Technical document

1. Current Implementation Status

The library implementations are at a point such that an end-to-end process can be run: data with training parameters are specified, and a CSCV based PBO is output.

A full list of implementations is as follows:

- Network Types
  - Feedforward Network
  - Restricted Boltzmann Machine
  - Stacked Auto Encoder
- Learning Algorithms
  - Contrastive Divergence-1 for RBM pre-training
  - Online Gradient Descent
  - Stochastic Gradient Descent
- Validation Algorithms
  - CSCV
  - PBO
- Data Processes
  - Log fluctuation implementation to format dataset with rolling windows
  - Library to generate synthetic data
- Optimizations
  - SGD: L1 & L2 Regularizations
- Neural Network Configurations
  - Activations: Sigmoid, ReLU, Softmax
  - Initialisations: Normal, Hinton, Xavier, He (Normal & Uniform variations)
  - Cost Functions: MSE, CE, Loglikehood
- Output Library
  - Effective graphing of training logs (epoch based performance, weight update rates, output reconstructions etc.)
- Technical Testing
  - Basic hyperparameter search method & outputs
  - Automated Unit Tests for CD1, OGD and
- Database Implementation
  - Recording results and configurations in SQL Lite database
  - Recording networks (SAE) in bson files

2. Current Experiment Testing Status

A full list of expected experiment is as follows:

- Synthetic SAE Tests
  - ReLU + Initialization (done in this version)
  - Sigmoid + Initialization
  - Sigmoid + Pre-training
- Synthetic FFN Tests
  - ReLU + Initialization (done in this version)
  - Sigmoid + Initialization
  - Sigmoid + Pre-training
- Real JSE Daily Close Tests: SAE
- Real JSE Daily Close Tests: FFN

3. Future Status and Next Iteration

The results of the current set highlight that overfitting is the biggest problem at the moment. A set of features, as noted below, will be implemented prior to proceeding with tests.

Next Iteration

- Change SAE testing to use cross validation. The current (constant) split on training and testing data is resulting in significant overfitting issues.
- Training: Regularization methods & Learning rate annealing schedule are implemented, but need to be tested and incorporated info configuration testing.
- Synthetic Tests: Run tests on synthetic data using the above features, and including Sigmoid activations with Weight Initialization & Pre-Training techniques.
- Extended Training: The predictive models in this iteration were not trained for very long - they could probably be easily improved in this way.

Critical Decisions and Points of Concern

- Data Scaling - How should this work for different activations and online learning?
- Finalise Dataset: Decide which data to use - JSE Close / intra day?
- Trading Strategy: Decide on the basic strategy used to derive returns
- SAE updates: Decide whether the SAE should be updated periodically or not
- Decide on data splits and what should be used for CSCV - MinBTL method?
- Decide on data windows - aribtrary choices or.. ?

Non-Critical Features

- Better hyper-parameter search technique, e.g. Random Search
- Denoising SAEs
- Learning Optimization: Momentum

Dissertation to-do list

- Literature Review: There are various inclusions and ammendments to be made here (particularly around data windows & varance initializations)
- Reformatting of Algorithms through document
- Papers and references to add: Wilcox & Debbie Hierarchical Paper; Crutchfield & Farmer: Geometry of Time Series; F Takens: Detecting strange attractors in turbulence; Reference Riaz
- General formatting changes necessary
- Various to-do notes throughout paper

4. Milestones & Deliverables

Finish Date	Item
Requires Corrections	1 Literature Review
In Progress	2 Data Collection
Done	3 Data Log Fluctuation Processing
Done	4 SAE Implementation
Done	5 Online FFN/LSTM Implementation
Done	6 CSCV Implementation
Done	7 Synthetic Data Generation
Done	8 Finalise Implementation Process
Done	9 Database Implementation
28/02	10 Learning Optimizations
28/02	11 Test full implementation on synthetic data
31/03	12 Test full implementation on actual dataset
31/03	13 Statistical Analysis (incl. profitability of trading strategy)
30/04	14 Write up & Revisions

5. Process Implementation

The steps below indicate how everything is expected to fit together at a high level perspective:

1 Data preparation

- The data is processed into day to day log fluctuations
- At each time point, rolling historic summations are calculated (e.g. the past 1, 7 and 30 days)
- At each time point, rolling future summations are calculated (e.g. the next 2 days)

2 Data Segregation

- The processed dataset is split into 2 parts, to be used for the following processes: SAE/SGD training and OGD training (e.g. 60%, 40%)
- Bailey's MinBTL method should be considered to determine this split length

3 SAE Training

- The dataset defined for SAE/SGD is used to train the auto-encoder networks (using either RBM pretraining/weight initialization algorithms + SGD)
- Notably, this SAE/SGD dataset will itself be split into training & testing portions in order to train and validate the SAE network
- The SAE training does not suffer from the same backtesting concerns as for the predictive network. A full set of models for difference encoding layers can be trained to sufficient performance, and then used as a configuration aspect for the predictive training.

4 FFN/SGD Training

- The same SAE/SGD portion of the data will then be used to train the FFN network to predict future prices, using the implemented SGD algorithm & optimizations.

5 OGD Training

- The second portion of data will be used to train and optimize the FNN from 4 for OGD. The purpose of this will be to optimize to train the network and produce returns.

6 Returns generations

- A basic trading strategy will be used to generate returns from the predicted and actual prices in the holdout dataset. These returns for each time period and each configuration of the model defined by steps 3-5 will form the M matrix for the CSCV process.

7 CSCV & PBO

- Using the M matrix from 6, the CSCV process will be run which will then allow a calculation of PBO.

6. Meeting Minutes

16th June 2018

- Literature Review Include: Wilcox & Debbie Hierarchical Paper; Crutchfield & Farmer: Geometry of Time Series; F Takens: Detecting strange attractors in turbulence; Reference Riaz
- Implementation
  - Key implementation point was the data segmentation choices
  - LSTM network would resolve this point for us (CNN convolution could also potentially work)
  - Non LSTM will run risk of correlating times with stocks
  - Representation needs a mixture of time scales as well as the features
  - Will be considering data for the entire index at any time point t
  - Will need to consider how to address the changing constituents
  - Aspect of how often the feature selection is updated (e.g. SDAE in online model)
  - Does it get updated in the online model ?
  - SDAE result should be similar to edge detection which will give set of predictions essentially
  - Use known test cases for CSCV method development
- Data
  - Use simple noise with Gaussian distribution & long trends to develop network models (6 variations of increasing, decreasing and flat with low or high variability)
  - Build backdated tests module with surrogate data afterwards
  - Not going to try intraday data due to different nature of problem
  - Implementation of either a flat file or MongoDB
  - Important to include Volume datav
  - Volume time -¿ might be far more predictable; volume and variance linked
- Other
  - Admin:Single document - table of contents (nice to have) with authors; sections etc. ; full doc with versioning; PDF + latex (always send both)
  - Further Correspondence: * Iterate over FFN/LSTM implementations rather than starting with LSTM * For the meantime, assume the models do not differ fundamentally enough to be considered as separate for the PBO calculations

Online non-linear prediction of financial time-series patterns

Joel da Costa

University of Cape Town

Abstract

We consider pattern prediction of financial time-series data. The algorithm framework and workflow is developed and proved on daily sampled OHLCV (open-high-low-close-volume) time-series data for JSE equity markets. The input patterns are based on input data vectors are equal size data windows pre-processed into a sequence of daily, weekly and monthly sampled feature measurement changes (here log feature fluctuations). The data processing is split into at online batch processed step where data is compressed using a stacked autoencoder (SAE) via unsupervised learning, and then batch supervised learning is carried out using the data-compression algorithm with the output being a pattern sequence of measured time-series feature fluctuations (log differenced data) in the future (ex-post) from the training and validation data. Weight initializations for these networks are implemented with Restricted Boltzmann Machine (RBM) pre-training, and variance based initializations. The historical simulation is then run using an online feedforward neutral network (FNN) initialised with the weights from the online training and validation step. The validity of results is considered under a rigorous assessment of backtest overfitting (Combinatorially Symmetrical Cross Validation), and the results are then considered in terms of test for statistical arbitrage with simple correction for transaction costs.

Keywords: online learning, feedforward neural network, restricted boltzmann machine, stacked autoencoder, pattern prediction, JSE, non-linear, financial time series, combinatorially symmetrical cross validation, backtest overfitting

Contents

1 Introduction

This dissertation explores several ideas for the purposes of effective and valid stock price prediction, and suggests a novel approach to the combination of techniques. The first part of the implementation focuses on the training of deep neural networks, and combines usage of several well known designs for data reduction, deep learning and online learning. The aim here is to produce a deep learning model that is able to effectively produce stock price predictions in an online process, using data reduction of high dimensional finance data in order to achieve better results.

The second part of the paper focuses on whether the model can be trained such that it does not succumb to backtest overfitting, namely by using methods suggested by Bailey et al. [4] in order to calculate the likelihood of overfitting having occurred. As they've noted, and discussed more fully in 2.6, backtesting overfitting for trading strategies have become problematically widespread in financial literature. Here we investigate 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 chapter 2 has a fuller discussion of work that has been done to precede the various techniques which have been implemented. A brief introduction to technical analysis in the financial sector, with it's perspective and history, is discussed and forms the basis for proposing and using the technical analysis methods throughout this paper 2.1. The literature review then moves onto covering the usage and history of Neural Networks, which has progressed enourmously in recent times. The section covers both the basic foundations, and also discusses the recent work which has resulted in the widespread use of deep learning models 2.2. This is extended to the coverage of Stacked Autoencoders, and their efficacy in data reduction for complex systems, which has lead them to be pivotal tools in deep learning models 2.3. Online learning methods are discussed in 2.5 with a coverage of both the historical basis as well as the more recent devopments which have allowed for further improvment in the algorithms. The chapter finishes off with discussing the impact of backtest overfitting and how some notable works developed by Bailey et al. have provided tools with which this may be avoided 2.6.

Chapter 3 provides the details around the data which has been used in the paper. The chapter starts with describing how the data is processed using log feature fluctuations, and then is expanded to include the fluctuations over rolling window periods. The synthetic data used is generated using Geometric Brownian motion, and is implemented such that each randomly seeded dataset consists of the following stocks:

- 3 stocks with an upward trend, and high, medium and low variance
- 3 stocks with a downward trend, and high medium and low variance
- 3 stocks with a sideways trend, and high medium andlow variance

There is a further summary of the real data used: OHLCV data for the JSE ALSI over a period of 20 years.

Chapter 4 provides more indepth details on the algorithms and structures used to implement the dissertation process. The structure of feedforward neural networks is discussed, as well as how they are trained using the backpropagation algorithm, and how that can be applied in a stochastic descent framework 4.2. The chapter also provides details for how these network weight initializations can impact performance, and how this can be improved with either RBM pre-training (as per [36]) or through newer variance based techniques - this includes the structures and training techniques used for the Stacked Autoencoders 4.4. How the CSCV techniques suggested by [4] are implemented and used to derive a Probability of Backtest Overfitting figure for the full training and testing processes is detailed in 4.7. The chapter ends with a summary of the full end to end process, from data preparation to the output of a PBO figure in 4.9.

Chapter 5 is the final chapter, and discusses the results found from the process.

2 LiteratureReview

2.1 Technical Analysis

Technical analysis is a financial analysis 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 [62].

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 [62]. 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 [31]. 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 [47]. 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 [71].

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 [46]. 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 [3]. Complex and dynamic systems such as these may often exist at the 'order-disorder border' - they will generate certain non-random patterned and internal organisation, which can be assessed and identified, however they will also exhibit a certain amount of randomness in their behaviours, or 'chaos' [18]. 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 nonlinearity in the system can make prediction challenging. Fittingly then, neural networks have become a popular choice for modelling within the 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 [75].

2.2 Neural Networks

A Neural Network (NN) 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 in context. The neurons are based on the same idea as synapses as seen in the brain - where a buildup of input results in a firing of output. The input here is determined by the models input (real numbers typically), and processed through the weights and activation functions of the neuron, which then results in an output value either at an intermediate level, or as the models final output. The system learns by considering input samples sequentially, and adjusting the weights between edges to result in more accurate outputs, which may either be classification or regression values.

Structured neural networks that learn to some extent have been around since the second half of the 21st century [69], though have been through several cycles of popularity. The first versions tended to be very simple with one one layer of hidden neurons [43]. It was only later, through the application of the backpropagation algorithm, that they started to become more practical and popular [85].

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 [74]. 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 [40]. 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, as elaborated on later.

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 [69]. While all of these configurations and paradigms have their benefits and uses, this paper will largely focus on FNN and RNNs, trained through SL and UL.

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. In line with this, the Backpropagation (BP) algorithm (as defined in 4.2.3) has played a pivotal part: while neural network (or perceptron) models were around long before NN popularity, they were largely deemed ineffective, at least in comparison to other available models of the time [61]. It was only during the 1980s that the BP algorithm was applied to NNs, and the field started to gain in popularity again [52, 86].

Rumelhart et al. showed that the BP algorithm as applied in NNs resulted in useful feature representations occuring in hidden layers and the empirical success that resulted thereof [67]. Shortly after, LeCun et al. 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) [53].

While many improvements were made during this time via gradient descent modifications (as expanded on in section **??**), 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 [63]. Essentially, 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 first suggested and primary solutions to the problem is to perform pre-training on the network through unsupervised learning [69], which is discussed more fully in 2.3.2.

There were also initial concern that the BP algorithm as applied to high dimensional neural networks would result in the network weights being trapped in local minima if a simple gradient descent was used (e.g. where no small changes to the configuration would reduce the average error rate) [54]. 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 [19]. Ge et al. have also shown that it is possible to escape saddle points and offer a guaranteed global convergence in certain non-convex problems [27].

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 might make them appropriate: Non-linearity allows for neural networks to operate as universal approximators, as shown in [41], 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 [87]. Lastly, the range of the function (infinite or finite) can impact both the stability and efficiency of the training.

Some of the most popular functions that have been used are the Sigmoid, TanH, ReLU and Softsign. There have been various studies showing the effectiveness of the different activations under varying initialization (or pre-training) for weights. Glorot and Bengio noted that the typical Sigmoid and TanH functions performed poorly with standard minimization, and result in slower convergence and worse minima, and showed that Softsign with a non-standard initialization resulted in quicker convergence [29]. Further research with Bordes & Bengio found that the rectifier (ReLU) functions were more effective in deep sparse networks compared to the TanH function [30].

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 pre-training of networks prior to supervised training. The effect was substantial enough to allow much deeper layered networks to be trained than before [10, 35].

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 pre-training 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 [35, 66, 36]. This is expanded up further in 2.3.2. Le Roux and Bengio were able to show that within the DBNs produced by Hinton [35], 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 [55].

FNNs had been shown to be effective in modelling high dimensional data even prior to the breakthroughs in deep networks [11], so it fits that the deep networks were shown to be extremely effective in high dimensional data classification. Early implementations shown increased efficacy in handwriting recognition, as well as pedestrian recognition [72]. 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 [44, 83].

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 [44, 30, 15]. When training for long enough, it was reported that the pre-training 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 certain attention was paid to the initialization, as well as the use of nonlinear activation units, it did suggest that pre-training largely acted as a prior which may not be necessary if large enough labelled datasets are available [12]. Naturally, pre-training was still implemented to prevent overfitting in smaller datasets.

2.2.4 Backpropagation Improvements

One of the instrumental improvements which aided the above achievements, was to modify the backpropagation algorithm using the dropout technique, as suggested by Hinton et al [38]. When training of large networks is attempted on small datasets, it often results in overfitting and poor results on out of sample data. Dropout helps resolve this by randomly excluding a certain percentage (usually 0.5) of feature detectors on each training iteration. The effect is to stop co-adaptations of feature detectors, and 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 resulting 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. used the dropout technique as the basis for their maxout activation function technique, which leverages and improves on dropouts 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 [28]. Further work was done by Wang et al., which improved on the dropout (and potentially maxout) techniques through fast sampling, resulting in an order of magnitude speedup in training [82].

2.3 Stacked Autoencoders

2.3.1 High Dimensional Data Reduction

As noted, machine learning techniques have been shown to be extremely effective at modelling non-linear inputs to outputs - neural networks have even been shown to be universal function approximators in this regard [41]. More traditional statistical models will typically process the available feature data to select the most significant features to be used in the model once its defined - evident in a processes such as subset selection [68]. 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, such as the neural networks described above.

Financial data, in line with the complex and dynamic system that it represents, is often of a very high dimensional nature, which offers opportunities through more sophisticated analysis, but also introduces the curses of dimensionality [21]. 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), as well as increase the risk of incorrect assumptions based on spurious variable collinearity [25]. Noise accumulation in high dimensional data can create further problems, resulting

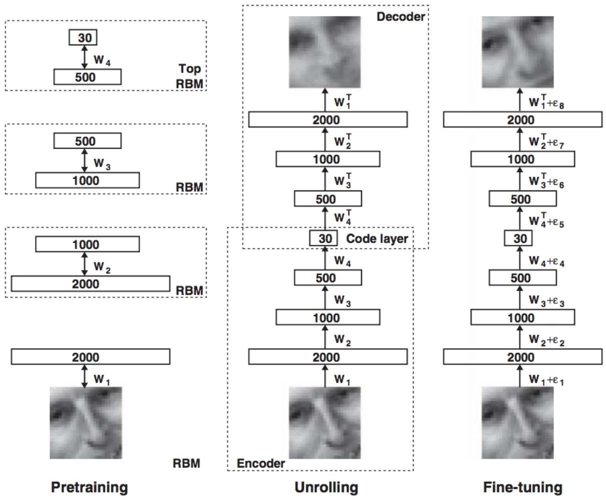

Fig. 1: The Autoencoder training steps [36]

in problems performing variable selection and ultimately having a large impact on classification and regression models [26].

Time series data can introduce its own set of challenges - there is often not enough data available to understand and predict the process [24], the time variable dependence creates complexity in how much past data to consider at any point, and the data is typically non-stationary [50]. 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), rather than static, and those which have performed feature selection processes rather than operating on the raw data (e.g. Auto-encoders) [50].

Two of the seminal pieces of research that have lead to the resurgence in machine learning and deep learning were the algorithms for training deep belief networks [35], as well as the usage of stacked auto-encoders [66, 10].

2.3.2 Deep Belief Networks

Autoencoders were suggested by Hinton et. al as a method of transforming high dimensional data to lower dimensional input vectors, in order to alleviate some of the problems detailed above, and increase performance of deep belief networks [36].

One of the more prominent classical techniques for dimension reduction is principal components analysis (PCA), which 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 nonlinear 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 transform 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 is 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 et al. 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) [37]. 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 1. They showed the deep autoencoder networks were significantly more effective than PCA or shallow autoencoders on multiple dataset types.

2.3.3 Stacked Denoising Autoencoders

The second important piece of work was the development of a denoising autoencoder (DAE), by Vincent et al. [79]. One of the problems identified in the DBN model (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. 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, with the key difference being 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), and so is forced to learn smarter feature mappings and extractions. The DAE suggested then is a stochastic variant of the autoencoder, which 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 [79] .

2.3.4 Pre-training

The methods described above follow a similar approach: greedy layer-wise unsupervised pre-training 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 results in significant performance gains [79]. However it is not immediately apparent, given the nature of backpropagation algorithms and the like, why this is the case. Erhan et al. performed extensive empirical simulations in order to suggest an explanation to the mechanism of pre-training [23].

While their results were not entirely conclusive, they did lend themselves to a reasonable hypothesis: the unsupervised pre-training 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 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 2.3.1. Due to the inherent difficulties with predictions in the financial system, it can sometimes be unclear if the shortcoming in results is due to this system complexity or if the methodologies used are unsuited for the purpose. In light of this it is worth pointing out that Stacked Autoencoder (SAE) implementations have been shown to be effective in many time series systems.

Lv et al. implemented a deep learning SAE model using the methods described in 2.3.3 in order to predict traffic flow at various time intervals (15, 30, 45 and 60 minutes) - a problem not so structurally dissimilar from what will be presented in this paper [58]. They were able to show that the deep SAE was able to offer prediction results which were both objectively good and also persistently outperformed the comparison models used (backpropagation neural network, random walk forecast, support vector machine and a radial basis function neural network).

In a review of unsupervised feature learning and deep learning methods on time series, Langkvist et al. noted 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 [50].

2.3.6 Financial Applications

There have of course also been successful applications of stacked autoencoders and deep learning models in finance as well. Takeuchi et al. 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 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 [76].

Zhao et al. 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 SVRs. 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 [91].

While much of the financial literature has focused on the use of RBM based models, Autoencoders and SAEs have recently been gaining popularity in performing feature reduction. Troiano et al. specifically investigate the use of different feature reduction models for trend prediction in finance [77]. In line with being primarily interested in the effect of feature reduction techniques, rather than the classification performance itself, only an SVM model was used to test results. Using various periods from historical S&P 500 data, they were able to show that AE 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. note that the research has been lacking with regards to whether SAEs should be used for financial prediction models or not [8]. They suggest a novel model which combines Wavelet Transformation, SAEs and a Long Short Term Memory (LSTM) network. Using data from several financial exchanges (considering a range of developed and undeveloped markets), they assess the models 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 suggesting the use of a Recurrent Autoencoder for multidimensional time series prediction [42]. There is a clear pattern through the literature that the use of AEs and SAEs both by themselves and when used as an assisting technique result in more accurate prediction results and less computationally expensive training.

2.4 Data Segmentation

One of the aspects of time series not yet discussed is how the data might be segmented for analysis. Financial pattern matching, as discussed in **??**, requires methodologies to decide the length of data to consider when determining whether a subset of data matches a pattern or not.

There are numerous classical approaches to this problem, which were widely applied in machine learning prior to the resurgence in deep learning. One of the earliest and more common approaches was the Sliding Window, where each the model input for each observation is padded by a predetermined number of observations that occurred both prior and after the one in question. This has the advantage of being fairly model agnostic, but fails to capture any correlations in the dependent variable values [16]. The Recurrent Sliding Window method aims to resolve this by including the same number of prior predictions made as part of the input. In this case, input would be $< x_{t-d}, ...x_t, , x_{t+d} > and < y_{t-d}, ..., y_t >$. This was shown to be significantly more effective than the plain sliding window method [7]. Notably, the sliding window approach can only be implemented in an offline model.

These methods can be adapted to take on online formats, as well as incorporate data reduction benefits through algorithms such as Piecewise Linear Approximation (PLA), which adapts a linear representation to the leading portion of the window. Some novel approaches (Feasible Space Window and its stepwise adaption) were suggested by Liu et al. in order to compensate for the computational requirements of reprocessing the entire window at each step for online models [56].

Window based methods represent a fairly static and unsophisticated approach to data segmentation. A suggested solution to this is to segment the data by dynamically identifying significant partition points, known as perceptual important points (PIP) [17]. The computational cost of identifying PIPs was initially rather large, making them unsuitable for quickly changing and dynamic environments such as finance. However, Zhou et al. suggested a novel approach which reduced computational costs through intelligent binary tree traversal. They were able to show this approach was effective in identifying traditional financial stock patterns with the use of a layered neural network [93]. In a similar fashion, input data turning points (TP) were shown to potentially offer lower error rates than PIPs by Yin et al. [88].

More recently, Wan et al. conducted a review on the effect of segmentation on financial time series pattern matching, comparing perceptual important points, piecewise aggregate approximations, piecewise linear approximations and turning point based methods [80]. They use several pattern matching algorithms (template-based, rule-based, hybrid, decision tree, and symbolic aggregate approximation) for each of the segmentation methods in order to determine their effect on a broader level. The analysis was performed on real stock market data, as well as synthetic data generated to display common patterns such as positive/negative head and shoulders (H&S). Measuring accuracy, precision and recall, they showed that PIP based segmentation generally offered superior results to the others (though it is worth noting that there were various performance differences within the segmentation methods depending on the matching algorithm used).

2.5 Online Learning Algorithms and Gradient Descent

Most classic machine learning algorithms operate under the assumption that, for all intents and purposes, 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 growth of information grows in an exponential fashion, there are numerous areas where the expected training data for the model will continue to grow. 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 rather repetitively draw on single samples from the data on which the models parameters can be adjusted. The benefit is that they are able to quickly process a large number of observations and readjust the model, though 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 Cun argue that as the size of the dataset grows significantly, online algorithms advantages result in them outperforming offline models, despite any initial drawbacks [13]. 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 [51, 14]. Bottou and Cun showed in fact, that it is 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 [13]. Theoretical and empirical findings were presented to show that a stochastic online gradient descent (SGD) algorithm **[referenceappendix]** was able to outperform the batch model for parameter estimation, and was able to asymptotically outperform in the number of samples processed in a time period. The stochastic aspect of the algorithm is related to random observations from batch sample groups being used as the gradient basis. Theoretically, this slows down the convergence, but speeds up the processing speed of each batch - a technique which has later been shown to be generally successful [73, 92].

The SGD algorithm has resulted in a fair amount of further research due to its applicability to machine learning and the online benefit, which can largely be group into two categories: improvements affecting gradient learning and convergence rates, and processing improvements through parallelization.

One of the earlier improvements to convergence rates was the Momentum algorithm, as developed by Tseng [78]. 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 increasing movement towards directions consistent with previous gradients (this is achieved through combining gradient movements in opposite directions) .

There have also been several attempts to introduce effective regularization into the SGD process. Bartlett et al. presented Adaptive Online Gradient Descent, which implements an adaptive step size through a λ penalty on the learning rate, which was shown to be nearly optimal in a strong sense [9]. 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 L_1 regularization (similar to Lasso Regression). They were able to show that implementation performed effective feature reduction, while having little effect on performance [49]. 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 [22, 89].

A parallel implementation of SGD largely rests on the idea of splitting up data to be processed in individual runs of the gradient descent. The results are periodically aggregated, and the new model parameters are distributed to processing nodes for further training [90]. Depending on the configuration, the variance within results can result in poor convergence rates [59]. Mahajan et al. suggest a novel implementation which improves the distribution impacts through the use of better approximating functions in the processing nodes, which in turn improves the efficacy of the algorithms convergence [59]. Povey et al. presented a Natural Gradient SGD, which improves the learning rates through the use of a factor matrix used on the new gradients. They were able to show empirical evidence of this improving performance issues introduced from the parameter averaging typically used in parallelization [65].

There have been some more recent improvements focusing on making the SGD algorithm more dynamic, including the Inconsistent SGD (ISGD) and AdaBatch adaptations. The idea behind ISGD is to treat the training on a particular batch as a stochastic process, and adjust it according to the expected loss identified. Gradient updates from small loss batches are relatively small compared to large loss batches, and so by focusing efforts here ISGD is able to optimize performance. Wang et al. performed careful testing of SGD vs. ISGD and found inconsistent training to consistently outperform in terms of convergence speed and results [81]. It is worth noting that due to its nature, ISGD can be effectively combined with other SGD improvements (e.g. Momentum, as per the authors experiments). AdaBatch, as presented by Devarakonda et al., introduces dynamism through the adjustment of batch sizes. Small batch sizes in SGD offer the benefit of faster convergence in fewer training epochs, however larger batch sizes are more computationally efficient due to their applicability to parallelization. The algorithm uses a monotonically increasing batch size, which is started small to gain traction in convergence, and later increased to allow the benefits of data parallelism. The effect, as shown, is to offer improvements in training performance of up to 6x, with less than 1% of accuracy when compared to the fixed batch baselines [20]. Both of these algorithms have been shown to be effective and applicable in the context of online neural network training.

2.6 Backtesting and Model Validation

Much of financial academic literature is currently facing a problem in terms of validation and verification of results. The primary method of going about these ends in the past has been to perform historical simulations, or backtests , in order to prove profitability of a trading strategy. The recent advances in both technology and the algorithms available to construct these strategies has resulted in researchers being able to run so many iterations of a model or strategy configuration through these backtests, that its become increasingly difficult to control for spurious results, with some papers suggesting that most published research findings are false [45].

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. The problem lies in that millions of different model configurations might be tested, and if more sophisticated test measures are not in place (i.e. not just the standard Neyman-Pearson hypothesis testing framework is implemented), 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). The nature of financial data, where there is a low signal-to-noise ratio in a dynamic and adaptive system, and where there is only one true data sequence, makes it difficult to resolve these issues effectively [4, 60].

Overfitting is not a novel issue, and has of course 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. Other methods, such as hold-out, are arguably still faulty due to researcher knowledge while constructing models [70]. One of the downfalls of the typical IS-OOS set up in the financial context is also that the most recent (and relevant) data will not be able to be used for the model training.

There have been some suggestions to resolve the problem that is occuring in the literature as a result of this - some work 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 [64]. While the points made with regard to the review process and so on are certainly important, they don't aid with more effective model training for the researcher up front, and so will not be covered here.

2.6.1 Testing Methodologies

Considering the issues laid out above, there has been much work to develop alternative approaches to backtesting. 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, including that the data is often used regardless, or that awareness of the movements in the data may, consciously or otherwise, influence strategy and test design by the researchers [70]. For small samples, a hold-out strategy may be too short to be conclusive [84], and even for large samples it results in the most recent data (which is arguably the most pertinent) not being used for model selection [34, 4].

There has been work by several authors to try and lay out techniques to try and avert backtest overfitting. The Model Confidence Set (MCS), as developed by Hansen et al. [32], starts with a collection of models or configurations, and remove 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 et al. [2], 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. [4] 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 median OOS in comparison to the remaining configurations. They use this definition to develop a framework which measures the probability of such an event occuring, where the sample space is the combined pairs of IS and OOS performance of the available configurations. 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 (SR), 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.

2.6.2 Test Data Length

The CSCV methodology offers an important but highly generalised framework to assess models and backtest overfitting. It doesnt however indicate which metrics should be used to assess the IS and OOS performance, nor any indication on the amount of data needed to do so effectively. One 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 skillful - it could be that all of them are. Also, as pointed out, it should not be used as an objective function to avoid overfitting, but rather as an evaluation tool. To this end it helps assess overfitting, but not necessarily avoid it.

A typical measure of evaluation used for financial models is the Sharpe Ratio (SR), which is the ratio of between average excess returns and the returns standard deviation - a measure of the return on risk. In the context of comparing models, SR is typically expressed annually to allow models with different frequencies to be compared. Lo et al. [57] show that annulaized SR can be expressed as

$$SR = \frac{\mu}{\sigma}\sqrt{q} \tag{1}$$

Using sample means and deviations, $\hat{\mu}$ and $\hat{\sigma}$, SR can be shown to converge as follows (as y $\rightarrow \infty$)

$$\hat{SR} \rightarrow \mathcal{N}[SR, \frac{1 + \frac{SR^2}{2q}}{y}] \tag{2}$$

Thus, when using SR estimations, which follow a Normal distribution, it is possible that where 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. [5], 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, thich is the Minimum Backtest Length (MinBTL), ultimately showing that

$$MinBTL \approx (\frac{(1-\gamma)Z^{-1}[1-\frac{1}{N}] + \gamma Z^{-1}[1-\frac{1}{N}e^{-1}]}{\overline{E[max_N]}})^2 < \frac{2ln[N]}{\overline{E[max_N]}}^2 \tag{3}$$

The statistic highlights the relationships between: selecting a strategy with a higher IS SR than expected OOS, the number of strategies tested (N), and the number of years tested (y). The equation shows that as the number of strategies tested increases, the minimum back test length much 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 impossible to assess.

Bailey et al. 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 isnt controlled for [5].

2.6.3 Sharpe Ratio

The use of the Sharpe Ratio in financial backtesting is not just an arbitrary or persistent literature choice. The statistic offers two benefits: the effectively strategy-agnostic financial information contained, as well as being relatable to the t-statistic, and so simple to perform hypothesis testing. The SR ratio (estimate from sample as $\hat{SR}$) is defined as

$$SR = \frac{\mu}{\sigma} \tag{4}$$

The t-ratio is defined as

$$t - ratio = \frac{\hat{\mu}}{\hat{\sigma}/\sqrt{T}} \tag{5}$$

Evidently, the link heree ratio (though it can be generalized to another statistic with a probabilistic interpretation). Additionally, PBO is generally more in line with machine learning literature with the cross validation like approach on time series data.

It should be noted, that the literature detailing usage of the Sharpe ratio for strategy comparison is extensive, with numerous variations and methodologies offered [6]. However, the crux of this paper lies in whether an online neural network is able to make effective enough predictions that a strategy might use the predictions to be profitable. The subtlety here is that we will consider the usage of such forecasting *within* a strategy, rather than *as* a strategy itself. In line with this, statistics such as the Sharpe ratio will be used, but not form a critical consideration of the research here as the comparison of strategies used will be a secondary concern.

3 Data

3.1 Data Processing

All datasets were transformed into log feature fluctuation values, and were then expanded to include fluctuations over rolling window periods.

$$log\_diff(x_t) = ln(x_t) - ln(x_{t-1}) \tag{6}$$

This log feature fluctuation, is processed for each OHLCV feature and for each time point (from the previous time point). The log fluctuations have the benefit of taking compound effects into account in a systematic way and symmetric in terms of gains and losses.

The datasets are then expanded with the rolling window fluctuations both in the past, for input, and in the future, for predicted output. For example, at each data point t, the following time series are added:

- Log feature fluctuations over the periods $(t-7, t)$ and $(t-30, t)$ (with $(t-1, t)$ already having been processed)
- Log feature fluctuations for the period $(t, t+10)$, to be predicted

3.2 Data Scaling

Once the log difference data has been processed, the datasets are standardized to allow for better learning. This is done in the typical way, calculating for mean 0 and variance 1: each feature vector has its mean subtracted, and is divided by its standard deviation.

$$standardized(x) = \frac{(x - \bar{x})}{\sigma_x} \tag{7}$$

3.3 Synthetic Data

Synthetic datasets were generated to represent known trends with some variance in prices. Geometric Brownian motion was used to simulate stock price movements. Each dataset was generated with a random seed (using a Mersenne Twister pseudorandom number generator), and had a structure of 9 stocks:

- 2 stocks with an upward trend, and high and low variance levels
- 2 stocks with a downward trend, and high and low variance levels
- 2 stocks with a sideways trend, and high and low variance levels

The steps for Geometric Brownian Motion are described in 1, which allows the stocks simulations to be implemented with a trend (σ) and variance (μ) for each stock.

Algorithm 1: Geometric Brownian Motion Simulation

Input: σ, μ, S_0, *steps*
t = 1/steps;
prices = $[S_0]$;
foreach *i in 1:t* **do**
  z = random() $\sim N(0, 1)$;
  $S_t = S_{t-1} * e^{(\mu - \frac{\sigma^2}{2})t + \sigma\sqrt{t}z)}$;
  prices = [prices; S_t];
end
Result: prices

Simulated Dataset This section will detail the chosen σ and μ values for the dataset, as well as the expected results from training on them.

3.4 Real Data

This section will detail the use of real data - namely the JSE ALSI data collected, and the time periods collected for. Detail will be added once this dataset has been finalised.

4 Implementation

4.1 Process Overview

The implementation focuses on bringing together several ideas: data reduction, deep learning with pre-training/weight initialization, online learning and backtest overfitting validation for the purposes of stock price prediction. The process implementation is discussed fully in 4.9, but can be summarised as the following:

1 The dataset is split into 2 subsets: the SAE/FFN portion, and the OGD portion

2 The first subset is used to train the SAE and deep FFN using the SGD algorithm. These networks were implemented with pre-training and weight initialization techniques

3 The second subset of data is used to continue training the network in an online manner using OGD

4 The returns generated from OGD can then be used in the CSCV process, to estimate the probability of backtest overfitting

The rest of this chapter will detail the algorithms used to train the relevant FFN, RBM and SAE networks, as well as the trading strategy and CSCV & PBO testing procedures.

4.2 Feedforward Neural Networks

Feedforward Neural Networks (FFN) in the form of multilayer perceptrons is a well established network technique, providing effective nonlinear representations for both shallow and deep structures [69]. 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 does 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.

4.2.1 Notation and Network Representation

· The weights from the j^{th} node in the l^{th} layer from the k^{th} node in the $(l-1)^{th}$ layer are represented by w^l_{jk}

· The bias for the j^{th} node in layer l is represented by $w^l_0 j$

· The output for the j^{th} node is defined as $o(z_j)$, for an activation function o and weighted input z

· The weighted input for the j^{th} neuron in a layer is $z^l_j = \sum_{n=i} a^{l-1}_i w_{ij} + w_{0j}$

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

4.2.2 Activation Functions

As noted in 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, 2 of the most used were implemented here.

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

$$f(x) = \frac{1}{1+e^{-x}} \tag{8}$$

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 casuing what is referred to as node 'saturation'). The exponent calculation is also computationally expensive, relatively speaking.

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

$$f(x) = max(0, x) \tag{9}$$

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 (though 'Leaky ReLUs' can be used to help resolve this)

4.2.3 Backpropagation

The backpropagation algorithm, as discussed in 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 is Mean Squared Errors (MSE):

$$C = \frac{1}{2}\|y - a^L\|^2 \tag{10}$$

This allows the backpropagation algorithm to implemented as follows:

- Let N be a neural network of weights w, inputs x and outputs y, where the weights are (naively) initialized to random values
- The algorithm then iterates through the following:

  1 Select a set of m samples x_s

  2 Forward Pass:

    i Propagate the samples through the network using the functions defined in 4.2.1 to generate out output y_s

    ii Calculate the error term (aka cost), as

$$\lambda_x^L = \nabla_a C \otimes o'(z^L) = (a^L - y) \otimes o'(z^L) \tag{11}$$

  3 Backward Pass:

    i Propagate activation value back through the network to calculate the delta values (the difference between the network outputs and desired output values at each layer)

$$\lambda_x^l = ((w^{l+1})^T \lambda^{l+1}) \otimes o'(z^l) \tag{12}$$

    ii Each weights output delta and input activation are multiplied to find the weights gradient and this gradient is reduced by a factor of the learning rate η, and the mean of this is subtracted from the network weights

$$w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \lambda^{x,l} (a^{x,l-1})^T \tag{13}$$

  4 Repeat from step 1, unless a minima or other stopping criterion is reached

4.2.4 Gradient Descent Algorithms

Stochastic Gradient Descent (SGD), as discussed in **??**, has been shown to increase the speed at which the backpropagation algorithm can converge a minima in terms of cost. The algorithm runs backpropagation over the entire dataset for a 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).

· For each epoch:

1 Select k samples at random from the data which have not yet been sampled in this epoch

2 Use backpropgation on the network using the k samples to update the weights

3 Repeat 1-2 until all samples from the dataset have been sampled

4 Start a new epoch, or finish if a stopping condition has been reached

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, but for only one sample at a time (i.e. $m = 1$)

4.2.5 Gradient Descent Improvements

This section will detail and improvements that are used in the project - i.e. L1 and L2 regularization, or the learning rate schedule

write t tion

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 function - a typical RBM has one visible layer (input/output), and one hidden layer. A sample will be processed from the input layer to the hidden layer, and the activation values from the hidden layer will be used by the input layer to provide a recontruction. The hidden units thus correspond to feature detection of the visible unit data structures, and the learning process of the network results in effetive parameter estimation.

One of the primary differences from an FFN lies in the stochastic unit determination - the values in a hidden layer will typically be implemented such that the take on a binary value with a probablistic 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 joint configuration (v, h) of the visible and hidden units has an energy given by:

add Ho 1982 r

$$E(v,h) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_i j \tag{14}$$

where v, h correspond to the binary states of the visible and hidden units, with biases a, b, and weights w. It can be shown , that network weights can be adjusted to change the probabilities assigned to a particular training sample. The derivations of 14 show that performing a stochastic asent for the log probability of the data can be implemented through the following weight adjustment:

add hi erence

$$\Delta w_{ij} = \eta(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}) \tag{15}$$

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, the it is easy to get unbiased sampling of $\langle v_i h_j \rangle_{data}$

$$p(h_j = 1) = \sigma(w_{0j} + \sum_{i=1} v_i w_{ij}) \tag{16}$$

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

$$p(v_i = 1) = \sigma(w_{0i} + \sum_{j=1} h_j w_{ij}) \tag{17}$$

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

$$\Delta w_{ij} = \eta(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{reconstruction}) \tag{18}$$

While it's 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.

4.3.1 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. Thus, CD-1 is the following:

- For a random training sample V', sample H' from $P(V|H)$
- Sample V from $P(V|H')$
- Sample H from $P(H|V)$
- Return $(V, H) \sim R$, where R is the reconstructed distribution for $\langle vh \rangle$

Using the reconstructed distribution $R(V,H)$, the weight update for CD-1 is, as above, then

$$\Delta CD(W) = \eta(E_{P(H|V)D(V)}[VH^T] - E_{R(V,H)}[VH^T]) \tag{19}$$

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.

4.3.2 CD-1 and SGD

In the same way that the backpropagation learning algorithm in 4.2.3 can be implemented in an 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 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 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 4.2.3) to reconstruct the input, so 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 dimenstional 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 et al. for initialization of weights helps resolve this.

4.4.1 Sigmoid based Greedy Layerwise SAE Training

The algorithm for implementing the SAE training suggested by Hinton et al is as follows

1 Define a network structure which conforms to the requirements of an SAE, with L layers

2 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

3 For each layer l in $(2, L/2)$:

- Process the data through the previously trained layers using the forward propagation as defined in 4.2.1
- 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

4 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

5 This network can then be trained using the backpropagation and SGD algorithms, where cost of reproducing the network input is minimised

6 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

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 above.

1 Define a network structure which conforms to the requirements of an SAE, with L layers

2 Use an effective weight initialization, such as Xavier or He

3 This network can then be trained using the backpropagation and SGD algorithms, where cost of reproducing the network input is minimised

4 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

4.4.3 Denoising Autoencoders

If denoising is implemented (presumably Gaussian denoising), then this section will detail the implementation. decide

4.5 Variance Based Weight Initializations

Recent research, as noted in , 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 pre-training is required). The 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 would prove faulty), and so the methods are tested here as well. add LI ence h

A typical initialization would be uniform, and layer component agnostic, such as the Hinton initialization suggested as add re

$$w_{ij} \sim N(0, 0.01) \tag{20}$$

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

$$Y = W_1X_1 + W_2X_2 + ... + X_nW_n \tag{21}$$

It can thus be shown, that for $i.i.id$ samples with mean 0, then

$$Var(Y) = nVar(W_i)Var(X_i) \tag{22}$$

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

$$Var(W_i) = \frac{1}{n_{in}} = \frac{1}{n_{out}} \tag{23}$$

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

$$Var(W_i) = \frac{2}{n_{in} + n_{out}} \tag{24}$$

Using this as the weight values expectation function provides us with the Xavier Glorot initialization which can be used for Sigmoid activations, and is defined as follows add re

$$w_{ij} \sim U(-r, r), r = \sqrt{\frac{6}{n_i + n_j}} \tag{25}$$

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

The initialization for ReLU is different on accounts of the function being equal to zero for half it's potential input range - in this case it makes sense to double the weight variance, and so the He initialization is used add re

$$w_{ij} \sim U(-r, r), r = \sqrt{\frac{6}{n_i}} \tag{26}$$

4.6 Trading Algorithms

Will need to detail the trading strategies used here - currently just implementing a niave representaton of prediction accuracy.

4.7 CSCV & PBO

The combinatorially symmetric cross-validation (CSCV) method developed by Baily et al., as discussed in 2.6.1, can be used to assess the likelihood of backtest overfitting through comparison of IS and OOS return metrics. Formulaically, the definition of backtest overfitting is given by

$$\sum_{n=1}^{N} E[\overline{r_n} | r \in \Omega_n^*] Prob[r \in \Omega_n^*] \leq N/2 \tag{27}$$

Where the search space Ω consists of the N ranked strategies, and their ranked IS performance r and OOS performance $\bar{r}$. This allows the PBO, using the bayesian formula, to be defined as

$$PBO = \sum_{n=1}^{N} Prob[\bar{r} < N/2 | r \in \Omega_n^*] Prob[r \in \Omega_n^*] \tag{28}$$

Notably, the above definitions consider IS as the data made available to the strategy selection, rather than the models calibration (e.g. the full IS dataset, rather than, by was 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 as follows [4]:

1 Generate a TxN performance series matrix, M, representing the profits and losses by the N configuration trials over T time periods

2 Partition the M matrix by rows into S submatrices, each of even size (T/SxN)

3 Generate the combinations C_S of M_S, in groups of size $S/2$, for total $\binom{S}{S/2}$ of combinations

4 For each combination in $c \in C_S$:

  a Form a training set J by joinined $S/2$ M_S submatrices, in their original order. J is a matrix of order $(T/S)(S/2) * N$

  b Form the test set $\bar{J}$ as the complement of J in M, once again in the original order

  c Form a vector of R^c of performance statistics of order N, where the Nth component R_n^c of R^c reports the performance associated with the nth column of J

  d Repeat [c] for $\bar{J}$ to derive $\bar{R}^c$ and $\bar{r}^c$

  e Determine the element n^* such that $R_n^c \in \Omega_n^*$ - i.e. n^* is the best performing strategy IS

  f Define the relative rank of $\bar{r}_{n^*}^c$ by $\bar{\omega}_c := \bar{r}_{n^*}^c/(N+1) \in (0,1)$. 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 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 ovefitting)

5 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

$$f(\lambda) = \sum_{c \in C_S} \frac{\chi_\lambda(\lambda_c)}{\#(C_S)} \tag{29}$$

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

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

$$\Phi = \int_{-\infty}^{0} f(\lambda) d\lambda \tag{30}$$

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., 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 Performance Assessment

This section may detail any choices and implementation of the performance assessment to be used in CSCV if it's not just the Sharpe Ratio.

4.9 Full Process Implementation

The full process tested here, combining the techniques defined thus far in the paper, is defined below:

1 Data preparation

- The data is processed into day to day log fluctuations
- At each time point, rolling historic summations are calculated (e.g. the past 1, 7 and 30 days) update cided v
- At each time point, rolling future summations are calculated (e.g. the next 10 days) update cided v

2 Data Segregation

- The processed dataset is split into 2 parts, to be used for the following processes: SAE/SGD training, OGD training (e.g. 60%, 40%). The split here will be based on the MinBTL method developed by Bailey et al. update decide

3 SAE Training

- The dataset defined for SAE/SGD is used to train the auto-encoder network (using either RBM pretraining/weight initialization algorithms + SGD)
- Notably, this SAE/SGD dataset will itself be split into training & testing portions in order to train and validate the SAE network
- The full process will rely on a generated set of 'best' SAE networks at each encoding layer size, to be used for FFN training and prediction. This takes advantage of the fact that the SAE training will not suffer from backtesting concerns, and so a wider consideration of configurations is possible
- Once the SAE networks have been defined, they can be used to reprocess the datasets such that the input is encoded, and the output is as before. These encoded datasets will be used for all following steps

4 FFN/SGD Training

- The same SAE/SGD portion of the data will then be used to train the FFN network to predict future prices, using the implemented SGD algorithm & optimizations and the encoded datasets

5 OGD Training

- The second portion of data will be used to train and optimize the FNN from 4 for OGD. The purpose of this will be to optimize the OGD parameters

6 Returns generations

- The trading strategy, as per 4.6, will be used to generate returns from the predicted and actual prices in the OGD dataset. These returns for each time period and each configuration of the model defined by steps 3-5 will form the M matrix for the CSCV process.

7 CSCV & PBO

- Using the M matrix from 6, the CSCV process will be run which will then allow a calculation of PBO.

5 Results

5.1 Search Methods and Parameters

This section will detail the search methods used - grid search / random search, details are seed settings etc., and the final configuration parameters chosen to run the models for the results below.

The experiments will be designed around exploration of the following points (listed in order of importance), which will then be written up as results:

- Online FFN network performance: This is the crux of the thing - how effective is the online network learning for predicting feature changes, and whether it is able to achieve efficacy such that it can be generate returns.
- CSCV: To show that the first model can be achieved in a manner that does not result in high PBO. A discussion of how the CSCV method has been adapted to be used for a deep machine learning model will be appropriate here as well.
- Data Reduction: Expected results will be a trade off in model efficacy on producing returns vs. training time. Ideal results would be to show that the SAE results in effective feature selection, and the removal of noise thus allows the model to perform better (i.e. better returns are produced). A possible but slightly less favourable result would be to show that it allows for a simpler and faster online model in terms of computation time, for an acceptable reduction in performance and returns. A result which shows neither would probably indicate an implementation failure of some sort, or a misunderstanding of how things should have been applied in the first place.
- Profitablity: Consider the returns generated in terms of test for statistical arbitrage and a correction for transaction costs.
- Weight Initialization: Experiments here will explore the use of ReLU activations and variance based weight initialization, and whether they are able to compete effectively (or perhaps even better?) than the RBM pre-training and Sigmoid counterparts. If so, it would constitute a favourable contribution, considering that signficant reduction in computation processing time.
- Training Findings: It's possible that the training process will indicate some learnings which would be appropriate to note - e.g. that L2 regularization has an unexpetedly large impact on performance.
- CSCV and PBO: Note around the fact that will need to train for return results, and not for PBO results (which would be overfitting in itself)

5.2 Results for Synthetic Data

5.2.1 Dataset Generation

Data was generated and standardized, as per the methods detail in 3.3 and 3.2. The set was generated with the following configurations:

- Upward Trend, High variance: μ = 0.9 and σ = 0.5
- Upward Trend, Low variance: μ = 0.9 and σ = 0.2
- Downward Trend, High variance: μ = -0.8 and σ = 0.55
- Downward Trend, Low variance: μ = -0.8 and σ = 0.15
- Sideways Trend, High variance: μ = 0.05 and σ = 0.4
- Sideways Trend, Low variance: μ = 0.05 and σ = 0.1

The dataset was generated for 5000 steps, and had a 60/40 split for the SAE&SGD/OGD training. The within dataset split for training and validation was 80/20. The prior timewindows used were the past 1, 7 and 30 days, and the future window used for prediction was the next 2 days. The actual price points generated are visible in figure 2.

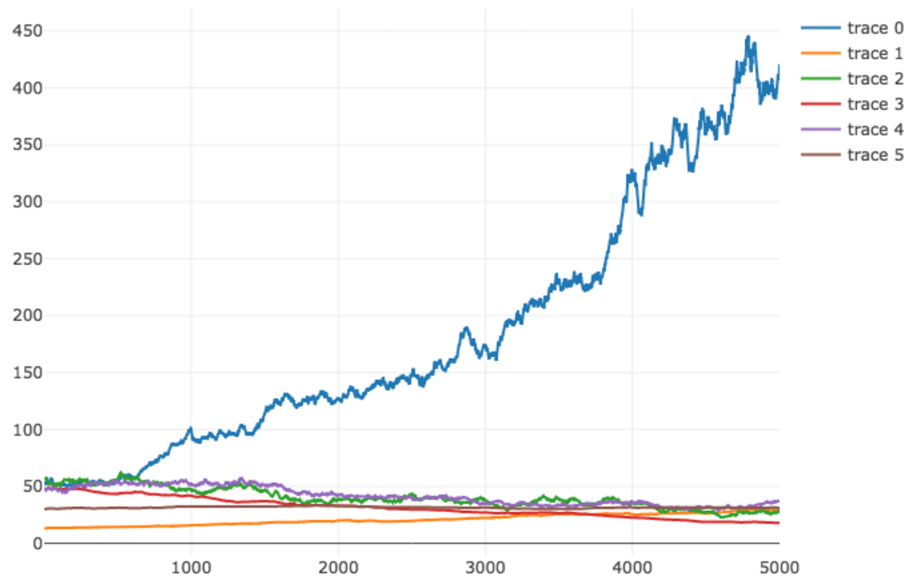

Fig. 2: Price Points

5.2.2 SAE Network and Results

The SAE networks considered were all based using ReLU activations and training methodology (as per 4.4.2), and were generated for all possible combinations of the following configurations:

- Learning Rates: 0.1, 0.01, 0.001, 0.0001
- Weight Initializations: XavierGlorot, He, Hinton
- Layer Sizes: 40, 80
- Number of Hidden Layers: 1, 2
- Encoding Layer Sizes: 3, 6, 9, 12, 15

Thus, a total of 240 different neural networks were trained on the synthetic data generated. Each was run for 2000 epochs, with a minibatch size of 20. The results were compared, with assessment ultimately based on best epoch MSE and MAPE as well as a view on the training vs. testing cost progression over epochs. The conclusions drawn from this are as follows:

- The Hinton initialization results in almost uniformly worse performance than the He & Xavier initializations.
- While the He initialization is considered ideal for ReLU activations, the Xavier seemed to show slightly better performance at an aggregate level (though there was no clear winner here). There may be some lack of stability introduced by He as it only considers the previous layer, as opposed to both the previous and following layers as with Xavier. In the context of an SAE with a smaller encoding layer, this may not be ideal behaviour, though this conclusion would warrant further investigation. The initialization MSE results are displayed in 3.
- The results for the 1 hidden layer SAE networks were typically better than their pairwise 2 layer version. There is no reason to believe that a deeper network would be inherently worse, but it did highlight that it would require greater effort to tweak the training parameters to account for the extra 2 layers in the SGD process.
- The learning rate 0.1 is too high and had uniformly worse results across configurations, especially for the He and Hinton initializations.
- The layer sizes (40 / 80) had mixed effects - the larger hidden layer sizes had better results for lower learning rates (0.001 and 0.0001), while at a higher learning rate of 0.01 the smaller layers outperformed. The suggestion, as confirmed through epoch assessment, is that the lower learning rate combined with a larger network can allow for a more consistent, though slower, improvement of the network. Higher learning rates allow for a quicker oscillation to a minima, which is more easily done in a smaller network.

- One unexpected result was the occurence of overfitting for smaller encoding layers with higher learning rates. Overfitting, as defined by the persistent decrease in training MSE and the increase in testing MSE, was more likely to occur as the encoding layer was reduced from 12 and lower, and as the learning rate increased from 0.001 upwards and as the number of hidden layer nodes increased. It would seem that the reduction in signal as a result from smaller encoding layers results in reducing the models ability to train properly, and so overfitting on training data occurs, which in turn is more easily induced through larger layer sizes and learning rates. The smaller layers, in this scenario, work as a form of regularization. It would require algorithmic changes, such as the use of SGD regularization, to try and avoid this issue in the noted configurations.

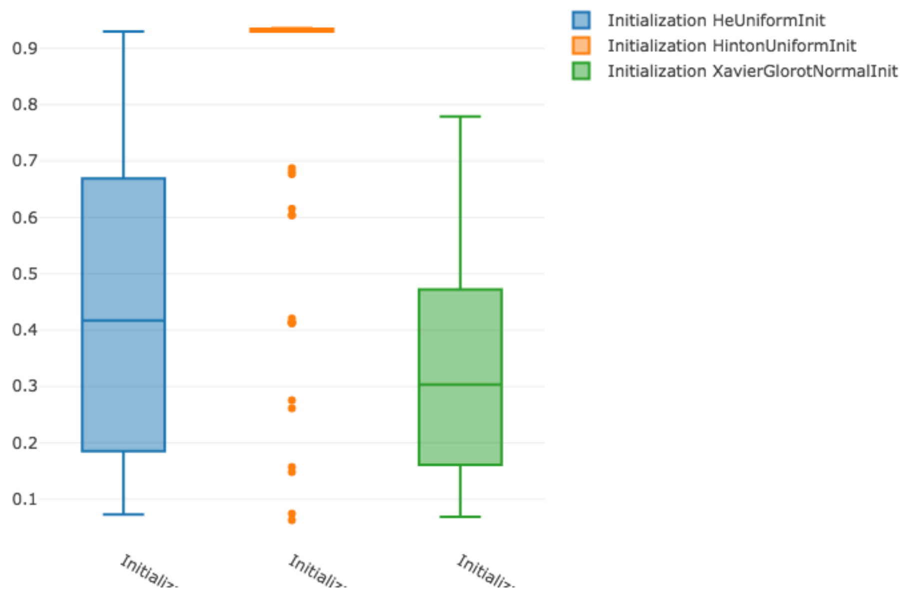

Fig. 3: Boxplot of SAE Minimum MSE

These conclusions were used to choose a set of networks to refine and train further, with the aim of producing one best network per encoding layer size, which can then be used as a configuration item in the training of the prediction network. All of the networks chosen had one hidden layer, were initialized using the Xavier technique and were trained for 15000 epochs. Notably, the input size here is 18 (6 stocks at 3 time windows).

Encoding Size	Hidden Layer Nodes	Learning Rate	Reconstruction MAPE	Best Epoch Test MSE
3	40	0.001	75.93%	0.5580
3	80	0.0001	86.8%	0.6810
6	40	0.001	67.99%	0.4247
6	40	0.0001	67.83%	0.4255
6	80	0.001	72.86%	0.4252
6	80	0.0001	68.82%	0.4269
9	40	0.0001	54.53%	0.2787
9	80	0.0001	56.8%	0.2886
12	40	0.01	39.62%	0.1521
12	40	0.001	38.23%	0.1481
12	80	0.0001	39.96%	0.1562
15	80	0.01	27.85%	0.0797
15	80	0.0001	26.03%	0.0820

There was some evidence of overfitting in the smaller encoding layers, something expanded on later.

5.2.3 Predictive FFN Network and Results

In light of the results presented in 5.2.2, the predictive FFN networks were trained on a set of configurations encapsluating all permutations of the features below:

- AutoEncoders: Of encoding layer sizes 3, 6, 9, 12 and 15
- Number of hidden layers: 1, 2, 3
- Size of hidden layers: 20, 40, 80
- SGD Learning Rate: 0.0001, 0.001, 0.01
- OGD Learning Rate: 0.00001, 0.0001, 0.001

Each configuration was run for 2000 epochs, with a minibatch size of 20. Results were considered in terms of the SGD MSE scores, as well as the overall profit a model was able to produce. The 'profit' calculation used here is more representative of practical performance than of expected effect, particularly as the comparison is being made on scaled log difference returns (i.e. the processed dataset output). The formula is as follows:

$$f(price_{actual}, price_{predicted}) = \begin{cases} abs(price_{actual}) & \text{if } sign(price_{predicted}) = sign(price_{actual}) \ . \\ -abs(price_{actual} - price_{predicted}), & \text{otherwise.} \end{cases} \tag{31}$$

The profit results for all combinations can be viewed in 4 below. While most are very poor, this is in part a result of the experiments being conducted to explore the configuration space quite widely. The highest profit calculated here was 781, where perfect knowledge of price movement would have resulted in a profit of 9511.

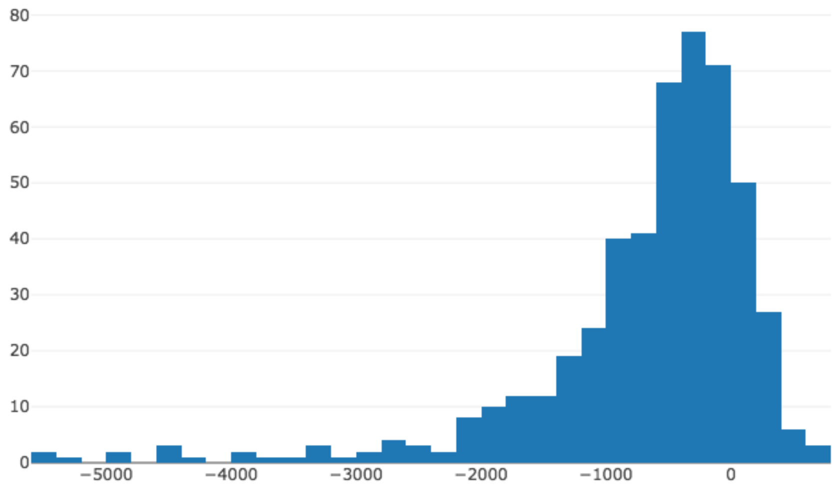

Fig. 4: Histogram of Total Profits

SAE Results

Figure 5 shows the effect of the different autoencoders on the profits generated, with encoding layer 9 performing the best for the most part. Encoding layer 12 seems to be out of pattern with 9 and 15 in the profits generated - it's possible that the trade off between feature reduction and feature representation is at a minima in the network used there. Aside from that, the results seem appropriate, showing effective feature reducton up to a point, but a loss in performance when the reduction is pushed too far (though this could be improved with better training).

Figures 6 and 7 show the effect of the different autoencoders on the best test score in the training epochs and the average score in the last 100 epochs, respectively. Noting the y-axis in figure 6 shows that the networks are largely able to achieve similar performances here, with no clear sense of one being better than the other - 15 has the lowest MSE score, but also the highest by far. Figure 7 shows the effect of overfitting on the training at the moment, with the ending MSE scores getting progressively worse.

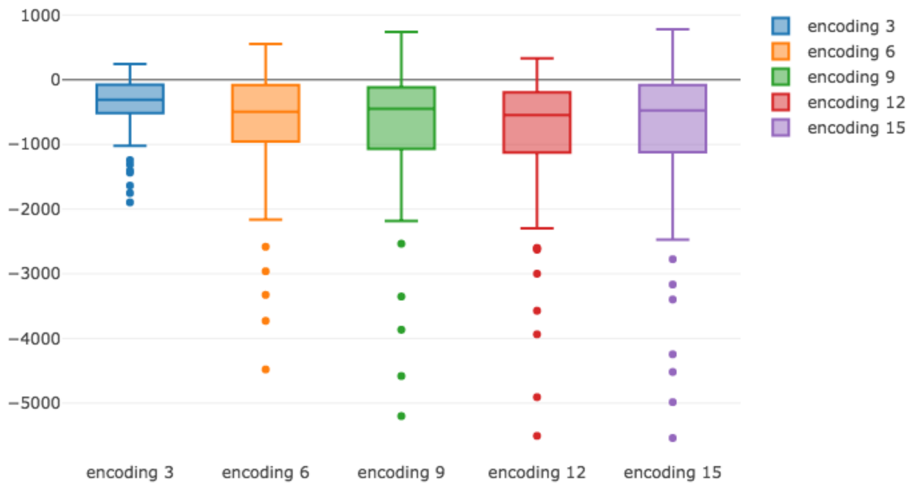

Fig. 5: Boxplot of SAE Profits Generated

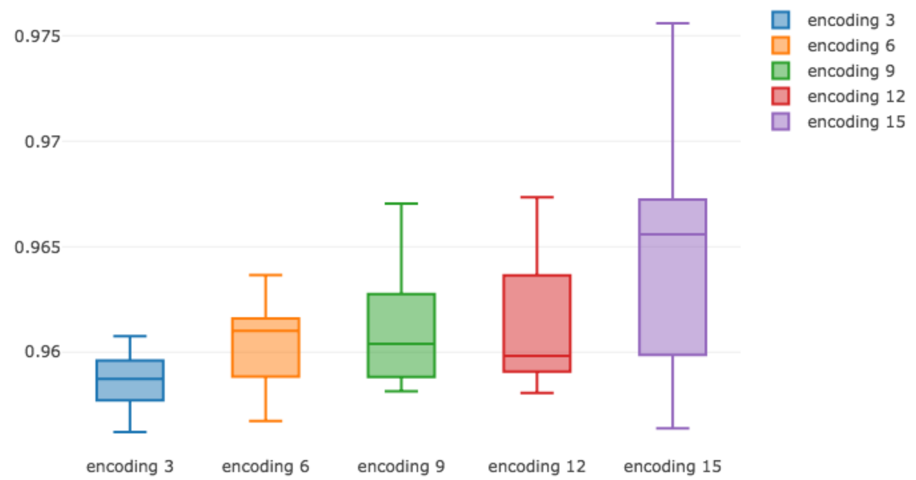

Fig. 6: Boxplot of SAE Best MSE Scores

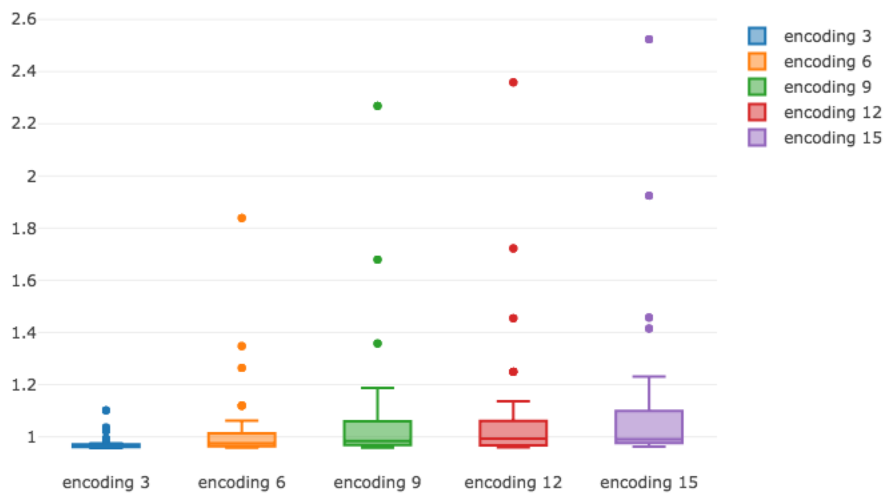

Fig. 7: Boxplot of SAE Last 100 MSE Scores

Layer Size & SGD Learning Rate Results

The same graphs can be seen grouped for the different layer configurations in figures 8, 9 and 10. Looking at the profits generated, the larger node 1 and 2 layer configurations are best, though training seems to start struggling at 3 layers, where the increase in layer size only has poorer effects.

The best MSE scores are largely within similar ranges, though are interestingly bounded in the networks with more, smaller layers, likely as a result of inherent regularization. Once again, the scores of the last 100 epochs show how the larger networks are overfitting and causing general performance to degrade significantly.

The learning rates shown here are that of the SGD training of the network prior to the OGD training, which is the network which will produce the profits. There is likely to be an optimal combination of the two which sufficiently prepares the network for OGD training without overfitting to a less useful minima.

The learning rate effects in figures 11 and 12 show that the lowest learning rate results in significantly better profits, but also generally worse MSE scores. Figures 13 and 14 put all this together, showing the scoring across layers and learning rates for MSE and profits. It would appear that larger nodes of 1 and 2 layers with a smaller learning rate have better capacity to learn effectively - this leads to both the higher profit on the best version networks, as well as the degradation of mean MSE scores as training progresses and the model begins to overfit.

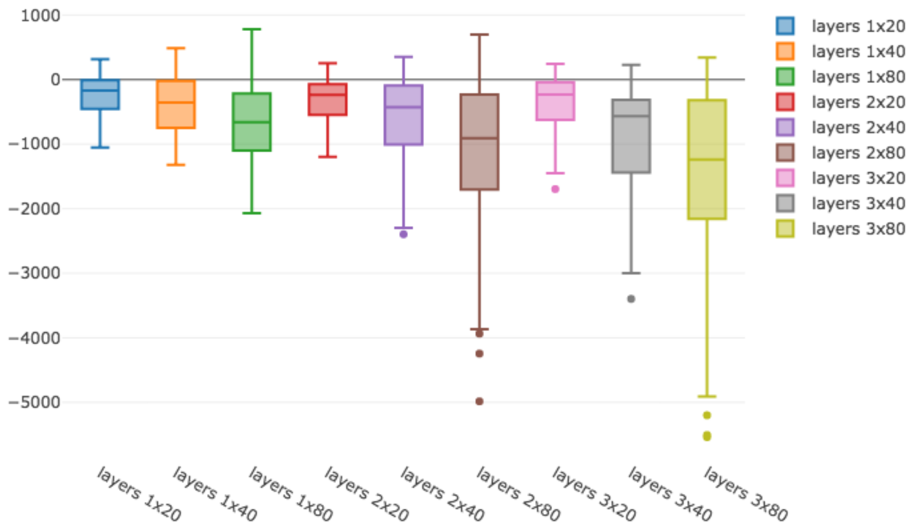

Fig. 8: Boxplot of Layers Profits Generated

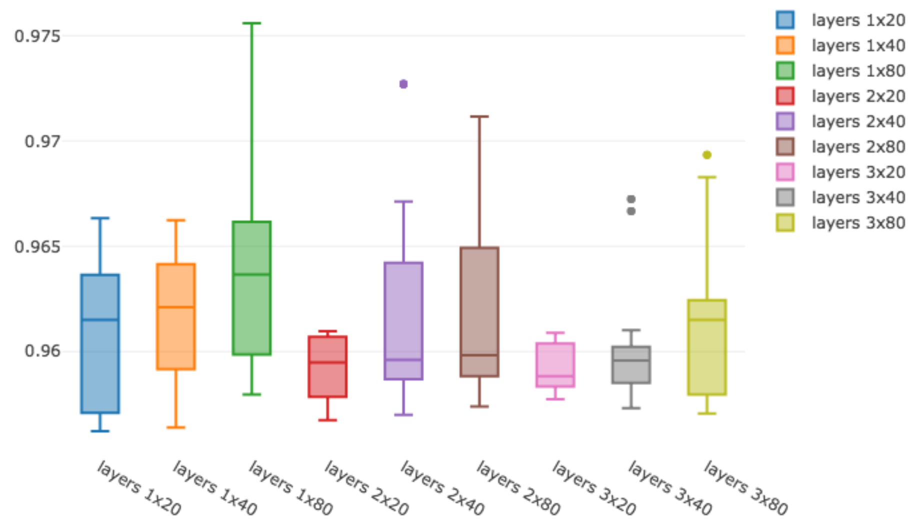

Fig. 9: Boxplot of Layers Best MSE Scores

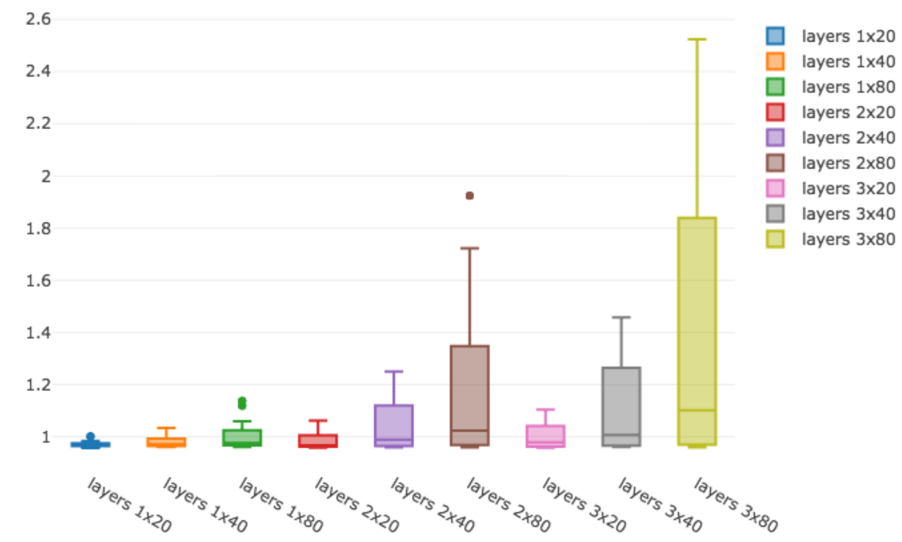

Fig. 10: Boxplot of Layers Last 100 MSE Scores

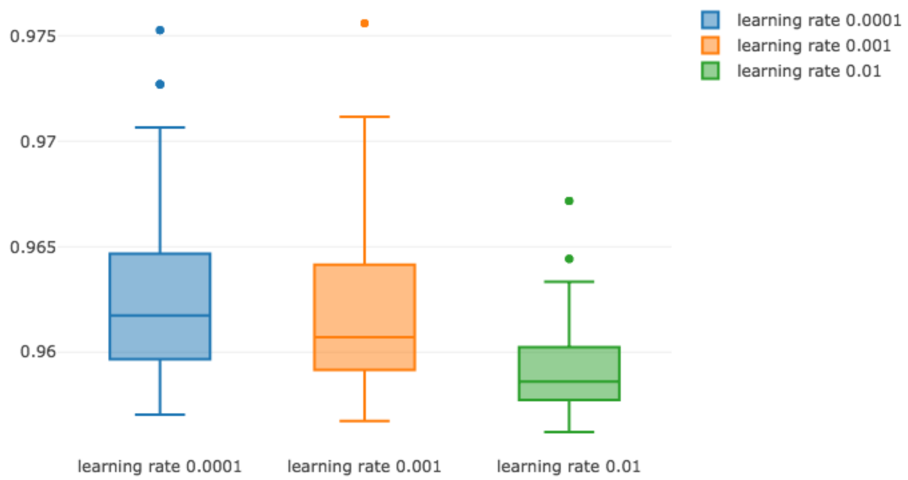

Fig. 11: Learning Rates MSE

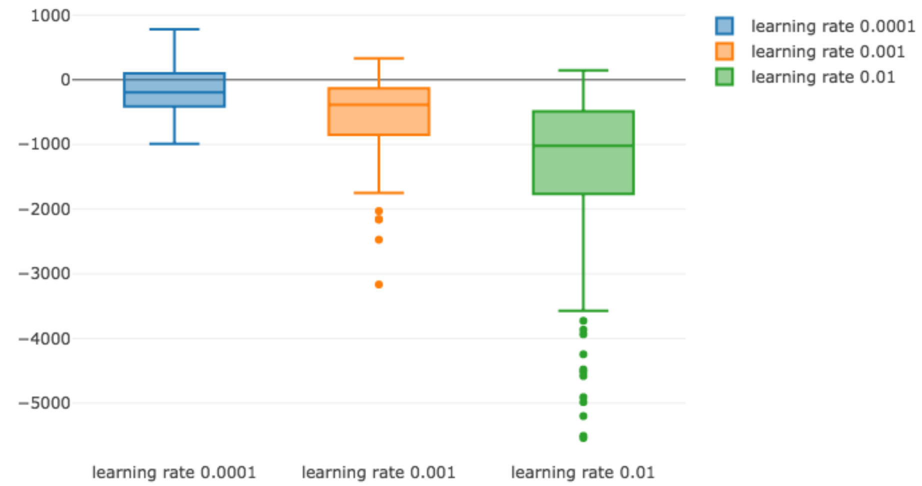

Fig. 12: Learning Rates Profit

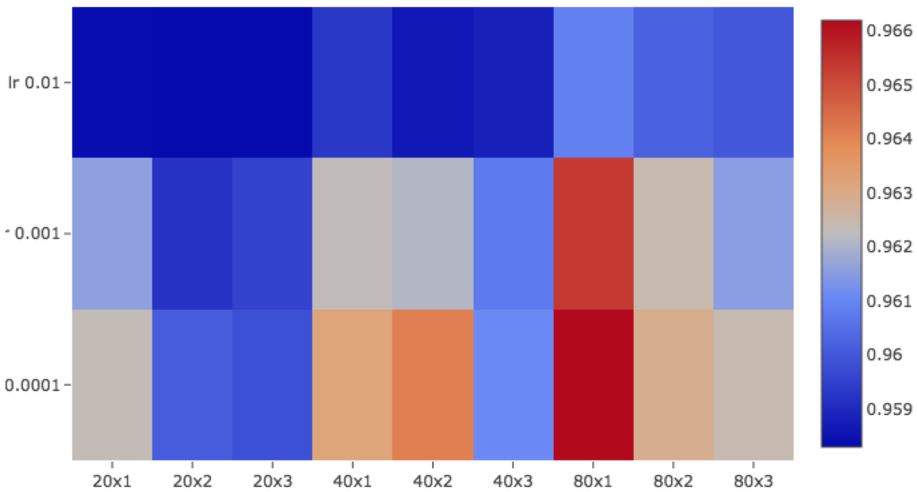

Fig. 13: Heatmap of Layer and Learning Rates Mean Minimum MSE

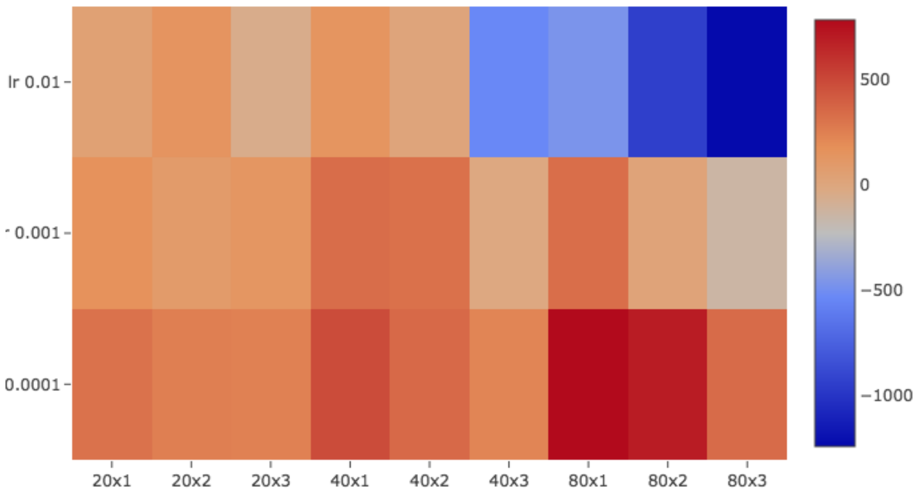

Fig. 14: Heatmap of Layer and Learning Rates Maximum Profit

OGD Learning Rate Results

OGD learning rates produce the best profits at 0.001, though also seem be oscillating more at that point, producing larger MSE values (0.01 produced exloding gradients and computational faults). This seems to make sense, 40% of the dataset was dedicated to OGD learning, and so higher learning values producing better networks would be appropriate. Network's that are in a poorer starting configuration may end up with larger errors as a result, leading to the MSE scores seen. A better look at the breakdown of MSE results in the 0.01 group would be warranted upon iteration.

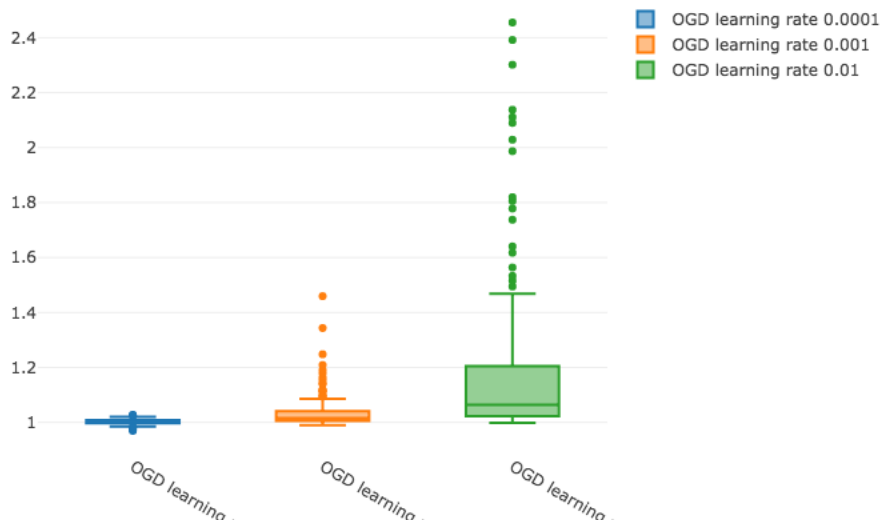

Fig. 15: OGD Learning Rates MSE

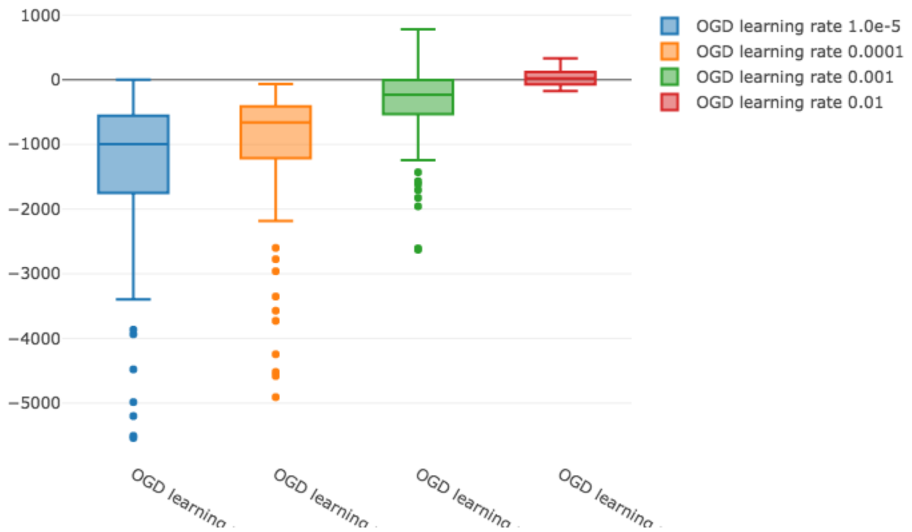

Fig. 16: OGD Learning Rates Profit

Prediction Plots

The model with the highest profit is, unsurprisingly, also the model with the lowest OGD training MSE costs. The model with the lowest SGD MSE score did not respond as well to the OGD training, and the two are compared in the graphs below, showing the actual and predicted changes in stock prices. The difference highlights the importance of the OGD phase and selecting a model that will respond to both that and the SGD training.

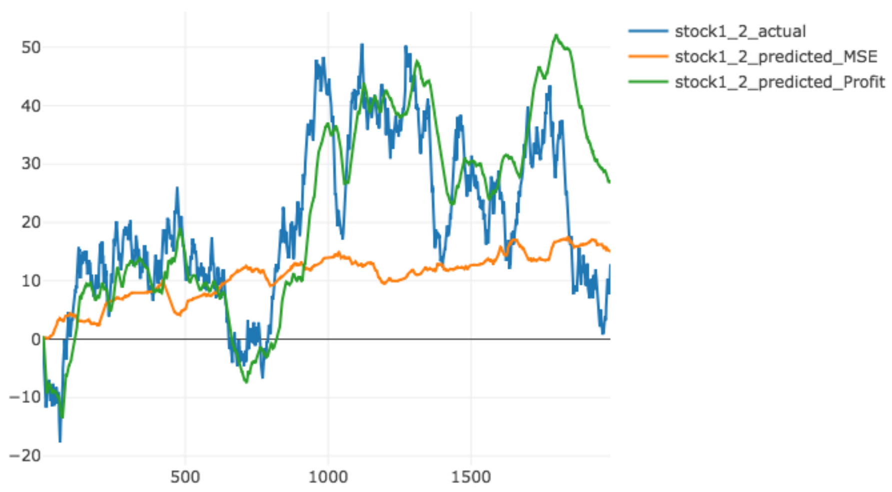

Fig. 17: Stock 1 Recreations

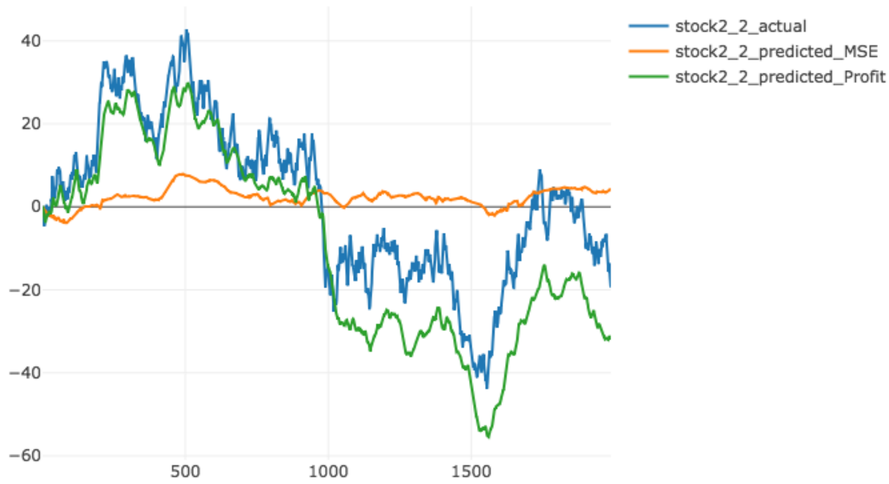

Fig. 18: Stock 2 Recreations

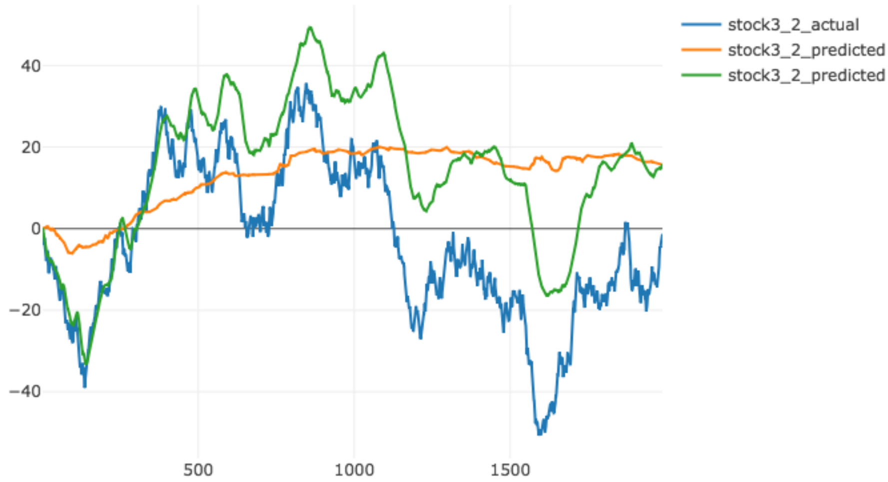

Fig. 19: Stock 3 Recreations

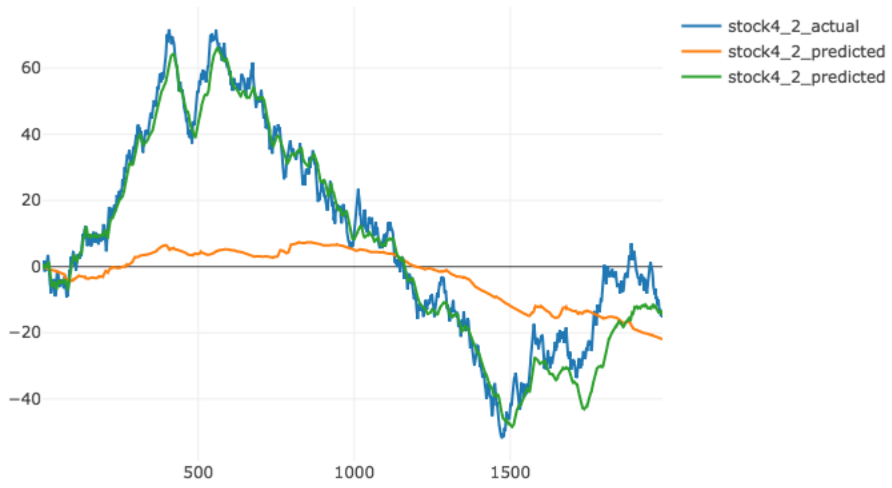

Fig. 20: Stock 4 Recreations

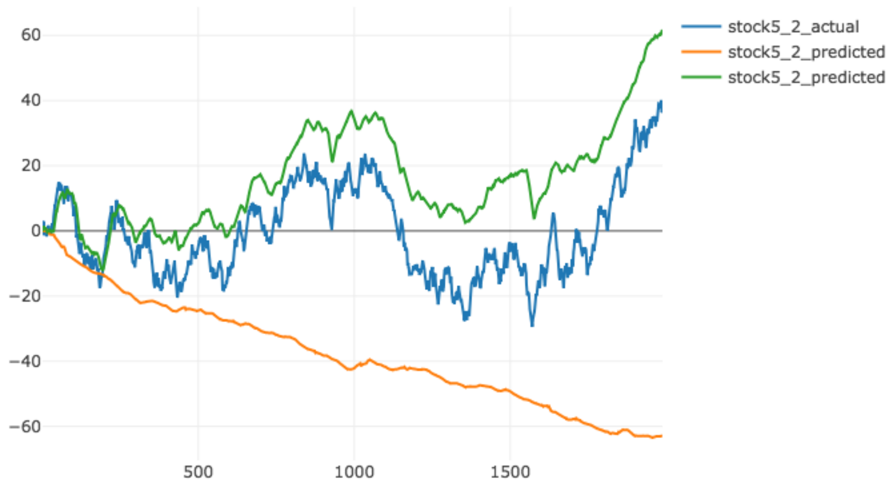

Fig. 21: Stock 5 Recreations

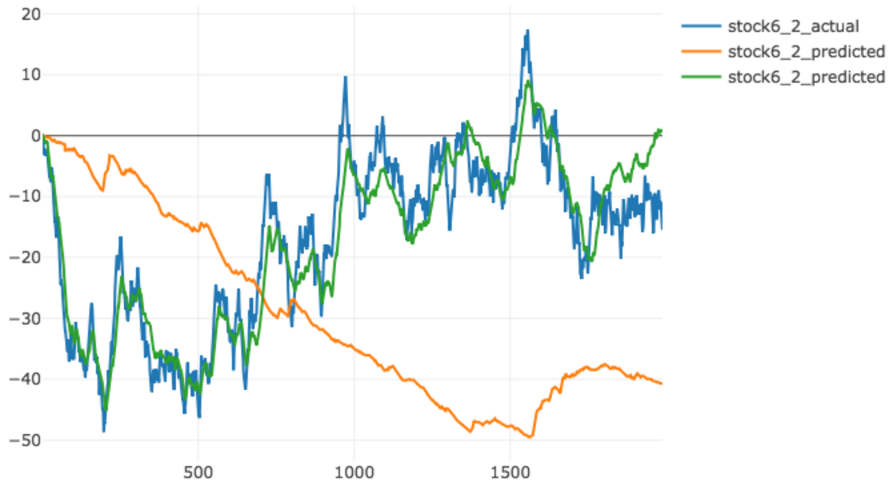

Fig. 22: Stock 6 Recreations

CSCV Plots

While not really necessary for this iteration, the CSCV method was run on all the configurations which were used to generate returns for synthetic data. The resulting logit distribution is displayed in 23. This results in a PBO figure of 0%.

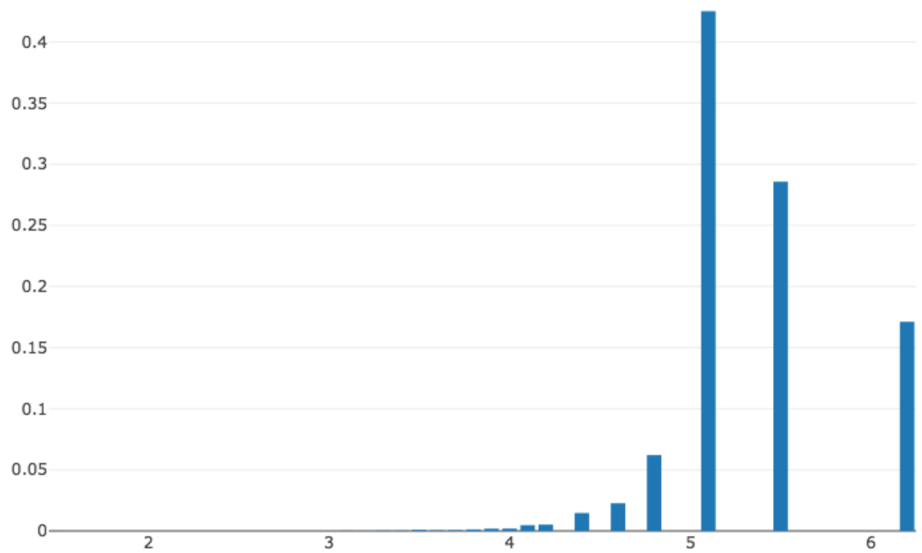

Fig. 23: CSCV Logit Distribution

5.3 Results for Real Data

6 Conclusions

7 References

[1] Albers S. Online Algorithms: a survey. Mathematics Subject Classification (1991): 68W25, 68W40. Available at: https://link.springer.com/content/pdf/10.1007

[2] Aparicio, Diego and Lopez de Prado, Marcos, How Hard Is It to Pick the Right Model? (December 2017). Available at SSRN: https://ssrn.com/abstract=3044740 or http://dx.doi.org/10.2139/ssrn.3044740

[3] B. Arthur (1995) Complexity in Economics and Financial Markets, Complexity 1 (1), 20???25.

[4] Bailey, David H. and Borwein, Jonathan and Lopez de Prado, Marcos and Zhu, Qiji Jim, The Probability of Backtest Overfitting (February 27, 2015). Journal of Computational Finance (Risk Journals), 2015, Forthcoming. Available at SSRN: https://ssrn.com/abstract=2326253 or http://dx.doi.org/10.2139/ssrn.2326253

[5] Bailey, David H. and Borwein, Jonathan and Lopez de Prado, Marcos and Zhu, Qiji Jim, Pseudo-Mathematics and Financial Charlatanism: The Effects of Backtest Overfitting on Out-of-Sample Performance (April 1, 2014). Notices of the American Mathematical Society, 61(5), May 2014, pp.458-471. Available at SSRN: https://ssrn.com/abstract=2308659 or http://dx.doi.org/10.2139/ssrn.2308659

[6] Bailey, David H. and Lopez de Prado, Marcos, The Deflated Sharpe Ratio: Correcting for Selection Bias, Backtest Overfitting and Non-Normality (July 31, 2014). Journal of Portfolio Management, 40 (5), pp. 94-107. 2014 (40th Anniversary Special Issue).. Available at SSRN: https://ssrn.com/abstract=2460551 or http://dx.doi.org/10.2139/ssrn.2460551

[7] G. Bakiri and T. G. Dietterich. Achieving high-accuracy text-to-speech with machine learning. In R. I. Damper, editor, Data Mining Techniques in Speech Synthesis. Chapman and Hall, New York, NY, 2002. 22

[8] Bao W, Yue J, Rao Y (2017) A deep learning framework for financial time series using stacked autoencoders and long-short term memory. PLoS ONE 12(7): e0180944. https://doi.org/10.1371/journal.pone.0180944

[9] Bartlett P., Hazan E. Adaptive Online Gradient Descent. Available at: http://papers.nips.cc/paper/3319-adaptive-online-gradient-descent.pdf

[10] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In Bernhard Scholkopf, John Platt, and Thomas Hoffman, editors, Advances in Neural Information Processing Systems 19 (NIPS06), pages 153160. MIT Press, 2007. Available at: http://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.pdf

[11] Bengio, Yoshua, and Samy Bengio. "Modeling high-dimensional discrete data with multi-layer neural networks." Advances in Neural Information Processing Systems. 2000. Available at: http://papers.nips.cc/paper/1679-modeling-high-dimensional-discrete-data-with-multi-layer-neural-networks.pdf

[12] Bengio, Yoshua, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives." IEEE transactions on pattern analysis and machine intelligence 35.8 (2013): 1798-1828. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6472238&tag=1

[13] Bottou L., Le Cun Y. Large Scale Online Learning. Available at: http://papers.nips.cc/paper/2365-large-scale-online-learning.pdf

[14] Bottou, L. and Murata, N. (2002). Stochastic Approximations and Efficient Learning. In Arbib, M. A., editor, The Handbook of Brain Theory and Neural Networks, Second edition,. The MIT Press, Cambridge,

[15] Cirean, Dan Claudiu, et al. "Deep, big, simple neural nets for handwritten digit recognition." Neural computation 22.12 (2010): 3207-3220. Available at: `https://www.mitpressjournals.org/doi/abs/10.1162/NECO_a_00052`

[16] Chu C., Time series segmentation: A sliding window approach. Information Sciences, Volume 85, Issues 13, July 1995, Pages 147-173. Available at: `ttps://www.sciencedirect.com/science/article/pii/002002559500021G#!`

[17] F. L. Chung, T. C. Fu, R. Luk, and V. Ng, Flexible time series pattern matching based on perceptually important points, in Proc. Int. Joint Conf. Artificial Intelligence Workshop Learning from Temporal and Spatial Data in International Joint Conference on Artificial Intelligence (IJCAI'01), Seattle, Washington, USA, 4-10 August, 2001, p. 1-7. Available at: http://hdl.handle.net/10397/48578

[18] J. P. Crutchfield, (2011), Between order and chaos, Nature Physics, Vol. 8, January 2012, 17-23

[19] Dauphin, Y. et al. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In Proc. Advances in Neural Information Processing Systems 27 29332941 (2014) Available at: http://papers.nips.cc/paper/5486-identifying-and-attacking-the-saddle-point-problem-in-high-dimensional-non-convex-optimization

[20] Aditya Devarakonda, Maxim Naumov & Michael Garland. ADABATCH: ADAPTIVE BATCH SIZES FOR TRAINING DEEP NEURAL NETWORKS. Available at: https://arxiv.org/pdf/1712.02029.pdf

[21] David L. Donoho, High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality (August 8, 2000). Available at: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.329.3392&rep=rep1&type=pdf`

[22] John Duchi, Elad Hazan and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research 12 (2011) 2121-2159. Available at: http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

[23] Erghan D., Bengio Y., Courville A, Manzagol P., Vincent P. Why Does Unsupervised Pre-training Help Deep Learning?, Journal of Machine Learning Research 11 (2010) 625-660 . Available at: http://www.jmlr.org/papers/volume11/erhan10a/erhan10a.pdf

[24] E.F. Fama, The behavior of stock-market prices, J. Bus., 1 (1965), pp. 34-105. Available at: https://doi.org/10.1086/294743

[25] Jianqing Fan, Runze Li, Statistical Challenges with High Dimensionality: Feature Selection in Knowledge Discovery (7 Feb, 2006). Available at: https://arxiv.org/abs/math/0602133

[26] Fan, J., & Fan, Y. (2008). High Dimensional Classification Using Features Annealed Independence Rules. Annals of Statistics, 36(6), 26052637. http://doi.org/10.1214/07-AOS504

[27] Ge, Rong, et al. "Escaping from saddle pointsonline stochastic gradient for tensor decomposition." Conference on Learning Theory. 2015. Available at: http://proceedings.mlr.press/v40/Ge15.pdf

[28] Goodfellow, Ian J., et al. "Maxout networks." arXiv preprint arXiv:1302.4389 (2013). Available at: http://proceedings.mlr.press/v28/goodfellow13.pdf

[29] Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." Proceedings of the thirteenth international conference on artificial intelligence and statistics. 2010. Available at: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

[30] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. 2011. Available at: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

[31] Griffioen, Gerwin A. W., Technical Analysis in Financial Markets (March 3, 2003).

[32] Hansen, Peter Reinhard and Lunde, Asger and Nason, James M., The Model Confidence Set (March 18, 2010). Available at SSRN: https://ssrn.com/abstract=522382 or http://dx.doi.org/10.2139/ssrn.522382

[33] Harvey, Campbell R. and Liu, Yan, Backtesting (July 28, 2015). Available at SSRN: https://ssrn.com/abstract=2345489 or http://dx.doi.org/10.2139/ssrn.2345489

[34] Hawkins, Douglas. (2004). The Problem of Overfitting. Journal of chemical information and computer sciences. 44. 1-12. 10.1021/ci0342472.

[35] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. Neural Computation, 18:15271554, 2006. Available at: https://www.mitpressjournals.org/doi/abs/10.1162/neco.2006.18.7.1527

[36] G. E. Hinton, R. R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks. Science 28 Jul 2006: Vol. 313, Issue 5786, pp. 504-507 DOI: 10.1126/science.1127647. Available at: http://science.sciencemag.org/content/313/5786/504/tab-pdf

[37] Geoffrey E. Hinton, Training Products of Experts by Minimizing Contrastive Divergence,Neural Computation Volume 14 — Issue 8 — August 2002 p.1771-1800 . Available at: https://www.mitpressjournals.org/doi/pdf/10.1162/089976602760128018

[38] Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580 (2012). Available at: https://arxiv.org/pdf/1207.0580.pdf

[39] Campbell R. Harvey & Yan Liu & Heqing Zhu, 2016. " and the Cross-Section of Expected Returns," Review of Financial Studies, vol 29(1), pages 5-68.

[40] Sepp Hochreiter, Jrgen Schmidhuber. Long Short-Term Memory. Neural Computation Volume 9 — Issue 8 — November 15, 1997 p.1735-1780. Available at https://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735.

[41] K. Hornik, Multilayer feed-forward networks are universal approximators, Neural Networks, vol 2, 1989

[42] Hsu D. Time Series Compression Based on Adaptive Piecewise Recurrent Autoencoder (17 August 2017). Available at: https://arxiv.org/pdf/1707.07961.pdf

[43] Ivakhnenko, A. G. (1971). Polynomial theory of complex systems. IEEE Transactions on Systems, Man and Cybernetics, (4), 364378. Available at: https://ieeexplore.ieee.org/document/4308320/

[44] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012. Available at: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

[45] Ioannidis JPA (2005) Why Most Published Research Findings Are False. PLoS Med 2(8): e124. https://doi.org/10.1371/journal.pmed.0020124

[46] N. Johnson, G. Zhao, E. Hunsader, H. Qi, N. Johnson, J. Meng & Brian Tivnan (2013). Abrupt rise of new machine ecology beyond human response time. Sceintific Reports 3(2627). DOI: 10.1038/srep02627

[47] Kahn, Michael N. . Technical Analysis Plain and Simple: Charting the Markets in Your Language, Financial Times Press, Upper Saddle River, New Jersey, p. 9. ISBN 0-13-134597-4.(2006)

[48] Stefan Knerr, Handwritten Digit Recognition by Neural Networks with Single-Layer Training. 962 IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 3, NO. 6, NOVEMBER 1992. Available at: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=165597`

[49] John Langford, Lihong Li and Tong Zhang. Sparse Online Learning via Truncated Gradient. Journal of Machine Learning Research 10 (2009) 777-801 Submitted 6/08; Revised 11/08; Published 3/09. Available at: http://www.jmlr.org/papers/volume10/langford09a/langford09a.pdf

[50] Martin Lngkvist, Lars Karlsson, Amy Loutfi, A review of unsupervised feature learning and deep learning for time-series modeling, Applied Autonomous Sensor Systems, School of Science and Technology, rebro University, SE-701 82 rebro, Sweden. Available at: `https://www.sciencedirect.com/science/article/pii/S0167865514000221#bi005`

[51] LeCun Y., Bottou L, Orr G, Muller K. Efficient Backprop. Available at: http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

[52] LeCun, Y. (1988). A theoretical framework for back-propagation. In D. Touretzky, G. Hinton, & T. Sejnowski (Eds.), Proceedings of the 1988 connectionist models summer school (pp. 2128). `Availableat:https://www.researchgate.net/profile/Yann_Lecun/publication/2360531_A_Theoretical_Framework_for_Back-Propagation/links/0deec519dfa297eac1000000/A-Theoretical-Framework-for-Back-Propagation.pdf`

[53] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., et al. (1989). Backpropagation applied to handwritten zip code recognition. Neural Computation, 1(4), 541551. Available at: http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf

[54] Yann LeCun, Yoshua Bengio & Geoffrey Hinton. Deep learning, Nature volume 521, pages 436444 (28 May 2015). Available at: https://www.nature.com/articles/nature14539

[55] Le Roux, Nicolas, and Yoshua Bengio. "Representational power of restricted Boltzmann machines and deep belief networks." Neural computation 20.6 (2008): 1631-1649. Available at: http://www.iro.umontreal.ca/ lisa/publications2/index.php/attachments/single/22

[56] Liu X., Lin Z., Wang H. Novel Online Methods for Time Series Segmentation. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 20, NO. 12, DECEMBER 2008. Available at: `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4445667&tag=1`

[57] Lo, Andrew W., The Statistics of Sharpe Ratios. Financial Analysts Journal, Vol. 58, No. 4, July/August 2002. Available at SSRN: https://ssrn.com/abstract=377260

[58] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic Flow Prediction With Big Data: A Deep Learning Approach. IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, VOL. 16, NO. 2, APRIL 2015. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6894591&tag=1

[59] Mahajan D., Keerthi S., Sundararajan S., Bottou L. A Parallel SGD method with Strong Convergence. Available at: https://arxiv.org/pdf/1311.0636.pdf

[60] McLean, R. David and Pontiff, Jeffrey, Does Academic Research Destroy Stock Return Predictability? (January 7, 2015). Journal of Finance, Forthcoming. Available at SSRN: https://ssrn.com/abstract=2156623 or http://dx.doi.org/10.2139/ssrn.2156623

[61] M Minsky, SA Papert, L Bottou (1988). Perceptrons: An introduction to computational geometry. Available at: https://books.google.co.za/books?hl=en&lr=&id=PLQ5DwAAQBAJ&oi=fnd&pg=PR5&dq=Minsky+Papert+1969+%E2%80%94+Perceptrons&ots=zyEDwMvmXX&sig=aVXF7DBJKAOxWO66S4UCyzhNARw#v=onepage&q=Minsky%20Papert%201969%20%E2%80%94%20Perceptrons&f=false

[62] Murphy, John J. Technical analysis of the financial markets: A comprehensive guide to trading methods and applications. Penguin, 1999.

[63] Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." International Conference on Machine Learning. 2013. Available at: http://proceedings.mlr.press/v28/pascanu13.pdf

[64] Lopez de Prado, Marcos, The Future of Empirical Finance (May 31, 2015). Journal of Portfolio Management, 41(4). Summer 2015. Forthcoming.. Available at SSRN: https://ssrn.com/abstract=2609734 or http://dx.doi.org/10.2139/ssrn.2609734

[65] Povey D., Zhang X., Khudanpur S. Parallel training of Deep Neural Networks with Natural Gradient and Parameter Averaging. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.745.6995&rep=rep1&type=pdf

[66] MarcAurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In B. Scholkopf, J. Platt, and T. Hoffman, editors, Advances in Neural Information Processing Systems 19 (NIPS06), pages 11371144. MIT Press, 2007. Aavilable at: http://papers.nips.cc/paper/3112-efficient-learning-of-sparse-representations-with-an-energy-based-model.pdf

[67] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, & J. L. McClelland (Eds.), Parallel distributed processing, vol. 1 (pp. 318362). MIT Press.

[68] Robert L. Schaefer (2012) Subset Selection in Regression, Technometrics, 34:2, 229, DOI: 10.1080/00401706.1992.10484917

[69] J Schmidhuber (2015). Deep learning in neural networks: An overview. Neural Networks Volume 61, January 2015, Pages 85-117. Available at https://www.sciencedirect.com/science/article/pii/S0893608014002135.

[70] Schorfheide, Frank, and Kenneth I. Wolpin. 2012. "On the Use of Holdout Samples for Model Selection." American Economic Review, 102 (3): 477-81.

[71] Jack D. Schwager. Getting Started in Technical Analysis, Page 2 (1999)

[72] Sermanet, Pierre, et al. "Pedestrian detection with unsupervised multi-stage feature learning." Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. IEEE, 2013. Available at: https://arxiv.org/abs/1212.0142

[73] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML-07), 2007.

[74] Siegelmann, H. (1992). Theoretical foundations of recurrent neural networks (Ph.D. thesis), New Brunswick Rutgers, The State of New Jersey: Rutgers. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.38.920&rep=rep1&type=pdf

[75] Skabar, Cloete, Networks, Financial Trading and the Efficient Markets Hypothesis (http://crpit.com/confpapers/CRPITV4Skabar.pdf)

[76] Takeuchi L, Lee Y. Applying Deep Learning to Enhance Momentum Trading Strategies in Stocks. Technical Report, 2013. Available at: http://www.smallake.kr/wp-content/uploads/2017/04/TakeuchiLee-ApplyingDeepLearningToEnhanceMomentumTradingStrategiesInStocks.pdf

[77] Troiano L., Mejuto E., Kriplani P.On Feature Reduction using Deep Learning for Trend Prediction in Finance (11 Apr 2017). Available at: https://arxiv.org/abs/1704.03205.

[78] Tseng P., AN INCREMENTAL GRADIENT(-PROJECTION) METHOD WITH MOMENTUM TERM AND ADAPTIVE STEPSIZE RULE. SIAM J. OPTIM. Vol. 8, No. 2, pp. 506531, May 1998. Available at: https://pdfs.semanticscholar.org/1a29/6a1577478654a54a9f801f93f71b7d853c53.pdf

[79] Vincent P., Larochell H., Lajoie I., Bengio Y., Manzagol P., Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. Journal of Machine Learning Research 11 (2010) 3371-3408. Available at: http://www.jmlr.org/papers/volume11/vincent10a/vincent10a.pdf

[80] Wan Y., Gong X., Si Y. Effect of segmentation on financial time series pattern matching. Applied Soft Computing Volume 38, January 2016, Pages 346-359. Available at: https://www.sciencedirect.com/science/article/pii/S1568494615006341

[81] Linnan Wang, Yi Yang, Renqiang Min, Srimat Chakradhar. Accelerating deep neural network training with inconsistent stochastic gradient descent. Neural Networks Volume 93, September 2017, Pages 219-229. Available at: https://www.sciencedirect.com/science/article/pii/S0893608017301399

[82] Wang, Sida, and Christopher Manning. "Fast dropout training." international conference on machine learning. 2013. Available at: http://proceedings.mlr.press/v28/wang13a.pdf

[83] Van Den Oord, Aaron, et al. "Wavenet: A generative model for raw audio." arXiv preprint arXiv:1609.03499 (2016). Available at: https://arxiv.org/abs/1609.03499

[84] Weiss, S. M, & Kulikowski, C. A. (1991). Computer systems that learn : classification and prediction methods from statistics, neural nets, machine learning, and expert systems. San Mateo (Calif.): Kaufmann.

[85] Werbos, P. J. (1974). Beyond regression: new tools for prediction and analysis in the behavioral sciences (Ph.D. thesis), Harvard University. Available at: https://www.researchgate.net/publication/244947191_1974_Beyond_regression_New_tools_for_predicting_and_analysis_in_the_behavioral_sciences

[86] Paul J. Werbos. Applications of advances in nonlinear sensitivity analysis. System Modeling and Optimization pp 762-770. Available at: https://link.springer.com/chapter/10.1007/BFb0006203

[87] Wu, Huaiqin. "Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions." Information Sciences 179.19 (2009): 3432-3441. Available at: https://ac.els-cdn.com/S0020025509002539/1-s2.0-S0020025509002539-main.pdf?_tid=fc9d5d20-c0fe-4107-947c-6a4d9e472aab&acdnat=1526319168_bc401e01e4ab496b50b8d568dcafb0b3

[88] Yin J., Si Y., Gong Z. Financial Time Series Segmentation Based On Turning Points. Proceedings of 2011 International Conference on System Science and Engineering, Macau, China - June 2011. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5961935

[89] Zeiler M. ADADELTA: An Adaptive Learning Rate Method. Available at: https://arxiv.org/abs/1212.5701

[90] Zinkevich M, Weimer M, Smola A, Li L. Parallelized Stochastic Gradient Descent. Available at: http://papers.nips.cc/paper/4006-parallelized-stochastic-gradient-descent.pdf

[91] Zhao Y., Li J., Yu L. A deep learning ensemble approach for crude oil price forecasting. Available at: https://doi.org/10.1016/j.eneco.2017.05.023

[92] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the Twenty-First International Conference on Machine Learning (ICML-04), pages 919926, 2004

[93] Zhou B., Hu J. A Dynamic Pattern Recognition Approach Based on Neural Network for Stock Time-Series. 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC 2009). Available at:https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5393674