processing power becoming cheaply available, there has been an increase in research to adapt computing techniques to technical analysis. The breadth and superhuman speed in which systems are able to perform technical analysis far outstrips what was possible before, and as such they have become the focus of competitive performance for many market participants [77]. To this end, there has been much research to apply machine learning algorithms to perform pattern recognition on stock price movements.

Financial markets have been shown to be complex and adaptive systems, where the effects of interaction between participants can be highly non-linear [4]. Complex and dynamic systems such as these may often exist at the 'order-disorder border' - they will generate certain non-random patterns and internal organisation, which can be assessed and identified, however they will also exhibit a certain amount of randomness in their behaviours, or 'chaos' [20]. Further, it has been shown that there is enough signal to reconstruct the phase space of chaotic systems from singular observations, encouraging that we might

be able to perform effective technical analysis in the context of financial markets [102, 118]. As a result, trying to identify these patterns and structures is a simultaneously reasonable and notoriously difficult goal. While it is often clear in hindsight that the patterns exist, the amount of noise and non-linearity in the system can make prediction challenging. Fittingly then, neural networks have become a popular choice for modelling within financial markets. Due to their structure, they are able to learn non-linear interactions between their inputs and outputs, with even early research showing their ability to achieve statistically significant results, which lends weight to the argument against the efficient market hypothesis [116].

The work presented here fits into the growing body of work which considers mechanistic and brute-force approaches of applying machine learning models to financial market data. In doing so, the financial market complexity in terms of non-linearity, noise and stability are highlighted through both the successes and challenges found in training these machine learning models. These difficult dynamics, and their notable difference when compared to other popular areas of ML research - which are often around Independently and Identically Distributed (IID) datasets - present fundamental problems to be explored, both in terms of prediction efficacy as well as validation. Gebbie and Loonat [58] have explored the application of online learning models in this space in the South African market, showing that direct online learning approaches would be able to identify and exploit trading opportunities on the JSE through assessment of Open High Low Close (OHLC) data. Murphy and Gebbie [98] later explore the use of online learning as applied to optimizing parameters which apply to maximising wealth trading zero-cost portfolio strategies, also in the JSE. They consider validation in terms of statistical arbitrage, as well the non-parametric Probability of Backtest Overfitting (PBO). Similar ideas are explored here, in the use of mechanistic batch and online learning in order to identify patterns in JSE closing data, as well as the use of non-parametric validation techniques for return assessment.

2.2 Neural Networks

A neural network is a learning model which was originally inspired by the biological mechanisms of neurons in brains. The structure is essentially that of a network system, with connected nodes and edges, or ‘neurons’ and ‘weights’. The neurons are based on the same idea as synapses as seen in the brain - where a build-up of input results in a ‘firing’ of output. The input here is determined by the model’s input (real numbers typically), and processed through the weights and activation functions of the neuron. The activation function then results in an output value either at an intermediate level, or as the model’s final output. The system learns by considering input samples sequentially, and adjusting the weights between edges to result in more accurate outputs. Output types may either be classification or regression values.

Structured neural networks that learn have been around since the second half of the 21st century [110], though they have gone through several cycles of popularity. The first versions tended to be very simple with one layer of hidden neurons [76]. It was only later, through the application of the backpropagation algorithm, that they started to become more practical and popular [126].

With the rise in popularity, many different network formations were developed and suggested. One of the initial suggestions was the conceptually simple Feedforward Neural Network (FNN) as described above - an acyclic graph where inputs are processed in a single direction until the output is reached. The other notable earlier model was the Recurrent Neural Network (RNN), which has a cyclic graph instead - this results in a more powerful computational system than the standard FNN, which was shown to be effective quite early on [115]. The Long Short-Term Memory (LSTM) network was another that used recurrent dynamics, though at a neuron level, in that the neuron is responsible for remembering values for an arbitrary time period [72]. Convolutional Neural Networks (CNN) have a non-recurrent structure, but implement separate pooling layers of neurons which consider the adjacent input values for each feature (e.g. pixels next to each other). These have been shown to be incredibly effective at tasks such as image recognition.

There are three primary learning paradigms used in neural network training - Supervised Learning (SL), where the network is trained on inputs with known outputs; Unsupervised Learning (UL), where the network is trained to identify unknown structures as an output; and Reinforcement Learning (RL), where environmental reactions are used as inputs to train a network for certain outputs [110]. While all of these configurations and paradigms have their benefits and uses, this paper will focus on FNNs and RNNs, trained through supervised and unsupervised learning.

2.2.1 Training and Backpropagation

Historically, the crux of neural networks' popularity has often been based on the development of novel training methodologies, and how they have increased performance. The Backpropagation (BP) algorithm (as defined in Section 4.2.3) has played a pivotal part: while neural network (or 'perceptron') models were around prior, they were largely deemed ineffective in comparison to other available models [96]. It was only during the 1980's that the backpropagation algorithm was applied to neural networks, and the field started to gain in popularity again [82, 125].

Rumelhart et al. [108] showed that the backpropagation algorithm, as applied in neural networks, resulted in useful feature representations occurring in hidden layers and the empirical success that resulted thereof. Shortly after, LeCun et al. [83] applied the BP algorithm to CNNs with adaptive connections.

They were able to show impressive performance for the time in classifying handwritten images, with the images as a direct input (rather than a feature vector).

While many improvements were made during this time via gradient descent modifications, the models were typically of a shallow nature due to problems encountered trying to train deeper networks. Early experiments with deep networks resulted in poor performance due to what is now widely known as the problem of either 'vanishing' or 'exploding' gradients [103]. As more layers are added to the network, the backpropagation algorithm (with typical activation function neurons) results in error signals that either shrink or grow out of bounds at an exponential rate. One of the first suggested solutions to the problem is to perform pretraining on the network through unsupervised learning [110], as discussed in Section 2.3.2. Variance based weight initialization techniques are another effective solution, as discussed in Section 2.2.4.

There were initial concerns that the backpropagation algorithm with a simple gradient descent, as applied to high dimensional neural networks, would result in the network weights being trapped in local minima (e.g. where no small changes to the configuration would reduce the average error rate) [85]. However, empirically, this tends not to so problematic, and large networks usually reach solutions of equitable performance. More recent research has shown that the solution spaces largely consist of many saddle points, each with varying gradients of the features, but which also tend to have similar values of the objective function [48]. Ge et al. [57] have also shown that it is possible to escape saddle points and offer a guaranteed global convergence in certain non-convex problems.

2.2.2 Activation Functions

One of the upfront configuration choices necessary is the activation function, which allows the mapping of input to output at the neuron level. There have been many suggestions and experiments with different functions, though there are some common features amongst functions which make them appropriate: Non-linearity allows for neural networks to operate as universal approximators, as shown in [73]; continuous differentiability allows for the use of gradient descent; and whether the function is monotonic has been shown to indicate whether the solution can be guaranteed to have a unique periodic solution [128]. Lastly, the range of the function (infinite or finite) can impact both the stability and efficiency of the training.

Some of the most frequently used functions are the sigmoid, tanh, ReLU and softsign (the equations for which are detailed in Section 4.2.2). There have been various studies showing the efficacy of the different activations under varying initialization (or pretraining) for weights. Glorot and Bengio [59]

noted that the typical sigmoid and tanh functions performed poorly with standard minimization, and result in slower convergence and worse minima. They continue to show that softsign with a non-standard initialization resulted in quicker convergence. Further research by Glorot et al. [60] found that the rectifier (ReLU) functions were more effective in deep sparse networks compared to the tanh function.

2.2.3 Deep Learning

As noted above, most of the earlier work using neural networks relied on shallow models with few layers. However, a resurgence in interest occurred in 2006 after several papers demonstrated the efficacy of unsupervised pretraining of networks prior to supervised training. The effect was substantial enough to allow much deeper layered networks to be trained than previously possible [12, 70].

The essential point behind the unsupervised learning was to initialize the weights in the network to sensible values in light of the problem context. The methods used trained each layer to be able to reconstruct the model of the features in the layer below (to a varying degree of accuracy). Sequentially pretraining and combining layers like this, the process generated a deep neural network with appropriate weights. Once done, a final output layer was added and the entire network could then be fine-tuned through backpropagation without suffering such performance degradation through vanishing or exploding gradients [70, 106, 68]. This is expanded on further in Section 2.3.2. Roux and Bengio [107] were able to show that within the DBNs produced by Hinton et al. [70], adding hidden nodes resulted in strictly improved modelling capabilities, and they suggested that increasing the number of layers is likely to result in increased representational ability (subject to efficacy of previous layers), thus establishing the argument for deep networks in theory as well as practice.

FNNs were shown to be effective in modelling high dimensional data prior to the breakthroughs in deep networks [11], so it follows that the deep networks were shown to be extremely effective in high dimensional data classification. Early implementations resulted in increased efficacy in handwriting recognition, as well as pedestrian recognition [113]. When it came to data types such as sound and images, CNNs were implemented on several occasions with record breaking model performances in recognition, notably in ImageNet and WaveNet [79, 100].

As more research into deep networks was conducted, it became apparent that with large enough datasets, the layerwise pretraining of networks was not actually necessary to achieve high performance standards [79, 60, 18]. When training for long enough, it was reported that the pretraining offered little to no benefit, though these models were typically using datasets far larger

than were attempted before (as a result of hardware improvements enabling as much). While these results did require that certain attention was paid to the initialization, as well as the use of non-linear activation units, it did suggest that pretraining largely acted as a prior which may not be necessary if large enough labelled datasets are available [13]. Naturally, pretraining was still implemented to prevent overfitting in smaller datasets.

2.2.4 Weight Initialization Improvements

One of the more critical innovations to allow effective training of deep networks without using pretraining was the development of more sophisticated techniques for weight initialization. One of the first of these was presented by Glorot and Bengio [59] for use in sigmoid based networks, and is commonly referred to as Xavier/Glorot initialization. The technique is layer specific and based on a linear activation hypothesis. It is designed such that the initial weights will maintain the same variance for input for information that is passed backwards, as in accordance with the nature of the activation function. While the assumption of linearity is not always applicable, they point out that at the start of the learning process it is typically the area of the activations where the gradient is close to 1 which is being explored, thus initially approximating a linear effect. The outcome is a technique that increases learning efficacy and optima found [59].

He et al. [66] extend the same methodology for the ReLU activation, once again based on the linearity hypothesis around the relevant activation function (Parametric Rectified Linear Units in this case). ReLU activations are computationally inexpensive and do not suffer from learning slowdown. The combination of these attributes and effective weight initialization techniques removed the need for pretraining and produced a seminal FNN training framework. He et al. [66] were able to achieve state of the art performance and produced the best known error rate at the time on the ImageNet dataset. The efficacy of these techniques has established them as norms in the training of deep neural networks.

2.3 Stacked AutoEncoders

2.3.1 High Dimensional Data Reduction

Machine learning techniques have been shown to be extremely effective at modelling non-linear inputs to outputs - neural networks have been shown to be universal function approximators in this regard [73]. More traditional statistical models will typically process the available feature data to select the most significant features to be used in the model once it is defined. This is evident in a process such as subset selection [109]. Machine learning techniques are no different in this regard, and feature data will typically be transformed

to smaller observations of more significance prior to being used as input to a model.

Financial data, in line with the complex and dynamic system that it represents, is often of a very high dimensional nature. This high dimensionality creates opportunities through more sophisticated analysis, but also introduces the curses of dimensionality [49]. The increased dimensionality can result in higher processing complexities when needing to do basic tasks such as estimating a covariance matrix (a commonplace necessity in finance). Also, it does increase the risk of incorrect assumptions based on spurious variable collinearity [54]. Noise accumulation in high dimensional data can create further problems, resulting in problems performing variable selection and ultimately having a large impact on classification and regression models [53].

Time series data can introduce its own set of challenges: there is often not enough data available to understand and predict the process [52]; the time variable dependence creates complexity in how much past data to consider at any point; and the data is typically non-stationary [81]. Thus, high dimensional time series data (which many financial problems focus on), require careful consideration on how to handle their inputs and analysis.

Deep learning techniques are a natural choice in this context, and much research has been done to show their (varying) efficacy on time series data. The most successful of these models have been ones which modify deep learning techniques to incorporate the temporal aspect of the data (e.g. Conditional restricted Boltzmann machines or Recurrent Neural Networks) and those which have performed feature selection processes (e.g. AutoEncoders) [81].

Two of the seminal pieces of research that led to the resurgence in machine learning and deep learning were the algorithms for training deep belief networks [70], as well as the usage of Stacked AutoEncoders [106, 12].

2.3.2 Deep Belief Networks

AutoEncoders were suggested by Hinton and Salakhutdinov [68] as a method of transforming high dimensional data to lower dimensional input vectors. This can alleviate some of the training problems noted, and increase performance of deep belief networks.

One of the more popular techniques for dimension reduction is Principal Components Analysis (PCA). PCA uses linear algebra to find the directions of greatest variance, and represent the observation samples features along each of these directions, thus maximising the variational representation. Hinton et al. show that AutoEncoders are a non-linear generalization of PCA. The structure and training algorithms of the AutoEncoder show it to be a specialised neural network - there is a multilayer encoder network which is able to trans-

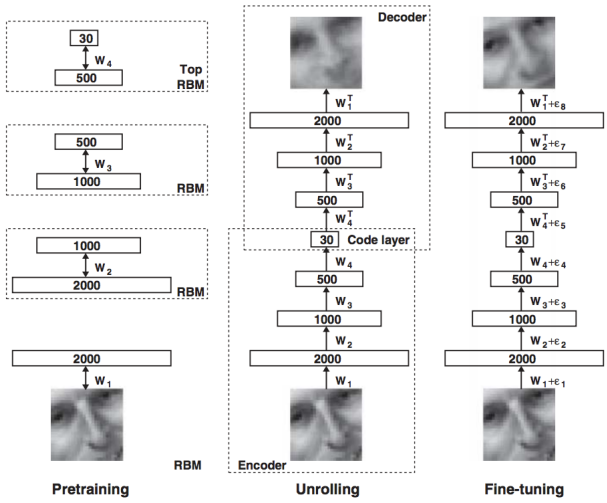


Figure 1: The AutoEncoder training steps [68].

form to a lower dimension, and a symmetrical decoder network to recover the data from the code as represented in Figure 1. As with neural networks, the gradient weights can be trained through the feedforward and backpropagation algorithms.

The primary challenge presented here was the initial weighting of the networks - with large initial weights the AutoEncoder will often find a poor local minima, and with small initial weights the gradients are too small to effectively train deep layered networks. The critical suggestion by Hinton [69] was to used layered restricted Boltzmann machines (RBM) in order to initialise the weights. For each layer of the desired AutoEncoder, a RBM is formed and trained with the previous layer (or RBM). Once all the layers have been trained in this way, they are mirrored to form the decoder network. This then forms the initial weights to be fine tuned further, as per the 'Fine-tuning' step in Figure 1. They found the deep AutoEncoder networks were significantly more effective than PCA or shallow AutoEncoders on multiple datasets.

2.3.3 Stacked Denoising AutoEncoders

The second important piece of work was the development of a Denoising AutoEncoder (DAE), by Vincent et al. [122]. One of the problems identified in the DBN model described in Section 2.3.2 (and those similar), is that if the encoder dimensions were too high, it is likely that the encoder would learn a trivial encoding - essentially creating a "copy the input" model. The one way of tackling this issue is to constrain the representation with bottlenecks and sparse AutoEncoder layers, which can be seen in Figure 1.

Vincent et al. [122] explore a very different approach to the problem, which was to develop an implementation of AutoEncoder which focused on partially

corrupting the input, and so force the network to 'denoise' it. The theory here is based on two ideas - the first, is that a higher dimensional representation should be robust to partial corruption of the input data; and the second is that the denoising process will force model focus to shift to extracting useful features from the input.

The algorithms and structures are largely the same as described for DBNs above. The key difference is that the model is trained to reconstruct the original input, but only using a corrupted version of the input (where noise has been added to it). This forces the model to learn smarter feature mappings and extractions. The DAE suggested is a stochastic variant of the AutoEncoder. It has the benefit of being able to implement higher dimensional representations without risking training of a trivial identity mapping. Notably, in the Stacked Denoising AutoEncoder (SDAE) formation, only the initial input is corrupted (as opposed to the input from layer to layer). It was shown that the SDAE model outperformed previous AE and DBN networks on numerous benchmark datasets [122] .

2.3.4 Pretraining

The methods described above follow a similar approach: greedy layer-wise unsupervised pretraining in order to determine initial weights, followed by supervised fine tuning to arrive at the final model. It is shown numerous times that the pretraining process can result in significant performance gains [122]. However, it is not immediately apparent, given the nature of backpropagation algorithms, why this is the case. Erhan et al. [51] performed extensive empirical simulations in order to suggest an explanation for the mechanism of pretraining.

While their results were not entirely conclusive, they did lend themselves to a reasonable hypothesis: the unsupervised pretraining results in a form of regularization on the model. Variance is minimized, and the bias introduced acts as a prior to direct the model configuration towards a sample space that is effective for the unsupervised learning generalization optimisations.

2.3.5 Financial Time Series Applications

The AutoEncoder papers reviewed so far in this section derive their results primarily from classification problems, and so do not necessarily account for the problems involved with time series as described in Section 2.3.1. In a review of unsupervised feature learning and deep learning methods on time series, Langkvist et al. [81] reported that the use of AutoEncoders, either as a technique in themselves or as an auxiliary technique to models such as convolutional neural networks, were able to offer performance increases in areas such as video analysis, motion capture data and bacteria identification.

There have of course also been successful applications of stacked AutoEncoders and deep learning models in finance. Takeuchi [119] performed some earlier work showing the use of AutoEncoders when applied to a momentum trading strategy. They implemented an RBM pre-trained DBN, as per Section 2.3.2, and assessed the networks classification performance for ordinary shares on NYSE, AMEX and Nasdaq. This showed that using a DBN network resulted in significant performance increases compared to the standard momentum strategy.

Zhao et al. [131] used SDAEs and combined them with the bootstrap aggregation ensemble method ('bagging') in a study of predicting the crude oil price. They compared the proposed model to a variety of benchmarks, including standard SAE, bagged and standard feedforward networks and Support Vector Regression (SVR) models. The results indicated that the SAE models were more accurate, with the bagged SAE model performing the best, though at a significant increase in computational costs in comparison to standard SAE.

While much of the financial literature has focused on the use of RBM based models, AutoEncoders and SAEs have also become more widespread in performing feature reduction. Troiano et al. [120] specifically investigate the use of different feature reduction models for trend prediction in finance. In line with being primarily interested in the effect of feature reduction techniques, rather than the classification performance itself, only a Support Vector Machine (SVM) model was used to test results. Using various periods from historical S&P500 data, they were able to show that the AutoEncoder outperformed the RBM model significantly in numerous accuracy measures, and was able to do so at a fraction of the training time.

Bao et al. [9] note that the research has been lacking with regards to whether SAEs should be used for financial prediction models or not. They suggest a novel model which combines Wavelet Transformation, SAEs and a LSTM network. Using data from several financial exchanges (considering a range of developed and undeveloped markets), they assess the model's applicability to OHLC prediction. Comparing the model to configurations without the SAE layers, and a RNN model as benchmark, they showed that the inclusion of SAEs resulted in less volatility and greater accuracy, which in turn offered higher profitabilities in a buy-and-hold trading strategy.

More novel AutoEncoder applications have also been attempted, with Hsu [74] suggesting the use of a Recurrent AutoEncoder for multidimensional time series prediction. There is a pattern through the literature that the use of AutoEncoders and Stacked AutoEncoders, both by themselves and when used as an assisting technique, result in more accurate prediction results and less computationally expensive training.

2.4 Online Learning Algorithms and Gradient Descent

Batch machine learning algorithms operate under the assumption that the full dataset has been collected and that the amount of training data for the model is both finite and immediately available. However, as the rate of information growth continues in an exponential fashion, there are many areas where the expected training data for the model will continue to increase. In these cases it would be disadvantageous to go through the full training and validation process again in order to incorporate the newly available data.

Online algorithms are designed to offset these issues by adjusting the batch training technique to repetitively draw on single samples from the data on which the model's parameters can be adjusted. The benefit is that they are able to quickly process a large number of observations and readjust the model. The downfall is that they are not always able to optimize the cost function to the same extent as offline batch algorithms [1].

Bottou and Lecun [14] argue that as the size of the dataset grows significantly, online algorithm's advantages result in them outperforming offline models despite any initial drawbacks. Previous research had shown that online algorithms typically perform as fast as batch algorithms during the 'search' phase of parameter optimization, but that 'final' phase convergence tended to fluctuate around the optima due to the noise present in single sample gradients [84, 15]. Bottou and Lecun [14] showed that it is actually more practical to consider the convergence towards the parameters of the optima, rather than the optima itself (as defined by the cost function) - the difference between the learning speed and optimization speed, respectively. Online learning methods are thus well suited to financial market modelling using neural networks. They allow effective and efficient incremental updates as more recent (and relevant) data becomes available. Further, the increased learning speed over optima convergence makes them a fitting choice when data is non-IID and constantly changing.

2.5 Gradient Learning Improvements

2.5.1 Gradient Adjustments and Regularization

One of the earlier improvements to convergence rates was the Momentum algorithm, as developed by Tseng [121]. As noted, stochastic descent often introduces significant oscillation around an optima, which slows down convergence. Momentum reduces this by decreasing movement in directions of high curvature, and increasing movement towards directions consistent with previous gradients (this is achieved through combining gradient movements in opposite directions).

There have been several attempts to introduce effective regularization into the SGD process. Bartlett et al. [10] presented Adaptive Online Gradient Descent, which implements an adaptive step size through a penalty on the learning rate, which was shown to be nearly optimal in a strong sense. Langford et al. demonstrated a variation named Truncated Gradient, which introduced an enforced weight sparsity parameter. The weight sparsity is able to achieve equitable effects to "L1" regularization (similar to Lasso Regression). They were able to show that implementation performed effective feature reduction, while having little effect on performance [80]. Other approaches, such as AdaGrad, aim to improve the robustness of gradient training by adjusting the updates to parameters according to frequency - e.g. larger updates to infrequent parameters, and smaller updates to frequent parameters [50, 130].

2.5.2 Dropout

An instrumental improvement to the backpropagation algorithm was the "dropout" technique, as suggested by Hinton et al. [71]. Training of large networks on small datasets can often result in overfitting and poor results on out of sample data. Dropout helps resolve this by randomly excluding a certain percentage of feature detectors on each training iteration. The effect is to stop co-adaptations of feature detectors. By rather training each neuron in a wide variety of internal configurations, it forces them to take on more usefully generalizable characteristics (it was noted that this is not a dissimilar technique to ensemble methods, or bagging). The authors were able to show that the method results in significant improvements on benchmark data sets (e.g. MNIST, CIFAR-10), and that a simpler model using dropout was able to achieve near comparable performance for the ImageNet dataset.

Goodfellow et al. [61] used the dropout technique as the basis for their "maxout" activation function technique. Maxout leverages and improves on dropout's fast optimisation and accuracy through averaging characteristics. The maxout model was shown to achieve state of the art performance on benchmark datasets, as well as have a strong theoretical grounding. Further work was done by Wang and Manning [123], which improved on the dropout (and potentially maxout) techniques through fast sampling, resulting in an order of magnitude speed-up in training.

2.5.3 Learning Rate Schedules

Several approaches look towards learning rate adjustment schedules rather than strategies to alter the weight update. A constant learning rate often suffers from one of two problems: if set too high, it can cause divergent behaviour in the loss function; though if set too low, it can result in slow learning or an

inability to escape saddle points effectively. Finding an optimal learning rate requires some degree of testing, though even then a singular value may fail to achieve the same degree of efficacy as a range of values explored throughout the solution space.

Learning rate scheduling has been proposed as a solution to this problem. These methods implement a repetitive cycle for the learning rate, such that it is set through a range of values between a minima and maxima, being adjusted slightly with each new epoch. Smith [117] suggests using the Cyclical Learning Rate (CLR) to achieve this, and points out that part of the benefit in learning rate schedules is the ability to jump out of sharp optima points which may not generalise well to unseen data. It is shown that this implementation can have significant performance effects in reaching either the same optima in fewer epochs or a better overall optima. This includes when used in conjunction with other learning optimizations noted in Section 2.5. These results were shown against standard datasets such as CIFAR-10, CIFAR-100 and ImageNet [117].

Loshchilov and Hutter [91] expand on this and present Stochastic Gradient Descent with Warm Restarts. The approach is implemented similarly, but rather follows an asymmetric cycle from a maximum to a minimum, starting once again at the maximum once the set number of epochs has passed. They too note the increased performance on CIFAR-10 and CIFAR-100 datasets in reaching optima quicker, as well as the efficacy of learning rate schedules even without restarts.

2.6 Backtesting and Model Validation

Financial academic literature is currently facing a problem in terms of validation and verification of results. Trading strategy profitability has typically been proven using historical simulations, or "backtests". However, the recent advances in technology and algorithms available to construct these strategies have resulted in researchers being able to test a number of variations that is factors above what used to be possible. This has made it increasingly difficult to control for spurious results. The problem is so extensive that some meta-research papers suggest that most published research findings are false [75].

The standard way of implementing backtests is to split the data into two portions: an In Sample (IS) portion which is used to train the model, and an Out of Sample (OOS) portion which is used to test the model and validate results. If millions of different model configurations are tested, then it is only a matter of time before a false positive result occurs which shows high performance both IS and OOS (i.e. overfitting) [8, 94].

The nature of financial data makes it difficult to resolve these issues effectively. There is a low signal-to-noise ratio in a dynamic and adaptive system,

and only one true data sequence. Traditional hypothesis testing frameworks (e.g. Neyman-Pearson) are not sufficient in this context, and more sophisticated techniques are necessary.

Overfitting is not a novel issue. This has been tackled in various literature areas, including machine learning. However, in that context, the frameworks are often not suited to the buy/sell with random frequency structure of investment strategies. They also do not account for overfitting outside of the output parameters, or take into consideration the number of trials attempted.

One of the common approaches to avoid backtest overfitting is the 'hold-out' strategy, where a certain portion of the dataset is reserved for testing true OOS performance. Numerous problems have been pointed out with this approach. The data is often used regardless, and awareness of the movements in the data may, consciously or otherwise, influence strategy and test design by the researchers [111]. For small samples, a hold-out strategy may be too short to be conclusive [124]. Even for large samples, it results in the most recent data (which is arguably the most pertinent) not being used for model selection [65, 8].

There have been some suggestions to resolve the problem that is occurring in the literature as a result of this. Some suggesting new frameworks, which this section will cover, and others which focus on the review process or how data and replication procedures are made available [87]. The points made with regard to the review and data processes are important, but do not aid with more effective model training for the researcher up front and so are not covered here.

2.6.1 Sharpe Ratio Assessment Methodologies

The Sharpe Ratio (SR), as introduced by Sharpe [114] in earlier literature, has become the favoured choice as a widely usable investment portfolio performance measure [5]. The measure indicates the amount of return relative to risk that a portfolio offers, and any portfolio which maximizes this can be shown to lie on Markowitz's efficient frontier. In line with its popularity, there has been extensive research around its distributional properties. Lo [86] shows that the estimated SR of a portfolio's returns has been shown to follow a normal distribution, regardless of whether the underlying returns are themselves normally distributed or not. Further work by Christie [17] and Opdyke [101] derived a limiting distribution, assuming only stationary and ergodic returns.

The Sharpe ratio is based on the assumption that the returns used are the result of a single trial (as is the case most standard performance measures). In consideration of the issues laid out above, it then becomes a misrepresentative performance measure. Bailey and Lopez de Prado [5] expanded on the work by

Christie [17] and Opdyke [101] and developed the Probabilistic Sharpe Ratio (PSR). The PSR estimates the likelihood that an observed best estimated $\widehat{SR}$ exceeds a provided benchmark SR^*, which might be expected from variance in the trials.

It is worth emphasising the distinction in investment strategies between a Family Wise Error Rate (FWER), which is the probability that one or more false positives occur, and a False Discovery Rate (FDR), which is the ratio of false positives to predicted positives. Where an FDR based approach may be applicable to a manufacturing process where there is an acceptable defect rate, investment strategy generations will tend to rely on the single best approach produced. Necessarily then, the FWER must be controlled for. Further work by Bailey and Lopez de Prado [6] developed the False Strategy Theorem (FST) with this in mind, allowing the assessment of whether a presented strategy is a false positive or not.

This allowed the development of the Deflated Sharpe Ratio (DSR) which calculates the likelihood that the true SR is positive under consideration of numerous trials being tested [6]. The DSR can be estimated using the PSR methodology as $\widehat{PSR}[SR^*]$ where the benchmark Sharpe ratio SR^* is no longer user defined, but rather calculated based on the False Strategy Theorem. That said, the calculation of SR^* requires both the variance of trial SR values and the number of independent trials. These are usually unknown on accounts of being meta-research variables which cannot be estimated from the selected strategy. It is not typical for researchers to track and report on these variables, and even if they were to, the trials would not typically be independent, thus further complicating the practical realities around calculating SR^*. Lopez de Prado and Lewis [90] provide critical work to this end with the Optimal Number Clusters (ONC) algorithm. They present a modified K-means methodology of clustering strategies and trial results, such that the dependent returns are categorised into subgroups with high intra-cluster correlations and low inter-cluster correlations. This clustering allows an estimation then of both the variance and number of trials, which in turn allows the DSR to be calculated. With this as a confidence level, one can accept or reject the notion that the observed $\widehat{SR}$ is positive.

The approach developed by Lopez de Prado and Lewis [90] (of combining PSR, FST, DSR and ONC) relies on several principles which may differ from prior methods suggested (such as those by Harvey and Liu [64]). The first is that returns do not always follow a normal distribution, which is supported by empirical evidence, and so their method rather takes into account the skewness and kurtosis of observed returns. The second is that Extreme Value Theory is a more appropriate technique than something such as Sidak's correction, due to the incorporation of return variance when a non-normal distribution is present. The third is that they do not assume a constant average correlation across trials, noting that there is often a hierarchical structure amongst returns

where there may be highly correlated clusters of strategies. Failing to take this into account would then bias the expected number of independent trials and so the false positive probability.

2.6.2 Generalised Assessment Methodologies

There has been work by several authors to try and lay out alternative techniques to try and avert backtest overfitting. The Model Confidence Set (MCS) developed by Hansen et al. [63] starts with a collection of models or configurations, and removes models iteratively according to a defined loss function. The confidence set is defined by the remaining models once a non-rejection takes place within the process, and these models are considered to be statistically similar within a certain confidence range. MCS is thus able to facilitate equitable model selection. However, Aparicio and Lopez de Prado [3] showed that while MCS is a potential strategy, in practice is is ineffective due to the inordinate requirement of signal-to-noise necessary to identify true superior models, as well as a lack of penalization over the number of trials attempted.

Bailey et al. [8] have developed a more robust approach to backtesting and how overfitting during strategy selection might be avoided, called Combinatorially Symmetric Cross-Validation (CSCV). Their research defines backtest overfitting as having occurred when the strategy selection which maximizes IS performance systematically underperforms the median OOS performance in comparison to the remaining configurations. They use this definition to develop a framework which measures the probability of such an event occurring, where the sample space is the combined pairs of IS and OOS measures. The Probability of Backtest Overfitting (PBO) is then established as the likelihood of a configuration underperforming the median IS while outperforming IS.

The CSCV methodology provides several important benefits over traditional testing frameworks, including the usual K-fold cross validation used in machine learning. By recombining the slices of available data, both the training and testing sets are of equal size, which is particularly advantageous when comparing financial statistics such as the Sharpe ratio, which are susceptible to sample size. Additionally, the symmetry of the set combinations in CSCV ensure that performance degradation is only as a result of overfitting, and not arbitrary differences in data sets. It is pointed out that while CSCV and PBO should be used to evaluate the quality of a strategy, they should not be the function on which strategy selection relies, which in itself would result in overfitting. In this sense, the methodology helps assess overfitting, but not necessarily avoid it. Another of the noted limitations of the framework is that a high PBO indicates overfitting within the group of N strategies, which is not necessarily indicative that none of the strategies are skilful - it could be that all of them are.

2.6.3 Test Data Length

When using SR estimations, where it is possible that the true SR mean is zero, we may still (with enough configurations attempted) find an SR measurement which optimises IS performance. This is shown by Bailey et al. [7], who propose the non-null probability of selecting an IS strategy with null expected performance OOS. Notably, typical methods such as hold-out once again fail, as the number of configurations attempted are not recorded. They add a further derivation, which is the Minimum Backtest Length (MinBTL), a statistic which highlights the relationships between: selecting a strategy with a higher IS SR than the expected OOS SR, the number of strategies tested, and the number of years tested. The equation shows that as the number of strategies tested increases, the minimum back test length must also increase in order to contain the likelihood of overfitting to IS SR.

As shown extensively throughout ML literature, increased model complexity and number of parameters is one of the primary causes of overfitting. In context of the MinBTL formula, model complexity affects the number of configurations that are available and which may be tested, which in turn will increase likelihood of overfitting. A lack of consideration, or reporting, of the number of trials makes the potential for overfitting difficult to assess.

Bailey et al. [7] expanded on this view with assessing the impact of presenting overfit models as correct. They were able to show that in lieu of any compensation effects (i.e. a series following a Gaussian random walk), there is no reason for overfitting to result in negative performance. However, where compensation effects apply (e.g. economic/investment cycles, bubble bursts, major corrections etc.), then the inclusion of memory in a strategy is likely to be detrimental to OOS performance if overfitting isn't controlled for.

3 Data Processing and Generation

3.1 Data Processing

The datasets used go through several transformations throughout the training and prediction process:

1. The raw data is log differenced and aggregated to rolling windows,
2. The transformed data is split into Training and Prediction subsets,
3. The predicted outputs have the scaling and log differencing reversed in order to reconstruct the actual price points.

3.1.1 Log Difference Transformation and Aggregation

All datasets are transformed into log feature fluctuation values, and are then aggregated to include fluctuations over rolling window periods. The log feature fluctuation for measured prices $\tilde{p}$ at timepoint t is calculated as:

$$\Delta \mathbf{p}_i = \ln(\tilde{\mathrm{p}}_\mathrm{i}) - \ln(\tilde{\mathrm{p}}_{\mathrm{i}-1}). \tag{1}$$

This log feature fluctuation is processed for each asset's closing price and for each time point i (from the previous time point). The log fluctuations have the benefit of taking compound effects into account in a systematic way and are symmetric in terms of gains and losses. Further, the log transformation provides an ergodic time series which is indicative what will happen to $\tilde{\mathbf{p}}$ over time, and has been shown to be the correct optimization choice [99].

The datasets are then expanded with the rolling window summations both in the past, for input, and in the future, for predicted output. A typical example would be past data aggregation windows of 1, 5 and 20, and a future prediction point of 5. These are calculated as summations of the log differences, such that for d days:

$$\mathbf{p}^{-}_{(\mathbf{d},\mathbf{t})} = \sum_{i=t-d}^{t} \Delta \mathbf{p}_i, \tag{2}$$

$$\mathbf{p}^{+}_{(\mathbf{t},\mathbf{d})} = \sum_{i=t+1}^{t+d} \Delta \mathbf{p}_i. \tag{3}$$

Only data points with a full set of features are used for training and prediction.

3.1.2 Data Scaling

Once the log differenced data has been aggregated using equations (2) and (3), the datasets are either standardized or normalized to allow for better learning. Typical implementations of these methods are detailed below:

3.1.2.1 Standardization

$$\mathbf{z_i} = \frac{(\mathbf{p}_\mathrm{i} - \bar{\mathbf{p}})}{\sigma_\mathrm{p}} \tag{4}$$

3.1.2.2 Normalization

$$\mathbf{n_i} = \frac{\mathbf{p}_\mathrm{i} - \min(\mathbf{p})}{\max(\mathbf{p}) - \min(\mathbf{p})} \tag{5}$$

The framework presented here only uses the suggested variations, dubbed "Limited Standardization" and "Limited Normalization", which are used when the data is split up into the Training and Prediction sets. In this case all the data needs to be scaled, but the variance or range should not be travelling from the Prediction set to the Training set through the use of aggregated measures such as $\bar{\mathbf{p}}$. Thus, if the data is split at point t out of n data points, the scaling would be implemented as follows:

3.1.2.3 Limited Standardization

$$\mathbf{q_{(i,t)}} = \frac{(\mathbf{p}_\mathrm{i} - \bar{\mathbf{p}}_{1:\mathrm{t}})}{\sigma_{\mathrm{p}_{1:\mathrm{t}}}}, \forall \mathrm{i} \in (1, \mathrm{n}) \tag{6}$$

3.1.2.4 Limited Normalization

$$\mathbf{v_{(i,t)}} = \frac{\mathbf{p}_\mathrm{i} - \min(\mathbf{p}_{1:\mathrm{t}})}{\max(\mathbf{p}_{1:\mathrm{t}}) - \min(\mathbf{p}_{1:\mathrm{t}})}, \forall \mathrm{i} \in (1, \mathrm{n}) \tag{7}$$

This log-differenced, aggregated and scaled data is then used as the input for the neural network models.

3.1.3 Reverse Data Scaling

The predicted data points are transformed back into actual prices for returns analysis. The first step is to reverse the scaling that is done in Section 3.1.2 in order to retrieve the log difference fluctuations. The reverse scaling equations are for a dataset of n observations, which has been split at point t.

3.1.3.1 Reverse Standardization

$$\mathbf{z_i^*} = \mathbf{z}_\mathrm{i} \cdot \sigma_\mathrm{p} + \bar{\mathbf{p}} \tag{8}$$

3.1.3.2 Reverse Normalization

$$\mathbf{n_i^*} = \mathbf{n}_\mathrm{i} \cdot (\max(\mathbf{p}) - \min(\mathbf{p})) + \min(\mathbf{p}) \tag{9}$$

3.1.3.3 Reverse Limited Standardization

$$\mathbf{q_i^*} = \mathbf{q}_\mathrm{i} \cdot \sigma_{\mathrm{p}_{1:\mathrm{t}}} + \bar{\mathbf{p}}_{1:\mathrm{t}}, \forall i \in (1, \mathrm{n}) \tag{10}$$

3.1.3.4 Reverse Limited Normalization

$$\mathbf{v_i^*} = \mathbf{v}_\mathrm{i} \cdot (\max(\mathbf{p}_{1:\mathrm{t}}) - \min(\mathbf{p}_{1:\mathrm{t}})) + \min(\mathbf{p}_{1:\mathrm{t}}), \forall i \in (1, \mathrm{n}) \tag{11}$$

3.1.4 Price Reconstruction

The predicted log fluctuations produced in Section 3.1.3 are used to reconstruct the predicted prices.

$$\mathbf{p_i^*} = \tilde{\mathbf{p}}_{\mathrm{i-d}} \cdot \mathrm{e}^{\hat{\mathrm{p}}_\mathrm{i}}, \tag{12}$$

where

$\tilde{\mathrm{p}}$ is the original asset price,

$\hat{\mathrm{p}}$ is the predicted log fluctuation,

i is the price timepoint, and

d is the price fluctuation prediction horizon.

These reconstructed prices are then used to assess model returns later in the Money Management Strategy (Section 4.6).

3.1.5 Decorrelation

The data processing does not incorporate any sort of decorrelation processes or data "whitening" transformations, thus keeping all correlation relationships between asset price fluctuations intact.

3.2 Synthetic Data Generation

Synthetic data generation is implemented using Geometric Brownian Motion (GBM) as described in Algorithm 1, which allows the asset simulations to be implemented with a drift (μ) and variance (σ) for each asset. The GBM simulation has the benefit of generating prices in a random walk, such that future movements are independent of past prices and so suitable for emulating an efficient financial market. The implications of using such data are discussed more fully in Section 8.5. Each dataset was generated with a random seed and using a Mersenne Twister pseudorandom number generator.

Algorithm 1: Geometric Brownian Motion Simulation

Input: σ, μ, $\tilde{p}_0$, n
$t = \frac{1}{n}$;
$\tilde{\mathbf{p}^{\mathbf{s}}} = [\tilde{p}_0]$;
foreach *i in 1:n* **do**
  z = random() $\sim \mathcal{N}(0, 1)$;
  $\tilde{p}_i^s = \tilde{p}_{i-1}^s \cdot e^{(\mu - \frac{\sigma^2}{2})t + \sigma\sqrt{t}z)}$;
  $\tilde{\mathbf{p}^{\mathbf{s}}} = [\tilde{\mathbf{p}^{\mathbf{s}}}, \tilde{p}_i^s]$;
end
Result: $\tilde{\mathbf{p}^{\mathbf{s}}}$

3.2.1 GBM Data Distributions

An important aspect of GBM generated data is that the price changes follow a Log-Normal distribution. If this data is then processed as per the steps above in Section 3.1, then after taking the log differenced values using equation (1), the values will follow a Normal distribution. Once scaled using standardization or normalization, these values will become centred and possibly be following a standard normal distribution. However, due to the effect of the limited scaling techniques used, only the Training dataset will be consistent in following this distribution, with the Prediction dataset having an increasing number of outliers as the timepoint moves further away from the Training set demarcations. These issues have notable impacts throughout the results, and are discussed more in Section 8.5.

3.3 Price Considerations

The "Actual10" dataset, as described in Section 7.1.1, is a price relative dataset. It should be noted that this does not account for inflation, which arguably makes the effective training of SAE and predictive networks more difficult. However, not being able to fully account for inflation would be an issue in any forward looking predictive model, and so it is sensible to keep it this way in our current dataset. It is also not expected to have a material impact on short term predictions, though it does make comparisons across time periods (i.e. IS and OOS) more difficult. Additionally, the dataset is price relative starting at 1.0, and so only price changes from the start onwards are accounted for and impact P&L, rather than the initial starting prices which might have then caused some imbalances. This does not have a large impact on the process, but does mean the P&L figures in the results section are of unit measures, rather than any particular currency.

4 Models and Algorithms

4.1 Process Overview

The implementation focuses on bringing together several ideas: data reduction, deep learning with pretraining/weight initialization, online learning and backtest overfitting validation for the purposes of stock price prediction. The process implementation is discussed fully in Section 5.1, but can be summarised as per the steps below.

1. The dataset is split into 2 subsets: the Training portion, and the Prediction portion.
2. The Training set is used to train the SAE and predictive FFN using the SGD algorithm. These networks are constructed with pretraining or weight initialization techniques.
3. Once the SGD training is complete for the predictive network, the same Training dataset is used to generate the IS predictions to be used for the CSCV process.
4. The Prediction set of data is used to continue training the network in an online manner using OGD.
5. The predictions made during steps 3 and 4 go through reverse scaling, and prices are reconstructed.
6. These reconstructed price predictions are used by the MMS to calculate returns and P&L.
7. The returns and P&L calculated by the MMS are used by the CSCV process, to estimate the PBO or DSR figures.

The rest of this section will detail the algorithms used to train the relevant FFN, RBM and SAE networks. It will also cover the algorithmic implementations of the MMS, and the CSCV, PBO and DSR methods.

4.2 Feedforward Neural Networks

Constructing Feedforward Neural Networks (FFN) in the form of multilayer perceptrons is a well established network technique, providing effective non-linear representations for both shallow and deep structures [110]. Specifically, a FFN is made up of several non-cyclical layers: the first and last are the input and output layers, respectively, and any inbetween are referred to as 'hidden' layers. Each layer is made up of nodes which are fully connected to the nodes in the previous and following layers, but do not have connections to nodes within the layer - information only travels forward. Each node has an activation function, which acts on the weighted input from the previous layers' nodes.

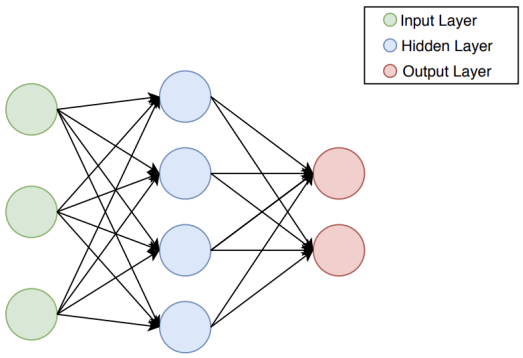


Figure 2: An example diagram of a Feedforward Neural Network with 1 hidden layer.

4.2.1 Notation and Network Representation

For the purposes of this section, the following notation will be used:

1. A network N will be constructed with L layers, each with e nodes,
2. The weights from the j^{th} node in the ℓ^{th} layer to the k^{th} node in the $(\ell-1)^{th}$ layer are represented by $\mathbf{w}_{jk}^{\ell}$,
3. The bias for the j^{th} node in layer ℓ is represented by $\mathbf{w}_{0j}^{\ell}$,
4. The output for the j^{th} node in layer ℓ is

$$\mathbf{a}_j^{\ell} = \theta(\mathbf{z}_j), \tag{13}$$

   for an activation function θ (with inverse θ') and weighted input $\mathbf{z}$, and
5. The weighted input for the j^{th} neuron in layer ℓ is

$$\mathbf{z}_{\mathbf{j}}^{\ell} = \sum_{i=1}^{e^{\ell-1}} \mathbf{a}_i^{\ell-1}\mathbf{w}_{ij} + \mathbf{w}_{0j}. \tag{14}$$

These definitions allow an input into the network to be propagated through it, having the original values processed through the weights and activation functions, and have an output in the form of the network's last layer.

4.2.2 Activation Functions

As noted in Section 2.2.2, there are 3 primary characteristics of concern for activation functions: non-linearity, continuous differentiability and monotonicity. While many different functions have been suggested and used, several of the more popular were implemented here.

4.2.2.1 Sigmoid

The sigmoid, or logistic, function is one of the most widely used activation functions historically, and is defined as:

$$\theta(x) = \frac{1}{1 + e^{-x}}, \tag{15}$$

$$\theta'(x) = \theta(x)(1 - \theta(x)). \tag{16}$$

The sigmoid function is in the range [0,1], making it a suitable choice for problems requiring a probabilistic output. The slope of the function curve is both a boon and a drawback: it allows for fast learning initially, but results in learning slowdown later (often causing what is referred to as node "saturation"). The exponent calculation is also computationally expensive, relatively speaking.

4.2.2.2 ReLU

The Rectified Linear Unit (ReLU) is a newer activation function which has been shown to be effective in deep learning networks. It is defined as:

$$\theta(x) = \max(0, x), \tag{17}$$

$$\theta'(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

The function has the benefits of quick learning which doesn't saturate, as well as being computationally cheap. The downside is that the non-gradient for the negative range of the function can result in 'dead' nodes, which stop updating with the learning process.

4.2.2.3 Leaky ReLU

The Leaky ReLU resolves the dying ReLU problem by adding a small gradient to the negative range of the function. This results in a slow learning being applied to 'dead' ReLUs, which may in turn result in them being used again if necessary. The choice of 0.01 as the negative range gradient in equations (19) and (20) has been shown to be generally effective [93], though further work might investigate different parameter values for this. The Leaky ReLU activation function is defined as:

$$\theta(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.01x, & \text{otherwise,} \end{cases} \tag{19}$$

$$\theta'(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0.01, & \text{otherwise.} \end{cases} \tag{20}$$

4.2.2.4 Linear Activation

Linear activations don't transform the input, and have a constant gradient, which can be useful for key layers where no loss in error signal is desired, such as the output or encoding layers. The linear activation function is defined as:

$$\theta(x) = x, \tag{21}$$

$$\theta'(x) = 1. \tag{22}$$

4.2.3 Backpropagation

The backpropagation algorithm, as discussed in Section 2.2.1, has allowed for effective training of FNNs for given data. The algorithm relies on incremental improvements of the model, as defined by decreasing the cost function. A common choice for cost (C) is Mean Squared Errors (MSE):

$$C = \frac{1}{2}\|\mathbf{y} - \mathbf{a}^L\|^2. \tag{23}$$

The conceptual steps for the backpropagation (fully detailed in Algorithm 2) are:

1. **Forward Pass**: The samples are propagated through the network, in order to generate the output $\mathbf{a}^L$.

   Vectorization is typically used to calculate the results for multiple samples at once. Conceptually though, each sample is used as input for the first layer, and subsequent layers use the weighted output from the previous

layer passed through the activation function as input.

Equation 14 is used to calculate the weighted output and the equations defined in Section 4.2.2 are used for the activation functions.

2. **Calculate Cost**: The cost between the training output $\mathbf{y}$ and the model output $\mathbf{a}^L$ is calculated.

   Using a cost function such as MSE (equation (23)), the rate of change for the error in the output layer by the rate of change in the output layer's activation functions is calculated. This rate of change for the error of the j^{th} node, δ_j, is expressed as:

$$\delta_j^L = \frac{\partial \mathbf{C}}{\partial \mathbf{a}_j^L} \theta' \left(\mathbf{z}_j^L\right). \tag{24}$$

   The vectorized version of this is often used, and expressed in equation (25). $\nabla_a C$ is a vector of components which are the partial derivatives, such that they represent the rate of change of cost C relative to the activation functions' outputs. Thus,

$$\delta^L = \nabla_a \mathbf{C} \otimes \theta'(\mathbf{z}^L), \tag{25}$$

   where $\otimes$ is the Hadamard (or element-wise) matrix product.

   If using quadratic cost, as per equation (23), then the term $\frac{\partial C}{\partial a_j^L}$ can be reduced to $(a^L - y)$. The equation for the output layer error rate of change is then:

$$\delta^L = (\mathbf{a}^L - \mathbf{y}) \otimes \theta'(\mathbf{z}^L). \tag{26}$$

3. **Backwards Pass**:
   (a) **Calculate Errors For Previous Layers:** The activation values are propagated back through the network to calculate the delta values at each layer.

   This uses the same rationale as equation (24), but rather uses $(\mathbf{w}^{\ell+1})^T$ by the next layer's error in order to get an indication of the error rate and how it moves backwards through the network:

$$\delta^\ell = ((\mathbf{w}^{\ell+1})^T \delta^{\ell+1}) \otimes \theta'(\mathbf{z}^\ell). \tag{27}$$

   (b) **Adjust Network Weights:** Network weights are adjusted in proportion to the error observed and chosen learning rate.

Each weight's output error and input activation are multiplied to find the weight's gradient, which is then reduced by a factor of the learning rate η. This adjustment value is subtracted from the current network weights:

$$\mathbf{w}^\ell \rightarrow \mathbf{w}^\ell - \eta\delta^\ell(\mathbf{a}^{\ell-1})^T, \tag{28}$$

where η, the learning rate, is a configured parameter.

Choices regarding the learning rates are discussed in Section 4.2.6 and the full algorithm is presented below in Algorithm 2.

Algorithm 2: Backpropagation

Input: Neural Network N, with randomly initialized weights $\mathbf{w}$, L layers, activation functions θ and learning rate η

Data: Testing set with inputs $\mathbf{x}$, and outputs $\mathbf{y}$

repeat
- Select a sample x_s from x;
- `// Perform the Forward Pass, to calculate the network output,` y_s`, for input sample` x_s
- $a_0 = x_s$;
- **foreach** ℓ *in 1:*L **do**
  - **foreach** j *in 1:*e^ℓ **do**
    - $\mathbf{z}_j^\ell = \sum_{i=1}^{e^{\ell-1}} \mathbf{a}_i^{\ell-1}\mathbf{w}_{ij} + \mathbf{w}_{0j}$;
    - $\mathbf{a}_j^\ell = \theta(\mathbf{z}_j^\ell)$;
  - **end**
- **end**
- `// Calculate the error term (aka cost), as`
- $\delta^L = (\mathbf{a}^L - \mathbf{y}) \otimes \theta'(\mathbf{z}^L)$;
- `// Perform the Backward Pass, to propagate the errors back and update the network accordingly`
- **foreach** ℓ *in (L-1):1* **do**
  - `// Calculate the delta values`
  - $\delta^\ell = ((\mathbf{w}^{\ell+1})^T\delta^{\ell+1}) \otimes \theta'(\mathbf{z}^\ell)$;
  - `// Update the weights`
  - $\mathbf{w}^\ell \rightarrow \mathbf{w}^\ell - \eta\delta^\ell(\mathbf{a}^{\ell-1})^T$;
- **end**

until *no new samples can be drawn from* x;

Result: Updated Network N

4.2.4 Gradient Descent Algorithms

The backpropagation algorithm is defined at a single sample level, and the learning from a dataset is usually repeated for a number of "epochs". As noted though, implementation is often implemented using a vectorized version

of the algorithm which either samples the entire dataset at once (batch), or a subset of samples (mini-batch). The latter is the Stochastic Gradient Descent (SGD), as discussed in Section 2.4, which has been shown to increase the speed and stability at which the backpropagation algorithm can converge to a minima in terms of cost. The algorithm runs backpropagation over the entire dataset for the number of epochs, and updates the network incrementally through the epoch. The stopping condition for the algorithm is usually defined as either a particular number of epochs being reached, or cost no longer decreasing for some number of epochs (i.e. a minima has been reached).

The changes to the initial backprop algorithm, detailed in Algorithm 2, are relatively minor:

1. Selection is now performed such that at each epoch m, s new samples are from x, forming the subset x_s.
2. The weight update is now performed such that it represents an aggregate update according the m samples that were selected:

$$\mathbf{w}^{\ell} \rightarrow \mathbf{w}^{\ell} - \frac{\eta}{m} \sum_{x} \delta^{x,\ell} (\mathbf{a}^{x,\ell-1})^{T}, \tag{29}$$

   where
   (a) w^{ℓ} is the ℓ^{th} layer's weights,
   (b) η is the configured learning rate parameter,
   (c) m is the size of the mini-batch,
   (d) δ is the rate of change in the error (calculated using equation (27)), and
   (e) a is the output of the activation function for the weighted input (calculated using equation (13)).

Online Gradient Descent Where SGD is appropriate and effective for scenarios where the entire dataset is available, Online Gradient Descent (OGD) is applicable for when the model is learning in an online fashion. In this case, the backpropagation is run as defined above in Algorithm 2 but with no repetition (i.e. only 1 epoch).

4.2.5 Regularization

Regularization is a commonly used technique in machine learning used to reduce a model's capacity to overfit to the data, and in so doing reduces the variance in the model results. One of the typical methods, "$L2$" (or "weight decay"), is implemented by adding an extra term to the cost function which is itself a function of the weights in the model. This extra term forces the learning process to favour smaller weights, and only allows large weights to

occur if they are able to offer an appropriate increase in performance. The modified cost function is calculated as:

$$\mathbf{C} = \frac{1}{2n} \sum_{x} \left\| \mathbf{y} - \mathbf{a}^{L} \right\|^{2} + \frac{\lambda}{2n} \sum_{w} \mathbf{w}^{2}, \tag{30}$$

where

1. n is the dataset sample size,
2. y is the input data,
3. a is the output of the activation function in the L^{th} layer (calculated using equation (13)),
4. λ is the configured regularization rate, and
5. w is a layer's weights.

The additional term is scaled by the configurable regularization parameter, λ, which if small approximates the original cost function or if large will increase the degree of regularization used.

In order to implement this in backpropagation, the weight update rule is changed to the following (while biases remain the same):

$$\mathbf{w} \rightarrow \mathbf{w} - \eta \frac{\partial \mathbf{C}_0}{\partial \mathbf{w}} - \frac{\eta \lambda}{n} \mathbf{w}. \tag{31}$$

For SGD, this can be simplified to the following:

$$\mathbf{w} \rightarrow \left(1 - \frac{\eta \lambda}{n}\right) \mathbf{w} - \frac{\eta}{m} \sum_{x} \frac{\partial \mathbf{C}_x}{\partial \mathbf{w}}. \tag{32}$$

4.2.6 Learning Rate Schedule

Learning rate schedules are implemented to allow for a more dynamic exploration of the possible solution space, and allow fine tuning around an optima. With a constant learning rate implementation, one has to choose between a larger learning rate which allows faster progress but less optimisation, or a smaller learning rate which learns slower but is more effective at optimising (by avoiding the learning algorithm from bouncing around a minima valley). Learning rate schedules aim to get the best of both scenarios by cycling through a minimum and maximum range across epochs.

The learning rate schedule is specified to cycle through from the minimum value to the maximum value every T epochs, from $\eta_{\min}$ to $\eta_{\max}$. Thus for the current epoch i, the learning rate is calculated as:

$$\eta_i = \eta_{\min} + \frac{1}{2} \left(\eta_{\max} - \eta_{\min}\right) \left(1 + \cos\left(\frac{i}{T}\pi\right)\right). \tag{33}$$

This is the Cyclical Learning Rates approach, though using sinusoidal form rather than linear [117]. For a minimum learning rate of 0.1, a maximum of

1.0 and an epoch cycle of 100, it will produce learning rates as displayed in Figure 3 below.

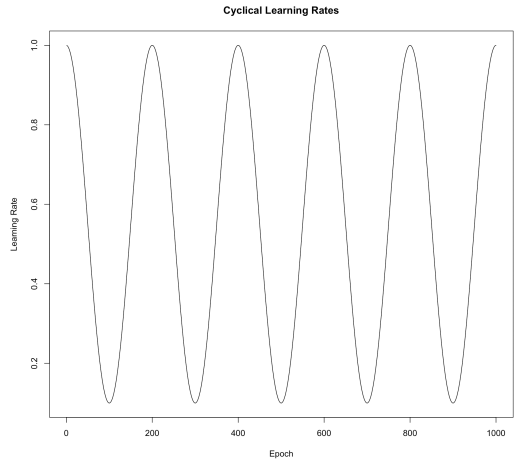


Figure 3: Learning rates calculated over 1000 epochs with $\eta_{\min} = 0.1$ to $\eta_{\max} = 1.0$ and $i = 100$.

4.2.7 Dropout

Input layer dropout has been implemented in order to perform regularization on the data features. The methodology used will set a percentage of the features for each sample to 0 during the SGD training process. While the percentage stays the same throughout the testing process, the features selected for each sample are reselected every time it is used (i.e. for subsequent epochs). Conceptually, the dropout as applied to the input layer may result in the network learning relations between assets and possible patterns that would improve predictive efficacy.

4.3 Restricted Boltzmann Machines

Restricted Boltzmann machines (RBM) are generative networks which can be trained to learn probability distributions over a dataset. They are structurally different from a FFN in that they have a recurrent weight structure and stochastic unit determination. Specifically, a typical RBM has one visible layer (input/output), and one hidden layer, and the values in the hidden layer will take on a binary value with a probabilistic likelihood.

Thus, the input and output have the same structure, and the processing from the hidden layer creates the generative process learned by the RBM. The hidden units correspond to feature detection of the visible unit data structures, and the learning process of the network results in effective parameter estimation.

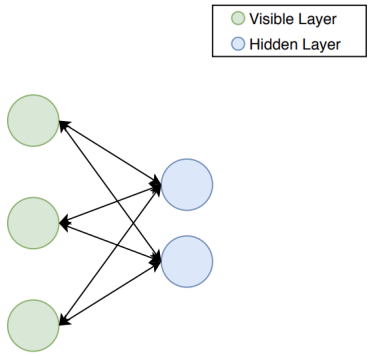


Figure 4: An example diagram of a restricted Boltzmann machine network.

RBM input processing goes through the following steps:

1. An input sample, $\mathbf{y}$, is processed from the visible layer to the hidden layer using typical feedforward functions.
2. Each node in the hidden layer takes on a value of 1 with a probability of its activation function output, or else is equal to 0.
3. The visible layer nodes are reconstructed to provide $\hat{\mathbf{y}}$, using the hidden layer values as the input.

4.3.1 RBM Stochastic Descent

Hinton [67] shows that the joint configuration (v, h) of the visible and hidden units has an energy given by :

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in \text{visible}} \mathbf{a}_i \mathbf{v}_i - \sum_{j \in \text{hidden}} \mathbf{b}_j \mathbf{h}_j - \sum_{i,j} \mathbf{v}_i \mathbf{h}_j \mathbf{w}_{ij}, \tag{34}$$

where

1. $\mathbf{v}$ and $\mathbf{h}$ are the binary state vectors of the visible and hidden units respectively,
2. $\mathbf{a}$ and $\mathbf{b}$ are the bias value vectors for the visible and hidden units respectively, and
3. $\mathbf{w}$ is the weights vector for the 2 visible and hidden layers.

It can be shown, that network weights can be adjusted to change the probabilities assigned to a particular training sample . The derivations of equation (34) show that performing a stochastic descent for the log probability of the data can be implemented through the following weight adjustment:

$$\Delta \mathbf{w}_{ij} = \eta(\langle \mathbf{v}_i \mathbf{h}_j \rangle_{\text{data}} - \langle \mathbf{v}_i \mathbf{h}_j \rangle_{\text{model}}). \tag{35}$$

The angular brackets here indicate expectations under the distribution specified by the following subscript. Due to the probabilistic nature of RBMs, the [0,1] ranged sigmoid activation function is typically used. Thus, for hidden nodes and a data sample v, it is easy to get unbiased sampling of $\langle v_i h_j \rangle_{\text{data}}$:

$$\mathrm{P}(\mathbf{h}_j = 1) = \sigma(\mathbf{w}_{0j} + \sum_{i=1} \mathbf{v}_i \mathbf{w}_{ij}). \tag{36}$$

Similarly, the visible states (or reconstruction), can then be calculated as

$$\mathrm{P}(\mathbf{v}_i = 1) = \sigma(\mathbf{w}_{0i} + \sum_{j=1} \mathbf{h}_j \mathbf{w}_{ij}). \tag{37}$$

Getting an unbiased sample of $\langle v_i h_j \rangle$ proves more problematic, and so the model reconstructions via Gibbs sampling are used instead (this is expanded on below), resulting in the weight updates of

$$\Delta \mathbf{w}_{ij} = \eta(\langle \mathbf{v}_i \mathbf{h}_j \rangle_{\text{data}} - \langle \mathbf{v}_i \mathbf{h}_j \rangle_{\text{reconstruction}}). \tag{38}$$

While it is important for the hidden units to take on a binary value (and so avoid communicating real values rather than learning structure), the visible units may be chosen to take on the probability values, rather than the stochastic samples, particularly if real valued output is necessary.

4.3.2 Contrastive Divergence

The process of sampling and resampling may be run for many iterations between the two layers before finishing on an output - this potentially long running and stochastic process results in the generative aspect of the network, and constitutes a Gibbs sampling chain. Multiple sampling steps in this chain is known as Contrastive Divergence, or CD-n, where n represents the number of steps, and which allows for effective parameter estimation. CD-1 is detailed in Algorithm 3.

Algorithm 3: CD-1

Input: RBM Network N, with visible nodes $\mathbf{v}$ and hidden nodes $\mathbf{h}$ and weight distribution $\mathbf{w}$ (representing $\langle vh \rangle$)

Data: Training sample dataset D

foreach $\mathbf{V^D}$ *in* D **do**

`// Sample` $\mathrm{H^D}$ `from` $\mathrm{P(H^D|V^D)}$

foreach n *in (1:j)* **do**

$\mathrm{P}(\mathbf{H^D}_n = 1) = \sigma(\mathbf{w}_{0j} + \sum_{i=1} \mathbf{V^D}_i \mathbf{w}_{in})$

end

`// Sample` V `from` $\mathrm{P(V|H^D)}$

foreach n *in (1:i)* **do**

$\mathrm{P}(\mathbf{V}_n = 1) = \sigma(\mathbf{w}_{0i} + \sum_{j=1} \mathbf{H^D}_j \mathbf{w}_{ij})$

end

`// Sample` H `from` $\mathrm{P(H|V)}$

foreach n *in (1:j)* **do**

$\mathrm{P}(\mathbf{H}_n = 1) = \sigma(\mathbf{w}_{0j} + \sum_{i=1} \mathbf{V}_i \mathbf{w}_{in})$

end

`// Calculate Change in Weights Using Reconstructed Distribution`

// $\Delta \mathbf{w}_{ij} = \eta(\langle \mathbf{v}_i \mathbf{h}_j \rangle_{\text{data}} - \langle \mathbf{v}_i \mathbf{h}_j \rangle_{\text{reconstruction}})$

$\Delta \mathbf{w} = \eta([\mathbf{V}^D \mathbf{H}^{DT}] - [\mathbf{V}\mathbf{H}^T])$

`// Update the weights`

$\mathbf{w} \rightarrow \mathbf{w} - \Delta \mathbf{w}$

end

Result: Network N with weights $\mathbf{w}$ updated for dataset D

There's no upper bound on the iterations used for CD, and running for many can prove more effective for certain purposes. In this case however, where RBMs are used for the purposes of weight initialization, CD-1 is usually deemed sufficient.

4.3.3 CD-1 and SGD

In the same way that the backpropagation learning algorithm in Section 4.2.3 can be implemented in a mini-batch SGD process for FFNs, so can the CD learning algorithm for RBMs. The framework is kept the same, with the implementation of epochs and weight updates based on stochastically chosen minibatches, but the calculation used to update the weights is CD-1 instead.

4.4 Stacked AutoEncoders

As noted in Section 2.3, the use of Stacked AutoEncoders (SAE) have resulted in significant improvements in deep learning networks, and allowed effective reduction of high dimensional data. A single layer AutoEncoder is a specialized type of FNN with 3 layers: one input, one hidden, and one output. The network is trained (using backpropagation as per Section 4.2.3) to reconstruct the input. The input and output layers have the same structure, and the hidden layer needs to have fewer nodes than the input. This forces the hidden layer to learn effective features of the data, and reduce the dimensional representation.

Stacked AutoEncoders follow a similar structure, but with multiple hidden layers. The only strict requirement of the hidden layers is that the middle one, which will be used as the encoder layer, still has fewer nodes than the input. This structure can still be trained using backpropagation, but with more layers it is likely to begin suffering from the vanishing or exploding gradient problem. As noted, the work by Hinton and Salakhutdinov [68] for initialization of weights helps resolve this.

4.4.1 Sigmoid Based Greedy Layerwise SAE Training

The steps for implementing the SAE training suggested by Hinton and Salakhutdinov [68] are detailed in Algorithm 4.

Algorithm 4: SAE Training - Sigmoid Greedy Layerwise

Input: Network structure N, which conforms to the requirements of an SAE with L layers; Input dataset X.

1. **Train First Layer:** For the first hidden layer, train as you would for an RBM with 2 layers - with the input layer, and the first layer as the hidden layer, using CD-1 and SGD.
2. **Train Subsequent Layers:** For each layer ℓ in $(2, L/2)$:
   (a) Process data X through the previously trained layers using the forward propagation as defined in Section 4.2.1.
   (b) This processed data then forms the input to the ℓ^{th} layer, which can be trained once again using CD-1 and SGD as if it were two layers.
3. **Mirror Network:** Once all the layers up until the encoder layer have been trained in this greedy layerwise fashion, mirror the weights and layers structures after the encoder to create a fully L layered FFN with pre-trained weights.
4. **Train Full AutoEncoder:** This network can then be trained using the backpropagation and SGD algorithms on dataset X, where the cost of reproducing the network input is minimised.
5. **Split to Encoder:** Once a minima or acceptable level of reconstruction has been reached, the network can be truncated as the encoder layer, and so the first $L/2$ layers are used as the SAE.

Result: Trained SAE network.

Notably, this weight initialization will only be effective if the RBM and SAE networks use the same activation function, which due to the RBM implementation, needs to be a function that can output a probabilistic value in [0,1].

4.4.2 ReLU based SAE Training

ReLU activations differ from sigmoid in that they are not fitting for probability estimations, which makes the algorithm suggest by Hinton et al. unsuitable. The process used here relies on effective weight initialization, and is a simplification of the greedy layerwise training (Section 4.4.1), as detailed in Algorithm 5.

Algorithm 5: SAE Training - ReLU Backpropagation

Input: Network structure N, which conforms to the requirements of an SAE with L layers; Input dataset X.

1. **Initialize Network Weights:** Use an effective weight initialization for all L layers, such as the Xavier or He Initializations (discussed fully in Section 4.5).
2. **Train Full AutoEncoder:** This network can then be trained using the backpropagation and SGD algorithms for dataset X, where cost of reproducing the network input is minimised.
3. **Split to Encoder:** Once a minima or acceptable level of reconstruction has been reached, the network can be truncated as the encoder layer, and so the first $L/2$ layers are used as the SAE.

Result: Trained SAE network.

4.4.3 Denoising AutoEncoders

As noted in Section 2.3.3, denoising can be used as an optimization technique for AutoEncoders in order to improve general performance and reconstruction. The methodology works by corrupting the input data for training, but using the non-corrupted data as the expected output sample. In doing so, this forces the AutoEncoder to learn more fundamental representations of the data rather then fitting to sample noise, hence "denoising". Two of the more typical techniques for achieving this have been implemented, as detailed below. In both cases, the noise is reapplied each time training samples are chosen.

4.4.3.1 Additive Gaussian Noise

With Gaussian noise, samples are corrupted such that a degree of variance is added to the input according to a parameterized Gaussian distribution. In this case, the distribution variance is a configurable training parameter:

$$\tilde{\mathbf{x}}|\mathbf{x} \sim \mathcal{N}\left(\mathbf{x}, \sigma^2 I\right). \tag{39}$$

4.4.3.2 Masking Noise

With masking noise, a fraction of the features in the data are set to 0, thus masking them for that sample. The percentage of features chosen is a configurable training parameter.

4.5 Variance Based Weight Initializations

Research by He et al. [66] has shown that weights can be initialized to maintain expected variance between the input and output layers. These methodologies have the immediate advantages of simpler implementation, as well as faster computation (as no pretraining is required). They also allow for effective weight initialization of non-probabilistic activation functions, such as ReLU. Whether they result in better reconstructions or predictions is less clear (especially as the linearity assumption could prove faulty), and so the methods are tested here as well.

A common initialization heuristic is to use a uniform, but not layer agnostic, initialization such as equation (40) below. While simple enough, it has been shown that this does not always lead to the best training results, and can be outperformed by variance based initializations [59].

$$\mathbf{w} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right], \tag{40}$$

where n is the number of nodes in the layer.

4.5.1 Initialization Rationale

The variance initialization methodology is based on balancing the variance of a linear network. For input X, with n components, linear neurons with weights W, and output Y:

$$\mathbf{Y} = \mathbf{W}_1\mathbf{X}_1 + \mathbf{W}_2\mathbf{X}_2 + ... + \mathbf{X}_n\mathbf{W}_n. \tag{41}$$

It can thus be shown, that for IID samples with mean 0:

$$\mathbf{Var}(\mathbf{Y}) = n\mathbf{Var}(\mathbf{W}_i)\mathbf{Var}(\mathbf{X}_i). \tag{42}$$

For the variance of both input X and output Y to be balanced on the forward and backward propagation then, it is necessary for

$$\mathbf{Var}(\mathbf{W}_i) = \frac{1}{n_{\text{in}}} = \frac{1}{n_{\text{out}}}. \tag{43}$$

In the instance where there are not an equal number of nodes in the two layers, the average can be taken, such that

$$\mathbf{Var}(\mathbf{W}_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}. \tag{44}$$

4.5.2 Initializations

4.5.2.1 "Xavier" Initialization

Considering that $\mathbf{Var}(U(-a, a)) = a^2/3$, then ensuring balance variance for weights as per equation (44) provides us with the Xavier Glorot initialization

[59] which can be used for sigmoid activations, and is defined as follows:

$$\mathbf{w}_{ij} \sim U(-r, r), r = \sqrt{\frac{6}{n_i + n_j}}, \tag{45}$$

where n is the number of nodes in the i^{th} or j^{th} layer.

4.5.2.2 "He" Initialization

The initialization for ReLU is different on accounts of the function being equal to zero for half its potential input range - in this case it makes sense to double the weight variance, and so the He [66] initialization is used:

$$\mathbf{w}_{ij} \sim U(-r, r), r = \sqrt{\frac{6}{n_i}}. \tag{46}$$

4.5.2.3 "He-Adjusted" Initialization

An adjusted initialization is presented here, dubbed "He-Adjusted", where the ReLU initialization uses a mean of the input and output layers in order to scale the weight variance. For networks with constant layer sizes, the initialization is the same as He, though for SAE networks which have layer size changes by definition, the He-Adjusted initialization will result in more appropriately sized weights:

$$\mathbf{w}_{ij} \sim U(-r, r), r = \sqrt{\frac{6}{(n_i + n_j)/2}}. \tag{47}$$

4.6 Money Management Strategy and Returns

In line with the general approach here, the Money Management Strategy (MMS) has been designed to be relatively simple, such that the results are more indicative of the underlying modelling rather than intricate trading strategies.

The MMS follows an arithmetic long strategy of buying any asset for which the predicted $t + D$ price is above the current price, and selling the stock at $t + D$. Trading costs have been included at 10% capital costs per annum for borrowing to purchase, and 0.45% for transaction costs as per [58]. This model does not take liquidity affects into account, though these can be accounted for in the costs. Results are compared to a benchmark model which has exact knowledge of the price in D days, which represents an upper bound on performance. This MMS implementation does not make any attempt to account for volume in the trades, and only trades one asset unit at a time.

4.6.1 Input Variables

The MMS takes the following values as input for each time point t and prediction point D (D is typically 5 in this paper):

1. $\tilde{\mathbf{p}}_t$, the closing price at time t,
2. $\tilde{\mathbf{p}}_{t+D}$, the closing price in D days, and
3. $\hat{\mathbf{p}}_{t+D}$, the predicted closing price in D days.

The naming conventions for the variables calculated are as follows:

1. s represents a trading signal indicating whether a trade should occur on a given day,
2. c represents the purchasing cost for an asset,
3. k represents the capital cost for purchasing an asset (i.e. the borrowing and trading costs),
4. f represents the full cost of purchasing an asset (i.e. $c + k$),
5. o represents the nominal P&L observed from trading an asset, and
6. r represents the return rate for trading an asset (with costs accounted for).

Capitalised variations represent the same variables, but aggregated across all assets.

4.6.2 Calculated Asset Variables

For each asset at each time point, the values in Table 1 below are calculated.

Variable Name	**Description**	**Equation**
Trading Signal	A trading signal indicating whether a trade will take place on t.	$\mathbf{s}_t = \begin{cases} 1, & \text{if } \hat{\mathbf{p}}_{t+D} > \mathbf{p}_t \\ 0, & \text{otherwise.} \end{cases}$ (48)

Trading Signal, Benchmark	A trading signal indicating whether a trade will take place on t in the benchmark model.	$$\mathbf{s}_t^b = \begin{cases} 1, & \text{if } \mathbf{p}_{t+D} > \mathbf{p}_t \\ 0, & \text{otherwise.} \end{cases} \quad (49)$$
Asset Cost	The cost incurred in buying the asset at time t.	$$\mathbf{c}_t = \tilde{\mathbf{p}}_t \cdot \mathbf{s}_t \quad (50)$$
Asset Cost, Benchmark	The cost incurred in buying the asset at time t in the benchmark model.	$$\mathbf{c}_t^b = \tilde{\mathbf{p}}_t \cdot \mathbf{s}_t^b \quad (51)$$
Transaction Capital Costs	The capital and transaction costs of buying the asset.	$$\mathbf{k}_t = \tilde{\mathbf{p}}_t \cdot (0.1/365 \cdot D) + \tilde{\mathbf{p}}_t \cdot (0.45/100) \quad (52)$$
Full Asset Costs	The full cost of trading the asset at t.	$$\mathbf{f}_t = (\tilde{\mathbf{p}}_t + \mathbf{k}_t) \cdot \mathbf{s}_t \quad (53)$$
Full Asset Costs, Benchmark	The full cost of trading the asset at t in the benchmark model.	$$\mathbf{f}_t^b = (\tilde{\mathbf{p}}_t + \mathbf{k}_t) \cdot \mathbf{s}_t^b \quad (54)$$
Nominal Observed P&L	The nominal observed P&L at $t + D$.	$$\tilde{\mathbf{o}}_{t+D} = (\tilde{\mathbf{p}}_{t+D} - \tilde{\mathbf{p}}_t) \cdot \mathbf{s}_t \quad (55)$$

Nominal Observed P&L, Benchmark	The nominal observed P&L at $t + D$ in the benchmark model.	$\mathbf{o}^b_{t+D} = (\tilde{\mathbf{p}}_{t+D} - \tilde{\mathbf{p}}_t) \cdot \mathbf{s}^b_t$ (56)
Nominal Expected P&L	The nominal expected P&L at $t + D$.	$\hat{\mathbf{o}}_{t+D} = (\hat{\mathbf{p}}_{t+D} - \hat{\mathbf{p}}_t) \cdot \mathbf{s}_t$ (57)
Fullcost Observed Return Rate	The fullcost observed return rate at $t + D$.	$\tilde{\mathbf{r}}_{t+D} = \frac{(\tilde{\mathbf{o}}_{t+D} - \mathbf{f}_t)}{\mathbf{f}_t}$ (58)
Fullcost Observed Return Rate, Benchmark	The fullcost expected return rate at $t + D$ in the benchmark model.	$\tilde{\mathbf{r}}^b_{t+D} = \frac{\left(\tilde{\mathbf{o}}^b_{t+D} - \mathbf{f}^b_t\right)}{\mathbf{f}^b_t}$ (59)
Fullcost Expected Return Rate	The fullcost expected return rate at $t + D$.	$\hat{\mathbf{r}}_{t+D} = \frac{(\hat{\mathbf{o}}_{t+D} - \mathbf{f}_t)}{\mathbf{f}_t}$ (60)

Table 1: Asset Level MMS Variable Calculations.

4.6.3 Calculated Strategy Variables

The MMS is then implemented using the values calculated in Table 1. These values are aggregated at each time point t, for all assets a, resulting in the calculation of the strategy level variables presented below in Table 2.

Variable Name	Description	Equation
Nominal Expected P&L	The nominal expected P&L for $t + D$ from all assets traded at t.	$\hat{\mathbf{O}}_{t+D} = \sum_{\text{a}\in\text{assets}} \hat{\mathbf{o}}_{a,t+D}$ (61)
Nominal Observed P&L	The nominal observed P&L for $t + D$ from all assets traded at t.	$\tilde{\mathbf{O}}_{t+D} = \sum_{\text{a}\in\text{assets}} \tilde{\mathbf{o}}_{a,t+D}$ (62)
Nominal Observed P&L, Benchmark	The nominal benchmark P&L for $t+D$ from all assets traded at t.	$\mathbf{O}^b_{t+D} = \sum_{\text{a}\in\text{assets}} \mathbf{o}^b_{a,t+D}$ (63)
Total Trade Cost	The cost of all trades at t.	$\mathbf{C}_t = \sum_{\text{a}\in\text{assets}} \mathbf{c}_{a,t}$ (64)
Total Trade Cost, Benchmark	The cost of all trades at t in the benchmark model.	$\mathbf{C}^b_t = \sum_{\text{a}\in\text{assets}} \mathbf{c}^b_{a,t}$ (65)
Total Full Trade Cost	The cost of all trades at t including transaction costs.	$\mathbf{F}_t = \sum_{\text{a}\in\text{assets}} \mathbf{f}_{a,t}$ (66)
Total Full Trade Cost, Benchmark	The cost of all trades at t, including transaction costs, in the benchmark model.	$\mathbf{F}^b_{p,t} = \sum_{\text{a}\in\text{assets}} \mathbf{f}^b_{a,t}$ (67)

Total Full Trade Cost Observed Return Rate	The full cost observed return rate at $t+D$.	$\tilde{\mathbf{R}}_{t+D} = \frac{\left(\tilde{\mathbf{O}}_{t+D} - \mathbf{F}_t\right)}{\mathbf{F}_t}$ (68)
Total Full Trade Cost Return Rate, Benchmark	The fullcost benchmark return rate at $t+D$ in the benchmark model.	$\tilde{\mathbf{R}}^b_{t+D} = \frac{\left(\tilde{\mathbf{O}}^b_{t+D} - \mathbf{F}^b_t\right)}{\mathbf{F}^b_t}$ (69)
Total Full Trade Cost Expected Return Rate	The full cost expected return rate at $t+D$.	$\hat{\mathbf{R}}_{t+D} = \frac{\left(\hat{\mathbf{O}}_{t+D} - \mathbf{F}_t\right)}{\mathbf{F}_t}$ (70)
Sharpe Ratio	The calculation of the Sharpe ratio for a set of model returns [86].	$\mathbf{SR} = \frac{\mathbb{E}(\tilde{\mathbf{R}})}{\sqrt{\mathbf{Var}(\tilde{\mathbf{R}})}}$ (71)

Table 2: Strategy Level MMS Variable Calculations.

4.7 Combinatorially Symmetric Cross-Validation and Probability of Backtest Overfitting

The combinatorially symmetric cross-validation (CSCV) method developed by Bailey et al. [8], as discussed in Section 2.6, can be used to assess the likelihood of backtest overfitting through comparison of IS and OOS return metrics. As discussed more extensively in Section 2.6, their research defines backtest overfitting as having occurred when the strategy selection which maximizes IS performance systematically underperforms median OOS in comparison to the remaining configurations. The N configurations are assessed according to a fixed performance measure, such that $\mathbf{R} = (R_1, R_2, ..., R_N)$ and $\bar{\mathbf{R}} = (\bar{R_1}, \bar{R_2}, ...\bar{R_N})$ represent the IS and OOS performance of each configuration, respectively.

The crux of the framework lies in comparing the ranking of a selected strategy, and so Ω is used to represent ranking space consisting of the $N!$ permutations of $(1, 2, ...N)$ indicating the ranking of the N strategies. The random vectors r and $\bar{r}$ are used to denote the IS and OOS ranking of the configurations making up $\mathbf{R}$ and $\bar{\mathbf{R}}$. Lastly a subset of Ω is defined, such that $\Omega_n^* = \{f \in \Omega | f_n = N\}$ - this encapsulates the highest ranking configurations.

The definition of backtest overfitting (where a strategy with optimal IS performance has an expected ranking below the median OOS) is given by:

$$\sum_{n=1}^{N} \mathbb{E}[\overline{r_n} | r \in \Omega_n^*] \mathbf{P}[r \in \Omega_n^*] \leq \frac{N}{2}. \tag{72}$$

Equation (72) allows the Probability of Backtest Overfitting (PBO) to be calculated using the Bayesian formula, such that:

$$\text{PBO} = \sum_{n=1}^{N} \mathbf{P}[\bar{r} < \frac{N}{2} | r \in \Omega_n^*] \mathbf{P}[r \in \Omega_n^*]. \tag{73}$$

Notably, the above definitions consider IS as the data made available to the strategy selection, rather than the model's calibration (e.g. the full IS dataset, rather than, by way of example, the number of days used in a moving average). This allows the model-free and non-parametric nature of the definition.

They further developed the CSCV framework as a methodology to reliably estimate the probability used in PBO, which allows a concrete application of the concept. The CSCV framework does not require using the typical hold-out strategy (and thus avoids credibility issues), and is ultimately able to provide a bootstrapped distribution of OOS performance. The full methodology is shown in Algorithm 6, as originally presented by Bailey et al. [8].

Algorithm 6: CSCV

Input: N configuration trials over T time periods with P&L

1. **Generate P&L Matrix:** Generate a $T \times N$ performance series matrix, M, representing the profits and losses by the N configuration trials over T time periods.

2. **Define P&L Submatrices:** Partition the M matrix by rows into S submatrices, each of even size $(\frac{T}{S} \times N)$.

3. **Define P&L Submatrix Combinations:** Generate the combinations C_S of M_S, in groups of size $\frac{S}{2}$, for total $\binom{S}{S/2}$ of combinations.

4. **Process Combinations:** For each combination in $c \in C_S$:

   (a) **Define IS Set:** Form a training set J by joining $\frac{S}{2}$ M_S submatrices, in their original order. J is a matrix of order $\left(\frac{T}{S}\right)\left(\frac{S}{2}\right) N$.

   (b) **Define OOS Set:** Form the test set $\bar{J}$ as the complement of J in M, once again in the original order.

   (c) **IS Performance Stats:** Form a vector of R^c of performance statistics of order N, where the N^{th} component R^c_n of R^c reports the performance associated with the n^{th} column of J.

   (d) **OOS Performance Stats:** Repeat [c] for $\bar{J}$ to derive $\bar{R}^c$ and $\bar{r}^c$.

   (e) **Find Best IS Strategy:** Determine the element n^* such that $R^c_n \in \Omega^*_n$ - i.e. n^* is the best performing strategy IS.

   (f) **Calculate Relative Rank of OOS Strategy:** Define the relative rank of $\bar{r}^c_{n^*}$ by $\bar{\omega}_c := \frac{\bar{r}^c_{n^*}}{N+1} \in (0,1)$. This is the relative rank of the OOS performance associated with the strategy chosen IS, which should systematically outperform OOS if no backtest overfitting has taken place.

   (g) **Calculate Logit Value:** Define the logit $\lambda_c = \ln \frac{\bar{\omega}_c}{(1-\bar{\omega}_c)}$. High values here indicate consistency between IS and OOS performances (and so low overfitting).

5. **Measure Overfitting:** The logit values can now be used to compute the distribution ranks of OOS, by collecting all λ_c for $c \in C_S$. The relative frequency for λ occurring across all C_S is:

$$f(\lambda) = \sum_{c \in C_S} \frac{\chi_\lambda(\lambda_c)}{\#(C_S)}, \tag{74}$$

   where χ is the characterization function and $\#(C_S)$ is the number of elements in C_S, and so $\int_{-\infty}^{\infty} f(\lambda) d\lambda = 1$.

Result: Logit distribution.

The CSCV framework and results thus allows the consideration of several notable statistics. First and foremost, the PBO may now be estimated using the CSCV method and using an integral over the $f(\lambda)$ function as defined above, which offers a rate at which the best IS strategies underperform the median of OOS trials. The PBO is estimated using

$$\Phi = \int_{-\infty}^{0} f(\lambda) d\lambda. \tag{75}$$

If $\Phi \approx 0$, it is evidence of no significant overfitting (inversely, $\Phi \approx 1$ would be a sign of probable overfitting). Critically then, a PBO measure may be used in a standard hypothesis test to determine if a model should be rejected or not. This can be extended, as shown by Bailey et al. [8], to show the relationship between overfitting and performance degradation of a strategy. It becomes clear that with models overfitting to backtest data noise, there comes a point where seeking increased IS performance is detrimental to the goal of improving OOS performance.

4.8 Deflated Sharpe Ratio

The crux of the DSR assessment lies on the acknowledgement that a given set of SR estimates will have a greater maximum than zero, even if the true SR is zero, i.e testing the null hypothesis of $H_0 : SR = 0$.

In order to reject this, the observed SR must exceed the expected SR after controlling for the number of trials. Using the work noted in Section 2.6.1, Lopez de Prado and Lewis [90] combine several ideas allowing the assessment of the Sharpe ratio under multiple trials. The Optimal Number of Clusters (ONC) algorithm is used to cluster the trials into hierarchical independent trial groups (Section 4.8.1). Theses clusterings can then be used to estimate the trial set variance and number of trials, which can in turn be used to estimate the maximum likely Sharpe ratio expected under the False Strategy Theorem (Section 4.8.3). The Deflated Sharpe Ratio can be calculated by using this as the benchmark for the Probabilistic Sharpe Ratio, which gives a likelihood of whether the best observed $\widehat{SR}$ is a false positive or not (Sections 4.8.2 and 4.8). These methods are explained in full detail below.

4.8.1 Optimal Number of Clusters Algorithm

The primary contribution by Prado and Lewis is the Optimal Number of Clusters (ONC) approach to estimating the number of independent trials, such that the correct variances between strategies can be calculated. The methodology aims to partition a series of returns into clusters with higher intra-cluster correlations, and low inter-cluster correlations.

The initial data, N strategies each with their own return series $\mathbf{r}$, go through the following steps for pre-processing:

1. **Construct the Correlation Matrix**: ρ is constructed, an $N \times N$ correlation matrix, such that each element at $\rho_{\mathbf{ij}}$ represents the correlation between $\mathbf{r_i}$ and $\mathbf{r_j}$.

2. **Construct the Distance Matrix**: A proper distance matrix is calculated with which the clustering can be done. The proper distance satisfies the necessity that higher correlations will have closer distances, as well as the classical axioms of non-negativity, identity, symmetry and sub-additivity:
$$\mathbf{D_{ij}} = \sqrt{\frac{1}{2}(1 - \rho_{ij})}. \tag{76}$$

3. **Calculate Global Distance Matrix**: The ONC algorithm works on a distance of distances matrix, such that the distances incorporate information about the entire set of returns, which in turn reduces noise and increases the robustness of the clustering [88]. The distance matrix $\tilde{\mathbf{D}}$ is then calculated such that:
$$\tilde{\mathbf{D}}_{\mathbf{ij}} = \sqrt{\sum_k (\mathbf{D}_{ik} - \mathbf{D}_{jk})^2}. \tag{77}$$

Using the distance matrix $\tilde{\mathbf{D}}$, a modified K-means clustering algorithm is run, which addresses two concerns with a typical K-means implementation: the number of clusters needs to be set prior to running, and the random initialization can lead to similarly random efficacy.

The first problem is addressed using an objective function which can then be optimised. The silhouette score, introduced by Peter [105] is used as base to measure the quality of an individual strategies' assigned cluster:
$$\mathbf{S_i} = \frac{b_i - a_i}{\max(a_i, b_i)}, \tag{78}$$
where a_i is the average distance between i and all other nodes in the same cluster, and b_i is the smallest average distance between i and all the nodes in any other cluster. $\mathbf{S}_i$ then ranges between -1 (poor fit) and 1 (good fit), and incorporates a measure of both inter-cluster and intra-cluster quality. This allows the definition of the objective function to be used, where for a given cluster:
$$\mathbf{q} = \frac{\mathbb{E}[\mathbf{S}_i]}{\sqrt{\mathbf{Var}[\mathbf{S}_i]}}. \tag{79}$$

The initialization problem is, in part, addressed through the use of nested clustering. Typical K-means clustering is run for k = 2, ..., N-1 clusters, for which each clustering is evaluated according to $\mathbf{q}$ (equation (79)). This process is itself repeated a number of times, thus obtaining the results for different

random initializations. Naturally, the clustering results with the highest $\mathbf{q}$ are used for the steps that follow.

A final modification is made in order to address lower quality clusters that may have occurred as a result of the initial clustering capturing larger and broader clusters, but missing smaller and more distinct ones. In order to assess this, any cluster k where $\mathbf{q}_k < \bar{\mathbf{q}}$ has its components aggregated into a partition, K_1, with the remaining components in K_0. The components in K_1 are then run through the full ONC algorithm. This recursive approach may produce a better clustering, and so the $\bar{\mathbf{q}}$ for the cluster before and after the recursive step are compared. If the new clustering is better, then the new clusters for K_1 are combined with the original K_0 clusters. If not, then the original clustering is used as the final result. These clusterings will be used below, where both the number and variances thereof will be used.

4.8.2 Sharpe Ratio Normality and Probabilistic Sharpe Ratio

While it has often been assumed in the past that strategy returns will follow a Normal distribution, it has been shown empirically that they often do not [16], which may in turn lead to an underestimation of false positive probability. Under the assumption of IID and non-normal returns, Mertens [95] was able to show the asymptotic distribution of measured Sharpe ratio $\widehat{\text{SR}}$ for a strategy with returns $r_t, t = 1, ..., T$, as

$$(\widehat{\text{SR}} - \text{SR}) \xrightarrow{a} \mathcal{N}\left[0, \frac{1 + \frac{1}{2}\text{SR}^2 - \gamma_3 \text{SR} + \frac{\gamma_4 - 3}{4}\text{SR}^2}{T}\right], \tag{80}$$

where

1. γ_3 is the skewness of r_t, and
2. γ_4 is the kurtosis of r_t.

Christie [17] and Opdyke [101] showed that this holds under the more general assumption of returns being stationary and ergodic, but not necessarily IID. Thus, $\widehat{\text{SR}}$ follows a Normal distribution, with a variance that depends on the skewness and kurtosis of returns, which in turn allows the metric to be expressed in a probabilistic space. Bailey and Lopez did so, producing the Probabilistic Sharpe Ratio (PSR) measure, an adjusted $\widehat{\text{SR}}$, removing the inflationary effect of short series with skewed and/or fat-tailed returns.

$$\widehat{PSR}\left[\text{SR}^*\right] = Z\left[\frac{\left(\widehat{\text{SR}} - \text{SR}^*\right)\sqrt{T-1}}{\sqrt{1 - \hat{\gamma}_3 \widehat{\text{SR}} + \frac{\hat{\gamma}_4 - 1}{4}\widehat{\text{SR}}^2}}\right], \tag{81}$$

where

1. T is the number of observed returns,
2. $\widehat{\text{SR}}$ is the non-annualized estimate of SR, computed on the same frequency as the T observations,

3. SR^* is the user provided benchmark,
4. $Z[.]$ is the CDF of the standard Normal distribution,
5. $\hat{\gamma}_3$ is the skewness of r_t, and
6. $\hat{\gamma}_4$ is the kurtosis of r_t.

This PSR calculation then gives the probability that the observed $\widehat{\text{SR}}$ exceeds SR^*. Expectedly then, PSR will increase with $\widehat{\text{SR}}$, T and $\hat{\gamma}_3$, but decrease with $\hat{\gamma}_4$.

4.8.3 False Strategy Theorem

Bailey and Lopez de Prado [6] provided a proof for the following statement, allowing for PSR to be properly adjusted for multiple trials:

$$\text{E}\left[\max_k\{\widehat{\text{SR}_k}\}\right]\left(\text{V}\left[\left\{\widehat{\text{SR}_k}\right\}\right]\right)^{-1/2} \approx (1-\gamma)Z^{-1}\left[1-\frac{1}{K}\right]+\gamma Z^{-1}\left[1-\frac{1}{Ke}\right], \tag{82}$$

where

1. $\widehat{\text{SR}_k} \sim \mathcal{N}\left[0, \text{V}\left[\left\{\widehat{\text{SR}_k}\right\}\right]\right]$,
2. $Z^{-1}[.]$ is the inverse CDF of the standard Normal distribution,
3. e is Euler's number, and
4. γ is the Euler-Mascheroni constant.

The critical takeaway from this, is that unless $\max_k\{\widehat{\text{SR}_k}\} > \mathbb{E}\left[\max_k\{\widehat{\text{SR}_k}\}\right]$, then it is likely that $\max_k\{\widehat{\text{SR}_k}\}$ represents a false positive trial.

4.8.4 Deflated Sharpe Ratio

Prado and Lewis use the Probabilistic Sharpe Ratio and the False Strategy Theorem (FST) to define the Deflated Sharpe Ratio (DSR) as $\widehat{PSR}[\text{SR}^*]$: the probability that the true SR exceeds SR^*, which no longer needs to be user defined. Following from the FST statement in equation (82),

$$\text{SR}^* = \sqrt{\text{V}\left[\left\{\widehat{\text{SR}_k}\right\}\right]}\left((1-\gamma)Z^{-1}\left[1-\frac{1}{K}\right]+\gamma Z^{-1}\left[1-\frac{1}{Ke}\right]\right), \tag{83}$$

where

1. $\text{V}\left[\left\{\widehat{\text{SR}_k}\right\}\right]$ is the variance across trials' estimated SR,
2. K is the number of independent trials,
3. $Z[.]$ is the CDF of the standard Normal Distribution, and

4. γ is the Euler-Mascheroni constant.

For any set of trials with strategy returns, it is expected that there will be some maximum $\widehat{\text{SR}}$ that is more than zero, even if the true SR is zero. This maximum can be defined using equation (83) for SR*. The PSR from equation (81) can then be used to assess if the observed SR is statistically significantly larger than the expected maximum, and in doing so possibly reject the null hypothesis that $H_0 : \text{SR} = 0$. Thus, DSR gives us a confidence level as to whether the observed SR is legitimate and not just a false positive.

4.8.4.1 Estimating $\mathbb{E}[\mathbf{K}]$ and Var $\left[\widehat{\mathbf{SR_k}}\right]$

The results from the ONC algorithm, having already determined $\mathbb{E}[K]$ by clustering the trials into K clusters, can then be used to estimate the variance of the cluster Sharpe ratios. In order to do so, a single time series return needs to be constructed for the strategies in each cluster. The minimum variance allocation from Lopez de Prado [88] is used in order to choose the best weights and reduce the effects of strategies with large variances.

For each cluster k, where $k = 1, ..., K$, with components C_k, the weights for each component can be calculated using equation (84):

$$\mathbf{w_k} = \frac{\Sigma_k^{-1} 1}{1' \Sigma_k^{-1} 1}, \tag{84}$$

where Σ_k is the covariance matrix for the components of C_k.

Using these weights, the cluster return time series can then be calculated using equation (85):

$$\mathbf{S}_{k,t} = \sum_{i \in C_k} \mathbf{w}_{k,i} \mathbf{r}_{i,t}, \tag{85}$$

where $r_{i,t}$ is the return series for component i.

With a single time series for each cluster k, $\widehat{\text{SR}}_k$ can be calculated normally (as per equation (71)), and then annualized using equation (86). In this paper, all strategies have the same trading frequency of $\frac{T_k}{Years_k}$ (as per the MMS presented in Section 4.6), though other implementations may need to adjust the frequency for each cluster. The annualized Sharpe Ratio (aSR) is calculated as:

$$\widehat{\text{aSR}}_k = \frac{\mathbb{E}[\{S_{k,t}\}] \text{ Frequency }_k}{\sqrt{\mathbf{Var}[\{S_{k,t}\}] \text{ Frequency }_k}} = \widehat{\text{SR}}_k \sqrt{\text{Frequency}_k}. \tag{86}$$

These annualized Sharpe ratios for each cluster can now be used to calculate

the inter-cluster variance using equation (87), with their frequency adjusted to match the SR^* presented for DSR in equation (83):

$$\mathbb{E}\left[\mathbf{Var}\left[\left\{\widehat{\mathrm{SR}}_k\right\}\right]\right] = \frac{\mathbf{Var}\left[\left\{\widehat{\mathrm{aSR}}_k\right\}\right]}{\mathrm{Frequency}_{k^*}}. \tag{87}$$

The results from equation (87) and K can then be plugged into equation (83) in order to calculate SR^*, the expected maximum SR from the set of trials. This SR^* with the observed maximum $\widehat{\mathrm{SR}}$ can then be plugged into equation (81) to calculate PSR, and assess the likelihood of the observed $\widehat{\mathrm{SR}}$ being a false positive.

5 Framework Implementation

5.1 Full Framework Process

Having covered the technical implementations in Sections 3 and 4, a full overview of the end to end experimental process can be detailed and discussed. The process here rests on two key principles: implementing a generalised version of a system which could offer exploration of more complex techniques, and ensuring an effective modularisation of steps such that the process can be reconfigured accordingly while maintaining its integrity. In doing so, a separable system is created which brings together data reduction, deep learning with pretraining/weight initialization, online learning and backtest overfitting validation.

The steps below detail the conceptual stages for the full end to end process. These are also shown visually in Figure 5, with the diagram numbers corresponding to stages or items produced by each of the steps below.

1. **Process Data:** The raw market data, $\tilde{\mathbf{P}}$ from the JSE (see Section 7.1.1), goes through data processing, as per the steps below in Section 5.6, to form $\mathbf{L}$ (a log differenced and aggregated time series).
2. **Split Data:** $\mathbf{L}$ is split to form the Training and Prediction datasets, $\mathbf{L}^{\mathrm{I}}$ and $\mathbf{L}^{\mathrm{O}}$.
3. **Train SAE Networks:**
   (a) Either RBM pretraining or variance based techniques are used to initialize the network weights (see Sections 4.3 and 4.5, respectively).
   (b) $\mathbf{L}^{\mathrm{I}}$ is split into two datasets: $\mathbf{S}^{\mathrm{T}}$ and $\mathbf{S}^{\mathrm{V}}$, used to train and validate the SAE networks, respectively.
   (c) The networks are trained using SGD (see Section 4.2.3) and $\mathbf{S}^{\mathrm{T}}$, and validated using the MSE score on $\mathbf{S}^{\mathrm{V}}$.
4. **Select Best SAE Network(s):** The network MSE scores can be used to select the best SAE networks to be used for the following steps. Only the best need be selected out of each configuration category that is desired to be carried forward (e.g. 1 for each configuration of data horizon aggregations used).
5. **Encode Datasets:** The Training and Prediction datasets, $\mathbf{L}^{\mathrm{I}}$ and $\mathbf{L}^{\mathrm{O}}$, are encoded using the SAE networks to form $\mathbf{E}^{\mathrm{I}}$ and $\mathbf{E}^{\mathrm{O}}$.
6. **Train Predictive Network on IS Data:** The predictive network is initialized using variance based techniques, and trained using SGD on $\mathbf{E}^{\mathrm{I}}$. After training, $\mathbf{E}^{\mathrm{I}}$ is used to generate and record the IS predictions, $\hat{\mathbf{L}}^{\mathrm{I}}$.
7. **Train Predictive Network on OOS Data:** The predictive network is trained on $\mathbf{E}^{\mathrm{O}}$ using OGD (see section 4.2.4). The predicted outputs from the training, denoted $\hat{\mathbf{L}}^{\mathrm{O}}$, are also recorded.

8. **Reconstruct Price Predictions:** Price prediction sets $\hat{\mathbf{L}}^{\mathrm{I}}$ and $\hat{\mathbf{L}}^{\mathrm{O}}$ are processed to reconstruct actual timepoint prices using the steps in Section 3.1.4, denoted $\hat{\mathbf{P}}^{\mathbf{I}}$ and $\hat{\mathbf{P}}^{\mathrm{O}}$, respectively.
9. **Calculate MMS Variables:** $\hat{\mathbf{P}}^{\mathrm{I}}$ and $\hat{\mathbf{P}}^{\mathrm{O}}$ are used by the MMS to calculate variables at each time point, as per Section 4.6. One of the primary calculations is the trade return rates, which can be used to calculate the Sharpe ratio.
10. **Calculate CSCV and PBO:** The CSCV algorithm is run using both the $\hat{\mathbf{P}}^{\mathrm{I}}$ and $\hat{\mathbf{P}}^{\mathrm{O}}$ MMS return rates in order to calculate a PBO score, as detailed in Section 4.7.
11. **Calculate DSR:** The CSCV algorithm is run using $\hat{\mathbf{P}}^{\mathrm{O}}$ MMS return rates to calculate a DSR score, as detailed in Section 4.8.

5.2 Process Diagram

The flow diagram below provides a graphical representation of this process described above in Section 5.1 (the MMS calculations and CSCV process are not included). The process may be repeated accordingly in terms of SAE structures that are desired for testing, such that the "Best SAE Network" is different for different encoding layer sizes, or data aggregations used (as was the case in many of the experiment configurations which will be discussed in Section 8).

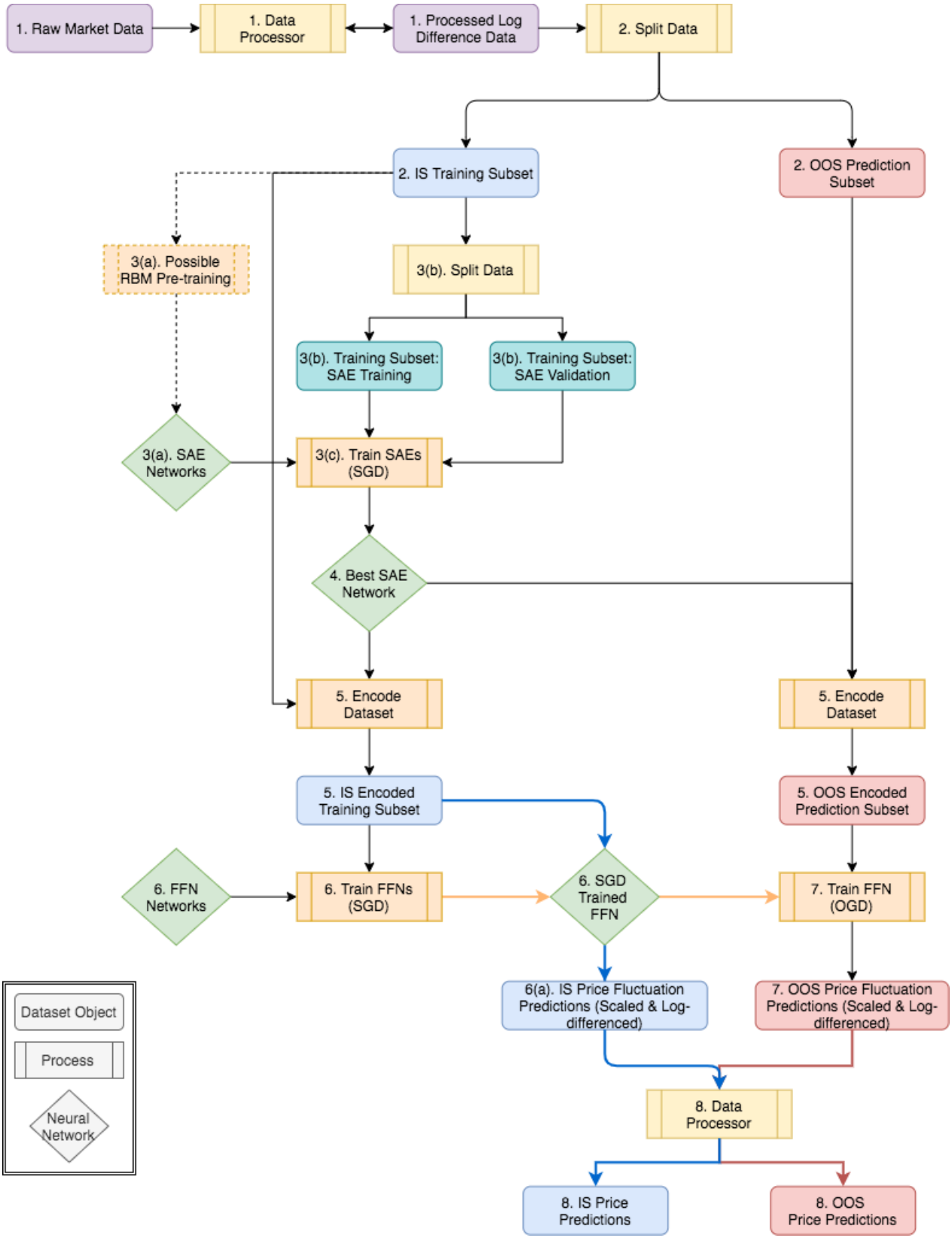


Figure 5: Overall Process Flow
The flow diagram here offers a visual representation of the process set out in Section 5.1, with the numbers corresponding to the steps of the process.

5.3 Parameter Space Exploration

The parameter space is explored using a phased grid search approach. For each stage, the relevant parameters are specified as a set of values (or one, if so chosen), and all sets are then used to generate the full combinatorial space, such that each possible combination of the specified parameters is tested.

1. **Stage 1:** The data configuration (i.e. data windows, prediction point, scaling, data split points) as well as the SAE configuration (network size, learning rates, learning optimization parameters, SGD epochs) are set in Stage 1 and used to train the SAE networks.
2. **Stage 2:** The preferred SAEs are chosen from Stage 1 manually, and determine the data configuration used for Stage 2. These are then used to encode the datasets, which will be used for FFN training. The FFN SGD and OGD parameters are set in this stage (network size, learning rates, SGD epochs etc.), and will be combined combinatorially with the SAEs that were chosen for testing as well.

5.4 Reproducibility

Reproducible results were ensured with the use of set random data seeds and a Mersenne Twister pseudorandom number generator. Each SAE training process is initialized with a dataseed which is recorded in the `dataset_config` schema (see Section 6 for more details). These are propagated through to the FFN training, such that any of the results presented could be reproduced using the data seeds and configurations noted in Appendix 11.5 (which are all detailed in the relevant figure captions as well).

5.5 Synthetic Data

The use of synthetic GBM generated data offers a way of exploring the parameter space without running configuration experiments on the actual data, which might then raise the probability of backtest overfitting. The synthetic data has enough similarity that it can allow a broad exploration to get ideas of optimization usage and parameter choices. This is an appealing option in light of the concerns around increasing the number of particularly poor performing networks when it comes to calculating PBO, as discussed in Section 8.9. That said, there are characteristic differences between GBM generated data and actual financial time series. With these in mind, the results section will make comparisons of the two in order to offer a sufficient understanding for when synthetic data can be relied upon to guide future training.

The process for generating synthetic data is detailed in Algorithm 1 (Section 3).

5.6 Data Preparation

The full dataset goes through several steps of pre-processing - these are covered in more detail in Section 3.1, though the steps are included below.

1. The data is processed into day to day log fluctuations (see equation (1)).
2. At each time point, rolling historic fluctuation summations are calculated (e.g. the past 1, 7 and 30 days, see equation (2)).
3. At each time point, rolling future fluctuation summations are calculated (e.g. the next 5 days, see equation (3)).
4. The dataset is truncated to only include points with all aggregations or predictions available.
5. The dataset undergoes scaling - the different functions are detailed in Section 3.1.2, though Limited Normalization (as per equation (7)) was used for most of the results produced.

5.6.1 Data Window Aggregations

The time aggregations were chosen according to domain knowledge - it is expected that prices may move in daily, weekly, fortnightly, monthly and quarterly patterns as trading is actioned by day traders such as speculators to institutional investors such as mutual funds. Indeed, it can be shown that markets can be modelled such that they are hierarchical in space and time, with actors at different hierarchical levels within the market interacting under different causative structures [127]. The effect is such that price trends and patterns can be expected to exist in time at multiple levels, with price being set as an emergent property of these hierarchies. Taking this into account, variations of 1, 5, 10, 20 and 60 working days were considered throughout the configuration tests, in the following combinations:

1. Horizon Tuple 1: [1, 5, 20],
2. Horizon Tuple 2: [5, 20, 60], and
3. Horizon Tuple 3: [10, 20, 60].

The calculations for the aggregated values can be seen in equations (2) and (3).

5.6.2 Point Predictions

Various configurations were tested in terms of the historic summations used, while the fluctuation predictions were set at 5 days. It's worth noting that the prediction horizon is configurable, and in the case of implementing a more complex trading strategy, one would likely choose to predict multiple points across the horizon in order to to develop a distribution. In this case, the MMS was kept purposefully simple, and so didn't warrant the more intensive implementation.

One point of concern is that point predictions at 5 days in advance could open the process up to aliasing error, as it is unlikely to be predicting at the Nyquist frequency, and may end up fitting to the band ripple rather than the underlying signal. Conversely, 1 day predictions for closing price might offer a high degree of accuracy, but are not practical for trading due to market auction and bidding structures. The system implementation veered on the side of practicality in this case, and was configured for longer term predictions despite the risk of aliasing. While some of the configurations were run with 2 day predictions, it was seen that once costs were considered it was difficult for even the benchmark to produce desired P&L figures and so a 5 day prediction was implemented in subsequent tests.

5.6.3 Scaling

When testing the two limited scaling methodologies, standardization and normalization, it was found that the results for standardizing were significantly worse than normalization (discussed in Section 8.6). Outliers in price fluctuations and changing variance through time (which is not captured effectively through the limited variation) resulted in a less informative representation and worse performance of the FNN models which can be prone to error maximisation. As a result of this, the limited normalization process was used for most of the tests run. It's worth noting that an implementation which incorporates processing of the data through the Error Function in order to reduce the impact of outliers would likely prove a worthwhile endeavour, and could be easily swapped in due to the systems modularity. The implementation for "Limited Normalization" can be seen in equation (5).

5.7 Data Partitioning

Once processed, the dataset is split into 2 portions:

1. Training: The first set is used for the SAE training, as well as the initial SGD training on the predictive FFN network. This is the IS dataset and is used as what would typically be the historical training set.
2. Prediction: The second set is used only for the OGD training in order to generate returns from the predictive network. This is the OOS dataset and is used as what would typically be the testing / validation set.

The predictions from both the Training and the Prediction sets are used by Bailey's CSCV method to assess overfitting (the use of the CSCV technique negates the need for a hold-out portion of the dataset [8]), however only the OOS prediction returns are used in order to assess DSR or model P&L figures and performance.

In this case, a simple split point of 60:40 was chosen for the Training and Prediction sets, respectively. In a more restrictive case, where there is less data

to work with, it would be advised to consider a method such as MinBTL as developed by Bailey et al. [7] and the estimated number of configurations to be tested in order to decide an appropriate split point.

5.8 Unsupervised Learning: SAE Training

The Training dataset is used to train the AutoEncoder network using either RBM pretraining or weight initialization algorithms and SGD training, as defined in Section 4.4. The Training portion of the dataset, as noted in Section 5.7, is itself split up into a training and validation set for the SAE. The efficacy of the SAE on the validation set is required in order to determine which networks to choose for subsequent steps in the process. The full testing process here will rely on a generated set of best SAE networks at each encoding layer size, to be used for FFN training and prediction (chosen by a minimum MSE score). The benefit of the modularised system is emphasised here, as the SAE training will not suffer from limitations due to backtesting considerations: any amount of configurations or processes can be tested for feature extraction without concern.

Once the SAE networks have been defined and chosen, they can be used to reprocess both the Training and Prediction datasets such that the input is encoded, and the output is as before. These encoded datasets can then be used for the subsequent steps in the process.

In a productionized system, where data is updated daily, there would be a set period after which the SAE network would be retrained to reflect more recent data. This step was not deemed necessary at the outset of developing this process, but results detailed in Section 8.2 suggest that it is an important implementation step.

5.9 Supervised Learning: Prediction Network Training

Typically, the SAE and predictive network might be presented as one and the same, however in the process presented we are able to separate the two and optimise for feature extraction prior to training for predictions. Having done so, the datasets can have their input encoded, but retain the same output. The predictive network is then trained using the Training set, as detailed in Section 4.2.3. There's no requirement to validate this part of the training, so the dataset is not split into subsets as it was for SAE training.

Once the predictive network is trained, the OGD process (as per Section 4.2.4) is run through the encoded Prediction dataset in order to generate the predictions for the asset prices that the model produces - thus emulating what

would have occurred in a live environment.

5.10 Price Reconstruction

In the interest of more interpretable results for figures such as P&L, the price fluctuations for both IS and OOS predictions are re-scaled back to their original price fluctuation predictions using the reverse scaling detailed in Section 3.1.3. These prices are then reconstructed using equation (12), such that they represent a predicted future price of the asset which can then be used by the MMS. It's worth nothing that the price reconstruction applies the predicted fluctuation to the known original price, such that it would represent a system that is run daily, rather then a system that is making rolling predictions over a period of time.

5.11 Money Management Strategy

The MMS, as described in Section 4.6, takes a proof of concept approach to the system being investigated. It is a simple long strategy with a set liquidation point, and no notable refinement. The naive approach is taken purposefully here, so as not to bias the perspective of the system as a whole by the effects of an impactful trading strategy. A more effective trading strategy would incorporate a shorting side as well, though ultimately this is not necessary in order to obtain a necessary indication of efficacy for the system (and conceptually, a consistently losing long strategy would be of interest in any case).

It is important, in the interest of effective optimization and development, that the pattern recognition in the prediction portion of the system is not tightly coupled with making it profitable. With this in mind, the modularity of the system is continued, with a distinct separation between the prediction signal and the MMS implementation which relies only on the predicted prices of each stock for a time point. The comparisons for the MMS are made against a benchmark of the same MMS, but with perfect information. This allows an indication for how effective the prediction system is (and need be) in order to generate comparable returns.

An effective production trading strategy would be likely to include stop losses or pyramiding stop losses for effective P&L optimization. An ideal implementation, and possible area for further work, would include an MMS strategy where the bands for this are defined using a machine learning technique (rather than being chosen from domain knowledge or heuristics).

The calculated variables used by the MMS are fully detailed in Section 4.6.

5.12 Probability of Backtest Overfitting

The CSCV and PBO parts of the system are essentially just an implementation of Algorithm 6 detailed in Section 4.7, though with a somewhat novel application to the predictive network and MMS combination. The CSCV process uses the IS and OOS returns from the MMS, which in turn used the prices from the predictions network, which then constitutes the M matrix used by the algorithm. Conceptually, the whole system comes into place here, as the results from the CSCV process are now indicative of not only backtest overfitting in the trading strategy, but also in the prediction network and without having to consider the impact of the many configuration tests for feature extraction.

5.13 Deflated Sharpe Ratio

The DSR calculations are a straight forward implementation of the process described in Section 4.8, with only minor changes from the authors recommendations. Unlike the CSCV process, the DSR calculation only uses returns generated from the predictive model and MMS on OOS data. As with the CSCV & PBO calculations, the DSR now offers an effective assessment of the experimentations results and their legitimacy. The value of a modularised approach is highlighted again, where the DSR process is not coupled to any aspect of the neural network training.

6 Software Libraries and Development

A significant aspect of this dissertation was the implementation of software libraries that fulfilled the criteria of generalised training of neural networks, an implementation of the full process as described in Section 4.1, the creation and management of database records for analysis and an extensive set of diagnostic tools in order to assess performance. This section details the implementation choices as well as the final libraries which were produced. A tutorial for using the provided libraries is included in the `Main_Tutorial` module, and discussed in Appendix 11.2.

6.1 Online Availability

As per the noted references, all the Julia and Python code developed has been made available via a Github repository and through Zivahub [46], including the JSE data used to run the trials (see Section 7.1.1). Unfortunately, it was not possible to host the full database that was created as it had a size of over 60GB. A truncated version of the database is made available [22], containing all of the aggregated performance results for configurations run on the "Actual10" dataset (see Section 7.1.1). The graphs contained in this paper can be reproduced using the code in the `ReportGraphs` module [43].

6.2 Programming Languages

The libraries developed were primarily written in Julia, a relatively new scientific computing language (initially developed in 2009) [46, 19]. The main motivation, both for Julia as a language as well as choosing it for this task, is its efficiency and speed. Julia was created with high performance tasks in mind, and with its use of a Just-In-Time (JIT) compiler, can approach similar benchmarks to C [104]. This stands in stark contrast to some of the more popular analytical languages used, such as R or Python, which often suffer from performance issues. In the context of a project which required extensive processing power, it was natural to prioritize this computational efficiency.

Julia's focus on technical computing is emphasised with its built in support for mathematical and statistical work, composable functional base and dynamic and optional typing schemes. Its ability to fulfil general purpose programming, familiar syntax (which also works effectively for mathematical commands) and its interactive (REPL) command line also make it a practical choice for the task at hand [104].

SQLite was chosen as the database management language. Relational schemas were a fitting choice for the nature of the data being captured, and SQLite offers a fast, effective and free implementation of SQL.

Python was used only for the DSR algorithm detailed in Section 4.8, so that the ONC code provided by Lopez de Prado and Lewis [90] could be used.

6.3 Data Generation & Processing

Data processing libraries were implemented to offer configurable processing for the steps outlined in Section 3. The `DataGenerator` module [27] implements Geometric Brownian Motion (GBM) generation for a set number of steps and assets with their mean and variances specified.

The `DataProcessor` module [28] offers a more extensive set of operations in order to implement the processing laid out in Section 3.1. The different scaling techniques (Normalize, Standardize, Limited-Normalize and Limited-Standardize) are implemented, as well as the methods to reverse scaling and reconstruct price predictions to the original format. The log differencing, aggregation of past and future horizons as well as dataset partitioning is also made available here, as per methodologies in Section 3. The module also includes dataset encoding through SAE networks, and adding Gaussian noise for the purposes of denoising training, as per Section 4.4.3.

The `DatabaseBatchProcesses` module [29] performs the analytical processing of asset price fluctuation predictions. The `RunBatchTradeProcess` function processes all predictions made, and writes the individual P&L observed and returns rates for each trade to the `config_is_trade_returns` and `config_oos_trade_returns` tables (detailed more in 6.7 below). The `RunBatchAnalyticsProcess` function uses the trade information to aggregate all strategy metrics, such as P&L and Sharpe ratios and also writes them to tables (e.g. `config_oos_pl` or `config_oos_sharpe_ratio`). These aggregate metrics are then used to perform the diagnostics and analyses in the rest of the paper.

6.4 Network Training

Neural networks were implemented with highly configurable training, such that the parameters are easily specified within Julia types [45] and then passed onto the training functions. The parameters are grouped accordingly to the different aspects of the training that they pertain to.

The network structures are easily configurable in terms of layers and layer sizes, as well as having the activation functions specified at each layer (i.e. linear, sigmoid, ReLU, Leaky ReLU, softmax, tanh as described in Section 4.2.2) [21]. The network weight initialization is also specifiable, with the noted initializations in Section 4.5 available: Normal Random, Xavier, He and He-Adj [39].

The Stochastic Gradient Descent (SGD) and Contrastive Divergence-1 (CD-1) learning algorithms, as per Sections 4.2.3 and 4.3.2, can be specified accord-

ing to minimum and maximum learning rates, and learning rate epoch cycles (as per the learning rate scheduling in Section 4.2.6). Training length can be specified in terms of maximum training epochs or a custom stopping function, which may rely on the Cost function hitting a particular point (with Mean Squared Error, Cross Entropy Error, Log Likelihood Error being made available [24]). Learning optimization hyperparameters for L1 Regularization or Denoising are also set here (implemented as per Sections 4.2.5 and 4.4.3). Dataset usage in terms of training / testing splits can also be specified accordingly.

The Online Gradient Descent (OGD) training parameters, as per OGD learning in Section 4.2.3, are specifiable through the learning rate and cost functions available.

The `NetworkTrainer` module [40] then offers 3 primary methods, given the relevant parameter configurations and dataset:

1. Train a Feedforward Neural Network, using variance based weight initializations.
2. Train an SAE, using variance based weight initializations.
3. Train an SAE, using RBM based pretraining for weight initialization.

These functions are supported by the following learning algorithm modules: `SGD` [44], `RBM` [42], `OGD` [41] and `GradientFunctions` [38].

6.5 Process Implementation

The process implementation, as described in detail in Section 4.1, is largely implemented in the `ExperimentProcess` module [33].

The `RunSAEConfigurationTest` method uses the `SAEExperimentConfig` Type to wrap up all necessary configurations and with a passed dataset runs through the following steps:

1. Either generate or process the raw dataset, according to the specified configuration.
2. Train the SAE, either with RBM or variance based weight initializations.
3. Pass back the SAE with original and reconstructed testing datasets.
4. Store the SAE as a bson object to be used later.

The `RunFFNConfigurationTest` method similarly uses the `FFNExperimentConfig` Type and a passed dataset to run through the following steps:

1. Either generate or process the raw dataset, according to the specified configuration.
2. Encode the datasets for the different steps of the process using the specified SAE.

3. Train the FFN using variance based weight initialization and SGD.
4. Use the FFN to generate predictions for IS data, reconstruct them and record them in the database.
5. Train the FFN using OGD on OOS data.
6. Use the FFN to generate predictions for OOS data, reconstruct them and record them in the database.

Two further modules allow the exploration of the configuration space, such that diagnostics can be performed on different aspects of the training. The `ExperimentProcessTrainFFN` [34] and `ExperimentProcessTrainSAE` [35] modules make use of the `ConfigGenerator` module [37] in order to generate a full grid configuration space for a given range of values for any chosen parameters, as per Section 5.3. Each of these combination configuration sets is then run as per the processes detailed above, with the various results recorded as usual.

6.6 Process Reproducibility

As noted in Section 5.4, it was ensured that the full process and results were reproducible. A Mersenne Twister pseudorandom number generator was set with a particular seed, which is recorded in the `dataset_config` schema in the database. This seed is propagated through from the SAE training process to the FFN training process. Consequently, any configuration set with the same parameters and random seed can be used to reproduce the results presented in this paper and otherwise.

6.7 Database Implementation

As noted, the database platform chosen was SQLite, with Julia modules handling the creation and management. The `DatabaseCreator` module [30] easily creates the database, with the schema names and purposes noted in Table 3, using the `CreateDatabase` function. The full relational schema diagram can also be viewed in Figure 60 in the Appendix.

The `DatabaseOps` module [31] handles the creation of records for these schemas, as well as the writing and reading of bson files which are used to store actual neural networks in the form of the `FFN` type [36].

6.8 Diagnostic Libraries

The diagnostic implementations were focused on 2 primary themes: Exploratory Data Analysis (EDA) through the use of visualizations of network results through configuration spaces, and formalised testing in the form of PBO and DSR.

The EDA makes use of the PlotlyJS library [92], and allows the net performance of any set of networks to be plotted along a particular dimension in

their configuration space. In the case of a set of SAE networks, this would typically be the MSE scores, and in the case of predictive FFN networks, this would typically be the P&L values generated from the MMS. A full list of diagnostic charts can be viewed in Appendix 11.6.

The CSCV and PBO methodologies can be found in the `CSCV` module [23], which encapsulates the generation and assessment of the logit distributions, as well as the PBO calculation.

As per Section 6.1, the graphs for the "Actual10" dataset contained in this paper can be reproduced using the `ReportGraphs` module [43] and the truncated version of the database made available [22].

6.8.1 DSR Libraries

Unlike the rest of the framework implementation, the DSR process was written in Python [55]. This was done so the code provided by Lopez de Prado and Lewis [90] for the ONC algorithm (see Section 4.8.1) could be used. The `DSR_PradoLewis` module [89] contains all the python code used which was presented in [90]. Modules `DSR_dataproc` [25] and `DSR_proc` [26] contain the code for returns processing and DSR calculations, respectively.

Schema Name	Description
configuration_run	The primary record for any trial configuration run.
dataset_config	The dataset configuration that may relate to a trial.
network_parameters	The network configuration for a trial.
training_parameters	The SGD/RBM/CD training configuration for a trial.
epoch_records	Details relating to ongoing training performance for the duration of a trial.
backtest_results	Predictions for IS data made by trial networks after SGD training.
prediction_results	Predictions for OOS data made by trial networks during OGD training.
config_is_trade_returns	MMS P&L observed and return rates for each IS trade for each trial.
config_is_trade_returns_cost	MMS P&L observed and return rates for each IS trade for each trial, with trading costs applied.
config_oos_trade_returns	MMS P&L observed and return rates for each OOS trade for each trial.
config_oos_trade_returns_cost	MMS P&L observed and return rates for each OOS trade for each trial, with trading costs applied.
config_is_pl	The total IS P&L for a trial.
config_is_pl_cost	The total IS P&L for a trial, with trading costs applied.
config_oos_pl	The total OOS P&L for a trial.
config_oos_pl_cost	The total OOS P&L for a trial, with trading costs applied.
config_oos_sharpe_ratio	The OOS Sharpe ratio for each trial.
config_oos_sharpe_ratio_cost	The OOS Sharpe ratio for each trial, with trading costs applied.
config_confusion	The percentages of correct trades made for a trial.
clusters	The clusters for the trials as determined by the ONC algorithm.

Table 3: Database schema descriptions, as discussed in 6.7.

7 Datasets Used

7.1 Actual Datasets

Several datasets have been constructed using JSE closing price relative data for 2003-2018 (more source information and a data snapshot are available in Appendix 11.4) [47]. They were processed following the steps set out in Section 3 and with a 60:40 split on the Training:Prediction subsets.

7.1.1 "Actual10"

The "Actual10" dataset is the primary closing price dataset that has been used throughout the experiment process. It was constructed with a focus on choosing prominent stocks from multiple sectors. The price fluctuations can be seen in Figure 6.

Code	Company	Sector
AGL	Anglo American	Resources
BIL	BHP Billiton	Resources
IMP	Impala Platinum Holdings	Resources
FSR	First Rand Limited	Finance
SBK	Standard Bank	Finance
REM	Remgro Limited	Finance
INP	Investec	Finance
SNH	Steinhoff International Holdings	Retail
MTN	MTN	Communication Services
DDT	Dimension Data	Technology

Table 4: Assets in the "Actual10" dataset, from 2003-2018.

7.1.2 "Scaling10"

This dataset was used specifically to test scaling techniques, and consisted of the following JSE closing price relatives: ACL, AGL, AMS, AOD, BAW, BIL, BVT, CFR, CRH, DDT. The price fluctuations can be seen in Figure 7.

7.1.3 "AGL" and "AGL&ACL"

The AGL and ACL data from JSE closing price relatives was used for some limited framework testing. The price fluctuations can be seen in Figure 8.

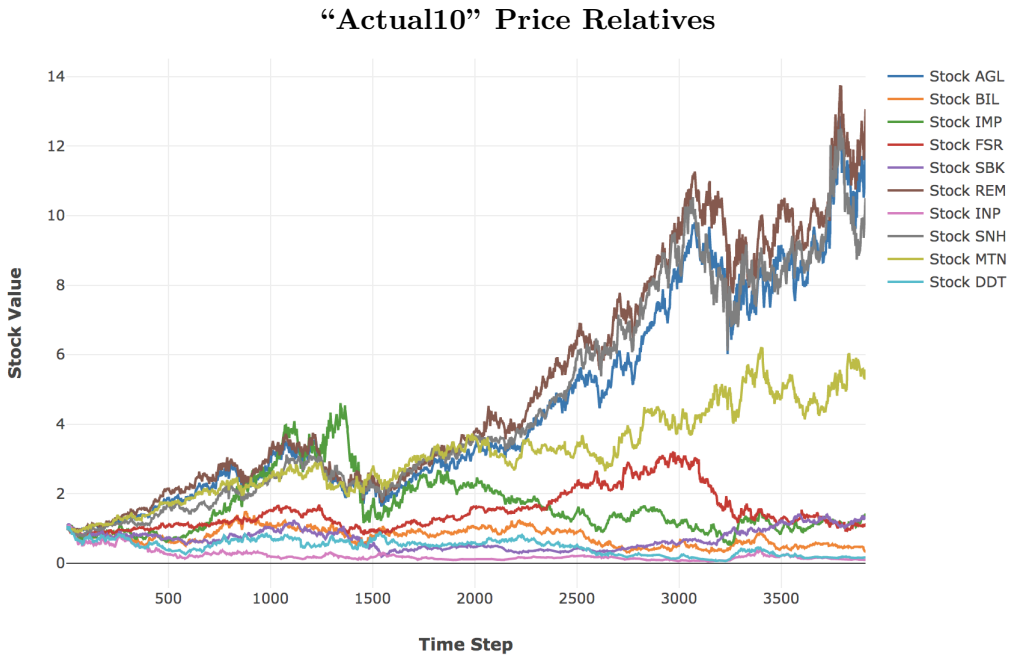


Figure 6: The primary dataset for actual price relatives is displayed here, as noted in Table 4. There is a varied collection of behaviours amongst assets, with upwards, downwards and sideways trends over the full period, as well as assets with very high and low variances. The dataset contains assets from most major sectors, and shows some strong correlations amongst groups of assets (e.g. AGL, REM and SNH).

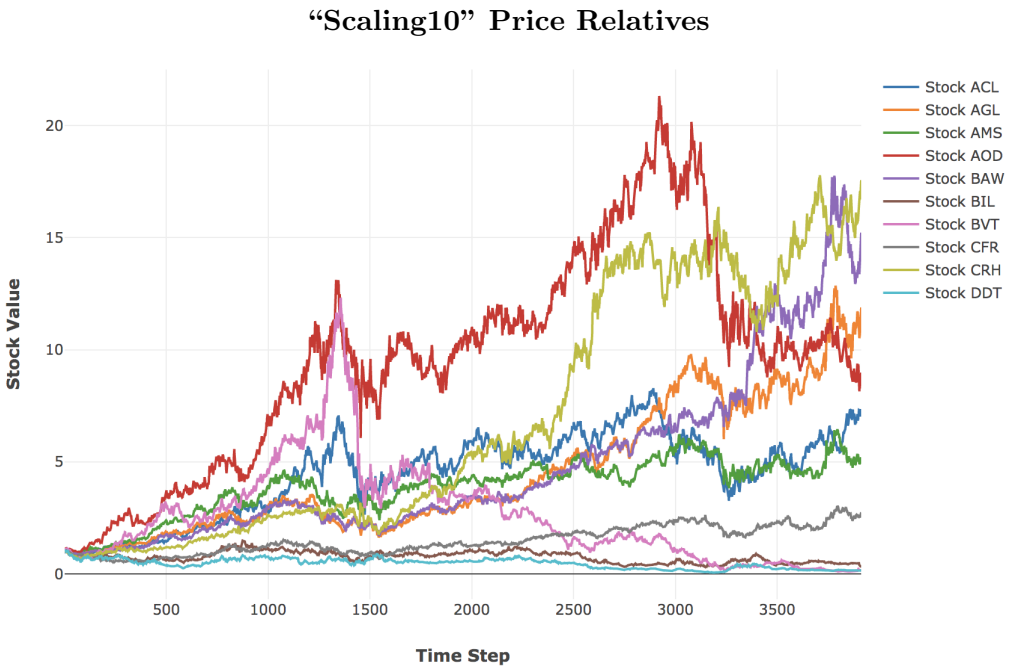


Figure 7: The figure here shows the prices relatives for the "Scaling10" Dataset as detailed in Section 7.1.2. The group of assets show a wide variety of behaviours, which was appropriate for the testing of training techniques on SAE networks. These results can be seen in Section 8.7.2.

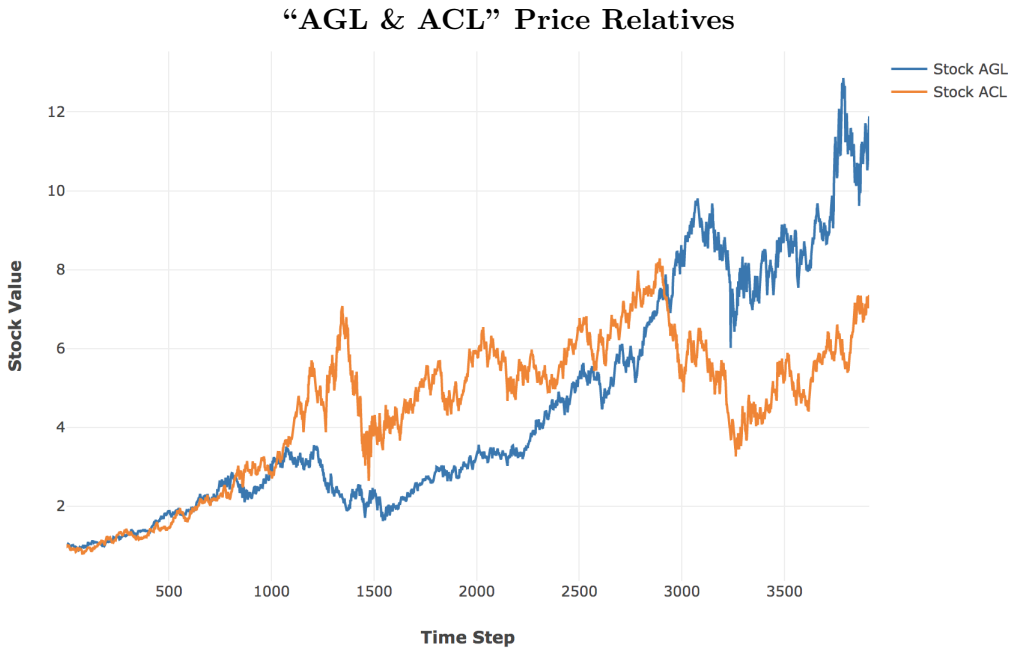


Figure 8: The figure here shows price relatives for AGL and ACL which make up the "AGL" and "AGL&ACL" datasets (7.1.3, 7.1.3), which were used for some limited testing (e.g. scaling as in Figure 27 and initializations as in Figure 22).

7.2 Synthetic Datasets

Data was generated and scaled, as per the methods detailed in Sections 3.2 and 3.1.2. The sets were generated with the following configurations, each for 5000 timesteps and with a 60:40 split on the Training and Prediction sets.

7.2.1 "Synthetic6"

The Synthetic6 dataset was configured to represent a combination of upwards, downwards and sideways Trends, each with high and low variance variations.

Trend Category	Variance Category	Trend Mean	Variance
Strong Upward	High	0.9	0.5
Strong Upward	Low	0.9	0.2
Upward	High	0.05	0.4
Upward	Low	0.05	0.1
Strong Downward	High	-0.8	0.55
Strong Downward	Low	-0.8	0.15

Table 5: Assets in the "Synthetic6" dataset, as detailed in Section 7.2.1.

7.2.2 "Synthetic10"

The Synthetic10 dataset was configured to represent a wide array of behaviours, including very high (positive and negative) mean stocks, as well as much lower mean stocks (which may be more representative of typical behaviour). These were chosen to make sure the network learning is able to differentiate and correctly learn across different asset categories.

Trend Category	Variance Category	Trend Mean	Variance
Strong Upward	High	0.9	0.5
Strong Upward	Low	0.7	0.2
Upward	High	0.05	0.5
Upward	High	0.05	0.4
Upward	Low	0.04	0.1
Sideways	Low	0.02	0.15
Sideways	Low	0.01	0.05
Downwards	Low	-0.1	0.2
Strong Downward	Low	-0.4	0.15
Strong Downward	High	-0.8	0.55

Table 6: Assets in the "Synthetic10" dataset, as detailed in Section 7.2.2.

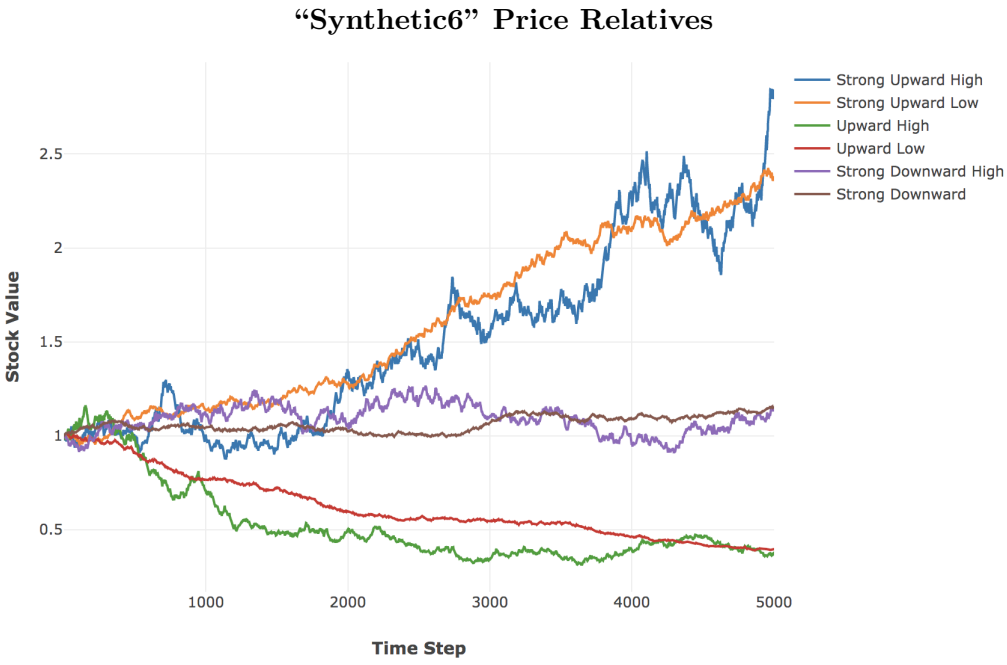


Figure 9: This figure shows the price relatives for the Synthetic6 dataset (Section 7.2.1), generated for 5000 days and with the attributes in Table 5. Assets were generated for high and low variances with upwards, downwards and sideways trends.

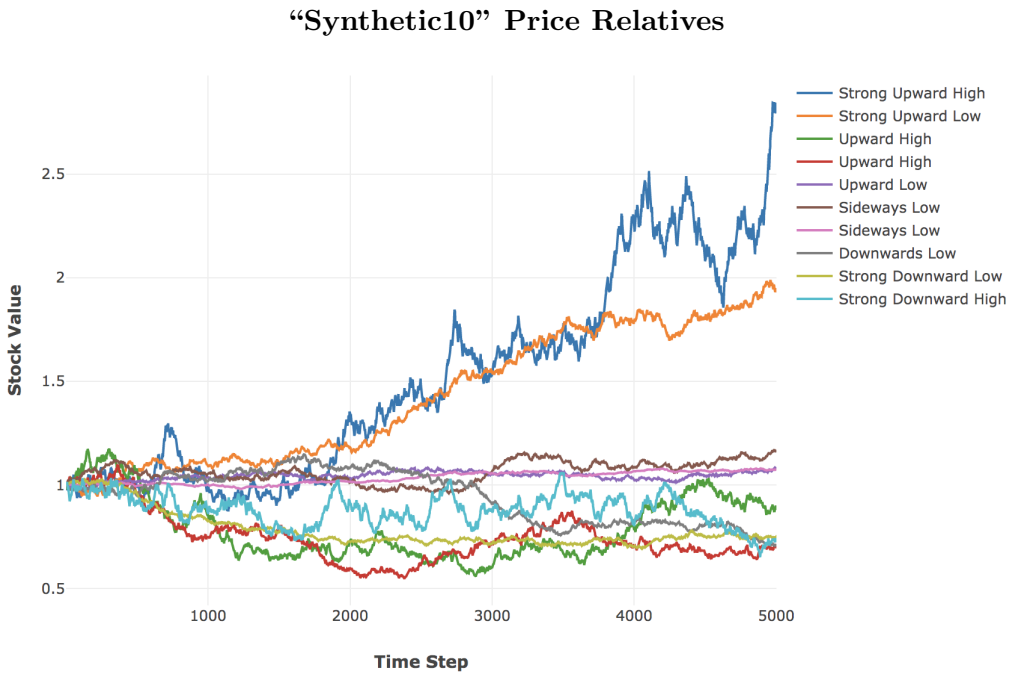


Figure 10: This figure show the price relatives for the Synthetic10 dataset (Section 7.2.2), generated for 5000 days and with the attributes in Table 6. A wide range of means and variances were chosen, such that the data might be representative of a non-synthetic dataset.

8 Results

8.1 Introduction

In this section we discuss the results seen at all stages of the framework. We provide takeaways in terms of technical training, as well developing a perspective on the phenomenology of financial markets, and how these both influence mechanistic learning approaches to pattern prediction.

OOS P&L: Section 8.2 discusses the primary determinants of OOS P&L seen in actual data results, covering data horizon aggregations, SAE feature selection and the OGD learning phase. We found that data horizon aggregations formed the primary determinant of the prediction strategies learnt, and that SAE networks learnt different classes of features according to the encoding size (which may differ in suitability for different prediction strategies). These are both considered in the context of the online OOS learning, which proved to be critical in effective learning. This is expanded on in Section 8.3, which shows the negligible impact that extensive training had on historical data, noting that reducing training epochs and datasets had limited effect, and that the online learning phase was most impactful for OOS P&L.

Network Initializations: Section 8.4 discusses network weight initialization techniques, noting their importance in context of financial time series where the value of extensive batch training was found to be of limited value (or even detrimental to performance).

Synthetic Data OOS P&L: Section 8.5 discusses the nature of Synthetic data, and how it differs in the key OOS determinants. We found that the fundamental differences in data distributions resulted in P&L performing better over longer data horizons (which was not definitive in actual data), and less complex features being learnt in SAE networks.

Complexity: The complexity of actual financial time series is discussed further in Section 8.6, where we found that results for data scaling and SGD learning optimizations were affected by the unstable nature of dynamics in financial markets.

Network Configuration: Section 8.7 gives an overview of the more typical network training components such as layer sizes and activation functions, and how they affected the performance results.

Trial Validation: The financial returns and Sharpe ratios from the prediction networks, as used by the MMS, are discussed in Section 8.8 with overall distributions shown with the relative benchmark. These results were validated and showed a low likelihood of backtest overfitting in Section 8.9, and a confidently positive Deflated Sharpe Ratio in Section 8.10.

Synthetica Data Assessment: Section 8.11 summarises the results presented for synthetic data throughout the framework, and notes its limited value in assessing behaviours for actual data.

Personal Learnings: Lastly, Section 8.12 covers some of the less quantifiable and personal learnings from the implementation of the project.

8.1.1 Technical Notes

While the predictive networks are trained using an MSE loss function, the performance will mostly be reported according to the P&L they were able to achieve through the MMS. Looking at Figure 11, we can see that while there is a strong correlation between predictive accuracy and P&L achieved, it is not absolute and there are many "worse" configurations according to OGD MSE which have higher P&L. This is partly due to OGD's potential to oscillate wildly in the beginning of training while it adjusts, and also representative of lower variance models being able to achieve competitive P&L. Also, in consideration of the task at hand, P&L is the primary metric of interest and so is a sensible analysis choice.

It is worth noting though that this P&L is a relative unit measurement, rather than a particular currency (as noted in Section 3.3), and that the MMS, as described in Section 4.6, only trades 1 stock unit at a time. As a result, P&L figures are relative indications rather than absolute limits. It is also usually reported without capital and trading costs (as per equation (52)) applied, though these costs are taken into account for the PBO and DSR validations.

The results for synthetic data are compared against actual data when appropriate, such that it might guide when synthetic data can and should be used for parameter space exploration (as discussed in Section 5.5). These results are discussed more completely in Sections 8.5 and 8.11.

Sample sizes for some configurations may differ greatly across results (particularly as results are aggregated across training sets and phases), though these are noted in the figures as well as the configurations in the appendix.

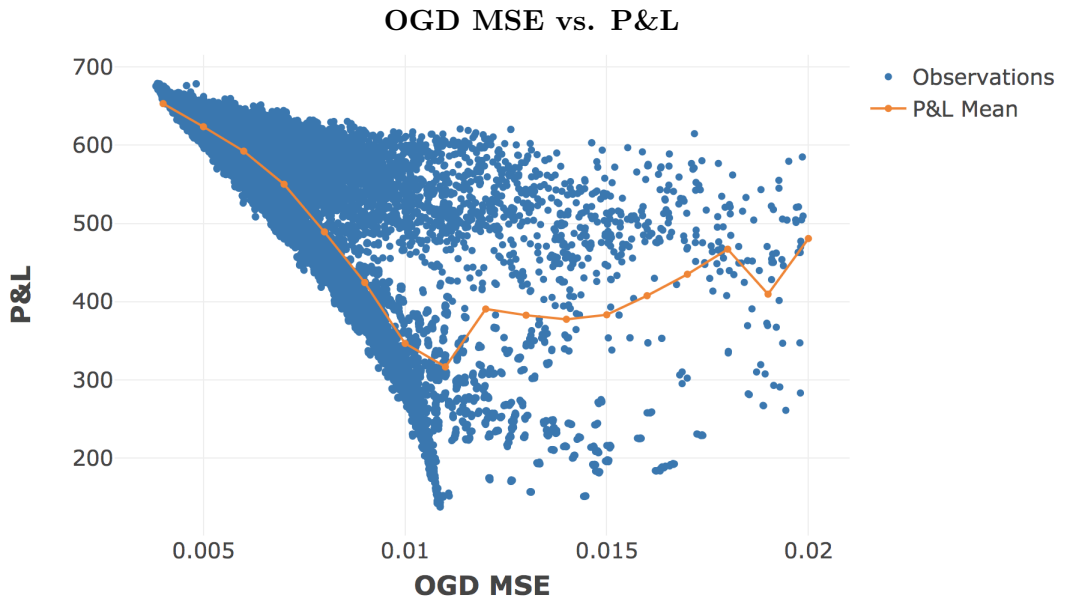


Figure 11: Dataset: Actual10 (7.1.1) ; Configuration 11 (11.5.11) & Configuration 16 (11.5.16)
The figure here shows the paired OGD MSE score and OOS P&L achieved for all 22248 trials run, excluding networks with null returns (a result of exploding gradients). There is a very strong correlation with $\rho = -0.74$, indicating the relationship of P&L increasing as OGD MSE decreases (or, as predictive accuracy increases). However, there are still many configurations which outperform in P&L despite having a worse OGD MSE score (reasons are noted above in Section 8.1). As a result, P&L is chosen as the primary assessment metric for the analyses on predictive networks.

8.2 Primary Determinants of OOS P&L

Our initial expectations were that the IS batch training using SGD for the predictive network would improve OOS P&L performance. Theoretically, the training on historical data might prime the network for predicting future data. However, we found that the effects of IS training had limited benefit. This limited effect highlights the complexity and dynamic nature of financial time series, where past relations and behaviours are not necessarily indicative of present state. These issues are discussed extensively in Section 8.3, however the follow through is that the primary determinants of OOS P&L are those which are present in the OGD (OOS) learning phase: the OGD learning rate, the data horizon aggregations and the SAE feature selection.

8.2.1 Effects of Data Horizon Aggregations

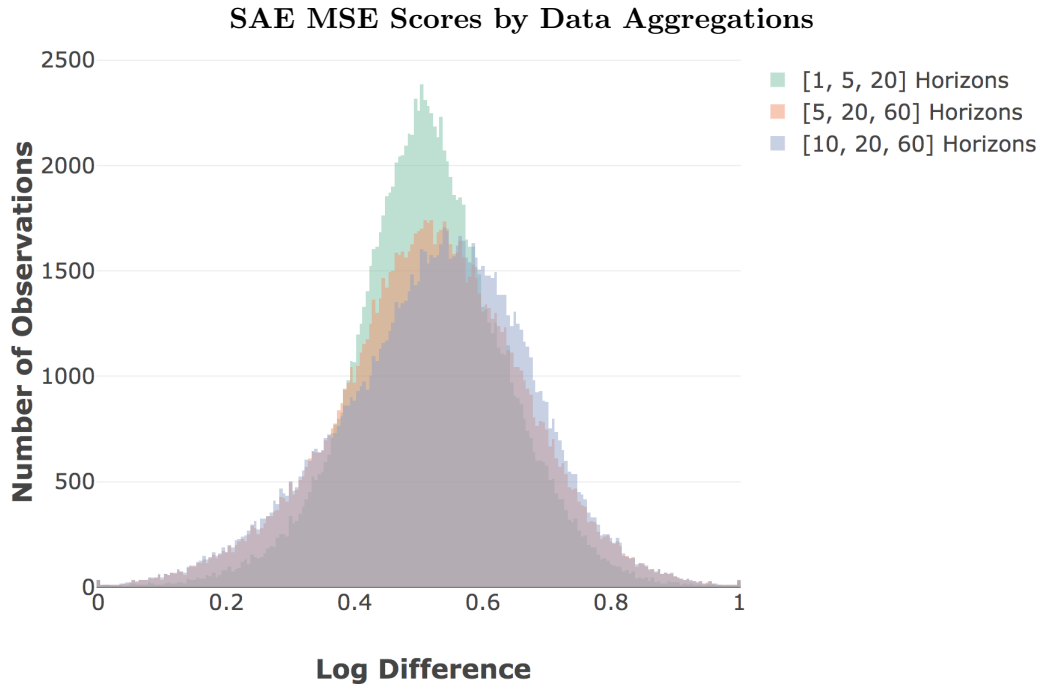
Input data was scaled to 3 different configurations to assess the effects of shorter and longer data horizons. Both SAE MSE and predictive P&L measures were used to assess performance. The configurations tested (in trading day window periods) were:

1. Horizon Tuple 1. [1, 5, 20],
2. Horizon Tuple 2. [5, 20, 60], and
3. Horizon Tuple 3. [10, 20, 60].

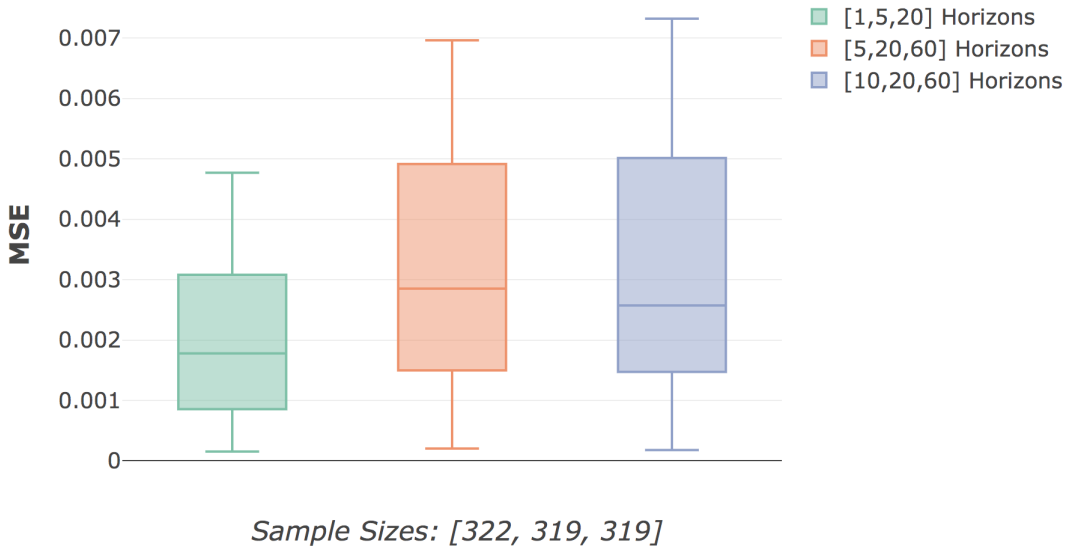
By way of example, using the 3rd configuration: the implementation of this is such that at each time point considered, the log differenced changes for the past 10, 20 and 60 days are available for each asset. This is described more extensively in Section 3.

We found that the predictive strategies learnt by the networks were determined by the data horizon aggregations chosen: predictions using short term fluctuations (with horizons of [1, 5, 20] days) and predictions using long term trends (with horizons of [5, 20, 60] or [10, 20, 60]). The differentiation between these groups is shown more robustly in Section 8.10, where it is determined to be the primary clustering attribute for trade correlations. We found that using short term fluctuations is a less reliable strategy, with higher variance in the returns delivered. However, the strategies using short term fluctuations also deliver the highest P&L and Sharpe ratios. The P&L comparison can be seen below in Figure 13.

We also found that the horizons impact the ability of the SAE network to reconstruct the aggregations. Lower variance in the shorter horizon aggregations result in easier replication, while longer horizons are more difficult (as indicated through MSE scores). The reproduction differences can be seen below in Figure 12.



(a) Log Difference Distribution



(b) SAE MSE Scores

Figure 12: Dataset: Actual10 dataset (7.1.1), Configuration 12 (11.5.12)
Figure (a) shows the distribution of values to be replicated by the SAE for actual data. Variances are quite different across the configurations ($\sigma_{[1,5,20]} = 0.118$, $\sigma_{[5,20,60]} = 0.146$, $\sigma_{[10,20,60]} = 0.150$). Smaller data windows are less likely to capture larger fundamental price changes, thus leading to these lower variances. This in turn, makes for easier replication for the SAE networks due to less variety in the sample set, which is seen in Figure (b) with the noticeably better performance in the [1, 5, 20] configuration.

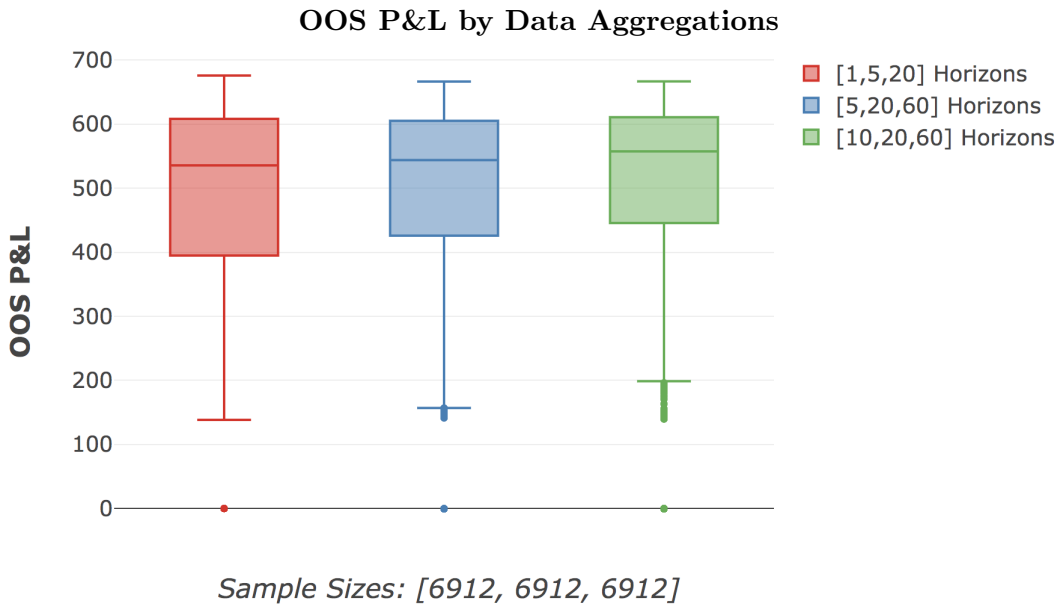


Figure 13: Dataset: Actual10 dataset (7.1.1), Configuration 13 (11.5.13)
The OOS P&L by data horizons shows the increased variance for shorter horizons, which is reduced as the horizons increase. The [1, 5, 20] configuration has a maximum which is 1.4% higher than the longer configurations. With lower medians and minimums though, it is not clear that the maximum is indicative of anything beyond variance. This is the first view which suggests the higher risk and rewards dynamic with short term predictive strategies, and more reliable returns with longer term strategies. However, it is necessary to consider the other determinants to see this more clearly, as is continued below.

8.2.2 OGD Learning Rate

We found the OGD learning rate has the most distinctive impact on P&L performance, as seen below in Figure 14. The large differences between rates further highlights the unstable nature of financial systems. If dynamics were consistent across time, we would not expect to see such large changes in OOS performance due to larger learning rates. That we do, shows how the network's ability to adapt to more recent information is critical in determining performance.

The effect of the OGD learning rate on networks can once again be separated by the prediction strategy, with lower and higher learning rates performing better for different strategies. The learning rates and data aggregations can be seen below in Figure 15.

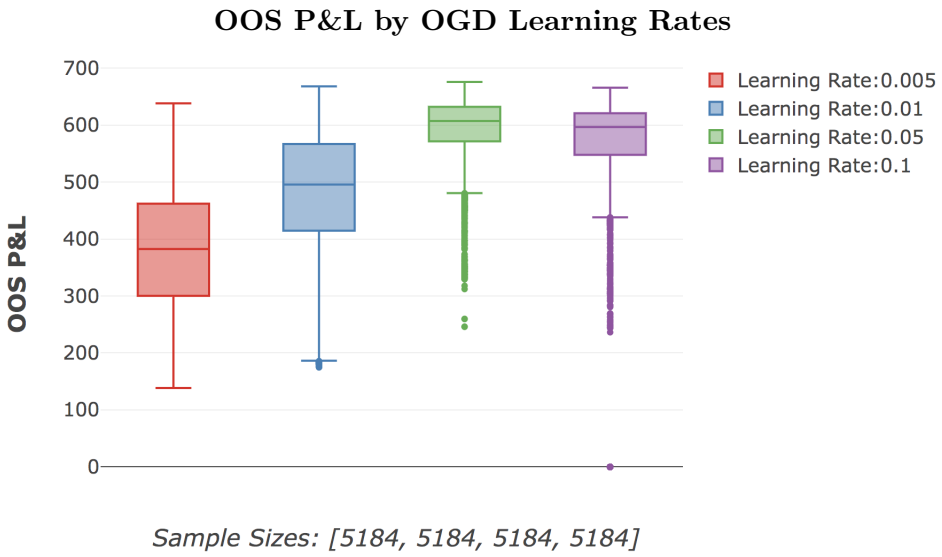


Figure 14: Dataset: Actual10 dataset (7.1.1), Configuration 13 (11.5.13)
The OOS P&L performance for OGD learning rates is expectedly non-linear. Performance increased significantly from 0.001 to 0.05, and degraded thereafter as the network over corrected to noise and short term changes in the market. The higher learning rates, with networks more able to adapt to changing signal dynamics, also resulted in significantly lower variance in the P&L delivered.

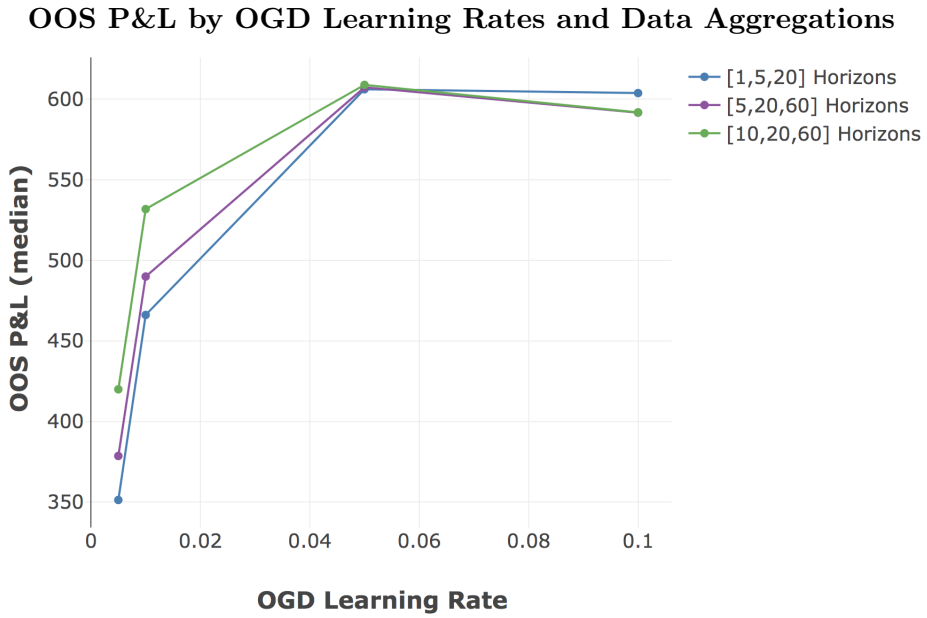


Figure 15: Dataset: Actual10 dataset (7.1.1), Configuration 13 (11.5.13)
The median OOS P&L by learning rate and data aggregation provides a view of the different prediction strategies. At lower learning rates, networks using long term changes and trends performed better as they benefit from smaller adjustments from one observation to the next. At high learning rates, networks using short term fluctuations performed better as they were able to more quickly adapt to changing dynamics and signals.

8.2.3 Feature Selection

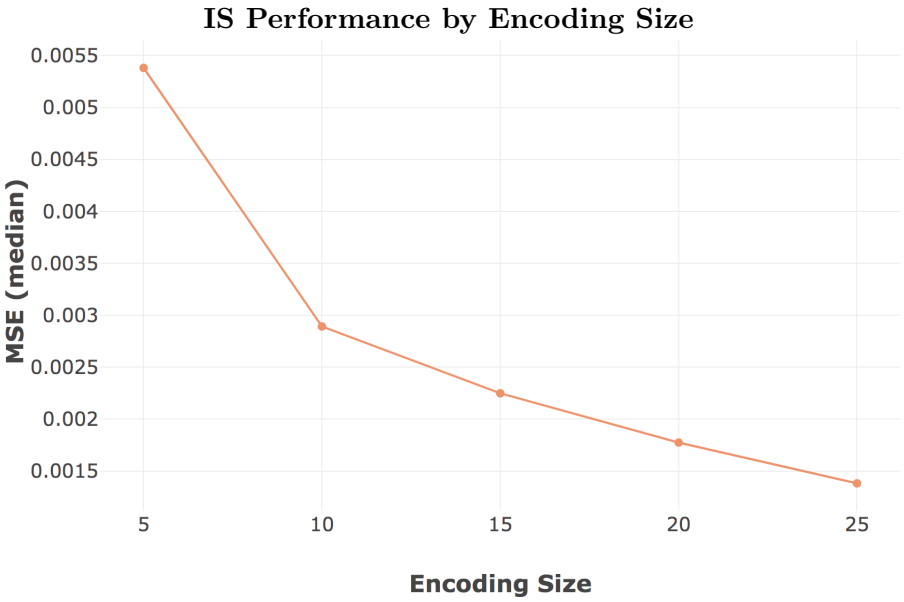
We found that feature selection through SAE dimensional reduction proves to be both possible and effective, though with complex behaviours and analyses. An important aspect to note is that the SAE networks were only trained on the first portion of IS data, and not updated afterwards. In light of changing dynamics in financial markets then, an effective SAE feature selection in this process is an optimization that may be limited to a certain time period. However, this optimisation may not generalise well in a different context (i.e. OOS).

Inspection of the nature of features learnt is difficult, though results show that the features can be categorised into 2 groups: larger encodings (25, 20, 15) which performed the equivalent of denoising or smoothing, and smaller encodings (10, 5) which performed more robust feature selection of legitimate properties[1].

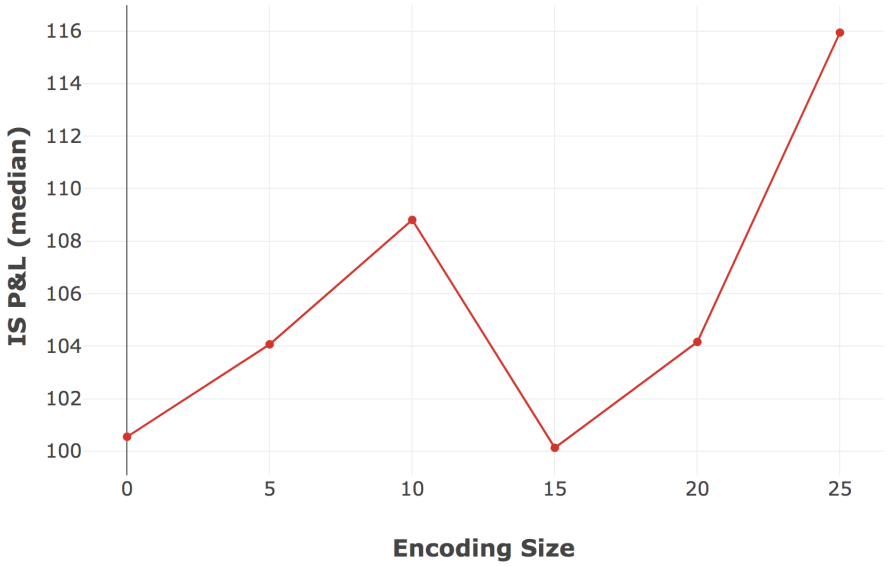
While the OOS performance of the larger encoding sizes mostly behaved as expected, the SAEs with encoding sizes of 5 and 10 were more inconsistent. In line with the view of changing dynamics within financial markets over time, we saw that SAEs with encoding of 10 overfit to particular features. These performed well for strategies focusing on long term trends, but were pathological for the strategies focusing on short term fluctuations. The SAEs with encoding size of 5 had more dependable performance, possibly because the small encoding layer acts as a form of regularization. The regularization would force the SAE to learn more consistently generalisable features. These dynamics are explored more in Figures 17 and 18 below.

The most noteworthy results we saw were SAE networks with encoding size of 5, which learnt generalisable features for 10 assets with 3 data aggregations each. Their performances were often on par or better than the higher encoding sizes, or no encoding at all, as seen in Figure 17a below. One of the 2 primary goals of this paper was to determine if feature selection is possible and beneficial, which we are able to answer here. The performance of the 5-encoding networks, with an 83% reduction in input data, is clear evidence of the efficacy and potential of feature selection in financial times series.

[1] Input data was 10 assets with 3 horizon aggregations each, resulting in an input size of 30 at each timestep.

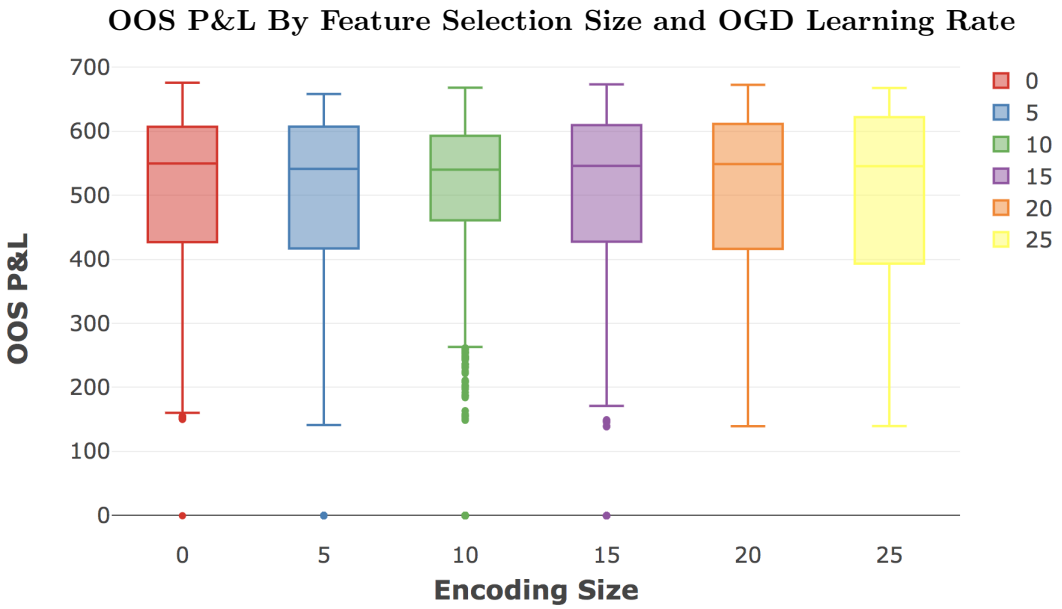


(a) MSE By Feature Selection Size



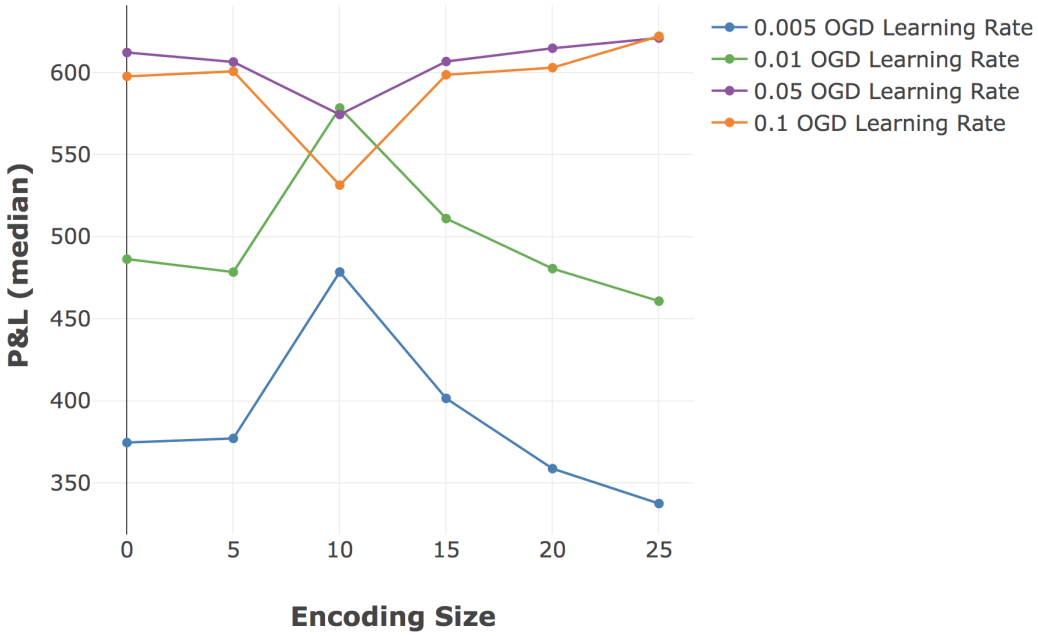
(b) IS P&L by Encoding Size

Figure 16: Dataset: Actual10 dataset (7.1.1), Configurations 11 and 12 (11.5.11, 11.5.12)
Figure (a) shows the MSE scores at different encoding sizes. While feature selection and reduction is possible through the SAE networks, the ability to reproduce input precisely does decrease monotonically with the encoding layer size.
Figure (b) shows the IS P&L for the same groups. This provides a different assessment, showing a clear increase in P&L at the 5 and 10 encoding sizes ("0" indicates no feature selection was done). This is indicative of the 2 different types of features learnt: sizes 15, 20 and 25 were smoothing or denoising the data and so reproductive MSE was correlated to P&L performance; conversely, at sizes 5 and 10 the SAE began learning more fundamental features of the data and so the relative reproductive MSE score is not a clear indicator for P&L performance.



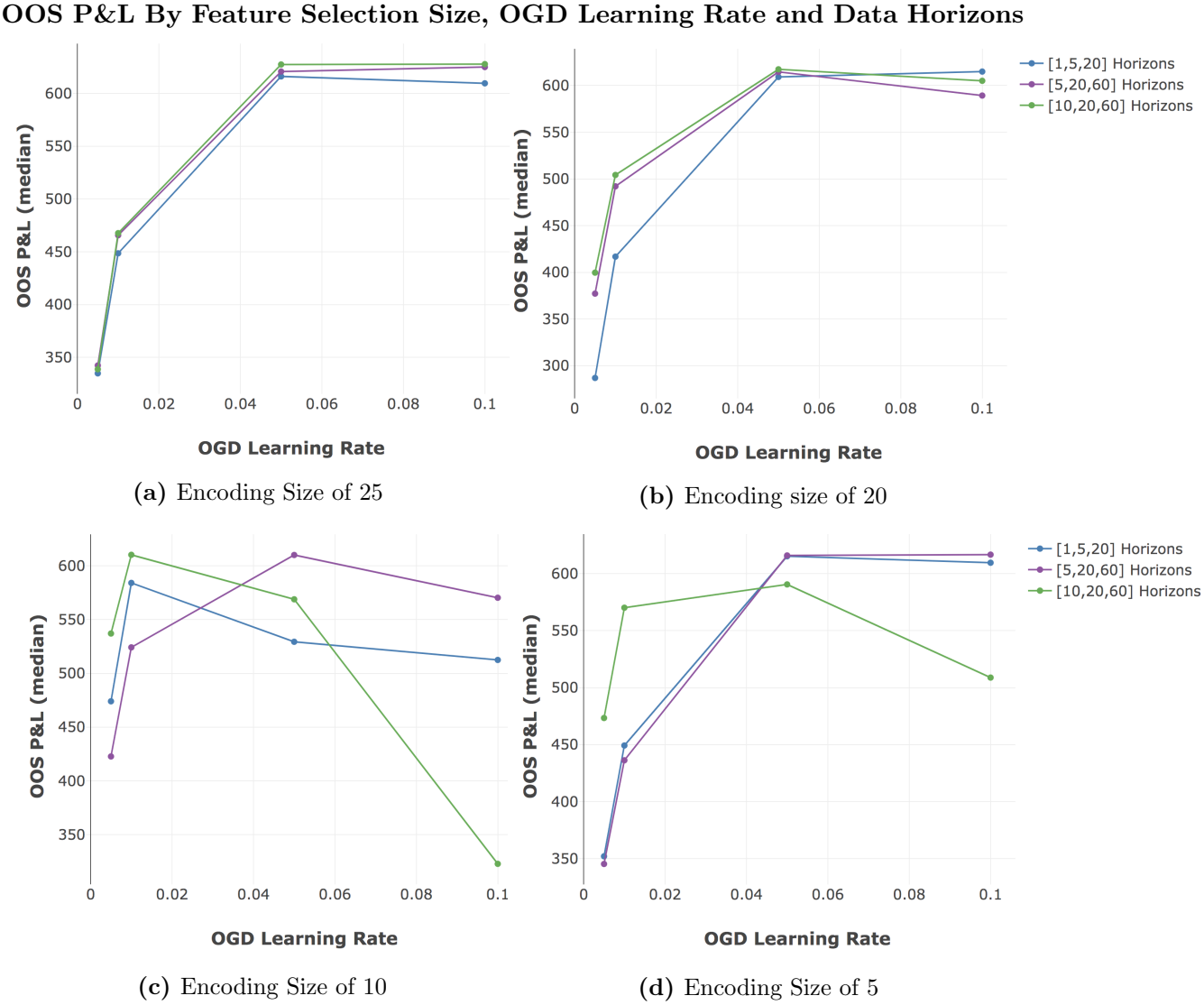
Sample Sizes: [3456, 3456, 3456, 3456, 3456, 3456]

(a) OOS P&L by Encoding Size



(b) OOS P&L by Encoding Size and Learning Rate

Figure 17: Dataset: Actual10 dataset (7.1.1), Configuration 11 (11.5.11)
Figure (a) shows the OOS P&L performance for all encoding sizes, with an encoding of 0 indicating no feature selection. It is interesting to note that at their best, almost all encodings were able to offer equitable performance. There was however a general variance reduction from 25 to 15, with 10 and 5 behaving differently. Figure (b) gives us far more insight into the dynamics causing the changing P&L. The lower learning rates (0.005, 0.01) performed best with strategies using long term trend pricing (as seen in Figure 15). The 10 feature encoding appeared to optimise specifically for this perspective. The optimisation caused outperformance at the lower learning rates and detrimental performance at higher learning rates (which perform best with short term fluctuation strategies). The 15 to 25 encodings showed a better association to the short term strategies, where higher encodings and higher learning rates offer the best performance. Notably, the 5 feature encoding offered very consistent performance across learning rates, further emphasising the learning of generalisable features. These dynamics are explored further in Figure 18.



(a) Encoding Size of 25

(b) Encoding size of 20

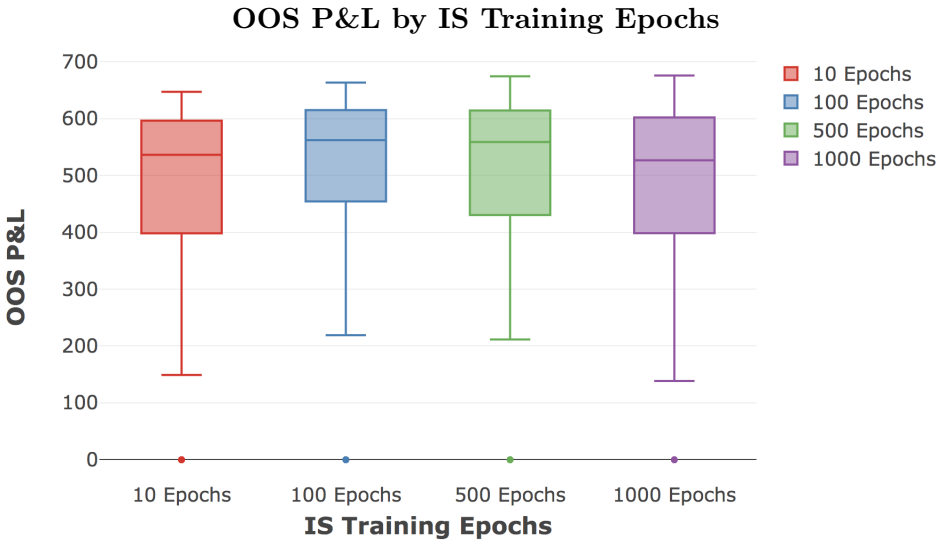
(c) Encoding Size of 10

(d) Encoding Size of 5

Figure 18: Dataset: Actual10 dataset (7.1.1), Configuration 11 (11.5.11)
The 4 plots above offer a cohesive view of all 3 primary determinants of OOS P&L. At encoding sizes of 25 and 20, which were largely smoothing the input, the feature selection had less material impact and so the relation between data horizons and learning rates is more prominent. Longer horizons deliver higher P&L with lower learning rates (and so smaller adjustments), while short term horizons outperform with large learning rates (and so an ability to adapt quickly). The SAEs with encoding size 10 showed the most inconsistent behaviour, clearly having overfit to the long term features. This resulted in high performance with the low learning rates and longer horizons, and extremely poor performance with higher learning rates and longer horizons. The encoding size 5 learnt a more robust feature selection, resulting in the best of longer horizons at low learning rates, and the best of shorter horizons at larger learning rates.

8.3 Value of Historical Signal

We ran experimental trials to test the hypothesis that the amount of historical IS data available is of limited use, and that the real value is in broader current and cross sectional data. This idea is in line with the understanding that financial markets are inherently complex, adaptive and dynamic, as discussed more broadly in Section 2.1. With this in mind, and especially so in light of a fundamentally changing macroeconomic landscape, there is limited reason to believe that the functions and relations that may have governed the asset prices 10-15 years ago would still be dominant today. We found the P&L results validate this idea, as seen in Figures 19 and 20. We saw that extensive training on past data may be akin to pretraining network weights at best, and counterproductive in overfitting to dynamics that no longer exist at worst.



Sample Sizes: [5184, 5184, 5184, 5184]

Figure 19: Dataset: Actual10 dataset (7.1.1), Configuration 13 (11.5.13)
The box plots above show OOS P&L grouped by the number of epochs in the SGD IS training phase (i.e. the number of times the IS data was trained on). In this set of configurations, 100 Epochs offered the best overall performance, and further training to 500 or 1000 epochs degraded performance due to the network overfitting on the IS data. The results here are noteworthy as they show that the benefit of historical data is limited - having a network become better at learning return relationships from 10 years ago did not lead to increased OOS P&L for more current data. The small difference in the upper half of observations between 10 and 100 Epochs further emphasises this point.

Figure 20: Dataset: Actual10 dataset (7.1.1), Configuration 16 (11.5.16) (some samples with 0 P&L were excluded for more effective visualization)
To further explore the effect of IS training on historical data, configurations were run with a percentage of the usual training data excluded, with the P&L results grouped above. The exclusion of up to 80% of the IS training data resulted in only a 2.2% drop in median OOS P&L for those networks. The training in these instances was not adjusted to increase the number of epochs according to the size of IS data, and so the configurations with more data excluded were also in essence trained less. These results, combined with those in Figure 19, show the limited use in training on historical data, and hence the far greater value in current cross sectional information.

8.4 Weight Initializaton Techniques

Neural network weight initialization has been one of the primary barriers to effective deep learning, with incorrect initializations leading to poor learning performance in deep networks. New techniques for initializing weights were one of the main reasons for the progress in the field [68]. In Sections 8.2 and 8.3, we show that historical data and training are of limited value. The most profitable models learn quickly from recent data, and have had less training on historical data. The issue of weight initialization then is even more prominent when in the context of financial time series. The two most popular methods, RBM pretraining and variance based initialization, have both been tested and detailed below.

8.4.1 RBM Pretraining for Sigmoid Networks

While previously considered the best approach for training deep neural networks, we found that the methodology of greedy layerwise RBM pretraining for sigmoid SAE networks (as described by Hinton and Salakhutdinov [68]) had detrimental effects on network performance, as seen in Figure 21.

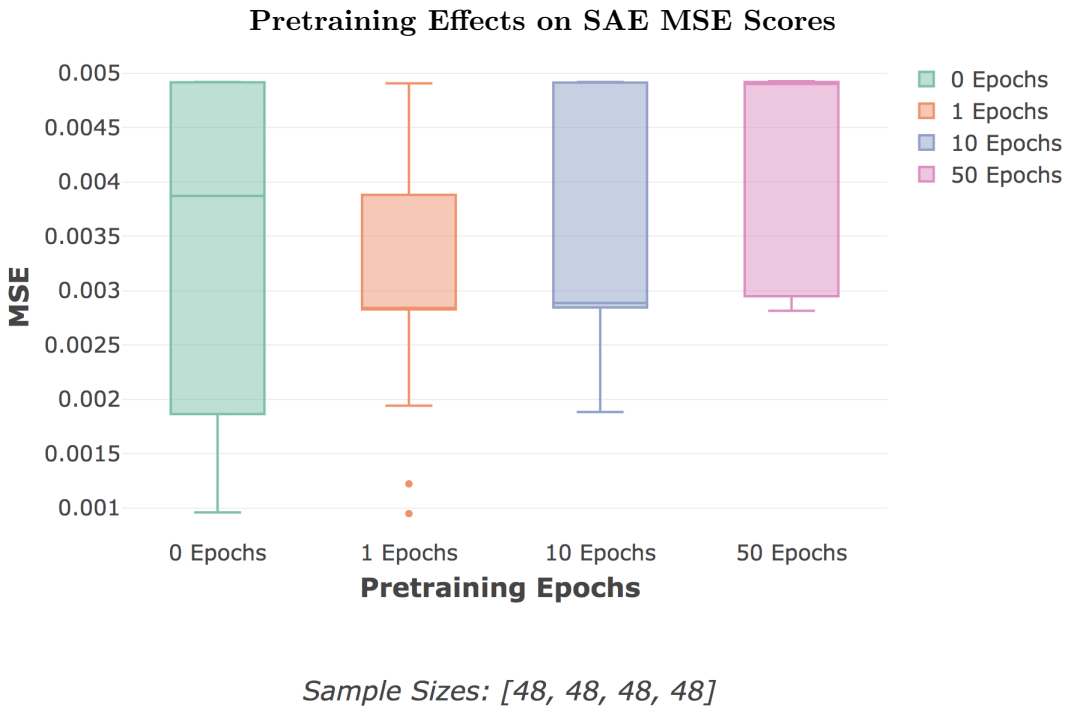


Figure 21: Dataset AGL&ACL (7.1.3); Configuration 6 (11.5.6)

The boxplots here show the summary of configuration performances, by minimum MSE achieved, grouped according to the number of pretraining epochs. There was an evident benefit to having no pretraining in this scenario as there was a clear decrease in performance as the number of epochs increased (the low value outliers that can be seen for the 1 epoch configuration were with learning rates which were low enough to approximate no epochs).

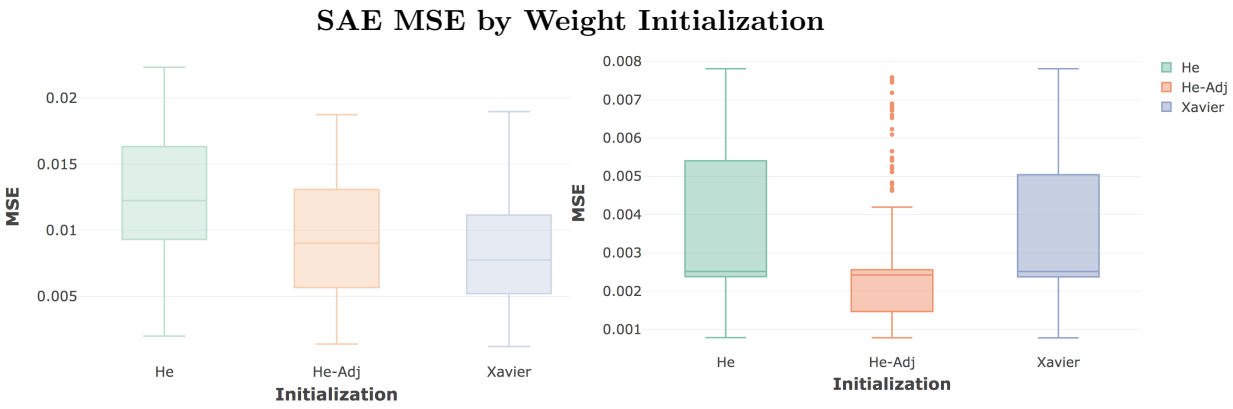
The RBM pretraining technique does assume that data is Independent and Identically Distributed (IID), and does ultimately traverse a different solution space and loss function. While it may be effective in some contexts[2], we saw it had a counterproductive exploration of the weight space for this non-IID dataset, and that ultimately the financial time series data is pathological for RBM pretraining. As discussed in Section 2.3.4, Erhan et al. [51] note that pretraining may be a form of regularization. This ties in with what we found in in Section 8.6.2, where "L1" regularization is also seen to have detrimental effects on most SAE networks (see Figure 29a).

8.4.2 Variance Based Weight Initialization Techniques

More recent research has focused on the use of weight initialization using variance based methodologies, as discussed fully in Section 4.5. Our expectation, theoretically, was that the He initialization would generally outperform Xavier due to it being more appropriate for the ReLU activations being used. Additionally, in networks with varying layer sizes, we expected He-Adj and possibly Xavier to outperform He which is subject to imbalanced initializations. He-Adj, as presented in Section 4.5, should present the best of both as an initialization that is suited to Leaky ReLU activations and the varying layer sizes found in SAE networks.

The results here mostly conformed to the theoretical expectations, though there are some inconsistencies. As discussed more fully in Section 8.5, the GBM data in synthetic datasets will have assets that are configured to certain means and variances. As these movements are processed, scaled and aggregated over time through the data processing, the movements will generally be more similar and less complex than those of actual stock data which will show greater variance over time (this is shown more concretely in Figure 12a). The learning process then is not required to be able to pass through error signals that change as dynamically over time, and so there is less pressure on the initial starting weights. This consideration may account for the unexpected performance of the Xavier initialization in the Synthetic SAE networks. Further, some of the assumptions of these techniques, such as data being IID, may not apply in the first place, making their results less predictable. Due to the generally better or comparable performance of He-Adj, it was the only initialization chosen for further trials.

[2]The RBM greedy layerwise training implementation was effectively tested and validated on known datasets such as MNIST, example results for which can bee seen in Section 11.1.2 of the Appendix.



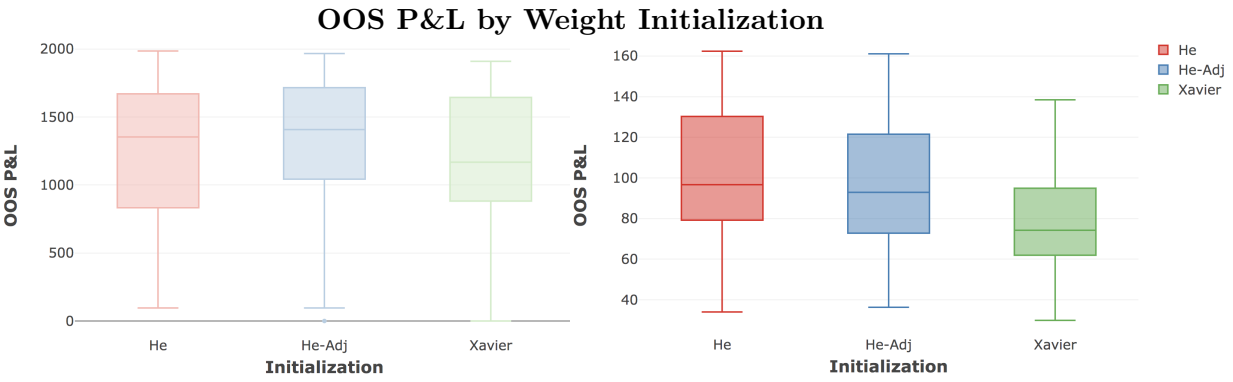
Sample Sizes: [350, 646, 1089]

(a) Synthetic Data

Sample Sizes: [400, 432, 432]

(b) AGL

Figure 22: Dataset: Synthetic10 (7.2.2), AGL (7.1.3), Configuration 7 (11.5.7) & Configuration 8 (11.5.8) Training SAE networks across the Synthetic10 and AGL datasets, we mostly saw the expected patterns emerge. He and He-Adj both suffered from exploding and vanishing gradients more frequently, leading to the smaller sample sizes seen when compared to Xavier. He-Adj consistently outperformed He due to being better suited to the network structures, as seen in the plots above. We also saw that He-Adj clearly outperforms Xavier for AGL data, but has performance that was mostly the same (or even marginally worse) than Xavier for the synthetic dataset. Similar trends as AGL were seen when SAEs were trained for a full 10 asset dataset, which can be see in Appendix 11.1.2.



Sample Sizes: [899, 1026, 1027]

(a) Synthetic Data

Sample Sizes: [432, 432, 432]

(b) AGL

Figure 23: Dataset: Synthetic10 (7.2.2), AGL (7.1.3), Configuration 9 (11.5.9) & Configuration 10 (11.5.10)
Figure (a) shows OOS P&L performance for synthetic data. The networks with more consistent layer sizes resulted in the performance between He and He-Adj becoming comparable, while both outperformed Xavier (as expected, with ReLU activations). In Figure (b), the P&L results for AGL data had clear outperformance of Xavier by the He based initializations, but a less clear comparison between He and He-Adj.

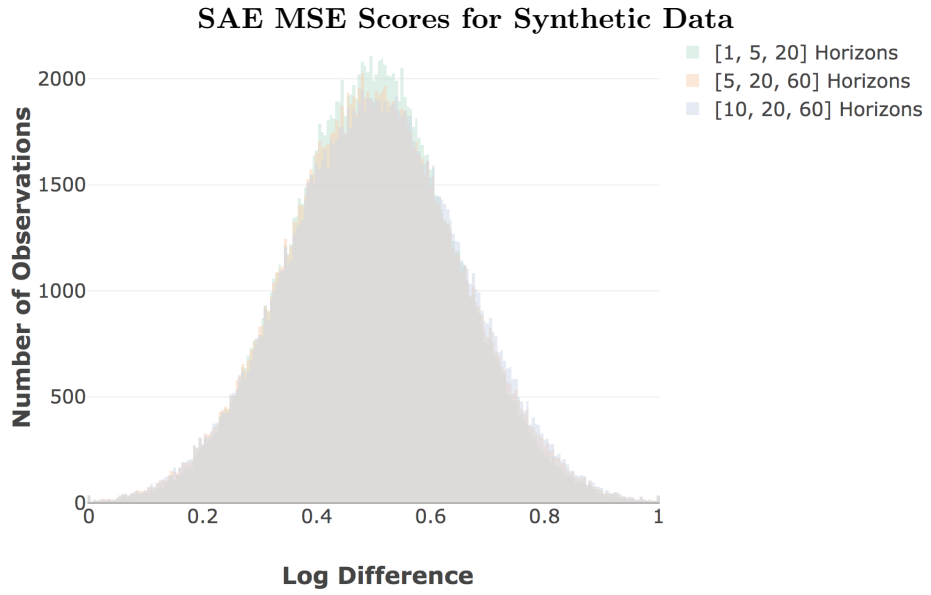
8.5 Synthetic Data

8.5.1 GBM Generated Data

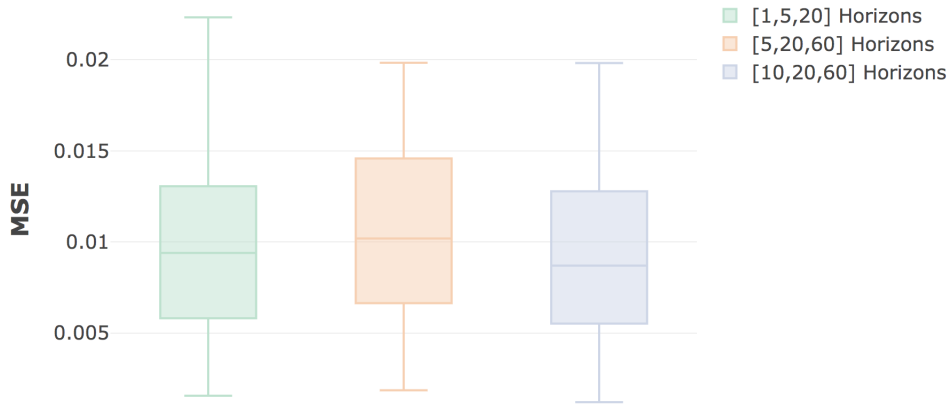
Geometric Brownian Motion (GBM), as discussed in Section 3.2, was used to simulate synthetic datasets for the purposes of testing implementations and configurations without influencing the likelihood of backtest overfitting. GBM is a popular choice for synthetic financial data as it is a Markov process, thus following a random walk and is generally consistent with the efficient market hypothesis in that the next price movement is conditionally independent of past movements. In line with this though, the series will exhibit a constant drift with price shocks according to its stationary configuration, and changes in price follow a particular distribution. It is worth noting that as GBM time series are non-ergodic, one should be wary of considering ensemble based predictions with much confidence [99]. GBM generated data provides a valuable tool in assessing ideas without taking on any risks, but these results need to be considered correctly.

This section details how the behaviour of synthetic data differed from actual data in some of the more critical areas of our experimentation process.

8.5.2 Synthetic Data Horizon Effects



(a) Log Difference Distributions



Sample Sizes: [700, 692, 693]

(b) SAE MSE Scores

Figure 24: Dataset: Synthetic10 dataset (7.2.2), Configuration 7 (11.5.7)
Figure (a) shows the distribution of values to be replicated by the SAE for synthetic data. Geometric Brownian Motion generates discrete changes that follow a log-normal distribution, the log difference and scaling of which (as per the data processing described in Section 3.1) result in normally distributed price changes. We see small differences in the distributions according to the data horizons used, as one would expect, with slightly lower variances occurring for smaller data horizons ($\sigma_{[1,5,20]} = 0.145$, $\sigma_{[5,20,60]} = 0.152$, $\sigma_{[10,20,60]} = 0.154$). There were some differences in SAE performance as indicated in Figure (b), though not to large extents and considering the very similar distribution of values, there was not any expectation of seeing fundamental differences in the networks ability to compress and replicate them.
Figure 12 shows the equivalent for actual data, where there is a significant difference in variances for longer data horizons which are more difficult to replicate.

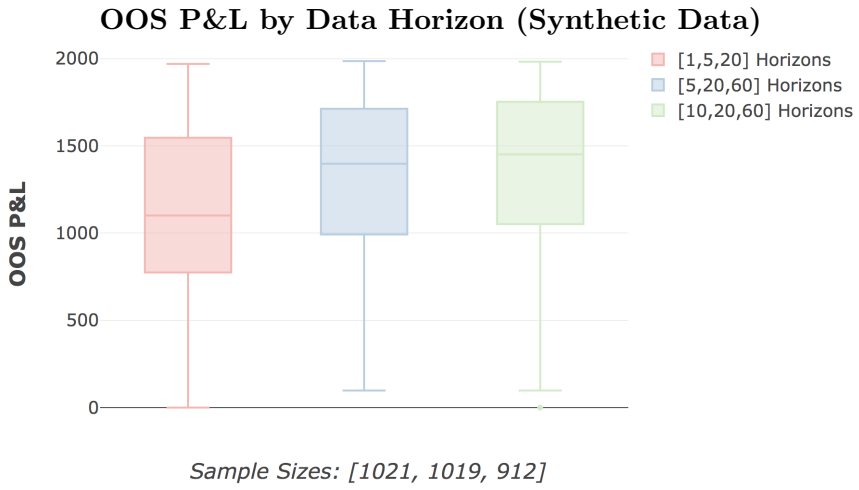


Figure 25: Dataset: Synthetic10 dataset (7.2.2), Configuration 9 (11.5.9)
The boxplots here show the P&L from the MMS grouped by the data horizon configurations. There was a clear trend of P&L increasing as the length of the windows increased. Shorter term GBM data would be more likely to represent fluctuations, whereas the longer term windows will be more representative of the constant mean in the asset price fluctuations, leading to easier predictive performance and higher P&L. This is in clear contrast to the effects seen in actual data, where both long term and short term strategies are present, with the highest P&L occurring for shorter horizon aggregations (these can be seen in Figure 13).

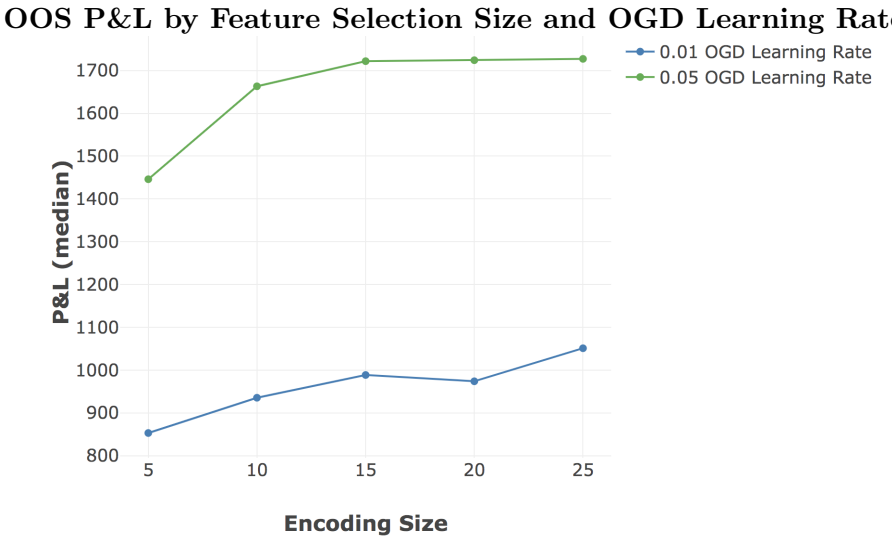


Figure 26: Dataset: Synthetic10 (7.2.2) ; Configurations 9 (11.5.9)
The trends in the synthetic data were straightforward in comparison to actual data. Feature reduction is possible, with OOS P&L increasing as encoding size increases. There is however an optimal point around 15, after which performance plateaus. Given the stabler nature of GBM data, this is to be expected. The results here offer a nice counterpoint to what is seen in the actual data. With the GBM data having constant characteristics throughout the whole dataset, there is no potential for the SAE to overfit to IS data and have a potentially detrimental effect as the learning rate increases (as was the case with actual data). Results for SAE MSE were largely the same, which can be seen in Appendix 11.1.1.

8.6 Complexity of Financial Time Series

The complex nature of financial markets, with a high noise to signal ratio and constantly changing dynamics, presents challenges during the learning process. A model's success lies in its ability to reduce noise without losing signal, and being able to capture determinant features of importance. This section considers how these issues affect data preparation in terms of scaling, and how SGD learning optimizations are able to help improve performance in SAE reproduction (Unsupervised Learning) and FFN prediction (Supervised Learning).

8.6.1 Data Scaling

We found the effects of different scaling techniques (Normalization and Standardization), as described in Section 3.1.2, had significant impacts on SAE performance as seen in Figure 27. The suggested implementation of the limited scaling approaches for the FFN networks was tested on Synthetic data (seen in Figure 28), showing a relatively small impact. The impact on FFN prediction is expected to be more significant when applied to actual data, which would contain unstable variances over time.

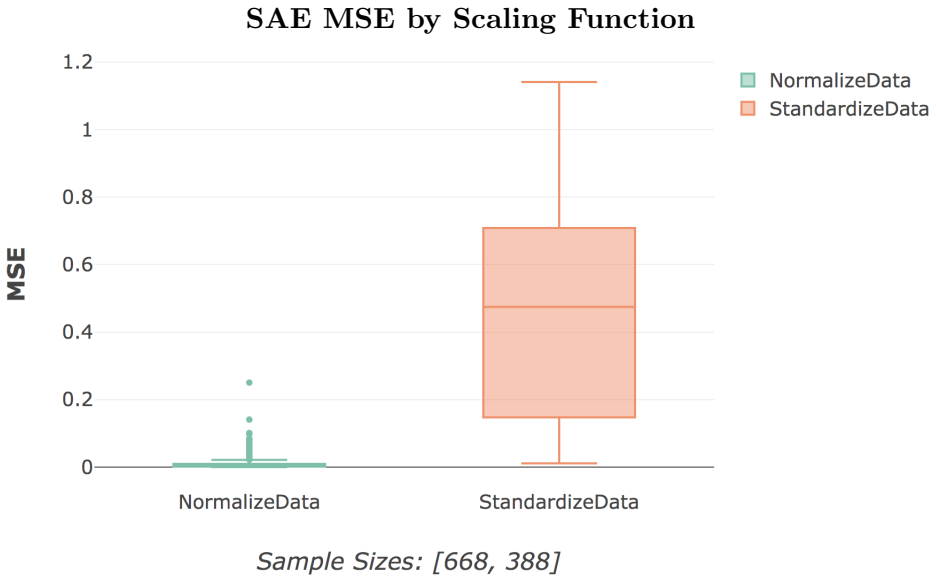


Figure 27: Dataset: Scaling10 dataset (7.1.2), Configuration 1 (11.5.1)
The figure here shows SAE MSE performance according to the scaling techniques described in Section 3.1.2. As discussed in Section 5.6, the use of standardization for scaling the data does not allow for effective outlier treatment, resulting in the significantly worse performance seen. This again shows the difficulty of dealing with financial time series, which are likely to have many outliers. The Standardize function was tried with fewer configuration choices due to the already apparent decrease in performance.

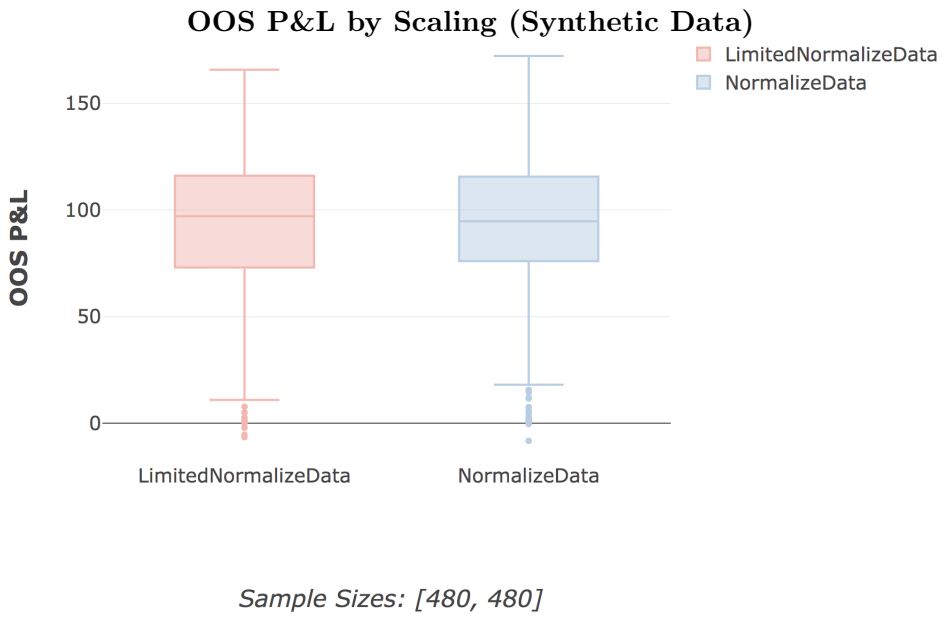
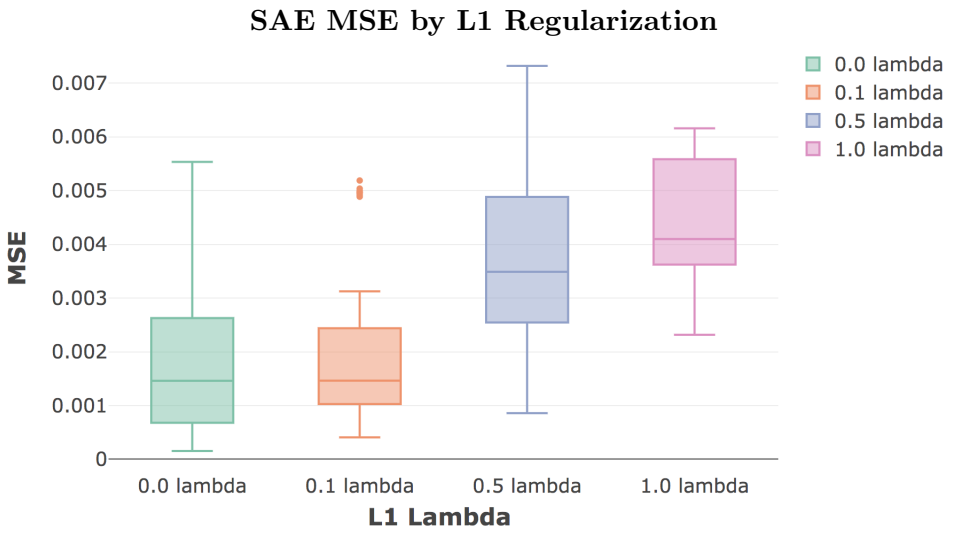


Figure 28: Dataset Synthetic6 (7.2.1), Configuration 3 (11.5.3) & Configuration 4 (11.5.4)
The figure here shows the impact of the limited scaling technique (as described in Section 3.1.2) in comparison to the non-limited version. There was a minor detrimental effect as a result of OOS no longer falling within the defined bounds for the scaling used, though the use of a Linear output layer was shown to assist in reducing the impact of this effect, seen in Figure 37. The implementation of this is not a choice when interested in simulating a real world implementation, and with real assets which have changing variance over time it is expected the impact is actually more significant

8.6.2 SAE Reproduction

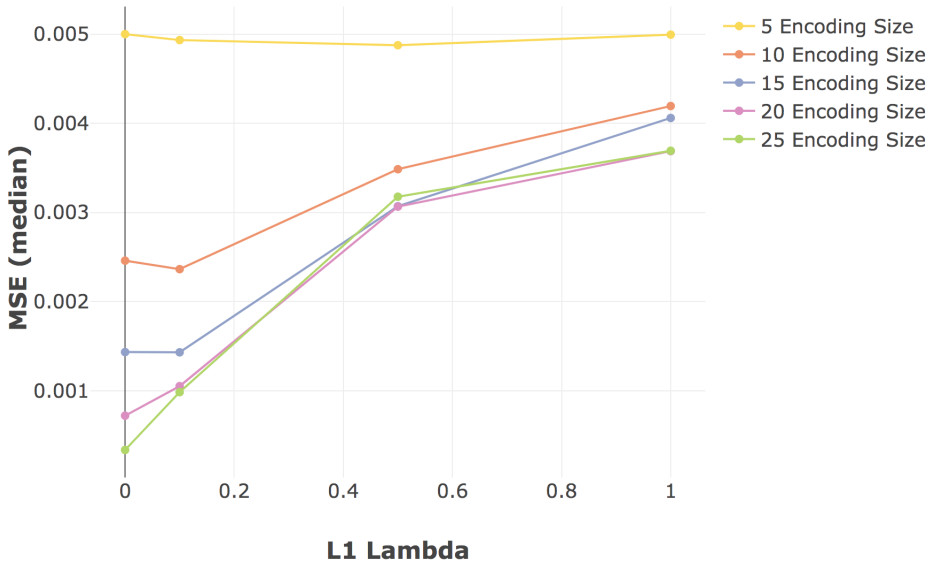
The ability of an SAE to reproduce data from a smaller encoding is directly related to the variability and complexity of the underlying data, and the extent to which it can be reduced to fundamental underlying features. The impact of data complexity and variance, as well as the different nature of encodings learnt is discussed fully in Section 8.2 for actual data, and Section 8.5 for synthetic data. What we discuss here is how learning optimizations can affect the traversal of the complex solution space.

We found that L1 Regularization (implemented as per Section 4.2.5) emphasised the different nature of features learnt at different encoding sizes, as discussed in Section 8.2. Smaller encoding sizes, which are generalizing by nature, benefit from the regularization. For larger encoding sizes, the regularization simply lowers model fidelity. These performance impacts can be seen in Figure 29 at both the aggregate and encoding size level. Figure 30 shows that denoising optimizations (implemented as per Section 4.4.3) had detrimental effects, highlighting the lack of benefits in adding noise to an already noisy feature space. Figure 31 shows the effects of learning rate schedules (implemented as per Section 4.2.6).



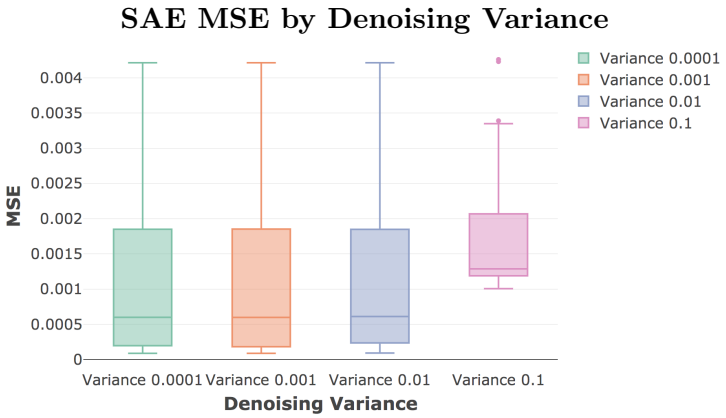
Sample Sizes: [451, 66, 410, 33]

(a) All Encodings



(b) Split by Encoding Size

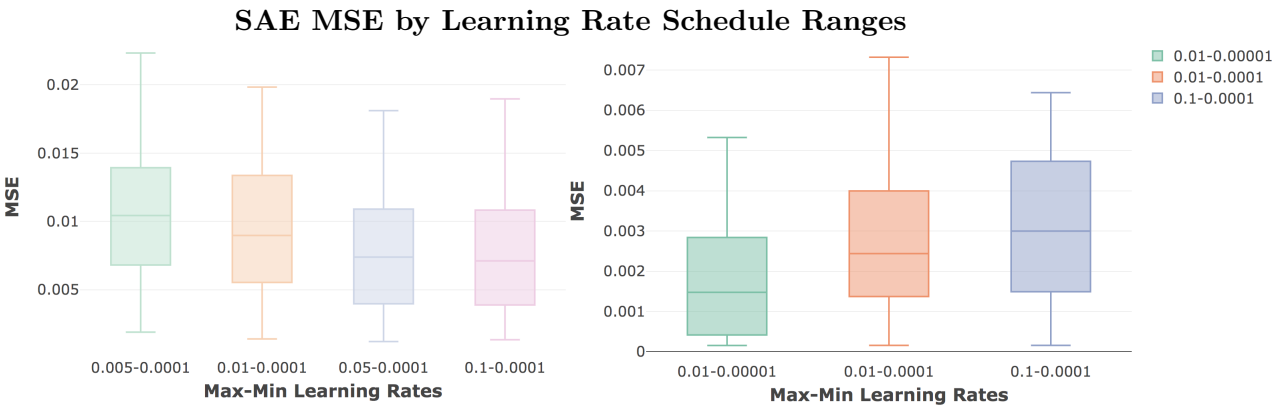
Figure 29: Dataset: Actual10 dataset (7.1.1), Configuration 12 (11.5.12)
Figure (a) shows the aggregate performance of SAE networks at different degrees of regularization, with performance decreasing as regularization increases (seen primarily in the 0.0 and 0.5 values, though supported by the smaller 0.1 and 1.0 configuration sets). Figure (b), grouping by encoding size, provides a better understanding. As per the results in Section 8.2, encodings of 5 are learning highly generalisable features, and so benefit from regularization with their best performance at $\lambda = 0.5$. Encodings of 10 have their best performance at $\lambda = 0.1$, learning less generalisable features. Higher encoding levels are operating in a different solution space, and do not benefit from the regularization.



Sample Sizes: [48, 48, 49, 49]

(a) Gaussian Denoising

Figure 30: Dataset: Scaling 10 (7.1.2) , Configuration 14 (11.5.14) & Configuration 15 (11.5.15)
This figure shows the effects Gaussian denoising had on SAE MSE scores, with scores worsening as the denoising increases. This once again reflects the differences between IID datasets with a model that is likely to overfit in comparison to financial time series data, which already has a significant amount of noise present.



Sample Sizes: [806, 639, 370, 270]

(a) Synthetic Data

Sample Sizes: [30, 684, 246]

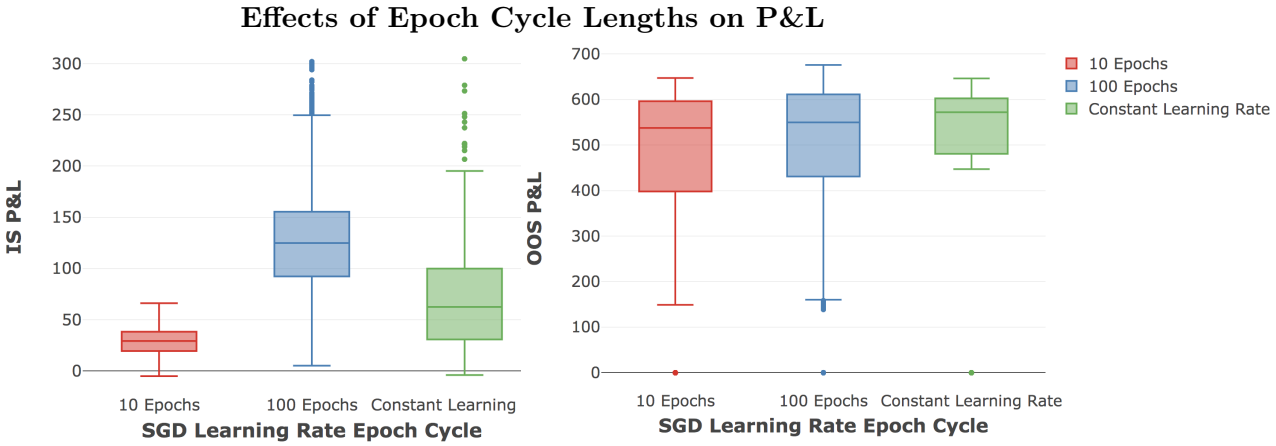
(b) Actual Data

Figure 31: Dataset: Synthetic10 (7.2.2), Actual10 (7.1.1), Configuration 7 (11.5.7) & Configuration 12 (11.5.12)
Figure (a) and (b) show the effects of learning rate ranges on SAE performance for synthetic and actual data, respectively. In both we found that a more informed approach to configuration choices was able to reduce variance in MSE performance (as seen in the sample sizes and upper bounds). However, the lower bounds for all ranges were quite similar, showing equitable performance in the best networks in different groups. This suggests that the learning rate range by itself is not enough to make significant impacts on performance. The different epoch lengths had a marginal effect, as seen in Appendix 11.1.3.

8.6.3 FFN Prediction

As we discussed in Section 8.3, the effects of supervised training on the IS dataset with SGD had limited benefits for OOS performance, and ultimately may be approximating the equivalent of pretraining weight initialization rather than an effective structural learning. The best performing networks were those which were able to quickly adapt to new solution spaces, rather than those which had trained effectively to old ones. Naturally then, the numerous SGD optimizations employed had very little effect on the OOS performance. That said, the effects they have on IS performance do give some insights into the nature of financial data being considered, and do suggest that implementing learning optimizations in the OGD phase could be beneficial.

Figure 32 shows the effects of learning rate schedules (implemented as per Section 4.2.6), highlighting the limited benefit of IS learning on OOS performance. Figure 33 shows the impact of L1 regularization (implemented as per Section 4.2.5), with results showing only a minor potential improvement and highlighting the unstable dynamics of financial time series. Lastly, Figure 30 shows that input Dropout (implemented as per Section 4.2.7) was ineffective in prompting the network to learn interactive relations between assets.



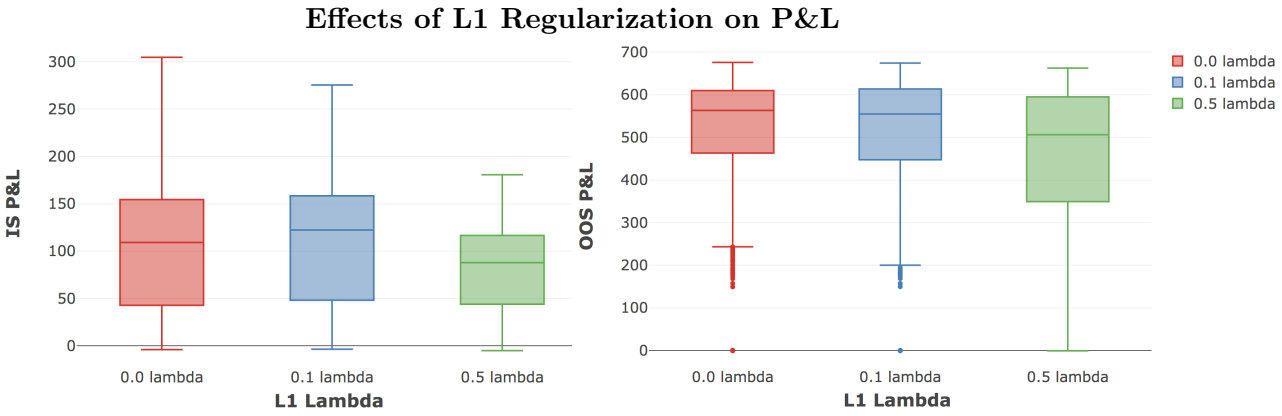
Sample Sizes: [5400, 15984, 648]

(a) IS P&L

Sample Sizes: [5400, 15984, 648]

(b) OOS P&L

Figure 32: Dataset: Actual10 dataset (7.1.1), Configuration 13 (11.5.13)
In Figure (a), the IS P&L for different learning rate epochs is shown. The different configurations (as indicated by the sample sizes) do not allow for a conclusive finding. That said, the upper and lower bounds do suggest that 100 epochs is a preferable exploration in comparison to shorter epochs or constant learning rates. What is of more significance in these figures is the pairwise comparison between IS and OOS in Figure (b). The 5% increase in upper bound performance in the '100 Epochs' group does suggest there is value to a more careful exploration, but the large differences IS do not persist OOS. The learning rate ranges themselves were set to the best values seen for the data in the SAE training, and small variations did not result in notable changes. These can be seen in Appendix 11.1.3.



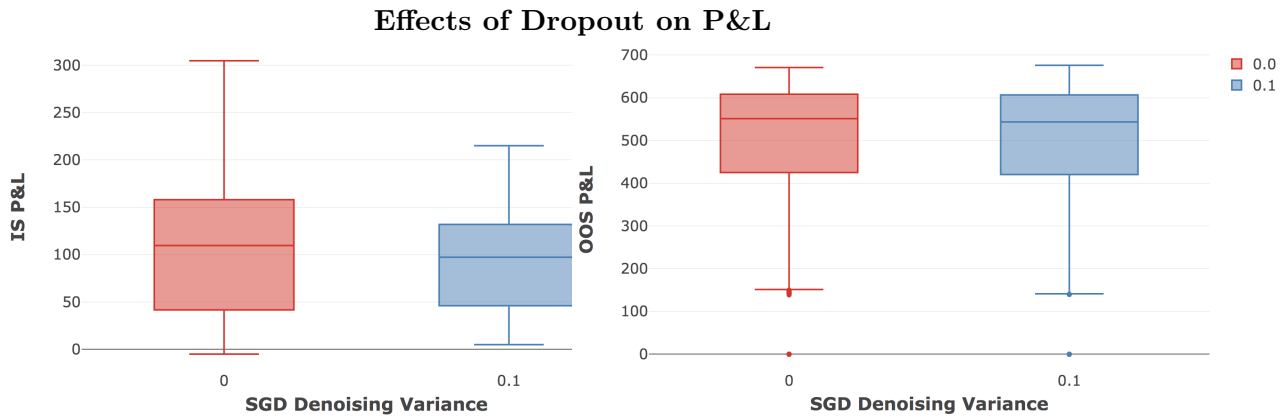
Sample Sizes: [8208, 6912, 6912]

(a) IS P&L

Sample Sizes: [8208, 6912, 6912]

(b) OOS P&L

Figure 33: Dataset: Actual10 dataset (7.1.1), Configuration 13 (11.5.13)
L1 Regularization in Figure (a) shows mostly poorer aggregate IS performance with lower P&L and with only detrimental follow through to the OOS performance, as seen in Figure (b). This highlights that even when a model generalises for IS data, the changing dynamics of financial time series means that it may not be generalising well for OOS data. L1 regularization is expected to work well in instances of IID data, but for time series financial data, where the model is not overfitting, it is just affecting model fidelity. This effect is amplified both IS and OOS when $\lambda = 0.5$.



Sample Sizes: [11664, 10368]

(a) IS P&L

Sample Sizes: [11664, 10368]

(b) OOS P&L

Figure 34: Dataset: Actual10 dataset (7.1.1), Configuration 13 (11.5.13)
Input Dropout was tested on the predictive FFN networks in the hope that it might encourage the network to increase learning about the relationships between assets. The results for 10% input dropout had a detrimental impact on IS performance, though with little impact on OOS P&L.

8.7 Network Structure

8.7.1 Effects of Network Size

We found that increasing network layers or layer sizes usually led to better performance in both SAE and FFN networks. The exception was for some of the networks with smaller sized layers (i.e. 120 nodes for SAE networks), which struggled with training. The effects can be seen for both SAE MSE and FFN P&L in Figure 35 below. The trends for both actual and synthetic data were largely the same, and so only actual data has been considered here.

It was interesting to see that the typically accepted SAE network structure, where layers decrease in size between the input layer and the encoding layer, resulted in worse performance (as compared to the same number of layers with a constant number of nodes). As detailed in Section 8.4, the RBM based greedy layerwise pretraining proved to be ineffective, and so variance based weight initialization with SGD training on the entire network was used instead. It was seen that for SGD training, larger layers at all stages improved performance. It is possible that a greedy layerwise training using variance based weight initialization may have shown increased performance in the decreasing layer size structure. The box plots showing these trends can be seen for actual and synthetic data in Appendix 11.1.4.

8.7.2 Effects of Activation Functions

We tested a range of activation functions (linear, ReLU and sigmoid) (as described in Section 4.2.2) on actual data for SAE networks to determine the best configurations going forward. We found that linear activations were best for the encoding and output layers, and that ReLU activations largely outperformed sigmoid and linear activations for the hidden layers (as seen in Figures 37 and 36 below). This makes sense with the non-linear benefit at hidden layers but with less loss of error signal and information at the output and encoding layers. These results were used for subsequent parameter choices.

8.7.3 Predictive FFN Activations

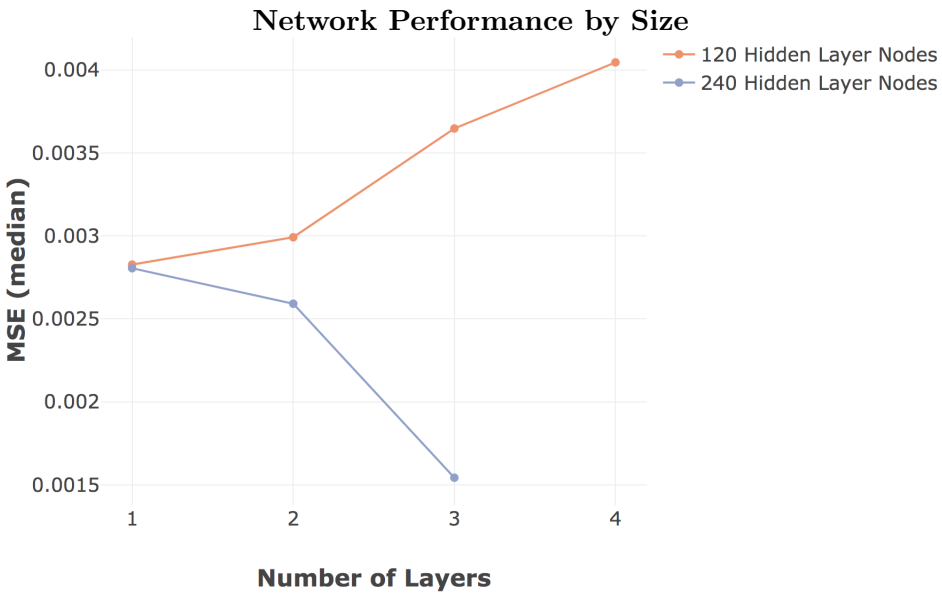
Predictive network activation functions were only tested on synthetic data. With the results for the SAE networks available, the risk of increasing PBO from more configurations did not seem necessary. We found that reconstruction of aggregated GBM data was able to be performed quite effectively through linear activations, so long as the networks (and encoding layers) were large enough to allow for sufficient transformations. Linear activations were not tested further on actual data, as there is no reason to believe the time series would continue to exhibit the constant drift with non-independent price jumps over time.

8.7.4 Sigmoid Activation Functions

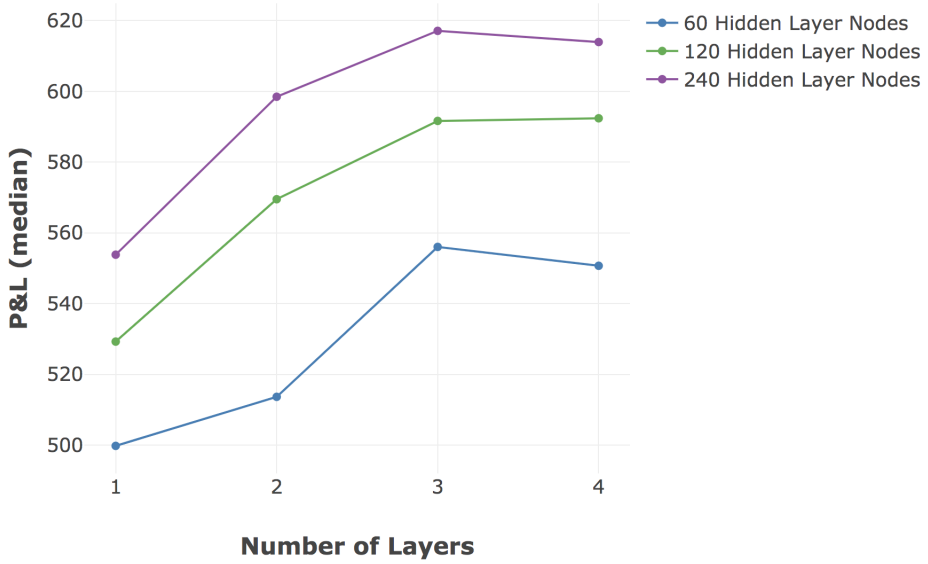
Due to the poor performance seen in sigmoid function based SAEs, as well as the poorer results when compared to ReLU activations, sigmoid functions were largely excluded from further configuration testing.

8.7.5 Leaky ReLU vs. ReLU

We found that the Leaky ReLU activations had only minor effects for SAE, which can be seen in Appendix 11.1.4. For the predictive network, the Leaky ReLU did offer some more significant improvements (3% - 10%) on OOS P&L for synthetic data, as seen in Figure 59. Subsequently, Leaky ReLU was used as the hidden layer activation of choice for most trials.

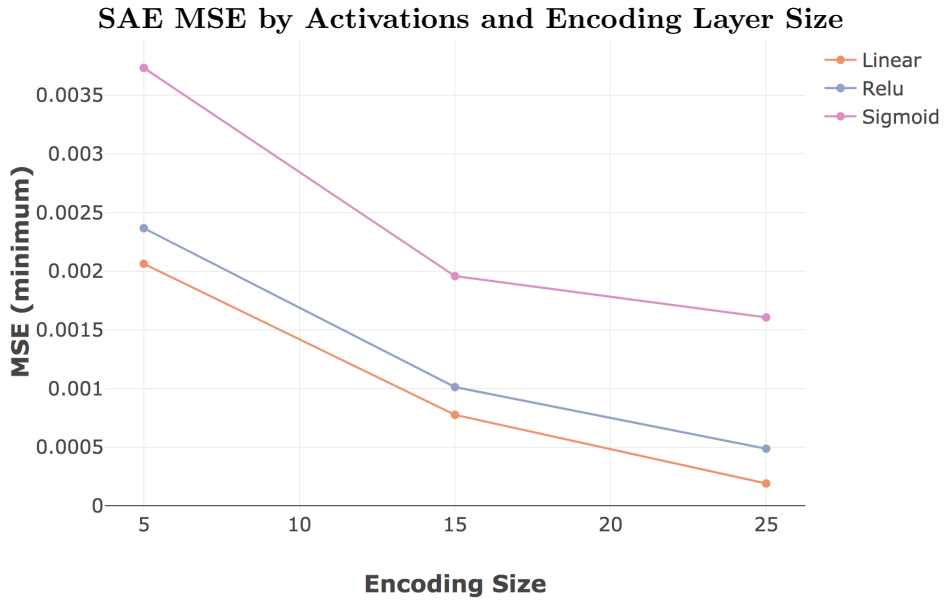


(a) SAE MSE Scores

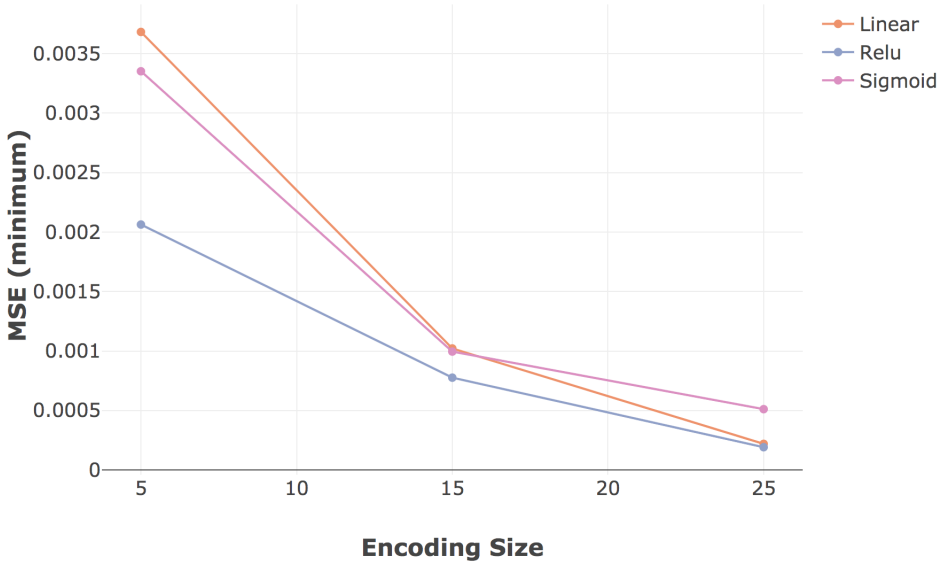


(b) Predictive P&L Scores

Figure 35: Dataset: Actual10 dataset (7.1.1), Configuration 12 (11.5.12) & Configuration 13 (11.5.13)
Figure (a) shows the median MSE by SAE networks achieved with the indicated number of layers and layer nodes. The number of layers here indicate the final SAE structure of N hidden layers, rather than the training structure of $2N + 1$ layers. The networks with 120 node layers struggled with training, and performance decreases as layers increased. This is to be expected if the learning parameters (e.g. SGD learning rate) are not ideal for the structure, the effects of which are amplified as network sizes increase. This was resolved in the larger 240 node networks which show significant improvement as more layers are added (as one would generally expect, subject to effective SGD).
Figure (b) shows the median OOS P&L achieved by predictive networks with the indicated number of layers and layer nodes. The behaviour was as expected here, where network performance usually improves with size. The exception was the four layer network of 60 nodes, where the training parameters were no longer effective in passing signal back (larger layer sizes are less likely to suffer from this, particularly with ReLU activations).



(a) Encoding Activations



(b) Hidden Activations

Figure 36: Dataset: Scaling10 dataset (7.1.2), Configuration 2 (11.5.2)
Figure (a) shows the best (minimum) MSE scores for different encoding activations at the different encoding layer sizes (with a network input of 30). There was consistent outperformance by linear activations in the encoding layer, which is expected from literature [68], and in line with reducing loss of error signal at critical points.
Figure (b) shows the best (minimum) MSE scores for different hidden layer activations at the different encoding layer sizes (with a network input of 30). At larger encoding layers, there was competitive performance from fully linear networks, where they are less pressured to take advantage of non-linear effects. As the size of the encoding layer is reduced, the advantages of non-linear activations become more apparent, with outperformance by ReLU and sigmoid activations. Sigmoid activations are known to suffer from learning slow down as a result of saturation and so have increased sensitivity to vanishing gradients [60]. SAE networks are deep by nature, and so it is not surprising that the sigmoid activations resulted in worse performance over the same training period when compared to ReLU activations.

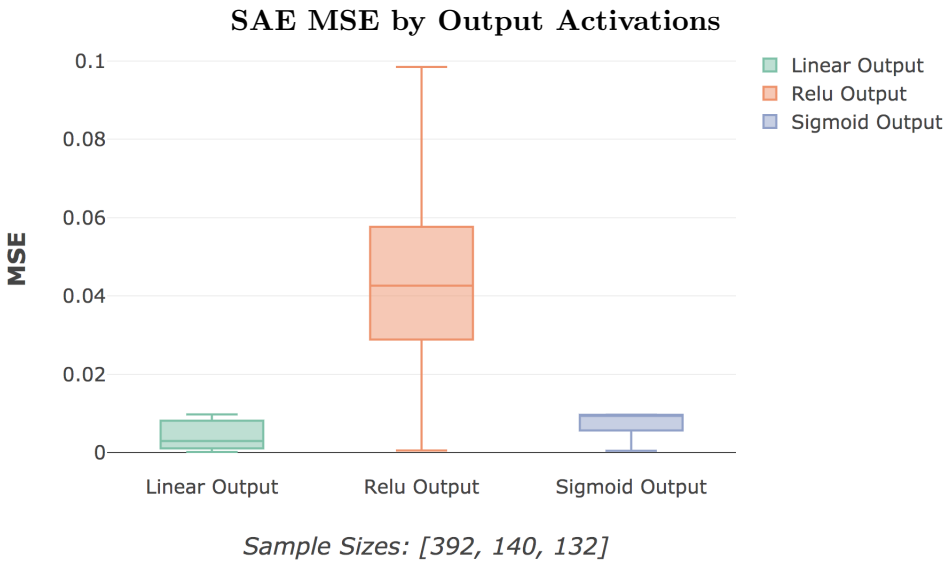


Figure 37: Dataset: Scaling10 dataset (7.1.2), Configuration 1 (11.5.1)
This figure shows the effects of the output activation for different configurations using Normalized scaling. The ReLU activations resulted in consistently worse performance when used as output activations, most likely due to their unbounded function range. While sigmoid and linear had more comparable performances, we found that linear typically had better results. This is likely due to reducing the loss of error signal in the output layer where it is most critical.

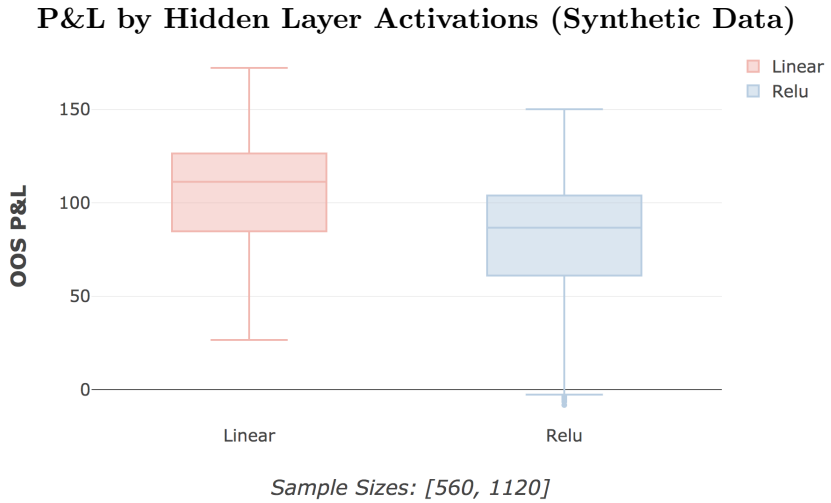


Figure 38: Dataset Synthetic6 (7.2.1)), Configuration 3 (11.5.3) & Configuration 4 (11.5.4)
This figure shows a comparison of linear and ReLU hidden layer activations on P&L, where the linear activations resulted in notably better P&L (even when more configurations were attempted with ReLU activations). Unlike the SAE networks, this persists even when the network size is decreased - this difference highlights the effects of GBM data and that it can be represented linearly, as per Section 8.5.1. We would see actual asset data benefit from a non-linear representation. Linear configurations for actual data were not run, as real stock data is not expected to follow patterns that are linear through time, thus offering little value.

8.8 Money Management Strategy

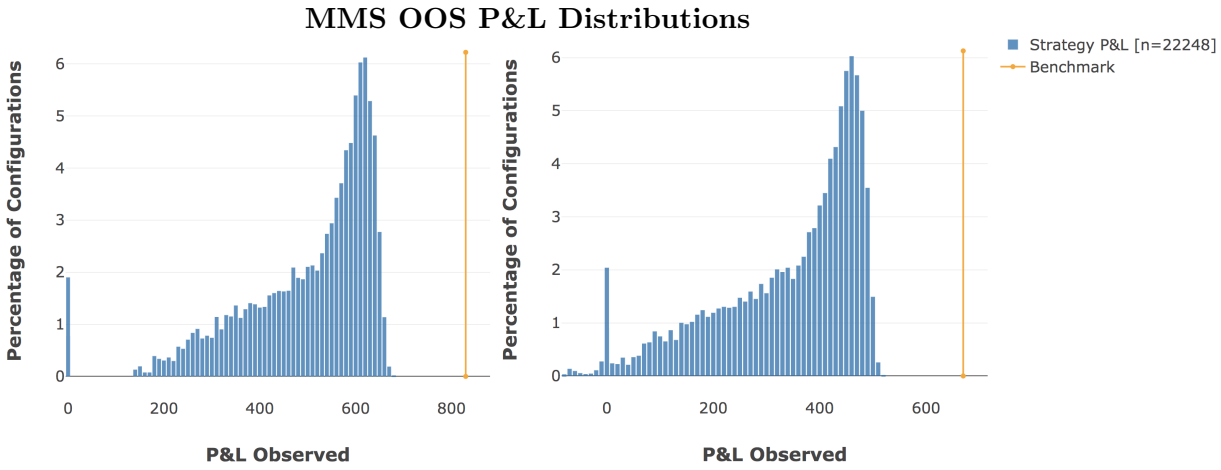
This section provides an overview of the financial returns generated from the MMS implemented, as per Section 4.6. We consider the strategies' OOS P&L and Sharpe ratios, with trading costs applied. The benchmark represents returns that would have been generated from the same MMS using a network making predictions with 100% accuracy, and so represents an upper bound on performance.

The benchmark full return rate is 2.4% with trading costs, over a period of 1555 trading days. So while the strategies' proximity to the benchmark do represent a framework success, they are not necessarily representative of a feasible market solution. Ultimately, this enforces the notion that the MMS implementation is of exceeding importance in a live trading process, and predictive accuracy is only able to achieve so much.

Figure 39 shows the distributions of OOS P&L with and without trading costs being accounted for. There were a significant number of configurations within the 20%-30% range of the benchmark. While the costs do have a notable impact on the overall P&L distribution, it does not indicate that the strategy would suffer from implementation limitations due to trading costs. The trials with 0 P&L are networks which suffered from either exploding or vanishing gradients, and were not able to make sufficient predictions.

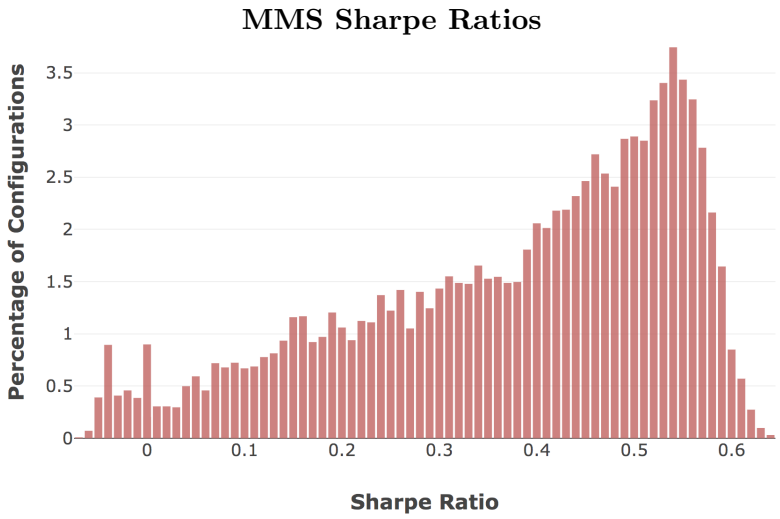
The distribution of Sharpe ratios are shown in Figure 40a, and followed an expectedly similar pattern to the P&L distributions. The Sharpe ratios are not exceedingly high, though this is as representative of the basic trading strategy and selection of stocks used as any performance assessment of the underlying networks.

As noted earlier, the P&L reported on here is a relative unit measurement, rather than a particular currency (as discussed in Section 3.3). Additionally, the MMS, as described in Section 4.6, only trades 1 stock unit at a time and so any P&L figures are relative indications rather than absolute limits.



(a) Without Capital & Trading Costs **(b) With Capital & Trading Costs**

Figure 39: Dataset: Actual10 (7.1.1) ; Configuration 11 (11.5.11) & Configuration 16 (11.5.16)
The distributions of all OOS P&L values, with the benchmark P&L indicated in orange, show an encouraging view of the results. There is a significant negative skew, with a proportionally small number of strategies resulting in negative returns, even with capital and trading costs applied (as per equation (52)). There were a large proportion of strategies near the OOS upper bound, which in turn is within 22% and 28% of the benchmark for Figure (a) and (b), respectively (these percentages apply similarly to return rates as they do to OOS P&L).



(a) Sharpe Ratios (With Costs)

Figure 40: Dataset: Actual10 (7.1.1) ; Configuration 11 (11.5.11) & Configuration 16 (11.5.16)
This figure shows the distribution of all strategies' Sharpe ratios, with costs taken into account. Naturally, this is highly correlated with the P&L, showing the same overall distribution patterns as those seen in the OOS P&L in Figure 39. The values and skew seen here indicate the successful implementation of the underlying framework.

8.9 Probability of Backtest Overfitting

8.9.1 Concerns Regarding the PBO Calculation

The CSCV and PBO techniques, as detailed in Section 4.7, were developed by Bailey et al. [8] in order to offer a robust methodology of assessing whether a selected strategy is likely to have been justified through backtest overfitting, a common and problematic phenomenon (as discussed in Section 2.6). The crux of the methodology lies on the idea that for overfitting to occur, the strategies that deliver the best IS performance will systematically underperform in the OOS datasets (thus reflecting the model having overfit to the IS patterns or noise).

Their methodology offers several key benefits:

1. The CSCV methodology is generic, model-free and non-parametric, allowing it to arguably be used in any model case.
2. There is no requirement of a hold-out set, which removes potential credibility issues regarding whether the holdout set was treated appropriately or not.
3. While the Sharpe ratio is the typically chosen performance metric, the technique is generic enough to allow any performance measure to be used.
4. The logit distribution developed through the assessment offers a useful view on the robustness of the strategies used and the nature of the PBO score.

While the methodology is in substance a model free approach to assessing overfitting, the application in a machine learning context is novel and has dynamics worth considering. The models we present here are trained through batch learning for IS data, and proceed with online learning for OOS data. The model parameters of a neural network are primarily its weights, which change throughout the learning process. This does stand in contrast to a static financial model which might use the same model parameters throughout IS and OOS. If a model performs unexpectedly well OOS due to an effective ability to learn, then it could be fairly indicative of the model's strength rather than overfitting. If different learning techniques are used IS and OOS (which is typical), then it is not inherently clear where learning ends and overfitting begins. Though as Bailey et al. [8] note, the PBO score does not necessarily invalidate a set of strategies which all have value, and it is still expected to pick up on cases where overfitting *has* occurred.

The use of the logit metric in the CSCV method also creates some potentially problematic issues. It has its basis in an ordinal ranking, and whether the best strategy in the IS set is higher than the median in the OOS set. The effect is that strategies which perform poorly both in the IS and OOS datasets are able to artificially bolster the ordinal position of the best strategy such

that it is past the median point and does not result in an increased likelihood of overfitting. As the authors correctly noted, the addition of trials that are doomed to fail will bias results, and if configurations are obviously flawed then they shouldn't have been tried in the first place. In practice though, the issue is more pervasive than suggested. The combinatorial grid search approach as outlined in Section 5.3 results in certain parameter combinations that cause networks to perform very poorly. These are the '0' P&L configurations in Figure 39a. Individually, these parameters may work well in other combinations though, and so there would be no obvious reason to exclude them to begin with. In this sense, an honest search of the solution space where there is no prior knowledge about what will work OOS may result in a PBO score that is lower than it should be. This is an obvious concern due to the potential of fraudulent use. It is even more concerning though, when considering that it may happen incidentally and without the researcher realising.

Further to this, the parameter space search methodology (Section 5.3) also results in a lower likelihood of PBO due to the way of combining parameters across IS and OOS stages. By way of example, any configuration which performs well IS will have all possible OOS parameters tested in combination with it. While some of these combinations may result in poor performance, there will always be a combination of the best IS and best OOS parameter choices. This makes it unlikely that the best configurations will be past the median point for the logit calculation, resulting in a systematically low PBO regardless of how many configurations are attempted.

Another potential implementation concern with CSCV and PBO is the choice of the S parameter, which is the number of splits for the data to be sampled into. As seen in Figure 41, the number of splits can have an overwhelming effect on the PBO score reported for the same data sets and results. This is not a problem if the implementation is done honestly and openly, but does allow for misuse and a potential inability to compare PBO scores. Further, while the authors do provide guidelines for choosing S such that the splits will be representative of trading periods, there is still no clearly defined method of choosing the time spans of these periods. The exponential growth in combinations to be tested as S increases also quickly results in reaching computational resource limits. This may further lead to the choice in S being made according to reasons that are not in line with prioritising the correct PBO score to report. These combination sizes can be seen below in Figure 42.

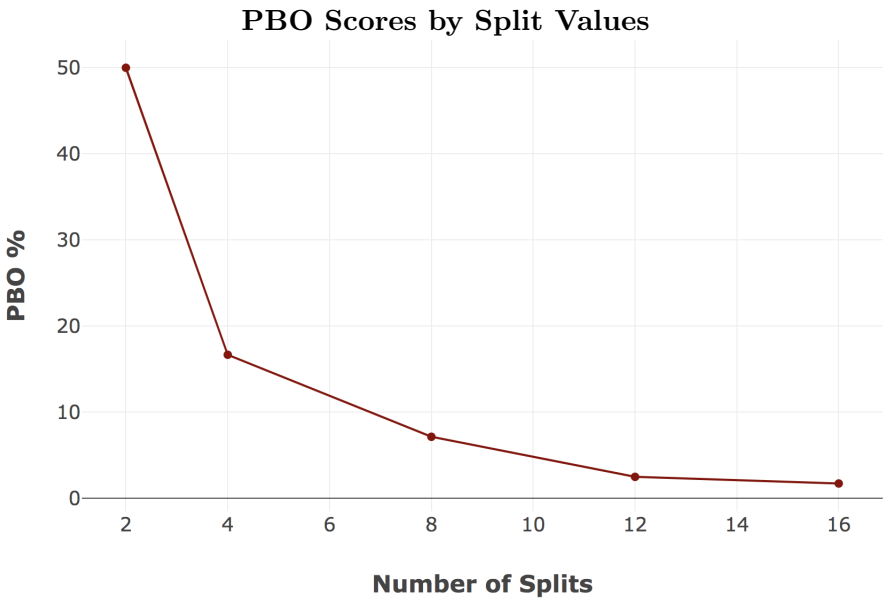


Figure 41: Dataset: Actual10 (7.1.1) ; Configuration 11 (11.5.11) & Configuration 16 (11.5.16)
The curve here shows the significant impact that the choice in the split parameter S has for the PBO score calculated, with a monotonically decreasing score as S increases (these scores were for the same set of results). This highlights the contextual nature of a PBO score and the problems raised with score-wise comparisons as well as upfront configuration choices.

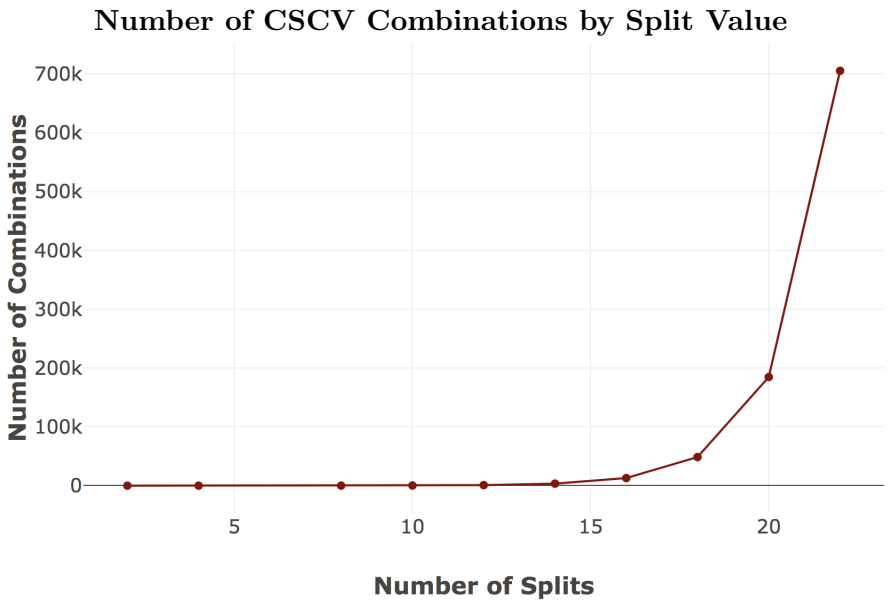


Figure 42:
The number of combinations that have to be processed by the CSCV method according to the value of the split parameter (S) chosen. The exponential increase in combinations to be processed as S crosses a value of 16 once again highlights configuration choice challenges. It is unlikely higher splits than 16 will be chosen, regardless of whether they are more appropriate or not.

8.9.2 PBO Results

We ran the CSCV algorithm on the majority of the configurations tested on the Actual10 (see Section 7.1.1) dataset, which resulted in a final PBO value of 1.7%. A subset of networks were excluded on accounts of 'null' predictions, resulting in a sample size of 21653 (out of a total of 22248 configurations). The CSCV algorithm was run with a split value of 16. There were 15 years of data, making 16 a reasonable choice as the split parameter (which needs to be even). Ideally, the splits would represent shorter periods, but the exponential increase in computational time (as seen in Figure 42) made this impractical, and 16 is also the typical choice recommended by the authors. The full logit distribution can be seen below in Figure 43.

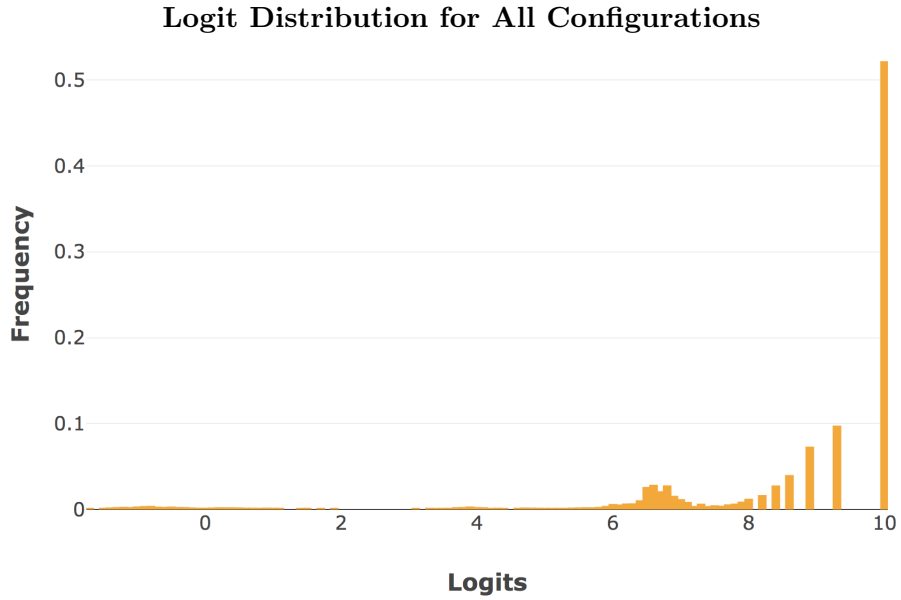


Figure 43: Dataset: Actual10 (7.1.1) ; Configuration 11 (11.5.11) & Configuration 16 (11.5.16)
The CSCV logit distribution for the 21653 configurations run, with a calculated PBO of 1.7%. The strong negative skew is indicative of IS and OOS strategy returns being closely matched in rank, thus resulting in the low PBO score. This is a favourable assessment for the efficacy of the full framework presented here, showing that training was able to occur without much risk of backtest overfitting.

We found interesting dynamics around the calculation of PBO, and the configurations contributing to the figure. The configuration process went through 2 primary phases: an extremely broad combinatorial grid search, consisting of 20736 configurations; and a second much narrower search of 1512 configurations. Assessing only the configurations from the second phase resulted in a PBO score of 6.3%, which was significantly higher than the overall PBO score. The logit distribution for the second phase can be seen in Figure 44 below. The effect here highlights two important aspects of the PBO calculation:

1. The PBO score was much higher for the configurations which were picked more specifically after having already seen a large number of results, which is correctly indicative of increased likelihood to overfit.

2. However, the PBO score is not monotonically increasing with N, as one would expect. This is counterintuitive and is in line with the concerns raised in Section 8.9.1 regarding the effects of increasing configuration sample size.

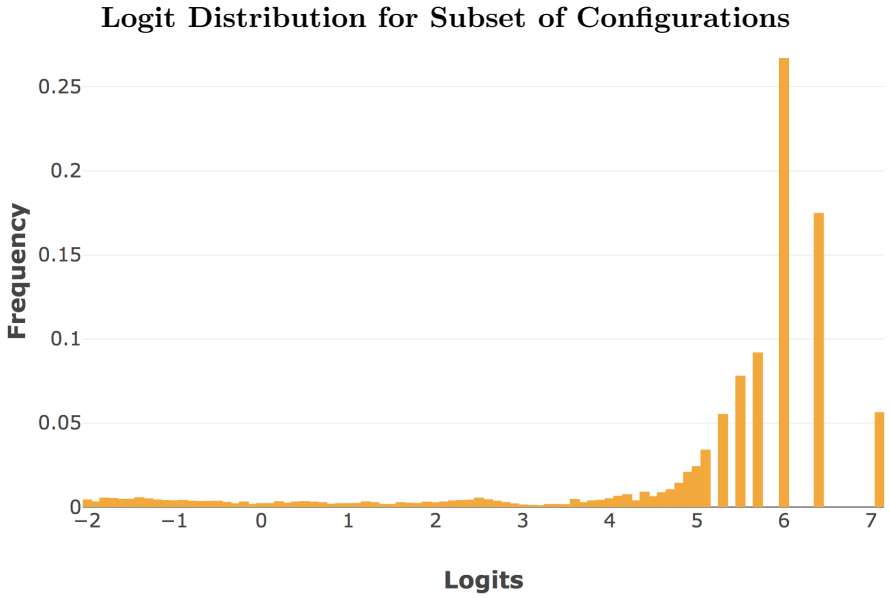


Figure 44: Dataset: Actual10 (7.1.1) ; Configuration 16 (11.5.16)
The CSCV logit distribution for a narrower subset of configurations run, with had a calculated PBO of 6.3%. This subset consisted of 1512 configurations (reduced to 1182 when 'null' networks were excluded), out of the total 22248 configurations run (presented in Figure 43). The higher PBO score (compared to the 1.7% for all configurations) is indicative of both a higher risk of PBO for the second phase of configurations tested, as well as a potential issues with PBO scoring, as raised above in Section 8.9.2.

8.9.3 Framework Success

While the CSCV and PBO implementations do raise some concerns, the results here also highlight the efficacy of the framework which has been implemented. The low PBO scores presented show that the suggested framework allows researchers to take on a broad exploration of the possible solution space while maintaining a low risk of backtest overfitting. This was able to be done while also achieving P&L results which are comparable to the benchmark, as seen in Section 8.8.

8.10 Deflated Sharpe Ratio

8.10.1 Computational Complexity

One of the primary drawbacks of the DSR algorithm presented by Lopez de Prado and Lewis [90] is the computational complexity which arises with the iterative K-means implementation. A standard Lloyds implementation of the K-means algorithm has a computational complexity of $\mathcal{O}(knt)$, for k clusters, n components and t iterations[3]. The suggested implementation modification (as discussed in Section 4.8.1) would be for $t = 10$ and to run the algorithm iteratively for $k = 1, ..., (n-1)$, which can in turn be expressed as $n(n+1)/2$. This leads to a computational complexity with an $\mathcal{O}(n^3)$ magnitude. Including the final modification suggested, which could run the algorithm recursively on subclusters, has the potential to increase this even further. This high degree of complexity presents a problem for cases where a large number of strategy permutations have been attempted, as is the case here (with $n = 22248$). With this in mind, the algorithm was implemented for a maximum of $k = 10$, though the results do not suggest this hindered the findings. The computational complexity and compute time for the clustering could be significantly reduced by implementing super-paramagnetic clustering using community detection [129]. This directly determines the number of clusters by solving a maximum likelihood representation of the cluster configurations.

8.10.2 ONC Results

The ONC algorithm produced three clusters: one consisting primarily of the negative Sharpe ratio configurations, and two containing the remaining configurations which were partitioned exactly by their data horizon configurations (as per the calculations in Section 3). The distributions for all three clusters' Sharpe ratios can be see below in Figure 45.

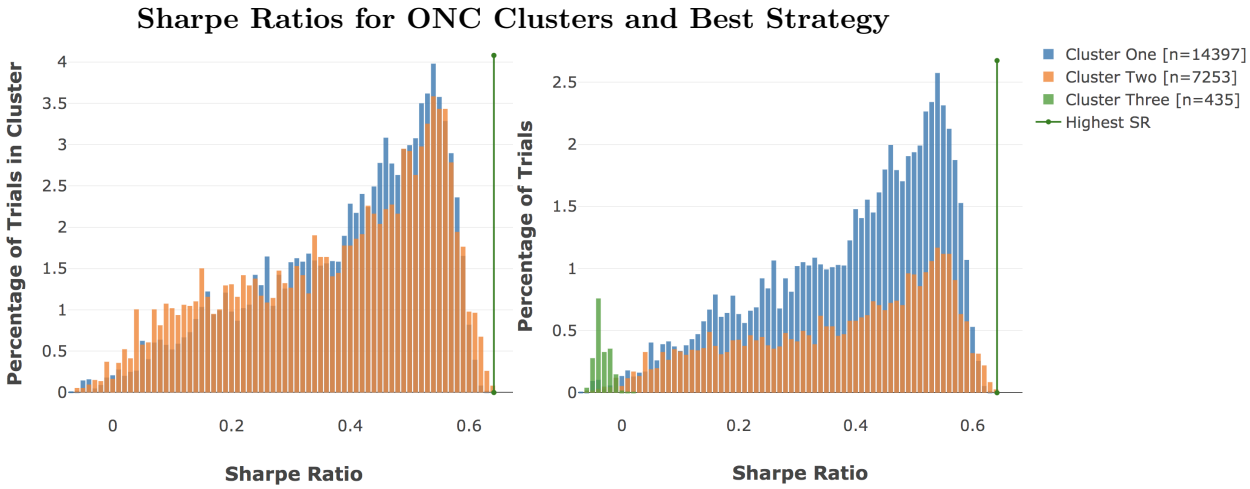
If we consider the two primary clusters, we see that Cluster One contained all the trials with horizon aggregations of [5, 20, 60] and [10, 20, 60], and Cluster Two contained all trials with horizon aggregations of [1, 5, 10]. The nature of the experimentation process, with the combinatorial parameter space exploration (as detailed in Section 5.3), is such that other parameters were evenly split across the 2 clusters (e.g. OGD learning rate, network sizes, initializations and so on).

Under consideration of a single and consistent MMS trading strategy, the clusters here indicate that the networks adapted to at least two different general strategies for predicting prices: one which which was more influenced by the long term fluctuations in the sixty day aggregation, and the second was more influenced by the short term fluctuations in the one day aggregation. The

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`

results presented in Section 8.2, such as the effect of the OGD learning rate on P&L, are then indicative of the networks ability to execute these overarching strategies effectively.

The best Sharpe ratio value (with trading costs applied) was 0.64 and part of Cluster Two, with the [1, 5, 10] price fluctuation horizon aggregations. The distributions seen in Figure 45 indicate that at a general level, Cluster One has more consistent performance, with a higher mean ($\mu_{c1} = 0.393$, $\mu_{c2} = 0.375$) and more negative skewness ($\tilde{\mu}_{3,c1} = -0.736$, $\tilde{\mu}_{3,c2} = -0.543$). Cluster Two on the other hand has higher variance ($\sigma_{c1} = 0.022$, $\sigma_{c2} = 0.028$), with more strategies at both the lower and higher range of Sharpe ratios. This shows that relying on the longer term trends for prediction can offer more reliable and consistent returns. Using shorter term fluctuations can offer higher rewards, though with more variance and risk involved, as per the efficacy of the implementation. These higher and lower variances in P&L returns were also seen in Figure 13. The same patterns are presented in the OGD MSE scores, as seen in Figure 46.



(a) Primary Two Clusters **(b) All Clusters**

Figure 45: Dataset: Actual10 (7.1.1) ; Configuration 11 (11.5.11) & Configuration 16 (11.5.16)
Figure (a) shows shows the distributions of all Sharpe ratios, grouped by the 2 primary clusters produced by the ONC algorithm, and an indication for the best Sharpe ratio, which is in Cluster Two. As noted above, Cluster One has more consistent values, with a higher mean ($\mu_{c1} = 0.393$, $\mu_{c2} = 0.375$) and more negative skewness ($\tilde{\mu}_{3,c1} = -0736.$, $\tilde{\mu}_{3,c2} = -0.543$). Cluster Two on the other hand has higher variance ($\sigma_{c1} = 0.022$, $\sigma_{c2} = 0.028$), with more strategies at both the lower and higher range of Sharpe ratios, including the highest Sharpe ratio from all trials. Figure (b) shows all three clusters produced, highlighting the comparison to the poorly performing configurations.

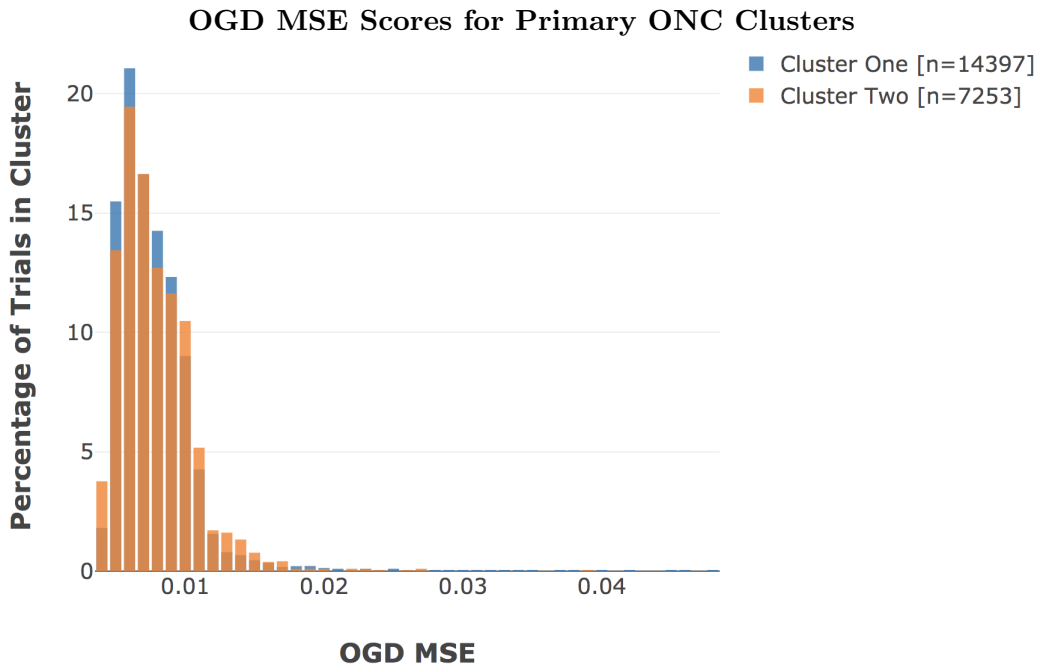


Figure 46: Dataset: Actual10 (7.1.1) ; Configuration 11 (11.5.11) & Configuration 16 (11.5.16)
The distributions of OGD MSE scores can be seen for both primary clusters here, indicating how precise the predictions were. The patterns tie in with those seen in Figure 45, with Cluster Two having a higher variance which leads to more configurations at either end of the MSE range, and including the best performing one.

The lack of further clusterings was probed manually for Cluster Two, running the algorithm for a range of values between 2 and 7000[4]. It was found that while there were sub-clusterings which resulted in a higher mean silhouette score (as per equation (78)), the variance of the silhouette scores also increased dramatically. This in turn, resulted in a lower score for the cost function **q** (as per equation (79)), and so the ONC algorithm did not cluster further.

8.10.3 DSR and PSR Results

Using the clusters produced by the ONC Algorithm (Section 8.10.2), the DSR process as presented in Section 4.8 was followed. The aggregate cluster time series returns were calculated using equations 84 and 85 (Section 4.8.4.1). These are annualized, which allows their variance estimates to then be used to calculate SR^*, which represents the maximum expected observed Sharpe ratio due to variance under the null hypothesis of $H_0 : \widehat{SR} = 0$ (as discussed in Sections 4.8.2 and 4.8.3). Using this SR^* as the benchmark, the PSR calculation ($\widehat{PSR}[SR^*]$) can then be used to determine if the observed $\widehat{SR}$ is a false positive or not (as per Section 4.8.4). This figure is the DSR as a confidence for observing a positive best SR. These steps and calculations are all detailed below.

[4] [2,3,...,49,50,500,1000,1500,2000,2250,2400,2500,2600,2750,3000,4000,5000,6000]

1 **Calculate the Aggregate Cluster Strategy Time Series Return:** Using the ONC results from Section 8.10.2, as well as equations (84) and (85) from Section 4.8.4.1, the 3 cluster SR metrics were calculated to be $\widehat{SR}_1 = 0.488759$, $\widehat{SR}_2 = 0.480554$ and $\widehat{SR}_3 = -0.040758$.

2 **Annualize the Cluster Strategy Sharpe Ratios:**

$$\widehat{aSR}_k = \widehat{SR}_k \sqrt{Frequency_k} \tag{88}$$

$$\therefore \widehat{aSR}_1 = 0.488759\sqrt{365.25} = 9.34093 \tag{89}$$

$$\therefore \widehat{aSR}_2 = 0.480554\sqrt{365.25} = 9.184119 \tag{90}$$

$$\therefore \widehat{aSR}_3 = -0.040758\sqrt{365.25} = -0.778954 \tag{91}$$

3 **Estimate the Variance of Strategy Sharpe Ratios:**

Using equation (87), the cluster strategy Sharpe ratios can be used to estimate the SR variance.

$$\mathrm{E}\left[\mathrm{V}\left[\left\{\widehat{SR}_k\right\}\right]\right] = \frac{\mathrm{V}\left[\left\{\widehat{aSR}_k\right\}\right]}{\mathrm{Frequency}_{k^*}} \tag{92}$$

$$= \frac{\mathrm{V}\left[\{9.34093, 9.184119, -0.778954\}\right]}{365.25} \tag{93}$$

$$= 0.061358 \tag{94}$$

4 **Calculate the Benchmark SR*:**

Using the estimated variance and equation (83), the benchmark SR^* can then be calculated (where $Z[.]$ is the CDF of the standard Normal Distribution and γ is the Euler-Mascheroni constant).

$$SR^* = \sqrt{\mathrm{V}\left[\left\{\widehat{SR}_k\right\}\right]}\left((1-\gamma)Z^{-1}\left[1-\frac{1}{K}\right] + \gamma Z^{-1}\left[1-\frac{1}{Ke}\right]\right) \tag{95}$$

$$= \sqrt{0.061358}\left((1-\gamma)Z^{-1}\left[1-\frac{1}{2}\right] + \gamma Z^{-1}\left[1-\frac{1}{2e}\right]\right) \tag{96}$$

$$= 0.211245 \tag{97}$$

5 **Calculate DSR:**

Using the benchmark, the final DSR calculation can be done with equation (81) (where T is the number of observed returns, $\hat{\gamma}_3$ is the skewness of r_t and $\hat{\gamma}_4$ is the kurtosis of r_t).

$$\widehat{SR} = 0.642632 \tag{98}$$

$$\widehat{PSR}\left[SR^*\right] = Z\left[\frac{\left(\widehat{SR} - SR^*\right)\sqrt{T-1}}{\sqrt{1 - \hat{\gamma}_3\widehat{SR} + \frac{\hat{\gamma}_4 - 1}{4}\widehat{SR}^2}}\right] \tag{99}$$

$$\therefore \widehat{PSR}\left[SR^*\right] = Z\left[\frac{(0.642632 - 0.211245)\sqrt{1555-1}}{\sqrt{1 - 1.969946 * 0.642632 + \frac{7.955865-1}{4}0.642632^2}}\right] \tag{100}$$

$$= Z\left[\frac{16.906872}{0.672459}\right] \tag{101}$$

$$= 1.0 \tag{102}$$

In this case, the benchmark SR^* calculated was 0.211245, and the best $\widehat{SR}$ observed was 0.642632, leading to a $\widehat{PSR}[SR^*]$ of 1.0, thus indicating that the trials certainly contain a strategy which has a positive SR rate. This is a reasonable conclusion, considering the SR distributions in Figure 45.

8.10.4 Framework Success

Much like the results from the CSCV and PBO methodologies, the DSR assessment here provides assurance that a large configuration space can be explored effectively and still have confirmable effects which can be successfully incorporated into a profitable trading strategy. In this case, the DSR indicates with full confidence that even with the expected variation amongst trial Sharpe ratios, there is still a strategy with a positive SR.

8.11 Summary of Actual and Synthetic Data Effects

We found that the use of GBM generated synthetic data to perform conceptual testing without increasing the risk of backtest overfitting did have some benefits, though the stabler characteristics of the data led to different results from actual data in many areas. The primary reason for the different behaviour links back to the complexity of financial time series with unstable dynamics. GBM data, conversely, has asset price fluctuations which follow a stable Log-normal distribution over time. This was seen clearly in the fluctuation distributions in Section 8.5.

As a result of this stable characteristic, longer data horizon aggregations led to better performance for synthetic data as they were more likely to smooth over noise and capture the constant long term means. This was seen in both SAE reproduction and P&L prediction in Section 8.5. Section 8.6 also showed with synthetic data that the Limited Normalization (5) did impact P&L, but within reasonable margins.

The other prominent difference in synthetic data was in the SAE features learnt. Results for SAE networks trained on actual data displayed different classes of features being learnt according to the input horizons, which had fluctuating performances across time periods. Synthetic data however has less inherent features to learn, and so responded with a monotonic performance according to feature size, as seen in Figure 26. In assessing data aggregations and encoding sizes, synthetic data was not an effective indicator for results on actual data.

The results for weight initialization techniques, as discussed in Section 8.4, were slightly different from actual data, but did still follow theoretical expectations and lead to similar conclusions. In this area, synthetic data was a good indicator. Similarly, while synthetic results for Linear network activations would be misleading for actual data, the results for non-linear activations led to similar decisions (as discussed in Section 8.7).

More typical implementation aspects, such as learning rates and network sizes, also followed similar patterns to actual data when training. These parameters tend to be discovered through exploration on any given data set though, and so the added benefit of synthetic data results is marginal.

Ultimately, we found that the inherent differences between synthetic and actual data led to more differences than similarities in the results. GBM generated data can be used to test process and framework implementations, but the behaviours seen in results should be considered very carefully before being applied to actual data (if at all).

8.12 Personal Thoughts on Mechanistic Machine Learning Approaches

The chapter thus far has focused on verifiable results and findings. The process of implementing and assessing the presented framework did however provide me with more learnings, and the development of some personal perspectives. The interested reader, or an individual considering a similar endeavour, may still find value in some of my less quantifiable thoughts.

8.12.1 On Backtest Overfitting and Validation

There was a seemingly trivial aspect of this process which had far more impact than I could have expected: the combinatorial generation of configuration parameters (as discussed in 5.3). As noted while discussing the PBO calculation in Section 8.9, "...any configuration which performs well IS will have all possible OOS parameters tested in combination with it. While some of these combinations may result in poor performance, there will always be a combination of the best IS and best OOS parameter choices. This makes it unlikely that the best configurations will be past the median point for the logit calculation...". This dynamic allowed for many thousands of neural networks to be trained, while incurring a PBO of 0%. The effect is emphasised by the use of the median on large datasets, and so was not present when running proof of concept tests on small, single asset datasets.

Unnecessary efforts spent trying to 'debug' the issue aside, it makes me question the validity of the technique. It would have been quite possible to continue training networks the same way, and I do expect that the PBO would have continued to be 0%. It was only when a more directed approach to configuration choices was taken, that the percentage increased. However, this directed approach is neither required nor necessarily advisable, and a grid search approach to parameters is hardly novel or unusual.

One could argue that this suggests that the approach then is simply not applicable to batch and online machine learning techniques. The context for the methodology and problem though, is one where multitudes of mechanistic models are trained. If the technique cannot be applied to these machine learning frameworks, I'm not sure if it can claim relevance in today's markets.

The ONC algorithm for DSR also had results that I did not expect. Suggesting that the sum total of 22248 predictive networks are fairly represented through 3 "independent trials" does not, on the face of it, seem entirely accurate. While the 2 primary cluster distributions are notably different, I question whether they are the most differentiable representation possible for this result set. I suspect that the symmetries in configurations and results is what prevented further effective clustering (though confirming this was unfortunately not possible given time constraints). The computer scientist in me also balks

at the computational complexity of the method suggested, which is particularly cumbersome.

Undoubtedly, both of these methods are significantly better than previous alternatives, or a naive hold-out test. However, having gone through the application and assessment of both, I feel like they are trying to solve big data problems under expectations of small data dynamics. My opinion come the end of this process is quite simple: backtest validation remains a very difficult, and unsolved problem.

8.12.2 On the Profitability of Machine Learning Models

If I had to name the single epistemic feat at which modern human civilization is most adequate, the peak of all human power of estimation, I would unhesitatingly reply,"Short-term relative pricing of liquid financial assets, like the price of S&P 500 stocks relative to other S&P 500 stocks over the next three months." This is something into which human civilization puts an actual effort. - Eliezer Yudkowsky.

In consideration of whether it would be profitable, or advisable, to try and leverage machine learning techniques in the market, I found 2 sets of results quite enlightening. The first, was the benchmark return noted in 8.8, which was 2.4% with trading costs applied. This was for a predictive model that was perfectly able to predict the price of assets the following week. Evidently, the short trading horizon impacts potential profitability (as well as asset selection and so on). A longer trading horizon however, would undoubtedly make accurate predictions a more difficult task. It may be tempting to get caught up in the hype and glory of machine learning's recent successes. And perhaps, employing every gradient learning technique that has been published, one might come upon a model with near perfect ability to predict next week's prices. Even with this though, one's returns might not even beat a fixed deposit account. The development of an effective MMS is as pivotal to profitability as predictive capabilities. Arguably, machine learning applied to the MMS rather than price prediction might be more beneficial.

Two economists are walking along the street, and one says, "Hey, someone dropped a $20 bill!" and the other says, "Well, it can't be a real $20 bill because someone would have picked it up already." - Ancient Proverb (give or take).

The joke usually goes on, with a game theory like approach to subsequent decisions, in assessing whether a $20 dollar bill on the street could be real or not. The results found in 8.6, showing the very limited benefit in IS training on historical data, were the second enlightening set of findings on this topic. Even if the difficulties of constantly changing financial markets were not present, historical data is in the public domain. The technologies discussed, and widely used, are also in the public domain. As such, one can reason that

most relevant information from historical data has been leveraged to the point of redundancy by now. This is certainly true for any low hanging fruits, so to speak.

There are enough successful quant funds to make it obvious that it is possible to find and exploit inefficiencies in the market. It becomes apparent then that it is most likely to find advantage through the processing of significant amounts of recent cross sectional data, rather than attempting to learn past relationships. Technical trading at this point, certainly in the realm of machine learning, is an endeavour which requires significant resources. As Eliezer notes, a significant amount of effort and resources are dedicated to this task in any form. While I am certainly of the opinion that these techniques can be used for financial profitability, I do also think it would be naive to suggest an individual might be able to compete against institutions on this level for any sustained period of time.

8.12.3 On the Development of Machine Learning Frameworks

While I thoroughly enjoyed working on this project, it surely took on a Sisyphean quality at times. For all of the framework's modularity as a final product, the intermediate development was far more integrated process. Also, though stating the obvious, machine learning models tend to take a *long* time to train. Developing the framework from scratch, I all too often found myself encountering the most painful realisation of something akin to an off-by-one error while analysing results or refactoring code. A couple of minutes to fix the error, and a couple of months to rerun the models.

Sadly, there are no magic solutions to this problem. I wrote unit tests, I double checked my work and I ran proof of concepts. Unfortunately though, when dealing with newly learnt, inherently complex and difficult to diagnose techniques (in a new programming language), these things are bound to occur. My advice, for anyone who might be considering such a task, would be to assume it will happen, and make sure you are easily able to respond to it. Capture as much data as possible at all stages, and ensure tasks are easily rerunnable. Also, consider how much rigour you should have in checking for correctness at the start of the project. Then, triple it.

When considering the nature of techniques to use, I would also advise caution when learning about techniques which have been proven on IID datasets (which, I'd estimate, is the majority of machine learning these days). One of my more fundamental takeaways from this has been a deeper appreciation for the differences in financial time series. One would assume this should neither need to be stated not learnt at this point. However, machine learning techniques such as RBM based pretraining are so well noted and spread (with time series applications included), that one expects them to at least help in this context. The hours of 'debugging' and validation could potentially be skipped

in these instances. The same would apply for the usage of synthetic data, with its own inherent differences from real financial time series.

9 Conclusion

The work we present here builds on mechanistic and predictive machine learning approaches to financial market data. In doing so, we provide a novel framework which was shown to be effective through sophisticated financial strategy validations. The primary goals of the paper were met: the framework experiment results proved the efficacy of such an approach through Profit and Loss (P&L) returns and validations, and financial feature selection was shown as both possible and effective. This was done whilst also developing a refined perspective on how these techniques behave under the complex and dynamic behaviour of financial markets. The availability of recent cross-sectional data, and the ability to process it effectively, is highlighted throughout the results as a core consideration for these models.

The framework we constructed embodies a configurable approach, based on decoupled modules throughout the process. To meet the goals set out, it incorporates several key techniques: deep learning neural networks for stock price fluctuation prediction; Stacked AutoEncoders for feature selection; Combinatorially Symmetric Cross-Validation (CSCV) & Probability of Backtest Overfitting (PBO) in order to assess the returns from the Money Management Strategy (MMS), and the likelihood that backtest overfitting has taken place; and Deflated Sharpe Ratio (DSR) in order to assess the likelihood of a positive Sharpe ratio having been observed. This was implemented with a scientific approach in mind, such that all results are reproducible using the libraries and configurations made available (see Sections 6 and 11.5, respectively).

The nature of neural networks as non-linear universal approximators combined with the brute force approach to learning market dynamics allowed us to establish market characteristics, and how they relate to models which, conceptually, could not be made more complex. The most pervasive observation in this sense, was the highly dynamic and complex nature of financial time series. The unstable behaviour of asset relationships over time was highlighted as one of the more challenging barriers to learning. While machine learning models are expected to excel in big data environments, in financial markets there is in fact a lack of relevant data with information and signal (especially if one considers the need of validation datasets).

To this point, we found in Section 8.3 that the IS training on historical data had a limited benefit, and could become detrimental if trained on extensively. This was confirmed by showing the negligible impact of increasing training time as well as the surprisingly small impact of large reductions in the training data (see Figures 19 and 20). It was made clear that the increased performance for In Sample (IS) data was not necessarily linked to Out of Sample (OOS) performance. This gives weight to the importance of online learning methods for financial applications, which in turn makes network initialization a

significant configuration choice. Methodologies for effective initialization were explored in Section 8.4. We found positive results in the He-adj initialization presented in Section 4.5 (see Figures 22 and 23), as well as poor performance in Restricted Boltzmann Machine (RBM) based pretraining (see Figure 21).

Naturally then, the primary determinants of OOS P&L were found to be those which affected the online learning data and the model's ability to adapt to it. The horizon length of input data aggregations was shown to be the primary distinguishing property of trade correlations in Section 8.10 (see Figure 45). In Section 8.2, the differences between these groups were explored. We found that networks provided with one day fluctuations adapted to short term price prediction strategies, and so excelled in configurations which allowed them to adapt quickly to new information. Conversely, networks provided with longer horizons, such as sixty day fluctuations, adapted to following long term trends and performed best under configurations which had more careful adaptations (see Figures 17 and 18). The short term strategies had the highest OOS P&L returns, but also the highest variance and a lower mean in these returns (see Figure 13). This highlighted both the increased value in recent information in financial markets, as well as the difficulty in using it due to the amount of noise present.

Reducing the noise through the usage of SAE based feature selection provided interesting results, as also noted in Section 8.2. We found that the SAE networks learnt fundamentally different features according to the encoding sizes made available. These different features then had a non-linear effect on OOS P&L. Larger encodings (15, 20, 25) had a smoothing effect on the input data, and performance was generally better at higher encodings when used in strategies focusing on short term fluctuations. Smaller encodings (5, 10) were shown to learn more inherent features, resulting in higher performance in many circumstances. These features were not necessarily stable through time though, which was shown through the inconsistent performance of features learnt for an encoding size of 10. This issue was not present in the networks which were limited to 5 features, which learnt more robust and generalisable features. These smaller networks had competitive performance when compared to networks learning larger feature sets. This showed that feature selection in financial time series is both possible and beneficial, despite the complexity present (see Figures 17 and 18).

We also explored the complexity of financial time series through the results of data processing and learning optimizations in Section 8.6. The effects of outliers was seen to be significant when comparing scaling techniques, with substantial differences between standardization and normalization (see Figure 27). A strong relation was shown between regularization benefits and the smaller SAE encodings, in line with the extent of generalisable features being learnt (see Figure 29). Learning optimizations for IS batch training, such as regularization and learning rate schedules, were shown to have benefits for IS

P&L results, but little carry through to OOS performance (see Figures 32 and 33). This once again indicated the complex but unstable nature of financial solution spaces.

Comparisons of results against synthetic data generated through Geometric Brownian Motion (GBM) were made throughout the findings, though particularly so in Sections 8.5 and 8.11. GBM generated data exhibits constant characteristics by design, which contrasts heavily against the non-stationary nature of financial time series. We found that, for the most part, these differences led to results for synthetic data being non-transferable to actual data. The use of synthetic data is still valuable in that it allows very effective framework and process testing without increasing the likelihood of backtest overfitting, but beyond that its usage is limited.

The full set of performance metrics was explored in Section 8.8, with the distributions of OOS P&L and Sharpe ratios being considered relative to the benchmark (a model with 100% prediction accuracy). We found that the framework presented is able to generate effective models, with return distributions showing significant negative skewness, and achieving performance in the region of 20%-30% from the upper bound benchmark (see Figure 39). That said, the benchmark itself only had a return rate of 2.4%. So, while the strategies were objectively profitable, they are being considered in a region that would not be competitive in the market. This speaks to a more foundational issue in financial markets, where the MMS put into place is a critical component of profitability. The predictability of trading signal in and of itself is not enough to generate competitive gains.

The most challenging aspect of such a mechanistic approach to learning is ensuring that the model has not resulted in backtest overfitting, as discussed extensively in Section 2.6. Probing and validation of the generalisation error in this case was done through the use of Bailey's Probability of Backtest Overfitting methodology [8], as well as the use of Prado's Deflated Sharpe Ratio [90]. We found through the PBO figure that there was a low likelihood of backtest overfitting having occurred. We also found through the use of the DSR figure, that there was a certain likelihood of a positive Sharpe ratio. These results are discussed in Sections 8.9 and 8.10, respectively. These verifications give weight to the validity and efficacy of the full framework presented, showing that the solution space for financial asset price fluctuation prediction can be explored in a brute force fashion without necessarily overfitting.

The system we presented then, was successful in achieving the aims set out for it. Further, in the results we provide a cohesive view of how components interact and might be changed in a future implementation. The structures, initializations and training of Stacked AutoEncoders and Feedforward networks were explored, and found to be effective for feature selection and price fluctuation prediction in financial time series. An extensive comparison of synthetic

and actual data was made, such that it might guide further usage in similar systems. The Combinatorially Symmetric Cross-Validation and Probability of Backtest Overfitting methodologies were implemented, considered, and shown that they are able to work for a novel implementation of backtest overfitting in machine learning models. Their results were further validated through the usage of the Optimal Number of Clusters and Deflated Sharpe Ratio algorithms to detect positive Sharpe ratios. A view on the phenomenology of financial markets was also presented, based on the experiment results. The value of recent cross sectional data over historical data is shown through limited in-sample training of the networks (both by length of training and dataset size), the effects of Stacked AutoEncoder feature selection developed in-sample, as well as out-of-sample learning configurations. In full, a framework was presented and found to be able to deliver legitimate profit and loss rates for out-of-sample data through mechanistic machine learning, and was done while incurring a low probability of backtest overfitting or spurious results otherwise.

10 Future Work

Several areas of the system have been highlighted as worthy of further work based on the results seen so far:

1. The incorporation of Open, High, Low and Volume data into the feature set would likely offer significantly beneficial signal gains for the predictive networks.
2. In light of the historical data redundancy seen, an improved and optimized online learning phase would be worth considering.
3. The updating of SAE models is the another likely area of beneficial development, possibly based on rolling time windows in order to fully take advantage of recent data benefits, or using a bootstrapped dataset in order to add time based weights to observations.
4. While RBM pretraining was shown to be ineffective, it is possible that a greedy layerwise approach using variance based weight initialization could still be used in order to train more efficient SAE networks.
5. Data processing methods such as the Error Function could be implemented in order to better deal with outliers, which are shown to be prominent in financial data.
6. Further exploration of using input Dropout for price fluctuation prediction is warranted, in order to try and enforce correlation and relationship recognition within the FNN.

11 Appendix

11.1 Additional Results

11.1.1 Additional Results for Section 8.2 - Feature Selection

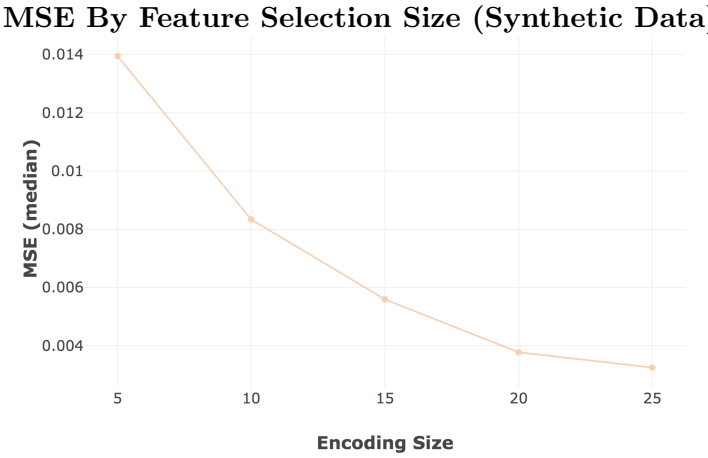


Figure 47: Dataset: Synthetic10 (7.2.2) ; Configuration 7 (11.5.7)
The MSE scores here show that while feature selection and reduction is possible through the SAE networks, the ability to reproduce input does decrease monotonically with the encoding layer size. This is not of real concern though unless exact reproduction (with noise included) is of interest.

11.1.2 Additional Results for Section 8.4 - Weight Initialization Techniques

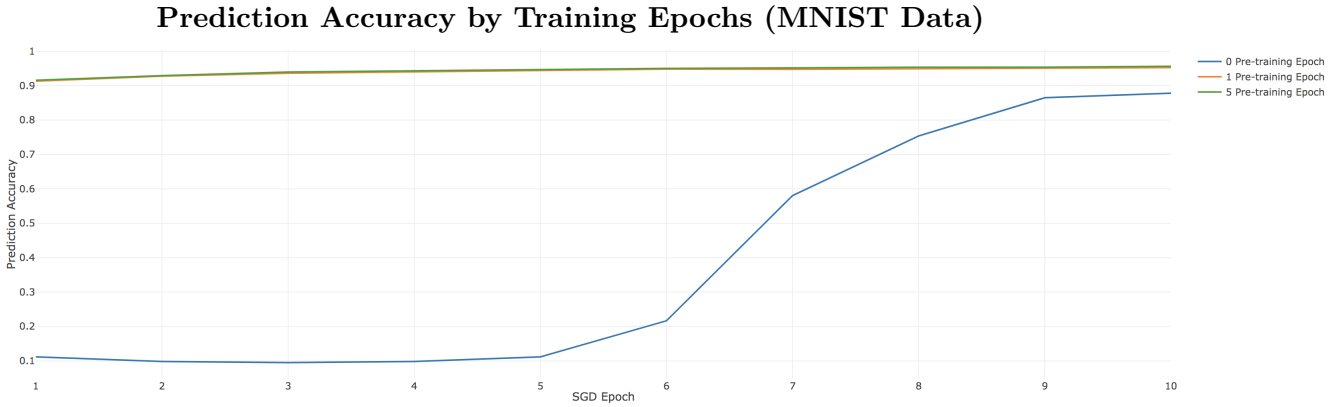


Figure 48: The plot above shows the prediction accuracy achieved on the MNIST dataset according to the number of pretraining epochs using the RBM greedy layerwise methodology as per Section 4.3. There is a clear indication that pretraining allows the network to achieve much higher accuracy much quicker.

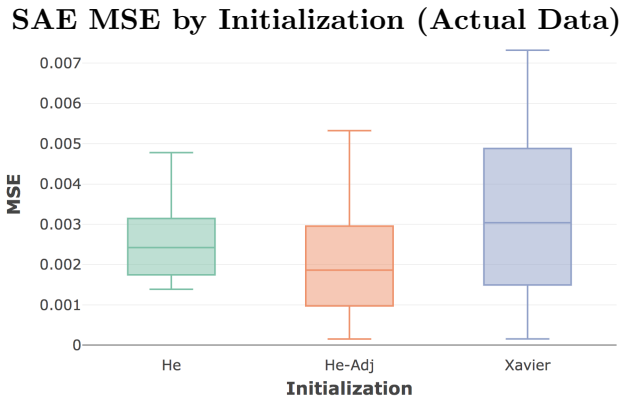


Figure 49: Dataset: Actual10 (7.1.1) ; Configuration 12 (11.5.12)
The SAE MSE results here continue the trend seen for AGL (as seen in Section 8.4), where He-Adj offers the best performance. Both He and He-Adj suffered with larger learning rates, resulting in dying ReLU's and networks with null outputs, hence the smaller sample sizes. It is possible a more suitable set of learning rates would lead to better results, though the results presented in Section 8.4 led to a favouring of He-Adj in any case.

11.1.3 Additional Results for Section 8.6 - Complexity of Financial Time Series

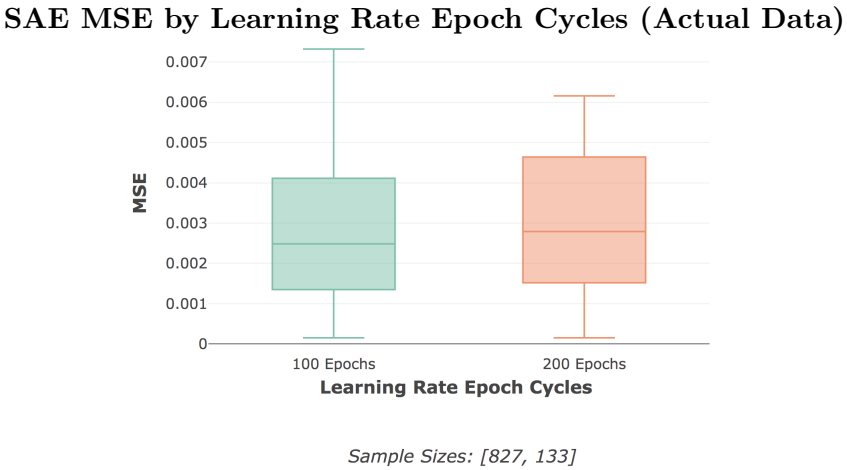


Figure 50: Dataset: Actual10 (7.1.1); Configuration 13 (11.5.13)
This figure shows the MSE scores for SAE networks. Both 100 and 200 Epochs achieved the same minima. The 100 Epoch configurations showed larger variance, but were for a far larger sample set. Neither shows a clearer benefit than the other here, and more testing would be required in order to establish a preferable epoch length. Synthetic data for the SAE MSE scores behaved similarly, as seen in Figure 51.

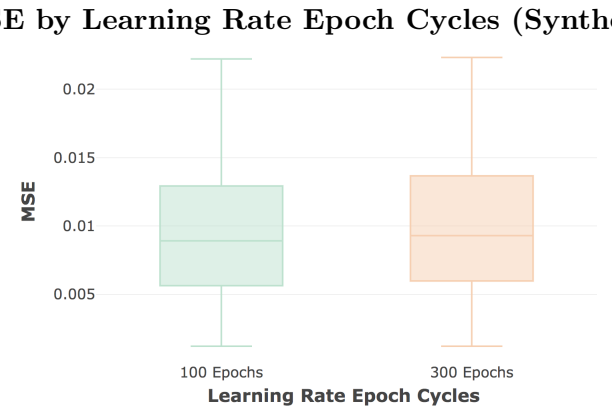
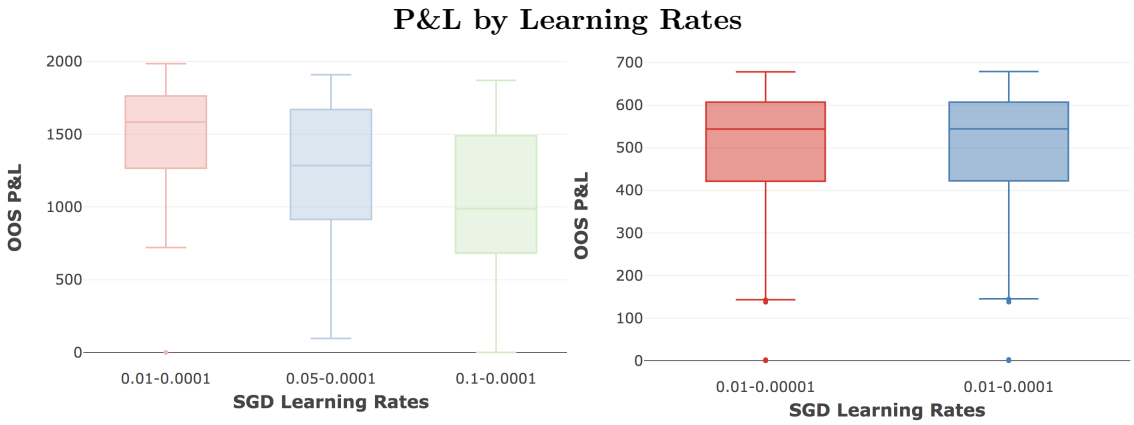


Figure 51: Dataset: Synthetic10 (7.2.2) ; Configuration 7 (11.5.7)
The SAE networks did not show significant differences according to the epoch cycles. As discussed in Sections 8.5.1 and 8.3, the structure of synthetic data for SAE replication is structurally less complex, and so learning optimizations are less likely to produce large differences.



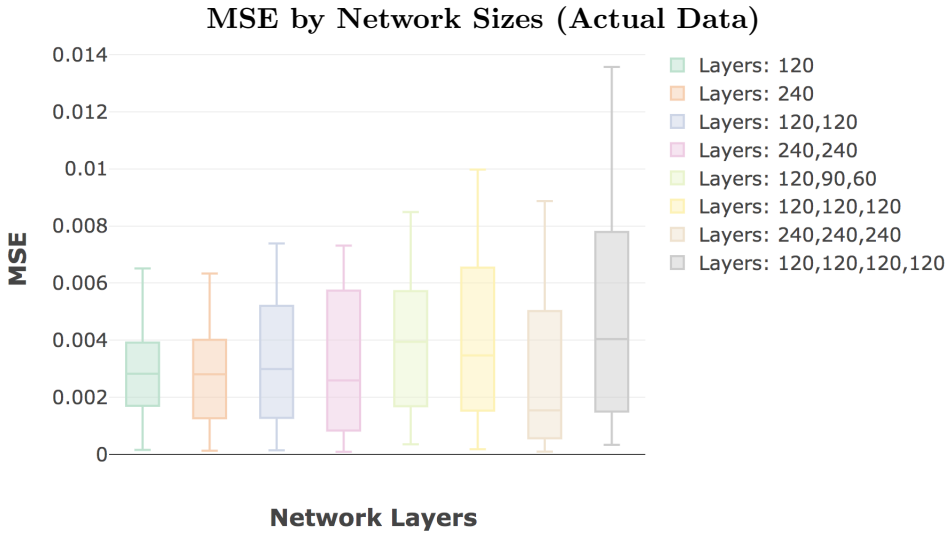
(a) Synthetic Data

(b) Actual Data

Figure 52: Dataset: Synthetic10 dataset (7.2.2), Actual10 dataset (7.1.1), Configuration 9 (11.5.9) & Configuration 13 (11.5.13)
The learning rates for actual data showing in Figure (b) here were chosen according to the best rates seen in the SAE training. There is no substantial difference shown by changing the lower bound by a factor. The results for synthetic data were interesting in that the networks had higher P&L when trained with smaller learning rates, suggesting that the predictive task still warrants a more careful solution space exploration despite the less complex nature of the data.

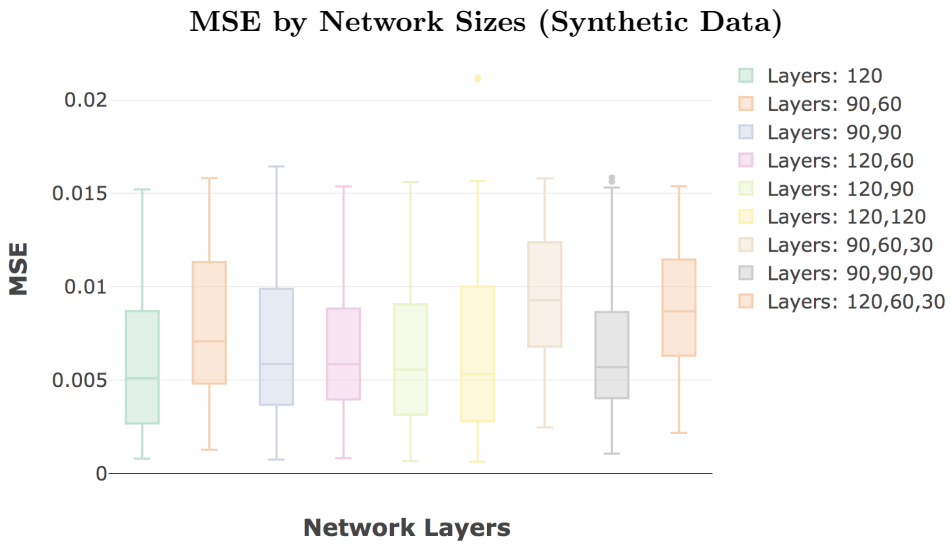
11.1.4 Additional Results for Section 8.7 - Network Structure and Training

Figures 53 - 56 show the effect of network layer sizes on SAE MSE and OOS P&L figures, for both actual and synthetic data. Training parameters such as number of SGD epochs were not adjusted for network size, and so some smaller networks achieved higher performance as they had more accommodating training. The general trends should be focused on as the indicator for more tailored training.



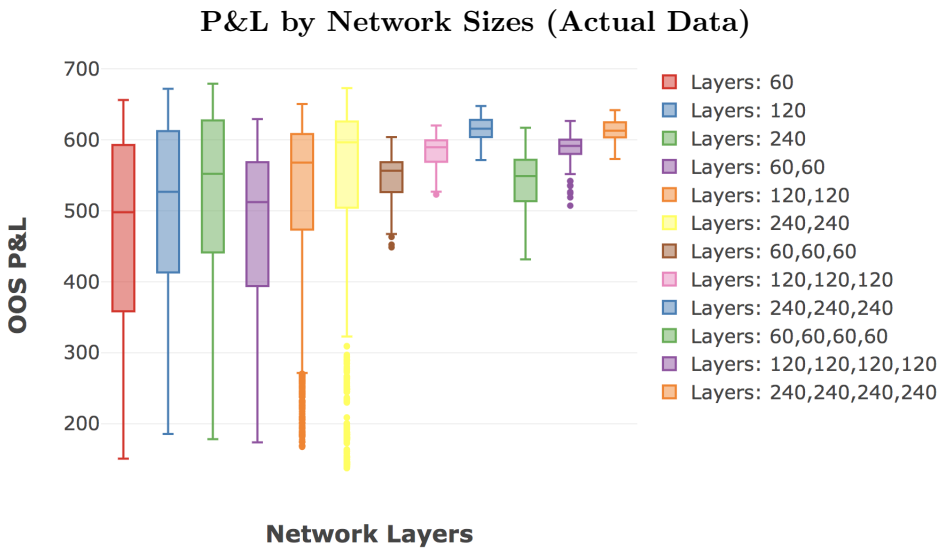
Sample Sizes: [110, 90, 90, 90, 90, 238, 162, 90]

Figure 53: Dataset: Actual10 (7.1.1) ; Configuration 12 (11.5.12)
This figure shows the MSE achieved by SAE networks with the indicated sizes, showing a general increase in performance as both layers and layer sizes increase (though variance also increased significantly with more layers). The network sizes here indicate the final N layers for the SAE, rather than the $2N + 1$ that are present in training. As noted in Section 8.7, the more typical structure of SAEs with descending layer sizes did not show better performance.



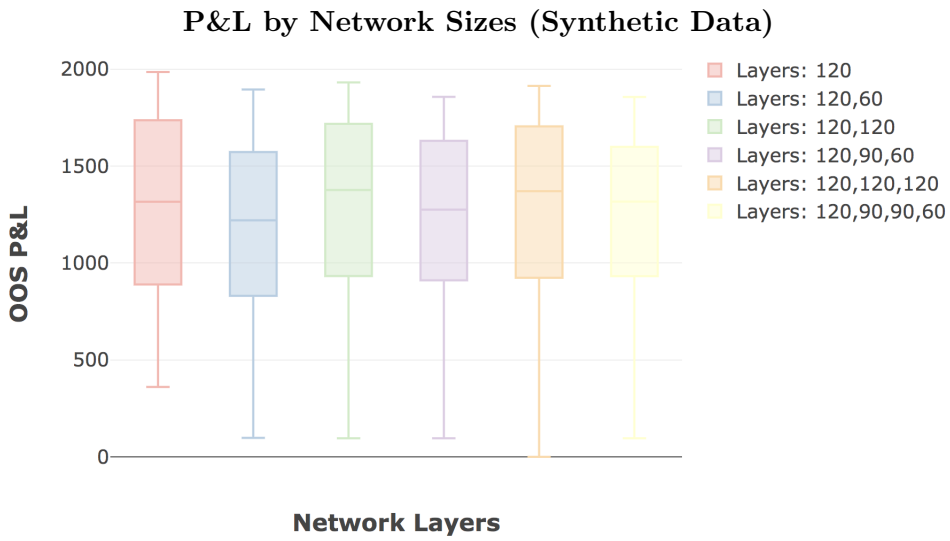
Sample Sizes: [264, 240, 234, 236, 236, 230, 214, 215, 216]

Figure 54: Dataset: Synthetic10 (7.2.2) ; Configuration 7 (11.5.7)
This figure shows the MSE achieved by networks with the indicated sizes. There is a clear increase in performance as layer sizes increase. The gains from increasing the number of layers is less definitive, highlighting the impact of training difficulties with deep networks. The network sizes here indicate the final N layers for the SAE, rather than the $2N + 1$ that are present in training. As noted in Section 8.7, the more typical structure of SAEs with descending layer sizes did not show better performance.



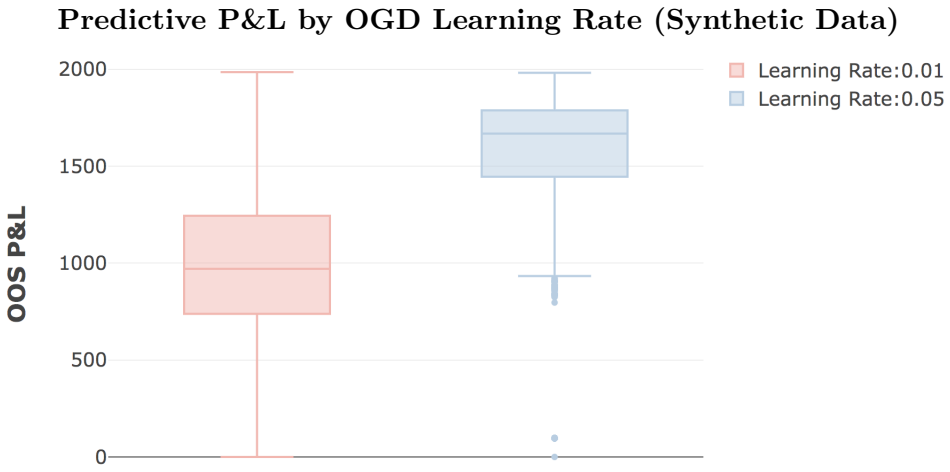
res: [3456, 3442, 3450, 3426, 3358, 3339, 203, 161, 145, 187, 146, 143]

Figure 55: Dataset: Actual10 (7.1.1) ; Configuration 13 (11.5.13)
This figure shows the P&L achieved by networks with the indicated sizes. There is a clear increase in performance as layer sizes increase. The gains from increasing the number of layers is less definitive, highlighting the impact of training difficulties with deep networks. Networks that suffered from exploding ReLU's (and hence have approximately 0 P&L) have been excluded for the more effective visualization.



Sample Sizes: [540, 180, 540, 540, 792, 360]

Figure 56: Dataset: Synthetic10 (7.2.2) ; Configuration 9 (11.5.9)
This figure shows the P&L achieved by networks with the indicated sizes. There is a clear increase in performance as layer sizes increase. The gains from increasing the number of layers is less definitive, highlighting the impact of training difficulties with deep networks. Networks that suffered from exploding ReLU's (and hence have approximately 0 P&L) have been excluded for the more effective visualization.



Sample Sizes: [1477, 1475]

Figure 57: Dataset: Synthetic10 (7.2.2) ; Configuration 9 (11.5.9)
The networks trained on synthetic data showed a sharp P&L increase as the OGD learning rate increased. It was not tested if they would experience the same turning point and begin to degrade as seen in networks for actual data in Section 8.2, though it seems reasonable to expect so.

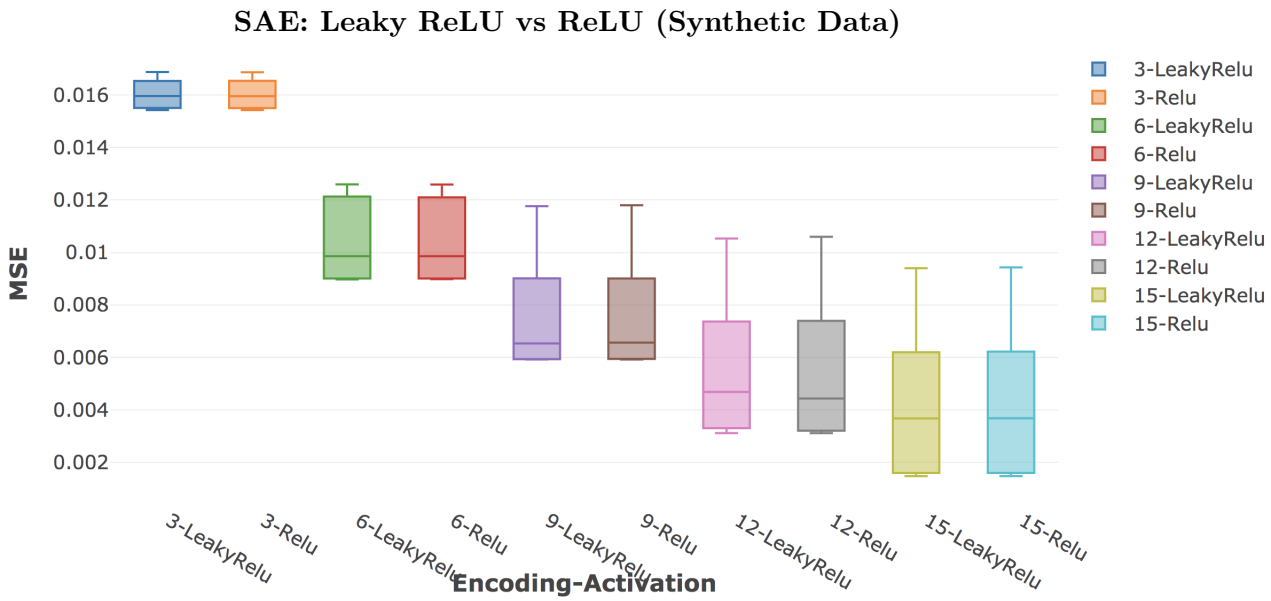


Figure 58: Dataset Synthetic6 (7.2.1) ; Configuration 17 (11.5.17)
The plot above shows the SAE MSE, grouped by encoding size and activations, showing a marginal increase in performance for Leaky ReLU activations.

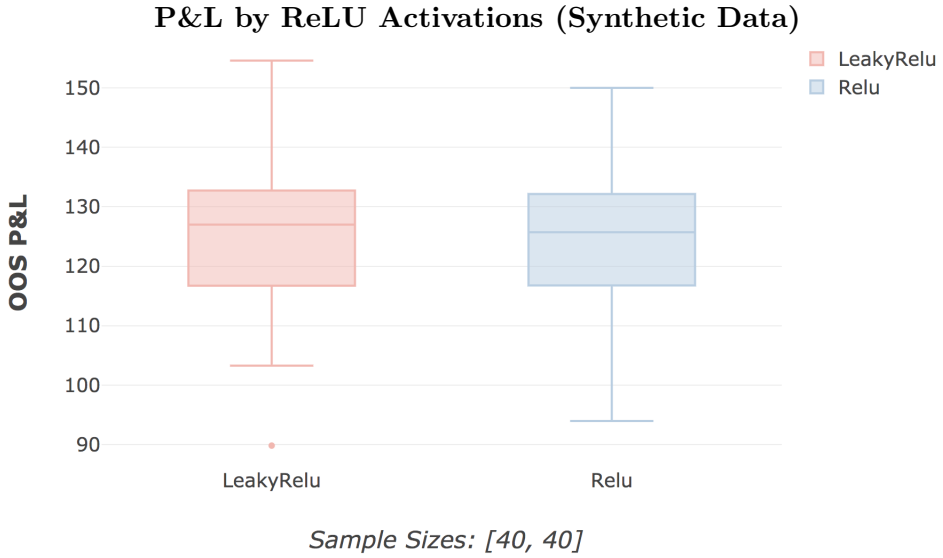


Figure 59: Dataset Synthetic6 (7.2.1) ; Configuration 5 (11.5.5)
The plot shows the FFN P&L, grouped by activation, showing some improvements from the Leaky ReLU activation. The effect is most noticeable in reducing the lower bounds of performance (10.8%), though has a clear benefit in the upper bounds as well (3.4%).

11.2 'How To' Guide for Julia Libraries

The `Main_Tutorial` module is made available to provide a step by step walk-through for using the provided libraries. Configurations and methods are detailed for some naive training on an AGL dataset (see Section 7.1.3). It should be noted that these are not necessarily the optimal parameter choices possible for this exercise.

11.2.1 Create the Database

The first step is to create the database which will be used with the `CreateDatabase` function [30]. Once created, the `DatabaseOps` module should be updated so that the `db` variable references the same database by name [31].

11.2.2 Data Specifications

The JSE dataset can be read into memory using `ReadJSETop40Data`, and particular assets can then be filtered on (as is done for AGL). The data specifications such as horizons, prediction points and segregation points are also set in this step.

11.2.3 Train SAE Networks

Once the database has been created, the SAE networks can be trained. This step uses the `RunSAEExperiment` method for training SAE networks, passing through the configurations chosen and noted in `Main_Tutorial` [35]. The configurations specify the parameters for the SAE networks and their SGD training, as detailed in Software Section 6.

11.2.4 Select Best SAE Networks

The selection of SAE networks is done semi-manually, and is dependent on the task at hand. In this case, the provided `GetBestSAE` function filters according to data horizons specified. This may differ according to what is being investigated. As the case may be, the selection of all 'best' SAE networks should be specified by `configuration_id` in the `sae_choices` vector.

11.2.5 Train FFN Networks

Much like the SAE networks, the FFN networks can be trained in a combinatorial manner using the noted configuration choices and the `RunFFNExperiment` function [34]. These are available as specified in Section 6.

11.2.6 Run Batch Process Diagnostics

The FFN training will include the recording of IS and OOS predictions for the specified data. This allows the running of the `RunBatchTradeProcess` and

`RunBatchAnalyticsProcess` functions from the `DatabaseBatchProcesses` module [29]. These processes record the per trade returns, as well as the aggregated performance metrics such as Sharpe ratios or strategy P&L.

11.2.7 Diagnostic Visualizations

Once the aggregated performance metrics have been calculated and recorded, they can be used for diagnostic visualizations [32]. A set of example visualizations have been included in the `Main_Tutorial` module, though any from the full set can be used, which are specified in Appendix 11.6.

11.2.8 CSCV & PBO

Subsequently, the CSCV analysis can be run using `ExperimentCSCVProcess` [23]. The set of `configuration_id`s for the FFN networks is specified, with the split values to be run. The method will save the logit distribution graph in the `GraphOutputDir` directory and return the PBO score.

11.2.9 DSR

The DSR implementation is split across the Julia and Python code base. In Julia, the `WriteCSVFileForDSR` method will write out the required return data for the specified FFN configurations. Using the Python module `DSR_proc`, the same CSV file can be specified and used to run the DSR analysis [25].

11.3 Relational Database Schema Diagram

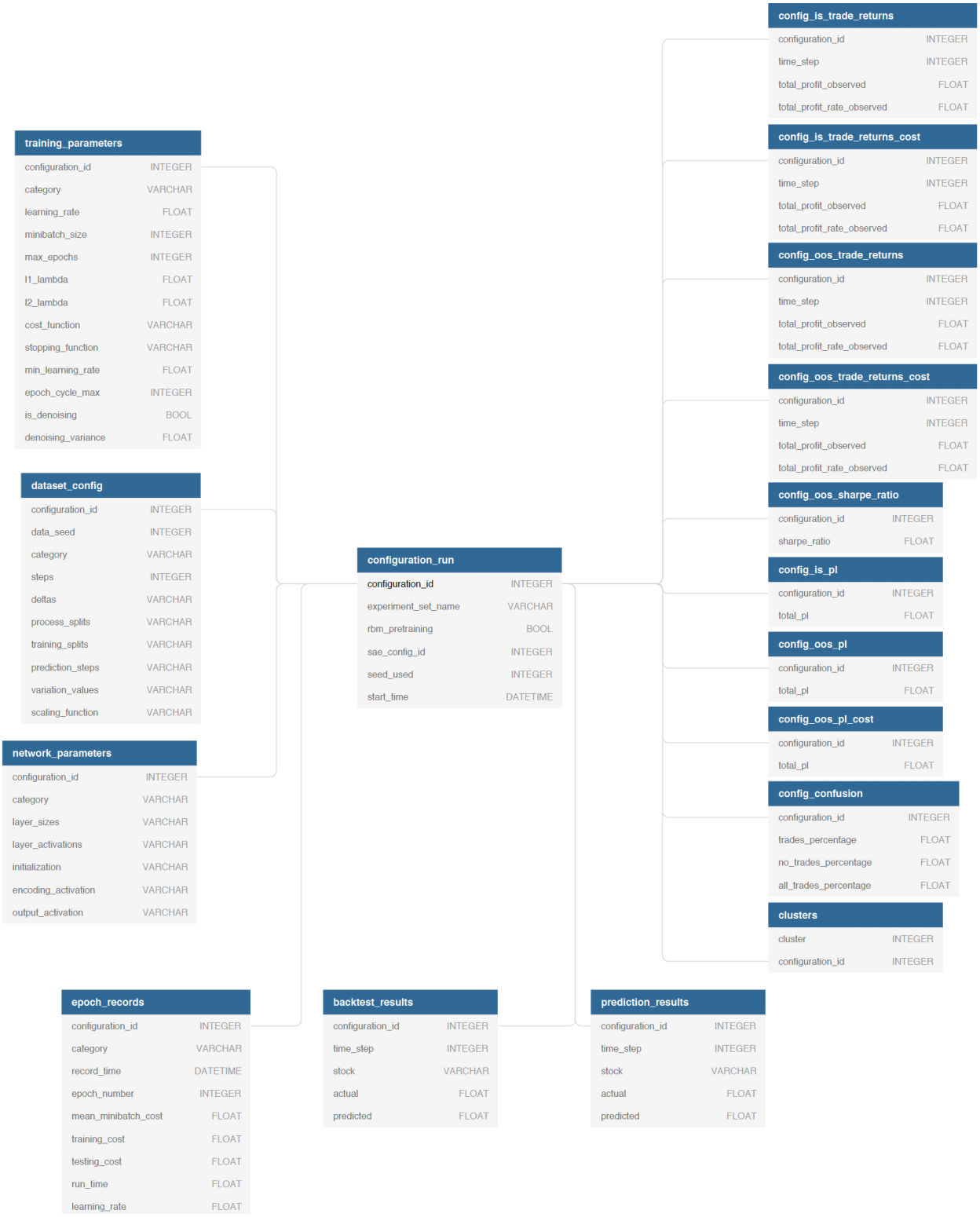


Figure 60: This diagram maps out the relational schema of the database created and maintained, as per Section 6.7. The schema is centered around `configuration_run` which is the primary representation for each trial. The 3 tables to the left (`training_parameters`, `dataset_config` and `network_parameters`) all capture processing and training configurations of the trials. The 3 tables to the bottom (`epoch_records`, `backtest_results`, `prediction_results`) capture the intermediate training data and all price predictions made during training. The remaining tables to the right mostly contain aggregate performance measures. These are calculated from `config_is_trade_returns` and `config_oos_trade_returns` which in turn are calculated from `backtest_results` and `prediction_results` (with respective 'cost' variations including trading costs).

11.4 Dataset Information

The "Actual10" dataset (as per Section 7.1.1) was used for the testing and validation of the framework presented. It was produced by Gant [56] using Bloomberg data and consists of daily asset price relative data for the constituents of the JSE Top 40 (as of January 2003) from 2003 to 2018 [47]. A snapshot of the data can be seen in Figure 61 below.

Snapshot of JSE Top 40 Dataset

date	ACL	ABG	ABI	AGL	AMS	ANG	AOD
2003-01-03	1.013216191960701	1.0020025116247497	0.9963768115942029	1.0095519864750633	1.0193266832917707	1.0	1.0081967213114753
2003-01-06	0.9576076416730379	1.0100602940180203	1.001090909090909	0.993234530687432 1	1.0198776758409787	1.0	1.008130081300813
2003-01-07	1.032689999975659	1.0076125959958415	1.007991282237559	1.020417797710374	1.0644677661169415	1.0092213114754098	1.0080645161290323
2003-01-08	1.00901336212078	0.9946082673234374	0.9657657657657658	0.9986616436998116	0.9645070422535211	0.9910659898477158	1.0
2003-01-09	0.9738555373607455	1.0227546513184314	0.9990671641791045	0.9759765721943714	0.990070093457944	1.0	1.0456
2003-01-10	0.9977644036450776	1.0254220651747155	1.0009337068160598	1.0123073793419113	0.9970501474926253	1.0038926449498053	1.009946442234124
2003-01-13	1.0369940306616758	1.053731533773651	1.0167910447761195	1.0135141924139663	1.0062130177514792	1.010204081632653	1.0795454545454546
2003-01-14	1.005406320201601	1.0254958364875095	1.0174311926605504	1.0239912758996728	1.0205821817112615	1.006060606060606	0.9964912280701754
2003-01-15	0.9784933095676758	1.013375852599876	1.0171325518485121	1.0	0.9853068280034573	0.9859437751004017	1.0
2003-01-16	1.0	0.9811188811188812	0.9663120567375887	0.9765708200212992	0.9853801169590644	0.9938900203665988	0.9929577464788732
2003-01-17	0.9879133360983332	0.9615407460204323	0.9908256880733946	0.996001454016721	0.9970326409495549	1.0	0.9645390070921985
2003-01-20	1.0011115802329549	1.010007103808259	1.0005555555555556	1.0174021234240211	1.0178571428571428	1.0051229508196722	1.0073529411764706
2003-01-21	1.0168886961535804	1.0120791413106633	0.9996298352766981	0.9828955306624925	1.0073099415204678	0.9991845056065239	1.0
2003-01-22	0.983391794778069	0.9589074208363549	1.0	0.9772395487723956	0.9854862119013063	1.0059171597633136	0.9635036496350365
2003-01-23	0.9722222884087979	0.9918389210990672	0.9979633401221996	0.9945168737692673	0.996759941089838	0.9979716024340771	0.9848484848484849
2003-01-24	0.9942847479725805	1.0193474600501955	1.0018552875695732	0.977963642570624	0.975177304964539	0.975609756097561	1.01
2003-01-27	0.9655187713058203	0.9850713706912672	0.9955555555555555	0.9718295109435541	0.9848484848484849	1.0	1.0053313023610053
2003-01-28	0.995238824640678	1.024171860663777	1.0044642857142858	1.0058010057471265	0.9907692307692307	0.9895833333333334	1.0075757575757576
2003-01-29	0.9868409239569573	0.9925859565660623	1.000925925925926	1.0014463510883347	1.0248447204968945	1.0	1.0150375939849625
2003-01-30	0.9818177092133137	1.0100837197721826	1.0120259019426456	1.0287604307823979	0.990909090909091	0.9896842105263158	0.9962962962962963
2003-01-31	0.9814809912104204	0.9906331844091819	0.9689213893967094	1.0	1.0061162079510704	0.9997872793022761	0.9747211895910781
2003-02-03	1.0163544406231033	1.008242356380828	1.0190566037735849	1.0223928453819089	0.9924012158054711	0.9787234042553191	1.0030511060259344
2003-02-04	1.0074232725033705	0.9937993583415595	1.0016663580818368	0.989048805709539	1.010719754977029	1.001086956521739	0.9885931558935361

Figure 61: This figure shows a snapshot of the dataset used as a source for the "Actual10" dataset (as per Section 7.1.1), showing the first month of closing prices for 7 out of the 40 assets.

11.5 Configuration Sets Used

11.5.1 Configuration 1

Parameter	Values
category	SAE
sae_config_id	0
seed_used	73
steps	5000
deltas	1,5,20
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	2
scaling_function	NormalizeData;StandardizeData
layer_sizes	(30,60,5); (30,60,60,5); (30,60,60,60,5); (30,120,5); (30,120,120,5); (30,120,120,120,5); (30,60,15); (30,60,60,15); (30,60,60,60,15); (30,120,15); (30,120,120,15); (30,120,120,120,15); (30,60,25); (30,60,60,25); (30,60,60,60,25); (30,120,25); (30,120,120,25); (30,120,120,120,25)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotUniformInit
encoding_activation	LinearActivation;SigmoidActivation;ReluActivation
output_activation	LinearActivation;SigmoidActivation;ReluActivation
learning_rate	0.001;0.1;1.0;0.01;0.05;0.25;2.0
minibatch_size	20
max_epochs	500
l1_lambda	0.0
min_learning_rate	0.0

epoch_cycle_max	0
ogd_learning_rate	0.0
Total Samples	**1129**

11.5.2 Configuration 2

Parameter	Values
category	SAE
sae_config_id	0
seed_used	73
steps	5000
deltas	1,5,20
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	2
scaling_function	NormalizeData
layer_sizes	(30,60,5); (30,60,60,5); (30,60,60,60,5); (30,120,5); (30,120,120,5); (30,120,120,120,5); (30,60,15); (30,60,60,15); (30,60,60,60,15); (30,120,15); (30,120,120,15); (30,120,120,120,15); (30,60,25); (30,60,60,25); (30,60,60,60,25); (30,120,25); (30,120,120,25); (30,120,120,120,25)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotUniformInit
encoding_activation	LinearActivation;SigmoidActivation;ReluActivation
output_activation	LinearActivation;SigmoidActivation;ReluActivation
learning_rate	0.001;0.1;1.0;0.01;0.05;0.25
minibatch_size	20
max_epochs	500
l1_lambda	0.0
min_learning_rate	0.0
epoch_cycle_max	0

ogd_learning_rate	0.0
Total Samples	**648**

11.5.3 Configuration 3

Parameter	Values
category	FFN
sae_config_id	3560;3574;3580;3590;3613;3620;3626;3638;3650;3671
seed_used	29;38;60;2;84;31;229;227;52;43
steps	5000
deltas	1,5,20
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	2
scaling_function	NormalizeData;LimitedNormalizeData
layer_sizes	(15,40,6); (12,40,6); (9,40,6); (6,40,6); (3,40,6); (15,40,40,40,6); (12,40,40,40,6); (9,40,40,40,6); (6,40,40,40,6); (3,40,40,40,6); (15,80,6); (12,80,6); (9,80,6); (6,80,6); (3,80,6); (15,80,80,80,6); (12,80,80,80,6); (9,80,80,80,6); (6,80,80,80,6); (3,80,80,80,6)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotNormalInit
encoding_activation	;LinearActivation
output_activation	LinearActivation;ReluActivation
learning_rate	1.0e-5;0.0001;0.001;0.01
minibatch_size	20
max_epochs	500
l1_lambda	0.0
min_learning_rate	0.0
epoch_cycle_max	0

is_denoising	0
denoising_variance	0.0
ogd_learning_rate	0.0001;0.001
Total Samples	**960**

11.5.4 Configuration 4

Parameter	Values
category	FFN
sae_config_id	3560;3574;3580;3590;3613;3620;3626;3638;3650;3671
seed_used	29;38;60;2;84;31;229;227;52;43
steps	5000
deltas	1,5,20
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	2
scaling_function	NormalizeData;LimitedNormalizeData
layer_sizes	(15,40,6); (12,40,6); (9,40,6); (6,40,6); (3,40,6); (15,40,40,40,6); (12,40,40,40,6); (9,40,40,40,6); (6,40,40,40,6); (3,40,40,40,6); (15,80,6); (12,80,6); (9,80,6); (6,80,6); (3,80,6); (15,80,80,80,6); (12,80,80,80,6); (9,80,80,80,6); (6,80,80,80,6); (3,80,80,80,6); (15,20,6); (12,20,6); (9,20,6); (6,20,6); (3,20,6);(15,20,20,6); (12,20,20,6); (9,20,20,6); (6,20,20,6); (3,20,20,6); (15,20,20,20,6); (12,20,20,20,6); (9,20,20,20,6); (6,20,20,20,6); (3,20,20,20,6)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotNormalInit
encoding_activation	;LinearActivation
output_activation	LinearActivation;ReluActivation
learning_rate	1.0e-5;0.0001;0.001;0.01
minibatch_size	20
max_epochs	500
l1_lambda	0.0

min_learning_rate	0.0
epoch_cycle_max	0
is_denoising	0
denoising_variance	0.0
ogd_learning_rate	0.0001;0.001
Total Samples	**1680**

11.5.5 Configuration 5

Parameter	Values
category	FFN
sae_config_id	8860;8872;8876;8888;8910;8800;8812;8816;8828;8850
seed_used	73;75;61;173;68
steps	5000
deltas	1,5,20
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	2
scaling_function	LimitedNormalizeData
layer_sizes	(15,40,6); (12,40,6); (9,40,6); (6,40,6); (3,40,6); (15,40,40,40,6); (12,40,40,40,6); (9,40,40,40,6); (6,40,40,40,6); (3,40,40,40,6); (15,80,6); (12,80,6); (9,80,6); (6,80,6); (3,80,6); (15,80,80,80,6); (12,80,80,80,6); (9,80,80,80,6); (6,80,80,80,6); (3,80,80,80,6)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotNormalInit
encoding_activation	LinearActivation
output_activation	LinearActivation
learning_rate	0.001;0.01
minibatch_size	20
max_epochs	500
l1_lambda	0.0
min_learning_rate	0.0
epoch_cycle_max	0

is_denoising	0
denoising_variance	0.0
ogd_learning_rate	0.001
Total Samples	**80**

11.5.6 Configuration 6

Parameter	Values
category	SAE
sae_config_id	0
seed_used	59
steps	5000
deltas	1,5,20
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	2
scaling_function	NormalizeData
layer_sizes	(6,15,5); (6,15,15,5)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotNormalInit
encoding_activation	LinearActivation;SigmoidActivation
output_activation	SigmoidActivation
learning_rate	1.0;2.0;4.0
minibatch_size	20
max_epochs	100
l1_lambda	0.0
min_learning_rate	1.0
epoch_cycle_max	1
ogd_learning_rate	0.0
Total Samples	**192**

11.5.7 Configuration 7

Parameter	Values
category	SAE
sae_config_id	0
seed_used	73
steps	5000
deltas	1,5,20;5,20,60;10,20,60
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	5
scaling_function	LimitedNormalizeData
layer_sizes	(30,120,60,25); (30,120,60,20); (30,120,60,15); (30,120,60,10); (30,120,60,5); (30,120,25); (30,120,20); (30,120,15); (30,120,10); (30,120,5); (30,120,120,25); (30,120,120,20); (30,120,120,15); (30,120,120,10); (30,120,120,5); (30,120,90,25); (30,120,90,20); (30,120,90,15); (30,120,90,10); (30,120,90,5); (30,90,60,25); (30,90,60,20); (30,90,60,15); (30,90,60,10); (30,90,60,5); (30,90,60,30,25); (30,90,60,30,20); (30,90,60,30,15); (30,90,60,30,10); (30,90,60,30,5); (30,120,60,30,25); (30,120,60,30,20); (30,120,60,30,15); (30,120,60,30,10); (30,120,60,30,5); (30,90,90,90,25); (30,90,90,90,20); (30,90,90,90,15); (30,90,90,90,10); (30,90,90,90,5); (30,90,90,25); (30,90,90,20); (30,90,90,15); (30,90,90,10); (30,90,90,5)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotUniformInit;HeUniformInit;DCUniformInit
encoding_activation	LinearActivation
output_activation	LinearActivation
learning_rate	0.005;0.01;0.05;0.1
minibatch_size	20

max_epochs	400
l1_lambda	0.0
min_learning_rate	0.0001
epoch_cycle_max	100;300
is_denoising	0
denoising_variance	0.0
ogd_learning_rate	0.0
Total Samples	**3266**

11.5.8 Configuration 8

Parameter	Values
category	SAE
sae_config_id	0
seed_used	73
steps	5000
deltas	1,5,20;5,20,60;10,20,60
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	5
scaling_function	LimitedNormalizeData
layer_sizes	(3,12,6,2); (3,12,2); (3,12,12,2); (3,12,9,2); (3,9,6,2); (3,9,6,3,2); (3,12,6,3,2); (3,9,9,9,2); (3,9,9,2); (3,12,6,1); (3,12,1); (3,12,12,1); (3,12,9,1); (3,9,6,1); (3,9,6,3,1); (3,12,6,3,1); (3,9,9,9,1); (3,9,9,1)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotUniformInit;HeUniformInit;DCUniformInit
encoding_activation	LinearActivation
output_activation	LinearActivation
learning_rate	0.005;0.01;0.05;0.1
minibatch_size	20
max_epochs	400
l1_lambda	0.0
min_learning_rate	0.0001
epoch_cycle_max	100;300
is_denoising	0
denoising_variance	0.0

ogd_learning_rate	0.0
Total Samples	**1296**

11.5.9 Configuration 9

Parameter	Values
category	FFN
sae_config_id	18140; 18914; 18259; 18311; 19481; 20662; 18917; 18260; 18314; 18766; 18119; 18191; 19343; 18344; 18751
seed_used	73
steps	5000
deltas	1,5,20;5,20,60;10,20,60
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	5
scaling_function	LimitedNormalizeData
layer_sizes	(25,120,120,10); (20,120,120,10); (15,120,120,10); (10,120,120,10); (5,120,120,10); (25,120,120,120,10); (20,120,120,120,10); (15,120,120,120,10); (10,120,120,120,10); (5,120,120,120,10); (25,120,90,90,60,10); (20,120,90,90,60,10); (15,120,90,90,60,10); (10,120,90,90,60,10); (5,120,90,90,60,10); (25,120,90,60,10); (20,120,90,60,10); (15,120,90,60,10); (10,120,90,60,10); (5,120,90,60,10); (25,120,10); (20,120,10); (15,120,10); (10,120,10); (5,120,10); (25,120,60,10); (20,120,60,10); (15,120,60,10); (10,120,60,10); (5,120,60,10)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotUniformInit;DCUniformInit;HeUniformInit
encoding_activation	None
output_activation	LinearActivation
learning_rate	0.01;0.05;0.1
minibatch_size	20
max_epochs	400

l1_lambda	0.0;0.1
min_learning_rate	0.0001
epoch_cycle_max	100
is_denoising	0
denoising_variance	0.0
ogd_learning_rate	0.01;0.05
Total Samples	**3313**

11.5.10 Configuration 10

Parameter	Values
category	FFN
sae_config_id	25339;25778;25684;25846;25640;25767
seed_used	73
steps	5000
deltas	1,5,20;5,20,60;10,20,60
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	5
scaling_function	LimitedNormalizeData
layer_sizes	(2,12,6,1); (1,12,6,1); (2,12,12,1); (1,12,12,1); (2,12,12,12,1); (1,12,12,12,1); (2,12,9,9,6,1); (1,12,9,9,6,1); (2,12,9,6,1); (1,12,9,6,1); (2,12,1); (1,12,1)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	HeUniformInit;XavierGlorotUniformInit;DCUniformInit
encoding_activation	None
output_activation	LinearActivation
learning_rate	0.01;0.05;0.1
minibatch_size	20
max_epochs	400
l1_lambda	0.0;0.1
min_learning_rate	0.0001
epoch_cycle_max	100
is_denoising	0
denoising_variance	0.0

ogd_learning_rate	0.01;0.05
Total Samples	**1296**

11.5.11 Configuration 11

Parameter	Values
category	FFN
sae_config_id	1533; 1497; 1554; 1639; 147; 1534; 1468; 1501; 318; 1284; 1535; 1553; 1059; 1508; 333
seed_used	73
steps	1
deltas	1,5,20;5,20,60;10,20,60
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	5
scaling_function	LimitedNormalizeData
layer_sizes	(25,60,10); (20,60,10); (15,60,10); (10,60,10); (5,60,10); (25,120,10); (20,120,10); (15,120,10); (10,120,10); (5,120,10); (25,240,10); (20,240,10); (15,240,10); (10,240,10); (5,240,10); (25,120,120,10); (20,120,120,10); (15,120,120,10); (10,120,120,10); (5,120,120,10); (25,240,240,10); (20,240,240,10); (15,240,240,10); (10,240,240,10); (5,240,240,10); (25,60,60,10); (20,60,60,10); (15,60,60,10); (10,60,60,10); (5,60,60,10); (30,60,10); (30,120,10); (30,240,10); (30,60,60,10); (30,120,120,10); (30,240,240,10)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	DCUniformInit
encoding_activation	None
output_activation	LinearActivation
learning_rate	0.01
minibatch_size	32
max_epochs	100;500;1000;10

l1_lambda	0.0;0.1;0.5
ifnull(tp.l2_lambda,0)	0
min_learning_rate	0.0001;1.0e-5
epoch_cycle_max	100;10
is_denoising	1
denoising_variance	0.0;0.1
ogd_learning_rate	0.005;0.01;0.05;0.1
Total Samples	**20737**

11.5.12 Configuration 12

Parameter	Values
category	SAE
sae_config_id	0
seed_used	73
steps	1
deltas	1,5,20;5,20,60;10,20,60
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	5
scaling_function	LimitedNormalizeData
layer_sizes	(30,120,25); (30,120,20); (30,120,15); (30,120,10); (30,120,5); (30,120,120,25); (30,120,120,20); (30,120,120,15); (30,120,120,10); (30,120,120,5); (30,120,120,120,25); (30,120,120,120,20); (30,120,120,120,15); (30,120,120,120,10); (30,120,120,120,5); (30,240,240,25); (30,240,240,20); (30,240,240,15); (30,240,240,10); (30,240,240,5); (30,240,240,240,25); (30,120,90,60,25); (30,120,90,60,20); (30,120,90,60,15); (30,120,90,60,10); (30,120,90,60,5); (30,240,25); (30,240,20); (30,240,15); (30,240,10); (30,240,5); (30,240,240,240,20); (30,240,240,240,15); (30,240,240,240,10); (30,240,240,240,5)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotUniformInit;HeUniformInit;DCUniformInit
encoding_activation	LinearActivation
output_activation	LinearActivation
learning_rate	0.01;0.1

minibatch_size	32
max_epochs	2000;10000;20000
l1_lambda	0.5;0.0;0.1;1.0
ifnull(tp.l2_lambda,0)	0
min_learning_rate	0.0001;1.0e-5
epoch_cycle_max	100;200
is_denoising	0
denoising_variance	0.0
ogd_learning_rate	0.0
Total Samples	**1665**

11.5.13 Configuration 13

Parameter	Values
category	FFN
sae_config_id	1533; 1497; 1554; 1639; 147; 1534; 1468; 1501; 318; 1284; 1535; 1553; 1059; 1508; 333
seed_used	73
steps	1
deltas	1,5,20;5,20,60;10,20,60
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	5
scaling_function	LimitedNormalizeData
layer_sizes	(25,60,10); (20,60,10); (15,60,10); (10,60,10); (5,60,10); (25,120,10); (20,120,10); (15,120,10); (10,120,10); (5,120,10); (25,240,10); (20,240,10); (15,240,10); (10,240,10); (5,240,10); (25,120,120,10); (20,120,120,10); (15,120,120,10); (10,120,120,10); (5,120,120,10); (25,240,240,10); (20,240,240,10); (15,240,240,10); (10,240,240,10); (5,240,240,10); (25,60,60,10); (20,60,60,10); (15,60,60,10); (10,60,60,10); (5,60,60,10); (30,60,10); (30,120,10); (30,240,10); (30,60,60,10); (30,120,120,10); (30,240,240,10); (10,60,60,60,10); (10,120,120,120,10); (10,240,240,240,10); (10,60,60,60,60,10); (10,120,120,120,120,10); (10,240,240,240,240,10)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	DCUniformInit
encoding_activation	None
output_activation	LinearActivation
learning_rate	0.01
minibatch_size	32

max_epochs	100;500;1000;10
l1_lambda	0.0;0.1;0.5
ifnull(tp.l2_lambda,0)	0
min_learning_rate	0.0001;1.0e-5
epoch_cycle_max	100;10;-1
is_denoising	1;0
denoising_variance	0.0;0.1
ogd_learning_rate	0.005;0.01;0.05;0.1
Total Samples	**22033**

11.5.14 Configuration 14

Parameter	Values
category	SAE
sae_config_id	0
seed_used	73
steps	5000
deltas	1,5,20
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	2
scaling_function	NormalizeData
layer_sizes	(6,30,5); (6,30,30,5); (6,30,3); (6,30,30,3); (30,60,5); (30,60,60,5); (30,60,60,60,5); (30,120,5); (30,120,120,5); (30,120,120,120,5); (30,60,15); (30,60,60,15); (30,60,60,60,15); (30,120,15); (30,120,120,15); (30,120,120,120,15); (30,60,25); (30,60,60,25); (30,60,60,60,25); (30,120,25); (30,120,120,25); (30,120,120,120,25)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotUniformInit
encoding_activation	LinearActivation
output_activation	LinearActivation
learning_rate	0.05;0.1
minibatch_size	20
max_epochs	1;1000
l1_lambda	0.0
min_learning_rate	0.0
epoch_cycle_max	0

is_denoising	1
denoising_variance	0.1;0.01;0.001;0.0001;1.0e-11
ogd_learning_rate	0.0
Total Samples	**243**

11.5.15 Configuration 15

Parameter	Values
category	SAE
sae_config_id	0
seed_used	73
steps	5000
deltas	1,5,20
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	2
scaling_function	LimitedNormalizeData
layer_sizes	(30,60,25); (30,60,60,25); (30,120,25); (30,120,120,25); (30,60,15); (30,60,60,15); (30,120,15); (30,120,120,15); (30,60,5); (30,60,60,5); (30,120,5);(30,120,120,5)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotUniformInit
encoding_activation	LinearActivation
output_activation	LinearActivation
learning_rate	0.001
minibatch_size	20
max_epochs	500
l1_lambda	0.0
min_learning_rate	0.0
epoch_cycle_max	0
is_denoising	1
denoising_variance	0.01;0.05;0.1;0.15;0.2;0.25

ogd_learning_rate	0.0
Total Samples	**73**

11.5.16 Configuration 16

Parameter	Values
category	FFN
sae_config_id	318;1508;1639
seed_used	73
steps	1
deltas	5,20,60;10,20,60;1,5,20
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	5
scaling_function	LimitedNormalizeData
layer_sizes	(10,60,60,60,10); (10,120,120,120,10); (10,240,240,240,10); (10,60,60,60,60,10); (10,120,120,120,120,10); (10,240,240,240,240,10)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	DCUniformInit
encoding_activation	None
output_activation	LinearActivation
learning_rate	0.01
minibatch_size	32
max_epochs	100;500;10
l1_lambda	0.0
ifnull(tp.l2_lambda,0)	0
min_learning_rate	0.0001
epoch_cycle_max	-1;100;10
is_denoising	0

denoising_variance	0.0
ogd_learning_rate	0.01;0.05;0.1
Total Samples	**1296**

11.5.17 Configuration 17

Parameter	Values
category	SAE
sae_config_id	0
seed_used	73
steps	5000
deltas	1,5,20
process_splits	0.6
training_splits	0.8,1.0
prediction_steps	2
scaling_function	LimitedNormalizeData
layer_sizes	(18,40,15); (18,40,40,15); (18,40,40,40,15); (18,80,15); (18,80,80,15); (18,80,80,80,15); (18,40,12); (18,40,40,12); (18,40,40,40,12); (18,80,12); (18,80,80,12); (18,80,80,80,12); (18,40,9); (18,40,40,9); (18,40,40,40,9); (18,80,9); (18,80,80,9); (18,80,80,80,9); (18,40,6); (18,40,40,6); (18,40,40,40,6); (18,80,6); (18,80,80,6); (18,80,80,80,6); (18,40,3); (18,40,40,3); (18,40,40,40,3); (18,80,3); (18,80,80,3); (18,80,80,80,3)
layer_activations	SigmoidActivation,ReluActivation,LinearActivation
initialization	XavierGlorotUniformInit
encoding_activation	LinearActivation
output_activation	LinearActivation
learning_rate	0.001;0.01
minibatch_size	20
max_epochs	500
l1_lambda	0.0
min_learning_rate	0.0

epoch_cycle_max	0
is_denoising	0
denoising_variance	0.0
ogd_learning_rate	0.0
Total Samples	**120**

11.6 Diagnostic Charts

The Tables below detail the available diagnostic charts made available through the `ExperimentGraphs` module [32]. Tables 24 and 25 provide SAE MSE and FFN P&L diagnostics along training and network dimensions. Table 26 provides charting functions for the price datasets used. Table 27 provides the performance for networks trained at the MMS level. Tables 28 and 29 provide the distribution graphs used in the CSCV and ONC/DSR diagnostics.

Function Name	Configuration Dimensions	Type
`PL_Scaling`	Scaling Methodology	Box Chart
`PL_ScalingOutputActivation`	Scaling Methodology, Network Output Activation Function	Box Chart
`PL_ScalingHiddenOutputActivation`	Scaling Methodology, Network Hidden Activation Function	Box Chart
`PL_LearningRates_MaxMin`	Min and Max SGD Learning Rates	Box Chart
`PL_L1Reg`	SGD L1 Regularization Lambda	Box Chart
`PL_L1Reg_Lines`	SGD L1 Regularization Lambda	Line Chart
`PL_ValidationSplit`	Percentage of SGD dataset used for training / testing	Box Chart
`PL_OGD_LearningRate`	OGD Learning Rate	Box Chart
`PL_Denoising`	SGD Denoising Variance	Box Chart
`PL_MaxEpochs`	SGD Training Epochs	Box Charts
`PL_EpochCycle`	SGD Learning Rate Epoch Cycle Length	Box Chart
`PL_LayerActivation_OutputActivation`	Network Hidden Activations and Output Activation	Box Chart
`PL_Activations`	Network Hidden Activations	Box Chart

PL_Activations_NetworkSize	Network Hidden Activations and Layer Sizes	Box Chart
PL_NetworkSize	Network Layers	Box Chart
PL_NetworkSizeLines	Network Layers	Line Graph
PL_SAE_Encoding_SizeLines	Feature Selection Size	Line Graph
PL_SAE_Encoding_Size	Feature Selection Size	Box Chart
PL_DataDeltas	Price Fluctuation Aggregation Horizons	Box Chart
PL_Init	Network Initialization Method	Box Chart
PL_OOS_OGDLR_Deltas	OGD Learning Rate, Data Horizons	Line Chart
PL_OOS_OGDLR_Delta_Encoding	OGD Learning Rate, Data Horizons, Encoding Size	Line Chart
PL_IS_Encoding	Encoding Size	Line Chart
PL_SAE_Encoding_SizeLines	Encoding Size, OGD Learning Rate	Line Chart
PL_Heatmap_LearningRate_MaxEpochs	OGD Learning Rate and SGD Training Epochs	Heatmap
PL_Heatmap_NetworkSize_DataAggregation	Network Number of Layers and Price Fluctuation Aggregation Horizons	Heatmap
PL_Heatmap_LayerSize_DataAggregation	Network Layer Sizes and Price Fluctuation Aggregation Horizons	Heatmap

Table 24: P&L Diagnostic EDA Functions.

Function Name	Configuration Dimensions	Type
MSE_SAE_Encoding_Size_Deltas_Lines	Data Horizons, Layer Sizes	Line Chart
MSE_LearningRateInit_Lines	Learning Rate, Initialization	Line Chart
MSE_ActivationsEncodingSizes	Network Activation Functions and Encoding Layer Size	Box Chart
MSE_LayerSizesLines	Network Layer Sizes	Line Graph
MSE_LayerSizes	Network Layer Sizes	Box Chart
MSE_LearningRate_MaxMin	SGD Min and Max Learning Rates	Box Chart
MSE_LearningRate_MaxMin_Lines	Learning Rate	Line Chart
MSE_EpochCycle_LayersLines	Epoch Cycle, Layer Sizes	Line Chart
MSE_Lambda1	SGD L1 Regularization Lambda	Box Chart
MSE_Reg_EncodingLines	SGD L1 Regularization Lambda, Encoding Size	Line Chart
MSE_EpochCycle	SGD Learning Rate Epoch Cycle Length	Box Chart
MSE_Init	Network Initialization Method	Box Chart
MSE_Deltas	Price Fluctuation Aggregation Horizons	Box Chart
MSE_Min_EncodingSize_Activation	Encoding Layer Activation Function and Size	Box Chart
MSE_Activation_Scaling_Filters	All Activation Functions and Data Scaling Methodology	Box Chart

MSE_OutputActivation_Scaling_Filters	Output Activation Function and Data Scaling Methodology	Box Chart
MSE_Output_Activation	Output Activation Function	Box Chart
MSE_Encoding_Activation	Encoding Layer Activation	Box Chart
MSE_Hidden_Activation	Hidden Layer Activation Functions	Box Chart
MSE_Scaling_Filters	Data Scaling Methodology	Box Chart
MSE_Pretraining	Pretraining Epochs	Box Chart
MSE_Denoising	SGD Denoising Variance	Box Chart

Table 25: MSE Diagnostic EDA Functions.

Function Name	Price Dataset	Type
PlotSynthetic6	Synthetic6 (7.2.1)	Line Chart
PlotSynthetic10	Synthetic10 (7.2.2)	Line Chart
PlotJSEPlots	All JSE Based Datasets (7.1.1, 7.1.3 , 7.1.3, 7.1.2)	Line Chart

Table 26: Price Plot Functions.

Function Name	MMS Variable	Type
SharpeRatiosPDF	Sharpe Ratio	PDF
PlotConfusion	Trade Confusion Percentages	PDF
AllProfitsPDF	P&L	PDF

`OGD_MSE_vs_PL`	P&L, OGD MSE	Scatter Chart

Table 27: MMS Diagnostic Functions.

Function Name	**CSCV Variable**	**Type**
`PlotCombinationSizes`	PBO	Line Chart
`PlotPBOBySplits`	PBO	Line Chart
`PlotLogitDistribution`	Logit Values	PDF

Table 28: CSCV Diagnostic Functions.

Function Name	**DSR Variable**	**Type**
`ClusterDistributionPercAll`	ONC cluster, Sharpe ratio	PDF
`ClusterDistributionPercCluster`	ONC cluster, Sharpe ratio	PDF
`ClusterOGDMSEPlot`	ONC cluster, OGD MSE	PDF

Table 29: ONC Diagnostic Functions.

12 References

[1] Susanne Albers. Online algorithms: a survey. *Mathematical Programming*, 97:3–26, 05 2003. doi: 10.1007/s10107-003-0436-0. URL https://link.springer.com/content/pdf/10.1007%2Fs10107-003-0436-0.pdf.

[2] Jasmina Hasanhodzic Andrew W. Lo. *Heretics of Finance*. Bloomberg Press, 2009. ISBN 978-1-57660-316-1.

[3] Diego Aparicio and Marcos Lopez de Prado. How hard is it to pick the right model? mcs and backtest overfitting. *Algorithmic Finance*, 7:53–61, 01 2018. doi: 10.3233/AF-180231. URL https://ssrn.com/abstract=3044740.

[4] B. Arthur. Complexity in economics and financial markets. *Complexity*, 1:20–25, 10 1995. doi: 10.1002/cplx.6130010106. URL https://onlinelibrary.wiley.com/doi/epdf/10.1002/cplx.6130010106.

[5] David Bailey and Marcos Lopez de Prado. The sharpe ratio efficient frontier. *The Journal of Risk*, 15:3–44, 12 2012. doi: 10.21314/JOR.2012.255.

[6] David Bailey and Marcos Lopez de Prado. The deflated sharpe ratio: Correcting for selection bias, backtest overfitting, and non-normality. *The Journal of Portfolio Management*, 40:94–107, 09 2014. doi: 10.3905/jpm.2014.40.5.094.

[7] David H. Bailey, Jonathan Borwein, Marcos Lopez de Prado, and Qiji Jim Zhu. Pseudo-mathematics and financial charlatanism: The effects of backtest overfitting on out-of-sample performance. *Notices of the American Mathematical Society*, 61:458–471, 4 2014. doi: 10.1090/noti1105. URL https://ssrn.com/abstract=2308659.

[8] David H. Bailey, Jonathan Borwein, Marcos Lpez de Prado, and Qiji Jim Zhu. The probability of backtest overfitting. *Journal of Computational Finance*, 20:39–69, 4 2017. doi: 10.21314/JCF.2016.322. URL https://www.davidhbailey.com/dhbpapers/backtest-prob.pdf.

[9] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLoS ONE*, 12, 07 2017. doi: 10.1371/journal.pone.0180944.

[10] Peter Bartlett, Elad Hazan, and Alexander Rakhlin. Adaptive online gradient descent. *Advances in Neural Information Processing Systems 20 - Proceedings of the 2007 Conference*, 01 2007.

[11] Y. Bengio and Samy Bengio. Modeling high-dimensional discrete data with multi-layer neural networks. 02 2001. URL http://papers.nips.cc/paper/

1679-modeling-high-dimensional-discrete-data-with-multi-layer-neural-networks.pdf.

[12] Y. Bengio, Pascal Lamblin, D. Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Adv. Neural Inf. Process. Syst.*, 19: 153–160, 01 2007.

[13] Y. Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35:1798–1828, 08 2013. doi: 10.1109/TPAMI.2013.50.

[14] L Bottou and Yann Lecun. Large scale online learning. *Advances in Neural Information Processing Systems 16*, 03 2004.

[15] L. Bottou and N. Murata. *Stochastic Approximations and Efficient Learning, The Handbook of Brain Theory and Neural Networks*. The MIT Press, 2 edition, 2019.

[16] Chris Brooks and Harry Kat. The statistical properties of hedge fund index returns. *Henley Business School, Reading University, ICMA Centre Discussion Papers in Finance*, 01 2001.

[17] Steve Christie. Is the sharpe ratio useful in asset allocation? *SSRN Electronic Journal*, 05 2005. doi: 10.2139/ssrn.720801.

[18] Dan Cirean, Ueli Meier, Luca Gambardella, and Jrgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22:3207–20, 12 2010. doi: 10.1162/NECO_a_00052.

[19] JuliaLang.org contributors. The julia programming language, 2019. URL https://julialang.org/.

[20] James Crutchfield. Between order and chaos. *Nature Physics*, 9:382–382, 06 2013. doi: 10.1038/nphys2639.

[21] Joel da Costa. Dissertation github - activationfunctions, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/ActivationFunctions.jl.

[22] Joel da Costa. Dissertation github - aggregated database, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/database_actual_agg.db.

[23] Joel da Costa. Dissertation github - cscv, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/CSCV.jl.

[24] Joel da Costa. Dissertation github - costfunctions, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/CostFunctions.jl.

[25] Joel da Costa. Dissertation github - dsrdataproc, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/DSR/DSR_dataproc.py.

[26] Joel da Costa. Dissertation github - dsrproc, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/DSR/DSR_proc.py.

[27] Joel da Costa. Dissertation github - datagenerator, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/DataGenerator.jl.

[28] Joel da Costa. Dissertation github - dataprocessor, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/DataProcessor.jl.

[29] Joel da Costa. Dissertation github - databasebatchprocesses, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/DatabaseBatchProcesses.jl.

[30] Joel da Costa. Dissertation github - databasecreator, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/DatabaseCreator.jl.

[31] Joel da Costa. Dissertation github - databaseops, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/DatabaseOps.jl.

[32] Joel da Costa. Dissertation github - experimentgraphs, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/ExperimentGraphs.jl.

[33] Joel da Costa. Dissertation github - experimentprocess, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/ExperimentProcess.jl.

[34] Joel da Costa. Dissertation github - experimentprocesstrainffn, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/ExperimentProcessTrainFFN.jl.

[35] Joel da Costa. Dissertation github - experimentprocesstrainsae, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/ExperimentProcessTrainSAE.jl.

[36] Joel da Costa. Dissertation github - ffn, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/FFN.jl.

[37] Joel da Costa. Dissertation github - config generator, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/ConfigGenerator.jl.

[38] Joel da Costa. Dissertation github - gradientfunctions, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/GradientFunctions.jl.

[39] Joel da Costa. Dissertation github - initializationfunctions, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/InitializationFunctions.jl.

[40] Joel da Costa. Dissertation github - networktrainer, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/NetworkTrainer.jl.

[41] Joel da Costa. Dissertation github - ogd, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/OGD.jl.

[42] Joel da Costa. Dissertation github - rbm, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/RBM.jl.

[43] Joel da Costa. Dissertation github - reportgraphs, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/ReportGraphs.jl.

[44] Joel da Costa. Dissertation github - sgd, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/SGD.jl.

[45] Joel da Costa. Dissertation github - trainingstructures, 2019. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries/TrainingStructures.jl.

[46] Joel da Costa. Dissertation github. 2020. doi: 10.25375/uct.11897358. URL https://github.com/joel11/Masters/blob/master/Code%20Libraries.

[47] JSE Top40 Data. Price relative from 2003-2018. doi: 10.25375/uct.11897628.

[48] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *NIPS*, 27, 06 2014.

[49] David Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, pages 1–32, 01 2000.

[50] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

[51] Dumitru Erhan, Y. Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 02 2010. doi: 10.1145/1756006.1756025.

[52] Eugene Fama. The behavior of stock market price. *Journal of Business - J BUS*, 38, 01 1965. doi: 10.1086/294743.

[53] Jianqing Fan and Yingying Fan. High dimensional classification using features annealed independence rules. *Annals of statistics*, 36:2605–2637, 02 2008. doi: 10.1214/07-AOS504.

[54] Jianqing Fan and Runze Li. Statistical challenges with high dimensionality: Feature selection in knowledge discovery. *Proc. Madrid Int. Congress of Mathematicians*, 3, 03 2006.

[55] Python Software Foundation. Python, 2019. URL `https://www.python.org/`.

[56] Michale Gant. Jse top 40 constituents daily price relatives 2003-2018 (bloomberg). 2019. doi: 10.17632/3nbgc4cygk.1. URL `http://dx.doi.org/10.17632/3nbgc4cygk.1`.

[57] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points — online stochastic gradient for tensor decomposition. 03 2015. URL `http://proceedings.mlr.press/v40/Ge15.pdf`.

[58] Tim Gebbie and Fayyaaz Loonat. Learning zero-cost portfolio selection with pattern matching. *PLOS ONE*, 13, 05 2016. doi: 10.1371/journal.pone.0202788.

[59] Xavier Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256, 01 2010.

[60] Xavier Glorot, Antoine Bordes, and Y. Bengio. Deep sparse rectifier neural networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statisitics (AISTATS) 2011*, 15:315–323, 01 2011.

[61] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Y. Bengio. Maxout networks. *30th International Conference on Machine Learning, ICML 2013*, 1302, 02 2013.

[62] Gerwin Griffioen. Technical analysis in financial markets. *SSRN Electronic Journal*, 03 2003. doi: 10.2139/ssrn.566882.

[63] Peter Hansen, James Nason, and Asger Lunde. The model confidence set. *Econometrica*, 79:453–497, 03 2010. doi: 10.2139/ssrn.522382.

[64] Campbell R. Harvey and Yan Liu. Backtesting. *The Journal of Portfolio Management*, 42:13–28, 07 2015. doi: https://doi.org/10.3905/jpm.2015.42.1.013.

[65] Douglas Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44:1–12, 05 2004. doi: 10.1021/ci0342472.

[66] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision (ICCV 2015)*, 1502, 02 2015. doi: 10.1109/ICCV.2015.123.

[67] G. Hinton. A practical guide to training restricted boltzmann machines[j]. *Momentum*, 9:926–947, 01 2010.

[68] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science (New York, N.Y.)*, 313:504–7, 08 2006. doi: 10.1126/science.1127647.

[69] Geoffrey Hinton. Article training products of experts by minimizing contrastive divergence. *Neural computation*, 14:1771–800, 09 2002. doi: 10.1162/089976602760128018.

[70] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18:1527–54, 08 2006. doi: 10.1162/neco.2006.18.7.1527.

[71] Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint*, arXiv, 07 2012.

[72] Sepp Hochreiter and Jrgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[73] K. Hornik. Multilayer feed-forward networks are universal approximators. *Neural Networks*, 2:359–366, 1989. doi: https://doi.org/10.1016/0893-6080(89)90020-8.

[74] Daniel Hsu. Time series compression based on adaptive piecewise recurrent autoencoder. 07 2017.

[75] John Ioannidis. Why most published research findings are false. *CHANCE*, 32:4–13, 01 2019. doi: 10.1080/09332480.2019.1579573.

[76] A. Ivakhnenko. Polynomial theory of complex systems. ieee trans syst man cybern. *Systems, Man and Cybernetics, IEEE Transactions on*, 1: 364 – 378, 11 1971. doi: 10.1109/TSMC.1971.4308320.

[77] Neil Johnson, Guannan Zhao, Eric Hunsader, Hong Qi, Nicholas Johnson, Jing Meng, and Brian Tivnan. Abrupt rise of new machine ecology beyond human response time. *Scientific reports*, 3:2627, 09 2013. doi: 10.1038/srep02627.

[78] Michael N. Kahn. *Technical Analysis Plain and Simple: Charting the Markets in Your Language.* Financial Times Press, 2006.

[79] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. doi: 10.1145/3065386.

[80] John Langford, Lihong Li, and Tong Zhang. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10, 07 2008. doi: 10.1145/1577069.1577097.

[81] Martin Langkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42, 06 2014. doi: 10.1016/j.patrec.2014.01.008.

[82] Yann Lecun. A theoretical framework for back-propagation. 01 1992.

[83] Yann LeCun, Y Boser, B Denker, Don Henderson, R. Howard, W. Hubbard, and Larry Jackel. Back-propagation applied to handwritten zipcode recognition. *Neural Computation - NECO*, 01 1992.

[84] Yann Lecun, Leon Bottou, Genevieve Orr, and Klaus-Robert Mller. *Efficient BackProp*, volume 1524, pages 546–546. 01 1998. doi: 10.1007/3-540-49430-8_2.

[85] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015. doi: 10.1038/nature14539.

[86] Andrew Lo. The statistics of sharpe ratios. *Financial Analysts Journal*, 58, 02 2003. doi: 10.2469/faj.v58.n4.2453.

[87] Marcos Lopez de Prado. The future of empirical finance. *The Journal of Portfolio Management*, 41:140–144, 07 2015. doi: 10.3905/jpm.2015.41.4.140.

[88] Marcos Lopez de Prado. Building diversified portfolios that outperform out of sample:. *The Journal of Portfolio Management*, 42:59–69, 07 2016. doi: 10.3905/jpm.2016.42.4.059.

[89] Marcos Lopez de Prado and Michael Lewis. Dissertation github - dsr-pradolewis, 2019. URL `https://github.com/joel11/Masters/blob/master/Code%20Libraries/DSR/DSR_PradoLewis.py`.

[90] Marcos Lopez de Prado and Michael Lewis. Detection of false investment strategies using unsupervised learning methods. *Quantitative Finance*, 19:1–11, 07 2019. doi: 10.1080/14697688.2019.1622311.

[91] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. 08 2016.

[92] Spencer Lyon. Plotlyjs, 2019. URL `http://spencerlyon.com/PlotlyJS.jl/`.

[93] A. Maas, A. Hannun, and A. Ng. Rectifier nonlinearities improve neural network acoustic models. *Proceedings of the 30 th International Conference on Machine Learning,*, 28, 2013. URL `https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf`.

[94] R. Mclean and Jeffrey Pontiff. Does academic research destroy stock return predictability? *The Journal of Finance*, 71, 05 2013. doi: 10.2139/ssrn.2156623.

[95] Elmar Mertens. Comments on variance of the iid estimator in lo (2002). *Technical report, Working Paper University of Basel*, 11 2002.

[96] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. 01 2017. ISBN 9780262343930. doi: 10.7551/mitpress/11301.001.0001.

[97] John Murphy. Technical analysis of financial markets. 01 1999.

[98] Nicholas Murphy and Tim Gebbie. Learning the population dynamics of technical trading strategies, 03 2019.

[99] Alexander Adamou Ole Peters. *Ergodicity Economics*. London Mathematical Laboratory, 2018.

[100] Aaron oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. 09 2016.

[101] J.D. Opdyke. Comparing sharpe ratios: So where are the p-values? *Journal of Asset Management*, 8, 12 2007. doi: 10.1057/palgrave.jam.2250084.

[102] Norman Packard, J.P. Crutchfield, and R. Shaw. Geometry from a time series. *Phys. Rev. Lett.*, 45:712, 09 1980. doi: 10.1103/PhysRevLett.45.712.

[103] Razvan Pascanu, Tomas Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *30th International Conference on Machine Learning, ICML 2013*, 11 2012.

[104] Jeffrey M. Perkel. Julia: come for the syntax, stay for the speed. *Nature*, 572:141–142, 2019. doi: doi:10.1038/d41586-019-02310-3.

[105] R.J. Peter. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, 20, 01 1987.

[106] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann Lecun. Efficient learning of sparse representations with an energy-based model. 01 2006.

[107] Nicolas Roux and Y. Bengio. 1 representational power of restricted boltzmann machines and deep belief networks. *Neural computation*, 20:1631–49, 07 2008. doi: 10.1162/neco.2008.04-07-510.

[108] D. Rumelhart, G. Hinton, and Williams RJ. Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1, 07 1986. doi: 10.1016/B978-1-4832-1446-7.50035-2.

[109] Robert Schaefer. Subset selection in regression. *Technometrics*, 34, 03 2012. doi: 10.1080/00401706.1992.10484917.

[110] J Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 01 2015. doi: https://doi.org/10.1016/j.neunet.2014.09.003.

[111] Frank Schorfheide and Kenneth Wolpin. On the use of holdout samples for model selection. *American Economic Review*, 102, 05 2012. doi: 10.1257/aer.102.3.477.

[112] Jack D. Schwager. *Getting Started in Technical Analysis*. Wiley, 1999. ISBN 0471295426.

[113] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann Lecun. Pedestrian detection with unsupervised multi-stage feature learning. *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 12 2012. doi: 10.1109/CVPR.2013.465.

[114] William Sharpe. Mutual fund performance. *Journal of Buisiness*, 39: 119–138, 01 1966. doi: 10.1086/294846.

[115] Hava Siegelmann. Theoretical foundations of recurrent neural networks. 11 2019.

[116] Andrew Skabar and Ian Cloete. Neural networks, financial trading and the efficient markets hypothesis. *ACSC '02: Proceedings of the Twenty-fifth Australasian Conference on Computer Science*, 4, 02 2002. doi: 10.1145/563857.563829.

[117] Leslie Smith. Cyclical learning rates for training neural networks. pages 464–472, 03 2017. doi: 10.1109/WACV.2017.58.

[118] F. Takens. Detecting strange attractors in turbulence. *Dynamical Systems and Turbulance serial Lecture notes in Mathematics*, 898, 01 1981.

[119] Lawrence Takeuchi. Applying deep learning to enhance momentum trading strategies in stocks. 2013.

[120] Luigi Troiano, Elena Villa, and Pravesh Kriplani. On feature reduction using deep learning for trend prediction in finance. 04 2017.

[121] Paul Tseng. An incremental gradient(-projection) method with momentum term and adaptive stepsize rule. *SIAM Journal on Optimization*, 8, 04 1999. doi: 10.1137/S1052623495294797.

[122] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Y. Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 12 2010.

[123] Sida Wang and Christopher Manning. Fast dropout training. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 118–126, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `http://proceedings.mlr.press/v28/wang13a.html`.

[124] Sholom M. Weiss and Casimir A. Kulikowski. *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991. ISBN 1-55860-065-5.

[125] Paul Werbos. *Applications of advances in nonlinear sensitivity analysis*, volume 38, pages 762–770. 01 1970. doi: 10.1007/BFb0006203.

[126] Paul Werbos and Paul John. Beyond regression : new tools for prediction and analysis in the behavioral sciences /. 01 1974.

[127] Diane Wilcox and Tim Gebbie. Hierarchical causality in financial economics. *SSRN Electronic Journal*, 08 2014. doi: 10.2139/ssrn.2544327.

[128] Huaiqin Wu. Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions. *Inf. Sci.*, 179: 3432–3441, 09 2009. doi: 10.1016/j.ins.2009.06.006.

[129] Lionel Yelibi and Tim Gebbie. Fast super-paramagnetic clustering. *Physica A: Statistical Mechanics and its Applications*, 2020. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2019.124049. URL `http://www.sciencedirect.com/science/article/pii/S0378437119322393`.

[130] Matthew Zeiler. Adadelta: An adaptive learning rate method. 1212, 12 2012.

[131] Yang Zhao, Jianping Li, and Lean Yu. A deep learning ensemble approach for crude oil price forecasting. *Energy Economics*, 66, 06 2017. doi: 10.1016/j.eneco.2017.05.023.