Imperial College London

Investigating the predictive ability of LSTMs on industrial metal prices

Oliver Boom ,

ob3618@ic.ac.uk

https://github.com/OliverJBoom

Applied Computational Science and Engineering Master's Thesis (ACSE9)

September 17, 2019

Supervisors:

Dr. Tristan Fletcher (External) Dr. Stephen Neethling (Internal)

Undertaken in partnership with ChAI

Address 116 Old Street, EC1V 9BG, London

Abstract

Foresight is a collection of tools built to forecast the future price movements of industrial metals, using Long Short-Term Memory networks (LSTMs). It was developed to investigate the predictive ability of LSTMs in the auto-regressive, multivariate and multi-task cases, predicting the price of both individual metals and a complex of metals. The mean squared error, mean absolute error and mean directional accuracy of the predictions were compared against ARIMA and naive benchmarks across 4 different forecast lengths; a week, a month, a quarter and half a year into the future. It was found that LSTMs with a multi-task learning formulation outperformed both benchmarks and single-task LSTMs when considering all metrics and forecast lengths. This led to the acceptance of the hypothesis that, due to the theory of co-movement within commodities prices, industrial metal price prediction would benefit from a multi-task framework. When progressing from an auto-regressive to multivariate framework, the multi-task learning frameworks saw improvement in performance, while the single-task framework saw a reduction in performance. It was found that multi-task LSTMs achieved superior performance, particularly at longer forecast lengths, with multivariate, multi-task LSTMs achieving performance that was independent of forecast length.

Keywords: Long Short-Term Memory Networks, Time Series Forecasting, Commodity Prices, Multi-Task Learning, Deep Learning

Nomenclature

AR	Auto-Regressive
$ARIMA$	Auto-Regressive Integrate Moving Average
HPC	High Performance Computer
IM	Industrial Metals
$LSTM$	Long Short-Term Memory (networks)
MAE	Mean Absolute Error

MDA	Mean Directional Accuracy
$MIMO$	Multi-Input Multi-Output
$MISO$	Multi-Input Single-Output
MSE	Mean Squared Error
MTL	Multi Task Learning
MV	Multivariate
PCA	Principal Component Analysis
RNN	Recurrent Neural Network
$SISO$	Single-Input Single-Output

1 Motivation and significance

The market price of commodities is information of importance to stakeholders at all points in the supply chain. It is this information that allows stakeholders to determine if raw materials can be economically exploited and allows them to strategically road-map for future operations. Price forecasting allows planning for future market conditions and adjustment of strategy to either curtail losses in adverse conditions or capitalize in favourable markets[1]. Industrial Metals (IMs) are a subset of the commodity sector. They are resources of paramount importance across economies of all stages of development and are the raw materials that unpin vast sectors. Their price fluctuations have been shown to have significant repercussions on macroeconomic performance[2][3]. Alongside industries heavily influenced by IM prices such as manufacturing, mining, etc, there is considerable effort spent on price forecasting by economic speculators[4].

Financial forecasting is challenging due to the high volatility and non-linearity of the underlying instruments and the extensive quantity of parameters both observed and hidden[5]. It is unknown whether the challenges in modelling arise from the technical challenge of modelling non-linearity or whether the driving factors are exogenously based and therefore cannot be captured by the models.

The purpose of this investigation was to compare the predictive performance of a state-of-the-art deep learning approach; Long Short-Term Memory (LSTM) networks on industrial metals, across several different regression frameworks. LSTMs have proven successful in cases where huge datasets and extensive computational power is available[6][7], however, these resources are often unavailable. This investigation sought to assess the performance of LSTMs in the case of relative data and computational scarcity. The time frame the data was taken from was briefer than 50% of the duration of the briefest datasets observed in other state-of-the-art cases and much smaller than that in most[8][9][6]. The primary hypothesis for this investigation was that high capacity models are required to tackle the complex problem of econometric time series prediction. It was believed that LSTMs should outperform other conventional computational methods in predicting IM prices. To test this hypothesis the predictions made by LSTM regression were compared against a naive solution and the popular ARIMA time series model.

There is extensive economic theory arguing that IMs exhibit excessive co-movement[10][2]. It was speculated that if this is true then there will be a common factor driving the Industrial Metals complex as a whole as well as idiosyncratic factors that drive individual metals. It was the second hypothesis of this investigation that, given the excessive co-movement literature, predictive performance should improve if using a Multi-Task Learning framework. A multi-task learning framework involves the optimisation of a model to predict a complex of metals simultaneously, even if only the price of one metal is of interest. It was the rationale of this investigation that given the relative data scarcity, a Multi-Task Learning framework would augment the available data and increase generalization. Closely tied to both these hypotheses was the third hypothesis; that increasing the amount of data included in the dataset and the complexity of the regression frameworks should result in improved performance. The focus of this hypothesis was a case where it is not possible to increase the temporal width of the dataset. Instead, the additional data must come from including additional (relevant) time series in the dataset from the same period.

To investigate these hypotheses Foresight was developed; a collection of tools to preprocess data and perform deep learning on a complex of Industrial Metals. The aim was to predict the price of copper across

different forecast lengths; 1 week, 1 month, 1 quarter and half a year, both in an auto-regressive, multivariate, multi-task auto-regressive and multi-task multivariate cases as will be detailed in Section 3.1. In addition to comparing regression frameworks, the effect of including feature spawning as a preprocessing step was also investigated as is discussed in Section 3.3. While being built for Industrial Metals prediction Foresight can be applied to any regression problem for the frameworks detailed, and allows for training and tuning neural networks at scale with its parallelised framework.

2 Literature Review

The barriers to entry for accurate price prediction have decreased over the years. Historically speaking it was only economic experts and analysts highly skilled in statistics who were capable of making reasonably accurate forecasts. Several primary challenges make price forecasting challenging. Commodity prices, as with many financial time series, exhibit highly non-linear, volatile behavior. In addition to this, there are significant exogenous factors that might be driving the price. How, for example, can models predict sudden changes in news or policy or they are only tracking financial time series? Even if these exogenous factors can be identified and included within the dataset, and models that can accurately capture the non-linearity are used, there is the very real possibility that major economic markets are efficient. If so, the underlying asset already fully reflects all the available information within the public domain, and it is impossible to forecast prices using technical analysis[11].

An extensive range of methodologies is available to researchers wishing to forecast prices. The most prevalent techniques can be categorized into; qualitative, econometric, stochastic, time series and machine learning based[1][12]. The most popular computational forecasting techniques are time series modelling and machine learning based modelling[1]. Time series modelling involves decomposing the signal of a time series and using the internal structure of a signal to forecast future trends[13]. Of the time series models developed, the Auto-Regressive Integrated Moving Average model (ARIMA) is one of the most competent, popular and flexible[14]. It is frequently used as a sophisticated benchmark, and was used as a benchmark for the single-task case in this work.

The ability of machine learning techniques to model dynamic, non-linear relationships means that there is significant potential for this methodology to outperform other techniques[12][9]. Deep learning; a subset of machine learning achieves best-in-class results in problems ranging from image classification to sentiment analysis[15][16]. Within deep learning Recurrent Neural Networks (RNNs) have been proven to perform extremely well on sequential data[17]. And a specially engineered case of RNN called Long Short-Term Memory networks or "LSTMs" can succeed in predictive tasks that require long term memory, a task previously inaccessible to RNNs[18].

LSTMs have application within speech recognition and text classification amongst other areas and their application to econometric time series is certainly a prominent area of research[19][20]. However, it is still a relatively new application with V. Ahti stating in 2009 that feed-forward neural networks are the "neural architecture of choice in time series econometrics"[21]. While LSTMs are frequently declared as being ideal models for financial time series prediction in theory, there is actually a relatively small amount of peer reviewed literature investigating their performance. The vast majority of literature declaring LSTMs applicable within finance appears on online data science publication such as Medium and Towards Data Science. These articles rarely appear with strong benchmarks, and the literature is not peer-reviewed. As G.Lemus noted, despite great success in other areas of sequential prediction, there is a relative lack of successful examples within financial time series[7]. This is a common problem with financially centred machine learning research, as it is often the case that performant models and frameworks are kept private to protect potential profit generation[22].

There has been published success of LSTM performance in predicting financial time series by Siami et al. who compared the predictive ability of LSTMs on stock prices against ARIMA. They discovered that LSTMs achieved an 84 - 87 percent reduction in error rates versus ARIMA[8]. Their data spanning a significantly longer period (1985-2018) and was sampled at a monthly frequency. LSTMs were used to predict directional

movement and daily returns prediction in Fisher et als. work, with the LSTMs outperforming random forest and logistic regression benchmarks[23]. In this case, their dataset was all S&P 500 stock prices between 1992 to 2015. Alongside these works, another paper that achieves successful out-of-sample performance is Sirignano and Cont's work "Universal features of price formation in financial markets"[6]. However, they had access to a truly comprehensive dataset with billions of transactions, across thousands of stocks for training data. They also trained the models on an incredibly powerful computer cluster comprising of 500 GPU nodes. There has been no literature on the performance of LSTMs on relatively small datasets (both in the number of values and the length of the time frame that it is across).

A major challenge in prediction is effective benchmarking. Quantifying how performant a model actually is, is a relatively difficult task. There is considerable economic literature that supports the view that economic markets exhibit at the very least a semi-strong form of market efficiency, with others claiming full market efficiency [11][24]. The more efficient markets are the more they behave like random walks[25]. By definition, it is impossible to predict a random walk therefore if the efficient market hypothesis holds then performant models achieve better results by chance. If a model follows a random walk then an optimal prediction is a naive forecast[26]. This involves taking the price for today as the predicted value for the next forecast length. I.e $y_{t+f} = y_t$ where y is the price, t is the time step and f is the forecast length. Conversations with quantitative analysts in hedge funds have confirmed that a naive forecast is considered a reasonably difficult benchmark to outperform. It was decided to use both an ARIMA forecast and a Naive forecast as the two benchmarking predictions.

2.1 Multi-Task Learning

There is considerable evidence that certain areas of the commodities sector exhibit significant co-movement[10] of prices. This is a large area of economic research with the observation that a "common factor" exists for the commodities universe[2]. There is a shared common factor both within the overall commodities universe and within the closer related sub-universes i.e precious metals, industrial metals, etc. And there has been research into which financial variables are correlated with the common factor to a statistically significant degree[2]. In predicting a singular metal e.g copper, there will be components of the prediction which are shared with the other IMs such as market risk-on/risk-off attitudes, US dollar effective exchange rate, global supply/demand dynamics, etc. But there will also be an idiosyncratic component of the price prediction which is unique to copper[27]. There is potential to exploit this shared information between these IMs using Multi-Task Learning.

Multi-Task Learning (MTL) is a machine learning framework which aims to leverage shared information between a collection of related tasks to improve the predictive performance in all tasks[28][29]. MTL is more a paradigm than a specific technique. A series of tasks are trained together simultaneously with the tasks being optimized as a collective. The objective here was to have a shared dataset of features used for training, and the prices of several metals as target variables. With each metal price being an individual task, these tasks could be optimised simultaneously, sharing common information to give a better predictive performance. Section 4 discusses the dataset in-depth but one of the main challenges of this investigation was a relatively small dataset. MTL is a technique that increases generalization of models, and it was believed that its use would decrease over-fitting.

There are different approaches to the parameter sharing of deep neural networks when using a multi-task learning approach. It was the choice of this investigation to focus on a framework where all of the parameters are shared between the individual tasks. In this formulation, there are no task-specific layers of the network. However, there are alternative frameworks for parameter sharing, which are explored in Ruder's paper "An Overview of Multi-Task Learning in Deep Neural Networks"[30]. One of the alternatives to total hard parameter sharing can be shown in Figure 1. Here there are shared layers between all tasks and then layers which are individual to that task specifically. This approach was not used but had been highlighted as a piece of future work that could improve the capabilities of a multi-task learning framework.

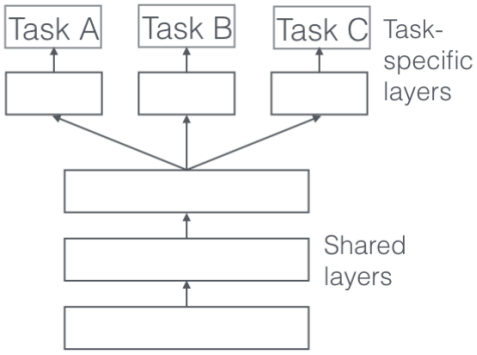


Figure 1: Deep learning multi-task learning framework[30]

2.2 ARIMA

Although ARIMA models are prevalent in economic forecasting there are several limitations. A fundamental assumption of ARIMA is the underlying time series is linear and that it follows a known statistical distribution[31]. It also traditionally requires statistical analysis to fit the correct parameters to the model. However, the recently developed Pmdarima library contains an `auto_arima` function which determines the optimal parameters for the model[32].

A series is said to be stationary if the statistical properties mean and variance does not depend on time[13]. It is an underlying assumption for most time series methods. A series can be defined as weakly stationary if 'the statistical moments of the process up to that order depend only on time differences and not upon the time of occurrences of the data being used to estimate the moments'[31]. The raw prices for IMs are not stationary, but they are weakly stationary meaning that by using differencing they can be transformed into a stationary signal. A major advantage of ARIMA is that it contains a differencing term allowing it to model weakly stationary series by including the amount of differencing required to transform a non-stationary series into a stationary series explicitly.

The ARIMA forecast is made using Equation 1, where x is the variable, t is time, a is the auto-correlation coefficients at different lagged values, η is the error residuals, which is a Gaussian white noises series with zero mean and variance σ_ε^2[8], c is a constant and w are the weights added to the ε term at different lags. A further mathematical treating of the subject can be found in Adhikari's 'An Introductory Study on Time Series Modeling and Forecasting'[31].

$$x(t) = \sum_{i=1}^{p} a_i x_{t-i} + \sum_{i=0}^{q} w_i \varepsilon t - i + c + \varepsilon \tag{1}$$

When specifying an ARIMA model in statistical packages, the model contains 3 parameters[33];

- **Auto-regressive Term (p)** - The auto-regressive term makes a prediction by assuming that the future value is a "linear combination of p past observations and a random error together with a constant term"[31].
- **Differencing Term (d)** - The differencing term is the number of time a series must be differenced to make it stationary.
- **Moving Average Term (q)** - The moving average term uses previous values/errors as "explanatory variables"[31], and its order is the number of lagged error terms to include in the model.

2.3 Long Short-Term Memory networks

Traditional neural networks do not have a natural means to incorporate temporal structure. Recurrent Neural Networks (RNNs) are a neural architecture engineered as a means to include time more naturally

than had been previously attempted using lags or feature engineering. RNNs achieve this by feeding the information from previous time steps into future predictions using hidden states, thus enabling sequential memory. The hidden state is a means of propagating forward information about all previous time steps. When the next item in a sequence is being predicted, the hidden state will contain the memory of previous time steps, their predictions and how accurate those predictions were.

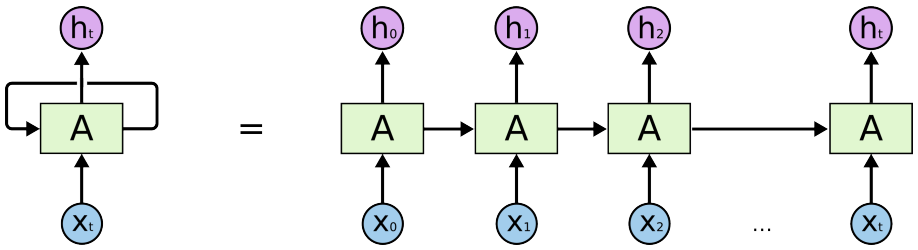


Figure 2: Recurrent Neural Network Architecture[34]

One of the major problems with RNNs is the vanishing gradient problem. The backpropagation algorithm that lies at the heart of deep learning, involves the network's weights updating proportionally to the gradient of the errors with respect to the network's weights[35]. Each layer of the network passes through an activation function/s which squashes the signal, causing the gradient to become smaller. RNNs perform back-propagation through time and each time step in an RNN can be thought of as a successive layer[36]. As the RNN back-propagates further back in time, through the network, the gradients become increasingly smaller due to these activation functions. As back-propagation is how the network's weights are updated i.e how they learn, this means that the updates to earlier time steps weights become diminishingly smaller. This means that pieces of information struggle to propagate far into the future, resulting in no long term memory[37].

LSTMs were designed to address this problem. They replace the individual cells of an RNN network with LSTM cells. These are mathematically sophisticated and have the net effect of allowing the networks to possess long term memory. They do this by having several different gates within the cells where information is copied, transformed, filtered, and can be forgotten or passed onto future hidden layers[18][8]. There are three separate gates within an LSTM cell. The input or memory gate, the forget gate and the output gate. The gates have the following purposes with the cell:

- **Input/Memory Gate**: Determine what new parts of the signal to include in the present cell state.
- **Forget Gate**: Determines what information to forget and what to keep. Information from previous hidden states as well as the current input passes through a sigmoid function filtering out a portion of it.
- **Output Gate**: Determines how much of the total cell state should be outputted from the cell.

The elegant behavior of LSTM cells, when used in an RNN network is that these gates do not have to be configured. The LSTMs learn what information is important, and what can be forgotten via backpropagation. The network tunes itself to remember events in the distant past when it recognizes that they could be important.

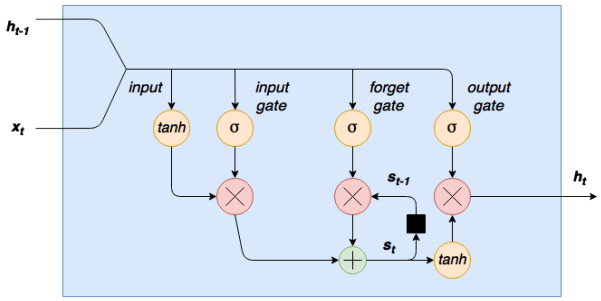


Figure 3: LSTM Cell[38]

3 Methodology

The investigation had several hypotheses that it sought to test. These hypotheses can be formalised as the following:

1. **Hypothesis**: Due to the problem of financial time series prediction being complex and non-linear, high capacity models capable of modelling non-linearity should achieve superior performance when compared to existing time series models and other conventionally used benchmarks.
   Test: Do LSTMs outperform naive and ARIMA benchmark predictions?
2. **Hypothesis**: Due to the theory of common movement within commodity prices, IM price prediction should benefit from an MTL framework.
   Test: Do multi-task predictions outperform single-task frameworks predictions?
3. **Hypothesis**: Increasing the amount of data included in the dataset should result in improved performance.
   Test: Do multivariate predictions outperform auto-regressive predictions in both the single-task and MTL cases?

To investigate these hypotheses numerous experiments were conducted to evaluate the performance of LSTMs against the benchmarks. This section details some of the technical considerations evaluated while formulating these experiments.

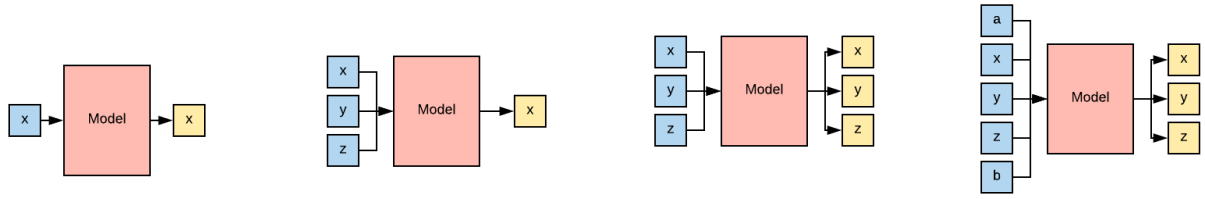
3.1 Regression Frameworks

The first consideration was how to structure the regression framework. Price forecasting can be framed in several different ways. It can take an auto-regressive, univariate or multivariate format. And in addition to this, it can take a single-task or multi-output format. Several of these frameworks come from the Multi-Task Learning literature[28][29]. The following frameworks were compared;

- **Auto-regressive Single-Input Single Output (AR SISO)** - Using a metals past price to predict its future price.
- **Multi-Input Single Output (MISO)** - Using a metals past price alongside other econometric time series to predict its future price. This aims to extract predictive information from other time series. Ideally, the target time series is highly correlated with the other time series but is lagged in time.
- **Auto-regressive Multi-Input Multi-Output (AR MIMO)** - Using multiple metals past prices to predict their future prices, simultaneously. This is not strictly auto-regressive, as information is shared, but to differentiate it from the full multivariate MIMO it will be referred to as the AR framework.
- **Multi-Input Multi-Output (MIMO)** - Using multiple metals past prices, alongside other econometric time series to predict their future prices simultaneously.

Both the AR MIMO and MIMO hoped to benefit from the advantages gained through using a Multi-Task Learning paradigm. These frameworks are visualized in Figure 4. In the blue of the figure are the inputs time series, and the yellow boxes represent the target time series. The exact number of variables shown in

Figure 4 is not accurate (there are significantly more input time series in the dataset and more target series in MTL, but was designed to show overall structure).



(a) AR SISO (b) MISO (c) AR MIMO (d) MIMO

Figure 4: Comparison of Regression Frameworks

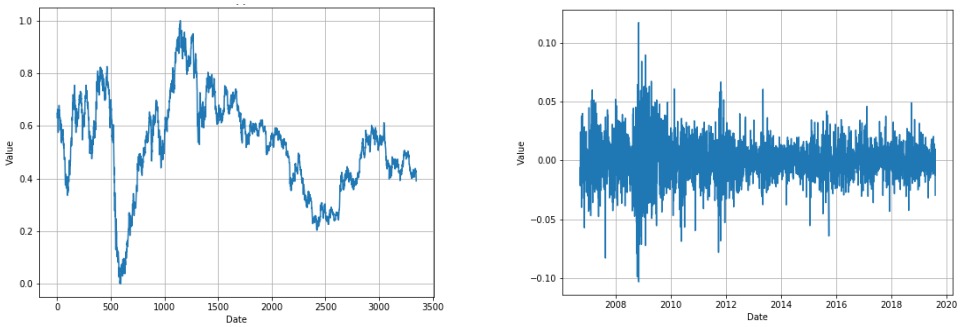
There was good reason to believe that the more driving factors that were included in the dataset the greater the potential predictive power of a dataset[2][1]. The second and third hypotheses of this investigation were that as the models became more complex, and more data was included that predictive performance should increase. If the hypotheses held, then the order of predictive performance should have been; SISO, MISO, AR MIMO and MIMO (with MIMO being the most performant).

3.2 Normalisation

It is always necessary to normalise the input data/targets as LSTMs perform best on correctly scaled data. In the planning phase of developing Foresight it was recognised that there were several approaches available for normalisation. Initially, there was a choice between trying to predict either raw prices, returns, or log returns. Log returns and returns are convenient as they are naturally normalised, while raw prices are not and require a normalisation method. The log return is shown in Equation 2 and involves taking the log of (1 + the return of a series).

$$[H]r_i = log\Big(1 + \frac{p_i}{p_{i-1}}\Big) \tag{2}$$

It was believed that log returns should be the most performant due to several statistically convenient properties that it possesses. Most notably that financial returns are believed to have a log-normal distribution. Machine learning models often perform best on normally distributed data, so the log of the returns if effectively transforming the data to a normal distribution. It was decided to investigate the comparative performance between predictions on scaled raw prices and log returns, as will be discussed in Section 7.



(a) Raw Price of Copper (b) Log Returns of Copper Price

Figure 5: The two target time series

3.3 Feature Spawning

While is always necessary to normalise the input data in machine learning, another consideration was whether additional feature engineering steps should be added to the preprocessing routines.

It was believed that additional information could be mined through feature engineering. The aim was to use feature spawning to extract additional value out of the dataset, and to see if it could improve performance. Feature spawning in this instance meant taking each time series in the dataset and passing it through a series of transformations that are believed to capture slightly different information about the signal in a very direct way. I.e passing the series through a series of filters and including these transformed series in the dataset. Then after normalisation, Principal Component Analysis (PCA) can be used to reduce the dimensionality of the data again. By doing this it was hoped that some additional insight can be captured in the data. The feature spawning transformations selected were:

- **Exponentially weighted moving average filters (EWMAs)**: These filters were used as a means of smoothing the data out. A series of filters with different half-life lengths were chosen. The half-lives used were 5 days, 22 days, 66 days, 132 days and 264 days. These represent 1 week, 1 month, 1 quarter, half a year and a year in business days.
- **Rolling volatility windows**: Rolling volatility was calculated on window sizes of different spans. The spans used were: 1 week, 1 month and 1 quarter.

It was decided to investigate the relative performance of including/not including the feature spawning preprocessing step, as well as including/not including Principal Component Analysis. PCA is a very common preprocessing technique to reduce the time required to train the models when using large quantities of data. There are obviously large amounts of overlap between a signal and spawned signals which are filters of that signal. PCA can improve performance by removing unnecessary data and allowing the models to converge to an optimum more easily. But converse arguments can also be made that it is the small amount of variance removed which could be the crucial pieces of information required to improve performance, and therefore PCA should not be used. An investigation into the effect of combining these different preprocessing steps was undertaken to clarify their applicability with LSTMs.

3.4 Window Size

An important experiment parameter to tune is the length that each time series that is inputted into the LSTM for each prediction should be. While the networks do tune themselves to learn the important features, giving the LSTMs input series that are too long is the equivalent of inputting useless information that will cause them to waste computational resources filtering the signal. "How far back is relevant to remember for future predictions?" was the question to investigate. Depending on the forecast length in the future, the size of the time window required changes. Instinctively it was assumed that for shorter predictions a shorter time series window would be required. The rationale for this is that it is unnecessary to remember years of data to make next day predictions. But using a window that is too short results in losing the benefits gained through using LSTM models as short input sequences would not require long term memory. The window sizes investigated were chosen qualitatively by considering the desired forecast length in the future and choosing a window of memory that would seem relevant for it.

It was anticipated that the further out the forecast length the less accurate the predictions would be. But additionally shorter forecast lengths are more susceptible to influence from noisy data. Hence why a next day prediction was not included in the investigation. Due to the challenges of predictions further into the future, it was believed that these might be the forecast lengths that would be easiest to outperform the benchmarks on, due to the relative simplicity of the benchmark methods. Naive is a very strong benchmark over short forecast lengths, but become increasingly weak for longer predictions.

Forecast Length (days)	Window Size (days)
5	44, 66, 132, 264
22	44, 66, 132, 264
66	66, 132, 264
132	66, 132, 264

Table 1: Window sizes investigated

3.5 Evaluation Metrics

To evaluate the performance of the time series in each experiment several metrics were used. Two measure of regressional accuracy were used and one measure of directional accuracy was used. The equations for the metrics are shown in Equations 3a, 3b, and 3c with Y being the vector of observed values and $\hat{Y}$ being the vector of predicted values. The reasons behind the selection of each of these different evaluation metrics are as follows:

1. **Mean Squared Error (MSE)**: Mean squared error has the advantages of additionally penalizing higher error due to the distance from the error being squared. This punishes poor performance on outliers heavily.
2. **Mean Absolute Error (MAE)**: Mean absolute error is very similar to MSE but is more robust to outliers due to the distance from the errors being absolute. Models that might be generally performant but that have a few really bad predictions can result in a high MSE but a reasonable MAE. Both are important.
3. **Mean Directional Accuracy (MDA)**: There are a lot of use cases in commodity pricing where the actual end price accurately is not that important, but correctly ascertaining which direction the price moves is important. Mean directional error is a means of measuring this. If a model completely mimics the movement of a price, but it the predictions are consistently offset, both MSE and MAE would register the model as a poor, while MDA would show that there is use in it.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{3a}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i| \tag{3b}$$

$$MDA = \frac{1}{n}\sum_{i=1}^{n}\Big(sign(Y_i - Y_{i-1}) == sign(\hat{Y}_i - \hat{Y}_{i-1})\Big) \tag{3c}$$

4 Dataset

Through a partnership with ChAI financial time series dating back to 2006 were acquired. Through a literature review and discussions with researchers at ChAI, financial time series which were believed to have the predictive ability on IM instruments of choice were selected to include in the training dataset. Statistical checks were performed by ChAI analysts to ensure that each time series was statically significant prior to inclusion in the dataset. The checks that were performed were; linear correlation, rank correlation and mutual information. And if any of those criteria achieved above a threshold value between the training series and target series, then it was included in the dataset.

The financial data available is of high quality, with most instruments being sampled daily and the time frame the data is drawn frame is relatively short. Deep learning thrives off of a large quantity of data, but there are certain financial instruments where this data has simply not been recorded for an extended period of time or procuring it is prohibitively expensive. There are certainly numerous exogenous factors that will be driving prices of IMs both as a collective and idiosyncratically, that could not be included in the dataset due to unavailability of data. A significant part of this investigation was determining if enough of the driving factors were contained within the dataset to capture enough percentage of the price variance, to allow for accurate forecasting.

It is worth noting that even if many of the key driving factors are contained within the dataset it was unclear whether there was enough data to train LSTMs and not over-fit to the data. It was deemed certainly probably that there might not be enough data within the dataset to allow for a well generalised LSTM predictor. With the greater issue being a lack of a wide temporal range, more than a lack of total data-points. Economic cycles can last in the order of a decade. As the data does not span multiple economic cycles and

contained one of the largest economic shocks in recent history; the 2008 financial crash, this presented a significant modelling challenge. Given the extraordinary events and a shorter dataset these events may have an unfairly dominant effect on the models learning.

For different regression frameworks, different time series were incorporated into the training dataset. For single-task the target was copper, and for multi-output the collection of metals being predicted were; copper, aluminium, tin, lead, nickel, although all the metrics were evaluated on copper price alone. All of these targets were the spot price on the London Metals Exchange. The training dataset for those target variables are as follows:

Framework	**Target**	**Training Dataset**
SISO	LME Cu price	LME Cu price
MISO	LME Cu price	LME Cu price, SHFE Cu price, COMEX Cu price, COMEX Cu stocks, Chillean Peso spot, Peruvian Sol spot, TED spread, Baltic Dry Index, CBOE Volatility Index, SKEW Index, Goldman Sachs Commodity Index
AR MIMO	LME Cu price, LME Al price, LME Sn price, LME Pb price, LME Ni price	LME Cu price, LME Al price, LME Sn price, LME Pb price, LME Ni price
MIMO	LME Cu price, LME Al price, LME Sn price, LME Pb price, LME Ni price	LME Cu price, SHFE Cu price, COMEX Cu price, COMEX Cu stocks, Chillean Peso spot, Peruvian Sol spot, TED spread, Baltic Dry Index, CBOE Volatility Index, SKEW Index, Goldman Sachs Commodity Index, LME Al price, LME Sn price, LME Pb price, LME Ni price, SHFE Al price, COMEX Al price, COMEX Al stocks, LME Al stocks, Chinese Yuan spot

Table 2: Datasets for each framework

Only weekdays were included and the data was truncated to the shortest time series included in the training dataset. Due to only weekdays being included when calculating the forecast length, predicting a week is 5 business days, a month is 22 business days, a quarter 66 and half a year is 132 business days. So when referring to a weeks prediction the value being predicted is $\hat{Y}_{t+5}$.

5 Software

5.1 Methodology

To assess the hypotheses detailed in Section 3 a collection of tools were built in Python 3. These tools collectively form Foresight. Foresight was built as a standalone piece of software using the major libraries listed in Table 7. The implementation of LSTMs was built using the popular deep learning library Pytorch. Initially, Keras was selected due to it's higher level of abstraction, allowing greater focus on the optimization of the problem and less time spent on the implementation. However, it was realised that Keras does not support fully deterministic mathematical operations, meaning that results would not be reproducible. Due to this factor and the greater control given by it's lower level abstraction, Pytorch was selected as the deep learning library of choice. The full list of requirements for Foresight can be found in Table 3.

There were two main separate phases of development; the software development phase, and the data science phase. And different development philosophies were taken for each. The software development phase was the phase required to develop the tools, infrastructure, work-flow and flow of information required for the investigation. While the data science phase involved experimenting with different feature selection/engineering methods, changes to the preprocessing approach and the tuning of the neural networks for each regression framework. The focus of the software development phase was to create an infrastructure on which to conduct data science. And the focus of the data science phase was the pursuit of better results. Importantly, the

ID	Requirement Detail	Implementation Detail
001	Software must load data into a centralised Pandas DataFrame	`universe_select`, `universe_dict` and `generate_dataset` provide this functionality
002	Software must clean the data of outliers	`clean_dict_gen` and `clean_data` provide this functionality
003	Software must ensure that there are no missing values in the final dataset	`generate_dataset` provides this functionality using forward fill imputing
004	Software must provide feature spawning capability	`features_spawn` provides this functionality
005	Software must have an implementation for an LSTM network	`LSTM`, `LSTM_shallow`, `LSTM_deeper` classes provide this functionality
006	Software must have a means to train/validate/test neural networks	`DeepLearning` class provide this functionality
007	The software should include early stopping functionality for deep learning training	`EarlyStopping` provide this functionality
008	Software must have a means to convenient visualise results	`live_pred_plot` and plotting examples
009	Software must be able to create a naive prediction as a benchmark	Examples in Metals Forecaster notebooks
010	Software must be able to create a prediction using ARIMA as a benchmark	ARIMA notebook provides this functionality
011	Software must be able to evaluate the predictions; MSE, MAE and MDA against other benchmarks	`evaluate` & `mean_directional_accuracy` provides this functionality
012	Software must allow for the training/validation/testing to be parallelised across multiple GPUs	Trained the models on multiple GPUs on the cloud
013	Software must allow for the saving on the models	`model_save` provides this functionality
014	Software must allow for the loading of the models	`model_load` provides this functionality
015	Software must allow for the saving of the deep-learning objects	Can pickle the `DeepLearning` and save
016	Software must allow for the loading of the deep-learning objects	Can pickle the DeepLearning and load
017	Software must be well documented	Sphinx documentation style and autodoc generator was used along with Read the Docs
018	Version control use must be incorporated in the development of the software	Github was used for version control
019	Continuous integration best practices must be used	Travis was used for CI
020	The software must be easily installable	Package hosted on Pypi and can be installed using `pip install ForesightPy`
021	The software must be able to produce deterministic results	Pytorch was used instead of Keras to allow this. The `set_seeds` function also aids this functionality

Table 3: Software Requirements

data science phase could not begin until the software development had completed a full iteration.

The waterfall development approach was taken for the software development phase. The waterfall development approach involves linear, sequential phases of planning. The reason for this approach was that there wasn't a large amount of time for the project, and the focus of the project was to investigate whether results could be obtained that would outperform the benchmarks. It was anticipated that the data science phase would require a lot of time for experiments. Therefore the focus was to complete the software development phase as quickly as possible, as the software provided the tools required to perform data science, which was the core focus.

The aim of the software development was to create a minimal viable product that met all of the requirements needed for the investigation. Trying to use an agile methodology made no sense for this context. Sprints in the agile methodology are typically 2-4 weeks. The project fell into the 3 months of summer, with

the first month being reserved for research and planning and the third month was set aside to improve results, tidy up documentation and write the report. This left the second month to build the software. Even if using an agile methodology, there was only time for one or two sprints. The aim was to sprint to the data science phase, but not to iterate the software phase until results for all regression frameworks had been obtained.

The data science phase had a different development approach. It had an Agile approach, where the aim was to obtain results for each regression framework and iterate to improve them as quickly as possible. All of the elements in the data science phase involve iterative changes to parameters and significant experimentation. Each experiment made to investigate a hyper-parameter requires full training of the model.

In the tuning of the hyper-parameters two major assumptions were made:

1. **The optimal hyper-parameters hold across regression frameworks**: Each of the 4 regression frameworks had an optimal set of hyper-parameters for each of the 4 different forecast length. As that would require 16 different parameter tuning cases the strong assumption was made that the hyper-parameters that performed well in the simpler frameworks would hold for the more complex frameworks. This was an incorrect assumption but was deduced to be appropriate in practice. This is as it was seen as preferable to tune the parameters to the simpler case because a simpler network is less likely to over-fit, and has the additional advantage of shorter training time. And when full hyper-parameter searches were conducted across all the forecast lengths, it was the hyper-parameters that performed best on average that were selected. After all the experiments had done, an investigation was undertaken into how reasonable this assumption was, by performing optimal tuning on a couple of cases and seeing how far they deviated from the parameters chosen. It was found that actually the best performing hyper-parameter did generally hold across frameworks and it was an appropriate assumption.
2. **For each new hyper-parameter tuned, prior optimal hyper-parameters hold**: There were a very large number of hyper-parameters and a grid search throughout the hyper-parameter design space would be extremely consuming both in time and computational power. This is as a full grid search has polynomial computational complexity (in regards to the number of different hyper-parameters being tuned). To address this, instead of performing a grid search through the design space, a depth-first decision tree search was conducted. This assumes that the optimal hyper-parameters lie on a hyper-plane where all of the previous optimal hyper-parameter selected hold. This is not the case but was a necessary assumption to avoid grid search.

5.1.1 Development Tools

Github was used as the platform for project management and version control. Due to the modular nature of the regression frameworks, separate branches were rarely needed because separate notebooks could be used to run experiments for the different frameworks. The issue tracker was used as a project management tool to plan feature enhancement, track bugs and note documentation requirements.

Continuous integration was employed using Travis CI, during the project. Much of the conventional CI use was minimised due to the project being primarily focused on data science. Therefore most of the tests were high-level checks making sure that new code didn't break the code-base. One of the challenges was that the environment that was being used to train the models was not the same as the environment that Travis uses for testing. The models were trained on multiple GPUs within a parallelised framework, and Pytorch serialization procedures mean that you can't load models or the pickled `DeepLearning` objects (that will be discussed in Section 6.2) unless they are being loaded into the same environment that they were initialized in. Ie Travis could not be used to load/test models that were run on the parallelised framework. So it was necessary to configure CPU compatible examples to enable for continuous integration and Travis was not used to validate the results.

To aid with documentation, Sphinx documentation software was utilized. To capitalize on the auto-documentation features the Sphinx docstring documentation style was followed throughout the project. As well as being a clear documentation style it also allowed for the generation of PDF and HTML compatible documentation. Then the Restructured Text Directives were used to give a more insightful documentation

information.

To augment the Sphinx generated documentation further, it was integrated with Read The Docs; a software documentation hosting platform. This allows for seamless integration of hosted documentation with code. Read the docs did have some challenges regarding package installation. It builds the documentation in Docker containers and requires that dependencies used in Foresight be installed using a requirements file in the Docker container. However, due to limited computational resources, it is unable to install memory-intensive dependencies such as Pytorch. To fix this, it was necessary to mock the Pytorch package. This allows Pytorch objects to be included in the hosted auto-documentation.

5.1.2 Computational Environment

Initially, it was known that eventually the models would run for an extended period of time. To capitalize on the computational advantages provided by highly vectorised tensor operations on Pytorch it was known that the models would be trained on GPUs. The project began with python notebooks which were designed to be compatible with Google Colabs notebooks, which provides free GPU access. However, it was quickly realised that Colabs does not have excellent integration with Github. Additionally, Colabs was not designed for extended computational runs and has issues disconnecting from runtime. Following this, it was decided to develop the notebooks locally, where testing and configuring the environment was simple. But also the goal of eventual migration to a parallelised framework on the cloud was kept at the forefront of development considerations.

When the software development phase was completed, the training/tuning of models was conducted on Google Clouds distributed computational network. The optimisation of the networks was parallelised across available GPUs, with each training/validation/test batch being split across the available processors. Experiments were conducted across 2 GPUs in the cloud. More GPUs could have been used, but it would have been unnecessarily expensive to add more GPUs when model training time was not very long. If the models had proven to take a very long time to train then it was planned to migrate the training to Imperials HPC, which had 16 GPUs available. However, the disadvantage of the HPC is that the visualization functionality that had been developed, such as the live plotting of results and predictions using `live_pred_plot`, would be unavailable. It was viewed as being more important to view results and make configuration tweaks on the fly, over submitting jobs to the HPC, which could run faster but had a slower feedback mechanism.

Initially, all of the financial time series available to ChAI were located in a PostGre SQL database. The amount and length of the time series were not so large as to have memory issues when pulling down the data, and all the data points could be held in memory easily. If this were not the case then it would be necessary rethink the order of some of the preprocessing, and conduct it in batches. However, as the datasets were tailored to each framework it was more convenient to save the CSVs into a folder for loading in each use case.

5.2 Algorithms

This section will describe some of the key algorithms used in Foresight. The training series need to be formatted in the correct manner to input into the LSTMs. Algorithm 1 details the process of taking the normalised time series and converting them into training and target datasets, and is the same for both the auto-regressive and multi-variate procedures.

Algorithm 1 Data Chunking Algorithm

P: Normalised time series — shape: [full length of time series, num of instruments]
F: Forecast length
L: Length of desired input series
$data\_X \leftarrow []$ — The destination for training dataset
$data\_y \leftarrow []$ — The destination for target dataset
procedure
    $y \leftarrow P[F:,:]$ — The target series is offset by the forecast length

$X \leftarrow P[:-F,:]$ The training series needs to be truncated to match the target series
for i in $L : len(P)$ **do**
$data\_X[i] \leftarrow X[i-L:i,:]$
$data\_y[i] \leftarrow y[i]$
end for
return $data\_X, data\_y$ shape: $data\_X$ [num windows, length of each series, num of instruments]
shape: $data\_y$ [num windows, num instruments]
end procedure

The algorithm used for LSTM is too lengthy to include here, and can be best understood by investigating the source code for the `DeepLearning` class. The algorithm used for ARIMA benchmarking is detailed in 2.

Algorithm 2 Auto-regressive ARIMA Training

F: Forecast Length
P: The auto-regressive parameter
D: The differencing parameter
Q: The moving average parameter
X: The time series
procedure FORECAST($hist, P, D, Q, L$)
$model \leftarrow ARIMA(hist, order = (P, D, Q))$
$model \leftarrow model.fit$
$pred \leftarrow model.forecast(F)[F-1]$ Takes the forecast at F into the future
return $pred$
end procedure
procedure ARIMA TRAIN
$train \leftarrow X[:0.6]$
$test \leftarrow X[0.6:]$
$predictions \leftarrow []$
for i in $len(test) - F + 1$ **do**
$pred \leftarrow Forecast(train, P, D, Q, L)$
$predictions.append(pred)$
$train.append(test[i])$
end for
return $predictions$
end procedure

6 Software Implementation

6.1 Capabilities

The package Foresight and the example notebooks allow the user to input one or more time series and to determine the relationship between the given time series and one or more target time series. The case problem this was built for was predicting the future values for given financial time series, but that does not have to be the use case. Fundamentally Foresight provides a means to model the relationships between time series, where historical data is available, and it is most applicable in a predictive context. It is capable of modelling non-linear and volatile time series and has the advantage over most time series methods that the target series does not need to be stationary. The Software Requirements Specification in table 3 details the full capabilities of the software.

The advantage of Foresight is that it makes access to LSTMs straight-forwards. The data can be prepared and used to train the models very easily and with relatively little understanding of the underlying theory or mathematics, as they have been abstracted away. Due to the focus on developing Foresight to be compatible with several different regression frameworks, it is also very flexible and capable of working on a variety of time series problems. Foresight was developed to investigate the performance of LSTMs on industrial metals prices. And, as this goal has been achieved, it was fit-for-purpose and delivered all on the requirements.

6.2 Architecture

The overall work-flow of Foresight can be shown in Figure 6 with the blue boxes in Figure 6 shows the outputs of the experiments. Foresight allowed the conducting of a variety of experiments, using different training/target datasets, regression frameworks and preprocessing steps to try and achieve better performance. By creating abstraction over these steps, conducting experiments in a systematic manner was straight-forwards. This allows the user to focus on the challenges associated with data science and price prediction instead of the challenges of managing a coherent code-base. If the PCA or Feature Spawn processing steps were not taken the flow of information is the same, just those transformation don't take place.

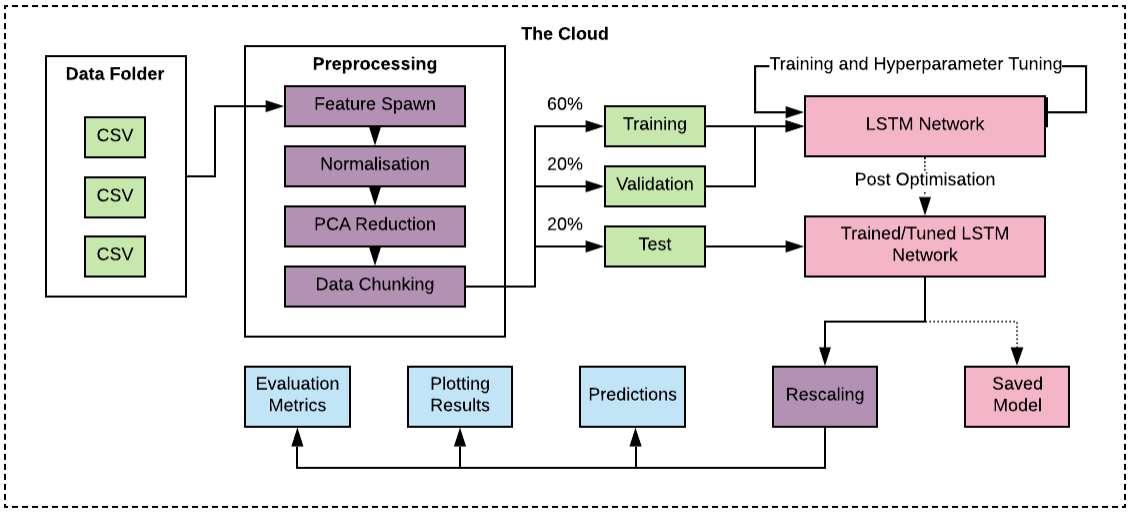


Figure 6: Process Flow for LSTM Predictions

The main reason for deciding to run models on machines hosted on Google cloud instead of the HPC was the ability to change model runs on the fly. A live plotting of predictions for the training/validation datasets allowed the tracking of model performance. If the model was not learning at a sufficient rate or appeared to require tweaking then changes could be made and the model re-initialized quickly. This saved a lot of time, by making the models' performance more transparent, and showing more information than just printing the loss metrics. In the case of the MTL, the training/validation predictions for the complex of metals are shown, as well as the predictions for the individual metal. Training can be stopped and resumed, meaning that the user can choose to evaluate the models on test results, re-scale the data before deciding if further training is required.

The main class that handles the training/validation of the LSTM models is `DeepLearning`. These objects were saved as `pickle` files for every experiment, and these files contain everything required to make predictions, investigate the data used, re-scale the data or investigate the model hyper-parameters, making for convenient re-visiting experiments. The only prerequisite as mentioned in Section 5.1.1 is that the objects must be loaded into the same computational framework as they were initialized in, due to the serialization back-end of Pytorch.

6.3 LSTM Architecture

There are several main hyper-parameters that determine the architecture of an LSTM network, which require tuning to optimise the network to the given problem. Different combinations of neurons in the hidden state, in the dense layer, as well as the number of dense layers was investigated. An example of an auto-regressive LSTM architecture can be seen in Figure 7. In the Figure the input layer takes in a number

of feature input series into the input layer (the first blue boxes), they are processed through the recurrent layers in the LSTM layer. The arrow above the LSTM layer indicates that it is the rolled-up visualisation of the LSTM layer. The unrolled visualisation would look similar to Figure 2, except for multiple inputs. The signal then passes through the dense layer and through to the final output layer where a singular prediction is made for each metal. The optimal hyper-parameter configuration can be found in Section 7.

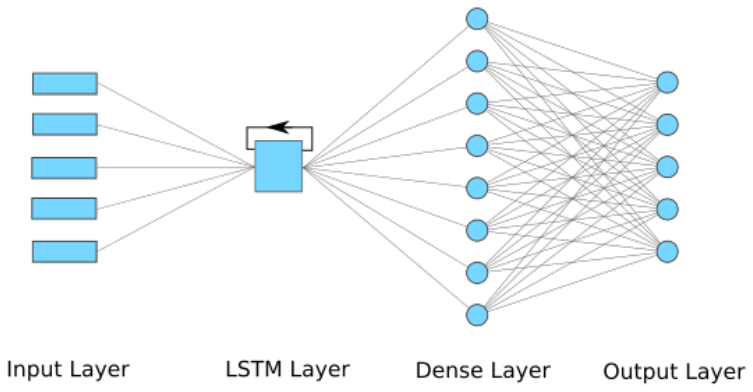


Figure 7: Neural Network Architecture

For each metal that was included in the MTL list of targets there was a corresponding neuron in the output layer that was associated with that metal/task. Each task is the prediction of a separate metal but the tasks share a common architecture. It is only the final output neuron that is not shared with the other tasks. An interesting formulation for future work could be including further layers for each task that are not shared with the other tasks. This would better help isolate idiosyncratic components from the common factor of the signals.

6.4 Verification and Validation

To verify the code-base several different methods were incorporated into the development work-flow. Using Jupyter Notebooks as the developing environment allowed the inspection and modification of variables, functions, and classes quickly. Jupyter notebooks do not provide static analysis, but as running cells is quick and easy, this static analysis isn't necessary as the functionality of the software is constantly being tested and inspected throughout the development process. When new functions had been prototyped in Jupyter Notebooks they were included in into python files using the Pycharm IDE, which does provide static analysis of code. Pycharm's provides code inspection functionality, and after using that to ensure consistent style the python files were run through Pylint, which is more thorough.

Validation processes had different functions within the software phase and the data science phase. Continuous integration was employed using Travis, and unit tests were developed and ensure that the code-base was not broken with each commit. To validate the performance of the model several different methods were used. For the Industrial Metal forecasts the true price of the instrument that is being predicted for serves as a validation method. Whether the models can learn the training data serves as a good validation routine. The purpose of the benchmarks is also to serve as a method of validation. If the data was completely random then the LSTM models might not be able to accurately forecast the price, however, the ARIMA and naive benchmarks can be compared to serve as a common-sense check.

In addition to the Industrial Metals forecasting examples, two generic notebooks were developed for auto-regressive and multivariate predictions. It was known that predicting financial data might prove too difficult so the generic notebooks served as a method of validation. They were developed to predict more simple problems, if they were able to outperform the naive benchmark on very simple cases i.e predicting sine waves or a linear combination of signals, then the models themselves were functioning correctly. A real test of

performance was whether the LSTMs could outperform an ARMA (another time series in the same family as ARIMA) benchmark on a signal generated by an ARMA process. As this was achieved it was validation that the deep learning and LSTM code-base was functional.

7 Results

Experimentation began using the log returns of the series. This was as theoretically log returns should perform best as discussed in Section 3.2. However, the log returns training and target series were extremely volatile and non-linear, much more than the raw price series because all trend components have been removed. The result was that the LSTMs performed very poorly. It was then thought that perhaps the log returns results could still be achieved by making predictions on the raw prices and then calculating the log returns of the predictions. The results for these experiments were better than trying to predict on purely log returns. However, it was still apparent that the models were failing to learn the datasets. Even just learning the training dataset was proving challenging for the models, with the networks predicting a flat result most of the time. Therefore the decision was made to change to scaling raw prices and use that as the input/target series, then to inverse scale the series when calculating the metrics and plotting the results. A `MinMaxScaler` was used on the time series, which adjust all the prices to between 0 and 1. This has the benefit of preserving the shape of the original series, and making no assumption about future values, meaning there is no look-ahead bias. All of the following experiments were made on the raw scale time series.

Investigations into the following neural network architecture hyper-parameters were taken for each of the forecast lengths, with the best performing hyper-parameter shown in bold;

1. Neurons in the LSTMs hidden state: 4, **8**, 16, 32
2. Neurons in the dense layer: 8, 16, **32**, 64
3. Number of additional dense layers: 0, **1**, 2
4. Dropout Percentage: **0%**, 10%, 20%, 50%
5. Number of LSTM Layers: **1**, 2

The general implication of these results is that the networks benefit from a medium level of complexity. An adequate capacity was required to be able to accurately model the data, but a low enough number of neurons was required to avoid over-fitting and generalise well. From the experiments on the window sizes, discussed in Table 1, it was discovered that the input window of 132 days (half a year) performed best for the forecast lengths of 1 week and 1 month, while a window size of 264 days was preferable for a quarter and half a year forecasts.

Predictions for the different regression frameworks can be found in Table 4 and 5. More performant models will be achieving a lower value for MSE and MAE, and a higher value for MDA. Due to space restrictions, some abbreviations have been used. AR denotes auto-regressive, MV denotes multivariate. So AR is the single-input single-output prediction, MV is the multi-input single-output prediction, AR MTL is the auto-regressive multi-input multi-output and MV MTL is the full dataset, multivariate, multi-input multi-output.

	1 Week			1 Month			1 Quarter			1/2 Year		
Method	MSE	MAE	MDA	MSE	MAE	MDA	MSE	MAE	MDA	MSE	MAE	MDA
ARIMA	21784	114	48.8	97648	245	46.2	287839	434	47.2	558965	620	47.2
Naive	21753	116	47.0	90972	234	43.8	270913	422	44.5	456655	541	45.2
AR	29803	138	49.6	93926	240	48.9	360432	492	47.1	702039	707	45.6
MV	46186	178	51.1	126064	288	47.2	860102	788	47.7	701929	555	44.3
AR MTL	183893	381	48.9	55079	193	48.7	153140	300	50.7	641360	677	46.1
MV MTL	55784	190	44.7	113348	271	45.7	97587	282	50.7	79190	242	51.4

Table 4: Predictions Evaluation

The ranking of the results from Table 4 can be seen in Table 5. The two MTL models outperformed both the single-task frameworks and the two benchmarks when comparing overall metrics, with the auto-regressive formulation narrowly outperforming the multivariate MTL. If looking at the regression only metrics

then naive was the top performer, but the multivariate MTL framework was able to outperform the ARIMA benchmark. The AR MTL performed well across the month and quarterly forecasts, while the MV MTL performed best over slightly longer forecast lengths; in the quarter and half a year forecasts. This will be discussed further in Section 8.

	1 Week			1 Month			1 Quarter			1/2 Year			Total	
Method	MSE	MAE	MDA	MSE	MAE	MDA	MSE	MAE	MDA	MSE	MAE	MDA	MSE/MAE	All
AR MTL	6	6	3	1	1	2	2	2	1	4	5	3	27	36
MV MTL	5	5	6	5	5	5	1	1	1	1	1	1	24	37
Naive	1	2	5	2	2	6	3	3	6	2	2	5	17	39
ARIMA	2	1	4	4	4	4	4	4	4	3	4	2	26	40
AR	3	3	2	3	3	1	5	5	5	6	6	4	34	46
MV	4	4	1	6	6	3	6	6	3	5	3	6	40	53

Table 5: Ranking Comparison

The predictions for the AR 1 week forecast be seen in Figure 8, while the comparison between naive and MV MTL can be seen in Figure 10 and 9 respectively. In Figure 8 it can be seen that the LSTM network outperforms the naive signal most of the time, but it overshoots on the peaks and troughs which results in a lower score overall. What is interesting about the MV MTL model is that it is not particularly accurate in the short term or long term. But it's performance is seemingly agnostic to forecast length i.e while it's accuracy does not improve, it's relative performance compared to the other forecasting methods over time does. This can be seen by comparing Figure 10 and Figure 9. In these Figures the blue line is the exactly the same signal for all 8 sub-figures. But due to the scale of the longer forecast length naive forecasts it is not even obvious that they are identical signals. The forecast length has been used as the scaling for the grid axes to impress just how far into the future these forecasts are predicting.

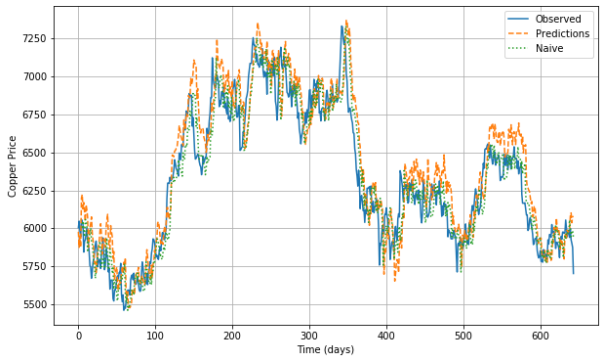


Figure 8: 1 week test predictions for the AR framework

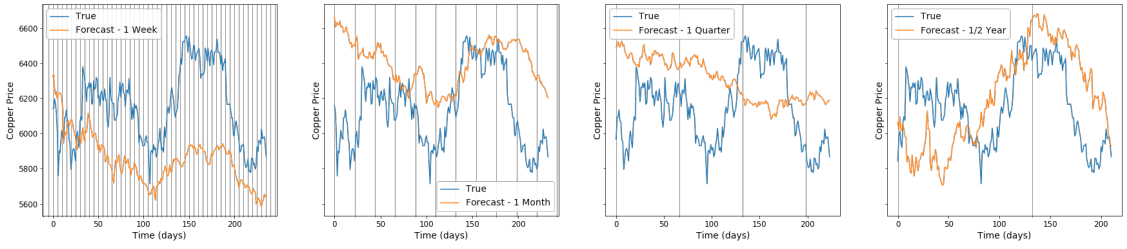


Figure 9: Comparison between forecast lengths using MV MTL framework. Grid lines spaced at one forecast length.

The comparison between including/not including a feature spawn step, and including/not including the PCA step can be seen in Figure 6 for the multivariate LSTM case. Y/N in the method column signified Yes or No, and FS denotes Feature Spawn. All the hyper-parameters were kept constant and only the preprocessing

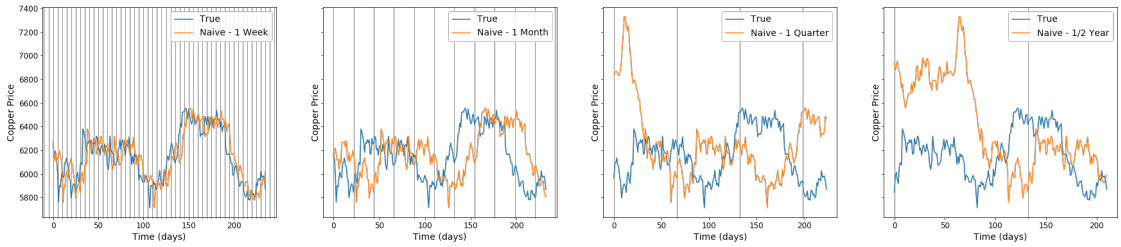


Figure 10: Comparison between forecast lengths using naive predictions. Grid lines spaced at one forecast length.

steps were changed. The most effective preprocessing routine was to use PCA without the feature spawning step.

	1 Week			1 Month			1 Quarter			1/2 Year		
Method	MSE	MAE	MDA	MSE	MAE	MDA	MSE	MAE	MDA	MSE	MAE	MDA
Naive	21753	116	47.0	90972	234	43.8	270913	422	44.5	456655	541	45.2
Y PCA N FS	46186	178	51.1	126064	288	47.2	860102	788	47.7	701929	555	44.3
Y PCA Y FS	128736	301	47.1	1438182	957	50.5	7811001	2156	45.1	5957215	2078	44.9
N PCA Y FS	115808	296	43.8	5809878	665	44.1	4753314	1927	46.3	291029	471	43.7
N PCA N FS	188958	351	47.0	459780	617	43.2	2396827	1461	44.4	1538614	1184	16.9

Table 6: PCA and Feature Spawn Comparison

8 Discussion

It is significantly more likely for a bad model to outperform a good model on MDA by chance than it is to be a bad model and perform well on MSE and MAE. False positives are highly unlikely with the MSE and MAE predictions. While the LSTM models generally outperform the ARIMA and naive forecasts on MDA, this metric does not carry the same kind of weight as the other two. If the models were displaying very high MDA (>60%) then it would be a more important metric. But as the MDA results are not exceptional from any of the LSTMs or benchmarks it is probably not as relevant as the regression metrics. Additionally, while the relative performance is important, and is the focus of Table 5, it must be combined with the absolute performance of the benchmarks/models. It doesn't provide much practical use for the LSTMs to outperform the benchmarks if both the benchmarks and the models give poor predictions. If all the models perform poorly, it could just be that it is not feasible to make predictions for that forecast length, given the information available.

With these considerations in mind, it is worth reviewing the implication of the results on the investigations hypotheses. Both on regression metrics only, and all metrics, the MV MTL formulation was capable of outperforming ARIMA, and on all metrics outperformed naive. Depending on the manner of evaluating the regression metrics the MTL may or may not have outperformed naive. If looking at the total sum of raw MAE and MSE scores the MTL formulation outperformed naive significantly, but if looking at the ranking sums it did not. It is pretty evident that naive and ARIMA are capable in the short term but not strong in the longer time horizons. While the longer time horizons are where AR MTL and MV MTL especially perform best. This could be because the nature of the common factor is less helpful in short term predictions but better at indicating longer-term trends. While the AR MTL ranked highest on all metrics when looking at a combination of relative and absolute metrics the MV MTL formulation has the best performance on average. It may not be highly accurate in the shorter time frame, but it is the most consistent across all forecast lengths.

It would be reasonable to say that the performance of MV MTL would lead to the partial acceptance of hypothesis 1, outperforming both benchmarks over the longer time horizons. The strong performance of both the MTL frameworks would lead to the strong acceptance of hypothesis 2; that the prediction of IMs would benefit from using Multi-Task Learning. Given the relatively small dataset the MTL frameworks aided to increase generalization and therefore performance out of sample. This evidence would concur with

literature that there is a common factor observed between commodities, particularly those within shared sub-universes. This investigation would serve as evidence to support these claims. The third hypothesis was that predictions should benefit from using a variety of data, and the multivariate formulations should outperform the auto-regressive formulations. This is the case for the MTL framework but not the case for the single-task framework. This leads to the partial acceptance of the hypothesis; with the acceptance in the MTL case but rejected in the non-MTL cases. These assessments of the hypotheses are based on the relative performance between the different frameworks.

It was surprising when it was shown by the experiments in Table 6 that the networks do not see additional performance when additional feature engineering was included in the preprocessing routines. The worse performance was seen in both the multivariate and multi-task cases indicating that the technique is generally not a beneficial preprocessing step for LSTM models. This spawning technique is known to work with other models from empirical evidence found through discussions with professional within quantitative finance. A hypothesis for the unsuitability of this preprocessing method could be that LSTMs are already inherently capable of the smoothing and volatility quantification ability that is directly given through these spawned series. Therefore by creating these spawned features, the LSTMs have to process more data but without gaining any of the additional benefits from it. Whereas other modelling techniques can benefit from it, because they do not inherently have the capability to smoothing and measure the volatility.

Ultimately when observing the absolute performance on the LSTMs and bench-marks, none of the models are sophisticated enough to instill confidence in their insight within financial markets. Depending on their use case the models might provide actionable insight. They allow access to predictions and are more sophisticated than a non-expert could make. However, they are not accurate enough that it would be advised to speculate on the price movements of Industrial Metals. It is easy to be critical of the models when you look at their predictions visually. However, it must be remembered that these models are predicting non-linear, volatile signals up to half a year in the future. And in this context, they are still more powerful a tool than the decisive power of an individual.

The most probable cause for this lack of confidence is the shortness of the time-frame in the dataset. The data only covers one economic cycle and had an extreme economic shock in the middle of the dataset. Without a wider historical context this confuses the models. This investigation believes that for the benefit of LSTMs to really be observed, a longer historical context if required. This could even be incorporated with less granularity, with the daily frequency probably being unnecessary. A weekly sampling would decrease much of the noise in the dataset, and help the LSTMs focus on the economic factors that drive the signals.

On reflection, too much attention was paid to investigating the relative performance between log returns, inverse log returns and price. This was as it was considered highly desirable to make predictions on naturally normalised time series. Therefore considerable effort was placed into trying obtain decent results using this formulation instead of using raw prices which obtained better results with very little effort. The reasons it was so difficult to obtain good results on the log returns formulation was that the target signal was extremely erratic. This meant any attempt the model made to learn the signal and deviate from the centre resulted in such high error metrics that it was 'safer' for the signal just to stay relatively flat. Perhaps this might not have occurred if mean absolute error was used as the loss function instead of mean squared error. The models did begin to learn the log return signals but only after extensive training, taking an order of magnitude more epoch training just to begin seeing learning, compared to the epoch required for full convergence on the raw price signal. Additionally it was difficult to tell visually if the log returns predictions were performing well due to the signal being much more erratic. While it was easy to see whether the raw price predictions were accurate using the `live_pred_plot` function.

9 Conclusions

This investigation aimed to investigate the predictive ability of LSTMs on industrial metal prices and there were three hypotheses that were the core focus. The first hypothesis stated that due to LSTMs abilities to model complex non-linear problems they should outperform time series methods and other conventional

benchmarks (in this case naive forecasting). This was observed to be true, with the multivariate LSTMs performing the best out of all the prediction methods when assessed across all the metrics/forecast lengths. It was expected that as forecast lengths increase so too would the error in predictions, which was the case. It can be generalised by saying that if predicting from a day to a month in advance then it is best to use a naive forecast but for longer time frame use MV MTL. This would lead to the view that for short term speculation that other techniques to LSTMs are most appropriate but for long term planning LSTMs are superior.

While multivariate LSTMs did outperform all the other models this is largely due to the other methods performing increasingly poorly with growing forecast length. The use of these networks in real applications depends on the importance of accuracy and precision. When looking at the predictions for the shorter time frames it is easy to have confidence in the predictions. For longer forecasts, while the LSTMs may outperform the benchmarks when looking at the predictions visually they do not appear particularly reliable. The utility of these predictions really depends on the use case. While the predictions do not look good enough to provide actionable insight ultimately there are many parties who are forced to engage in forecasting, whether they'd like to or not. For example, a soft drinks company will be forced to have to predict the price of aluminium to try and budget for can production. If the predictions from LSTMs can outperform state-of-the-art time series methods then even if there isn't a lot of confidence in the LSTM models visually it is still the best method available, and therefore the one to use. If the soft drinks company is required to forecast half a year into the future it is preferable to use a package like Foresight to forecast aluminum prices instead of the most common alternative which would be to plan based on the aluminium price today, which is equivalent to using a naive forecast. In this example, all the metrics are of importance, not just the regression metrics because forecasting the incorrect direction (which is more likely with ARIMA or naive) can result in the difference between making a profit or loss per can.

It was believed the theory that commodity prices exhibit excessive co-movement could be exploited by using multi-task learning, with the individual metal price predictions being individual tasks. This observation was shown to be valid, with both the multi-task learning frameworks outperforming the non-MTL and benchmark predictions across all metrics. Therefore it can be advised that if more data could be included in the dataset and further experiments are performed on IM prices, that the framework used would be multivariate MTL. As mentioned in Section 2.1 there are many ways that the neural parameters can be shared between tasks. The method chosen in this investigation is the simplest MTL deep learning framework, with all the parameters being shared. An interesting piece of future work would be including individual layers to the network that are task-specific. This is a very easy piece of additional work and would only involve creating the new architecture in Pytorch similar to `LSTM` but with different final layers.

The final hypothesis stated that the deep learning approaches should benefit from including more relevant data in the dataset. As data going back further in time was unavailable it was chosen to include other financial time series that could be driving the price movements. It was not surprising that deep learning methods improve with more data. However, it is surprising that this was only the case when used in the MTL case. Showing that the networks can capitalize greatest on additional data when the networks are structured in a manner that they have high-capacity but can also generalise well.

Overall, when assessing the performance of the LSTMs, while they did outperform the benchmarks really they are a technique that thrives on temporally wide datasets. The datasets contained a reasonably large amount of data, but the day-to-day price fluctuations, while they do contain some useful information also contain a lot of noise which can confuse the networks. It is generally known that for smaller datasets specific statistical techniques and machine learning methods designed for smaller datasets can outperform deep learning models. This dataset sits just past the cusp of being more performant than state-of-the-art time series models. The package Foresight was developed to complete this investigation and made the preprocessing, training and evaluation of models a simple process. Given the encouraging results of the experiments conducted it can serve as a good platform for further research into the other MTL LSTM architectures, or revisiting current architectures if older data could be acquired.

10 Required Metadata

Current software version	v1.0.8
Permanent link to code/repository used for this code version	https://github.com/msc-acse/acse-9-independent-research-project-OliverJBoom
Legal Software License	MIT
Code versioning system used	git
Software code languages, tools, and services used	python (>=3.6)
Installation requirements & dependencies	numpy (>=1.16.2), pandas (>=0.24.2), pmdarima (>=1.2.1), matplotlib (>=3.0.3), scikit-learn (>=0.20.3), statsmodels (>=0.9.0), torch (>=1.1.0)
Operating system requirements	MacOS, Windows
If available, link to developer documentation manual	industrial-metals-forecaster.rtfd.io
Support email for questions	ob3618@ic.ac.uk

Table 7: Software Metadata

Bibliography

[1] Gillian Dooley and Helena Lenihan. An assessment of time series methods in metal price forecasting. *Resources Policy*, 2005. ISSN 03014207. doi: 10.1016/j.resourpol.2005.08.007.

[2] Isabel Vansteenkiste. How important are common factors in driving commodity prices? *IMF Working Paper Series*, 1072, 2009.

[3] Christian Pierdzioch, Jan Christoph Rülke, and Georg Stadtmann. Forecasting metal prices: Do forecasters herd? *Journal of Banking and Finance*, 37(1):150–158, 2013. ISSN 03784266. doi: 10.1016/j.jbankfin.2012.08.016.

[4] A Olayiwola. Forecasting Copper Spot Prices: A Knowledge-Discovery Approach. *Studentnet.Cs.Manchester.Ac.Uk*, page 80 pp, 2016. URL http://studentnet.cs.manchester.ac.uk/resources/library/thesis_abstracts/MSc16/FullText/Olayiwola-Adegbenga-diss.pdf.

[5] W. Labys. *Modeling and Forecasting Primary Commodity Prices*. Ashgate, 2006.

[6] Justin Sirignano and Rama Cont. Universal features of price formation in financial markets: perspectives from deep learning. *Quantitative Finance*, pages 1–20, 2019. ISSN 14697696. doi: 10.1080/14697688.2019.1622295.

[7] G. Lemus. Why Financial Time Series LSTM Prediction fails, 2018. URL https://medium.com/datadriveninvestor/why-financial-time-series-lstm-prediction-fails-4d1486d336e0.

[8] Sima Siami-Namini and Akbar Siami Namin. Forecasting Economics and Financial Time Series: ARIMA vs. LSTM. pages 1–19, 2018. URL http://arxiv.org/abs/1803.06386.

[9] Sangyeon Kim and Myungjoo Kang. Financial series prediction using Attention LSTM. 2019. URL http://arxiv.org/abs/1902.10877.

[10] Joseph P Byrne, Giorgio Fazio, and Norbert Fiess. Co-movements , Common Factors and Fundamentals. *World Bank Policy Research Working Paper*, (February):1–33, 2011. URL http://www-wds.worldbank.org/servlet/WDSContentServer/WDSP/IB/2011/02/28/000158349_20110228101354/Rendered/PDF/WPS5578.pdf.

[11] Alexandra Gabriela Titan. The Efficient Market Hypothesis: Review of Specialized Literature and Empirical Research. *Procedia Economics and Finance*, 2015. ISSN 22125671. doi: 10.1016/s2212-5671(15)01416-1.

[12] C. A. Tapia Cortez, S. Saydam, J. Coulton, and C. Sammut. Alternative techniques for forecasting mineral commodity prices. *International Journal of Mining Science and Technology*, 2018. ISSN 20952686. doi: 10.1016/j.ijmst.2017.09.001.

[13] G P Nason. *Stationary and non-stationary time series*. Number 1994. 2003.

[14] Shuhao Zhang. Copper & Aluminium Price Forecasting. 2019.

[15] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. Deep Sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio Speech and Language Processing*, 24(4):694–707, 2016. ISSN 23299290. doi: 10.1109/TASLP.2016.2520371.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Handbook of Approximation Algorithms and Metaheuristics*, pages 60–1, 2007. doi: 10.1201/9781420010749.

[17] Alex Sherstinsky. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. pages 1–39, 2018. URL http://arxiv.org/abs/1808.03314.

[18] Hochreiter Sepp and Schmidhuber Jurgen. Long Short Term Memory. *Neural Computation*, 9(8):1–32, 1997.

[19] Haim Sak, Andrew Senior, and Franoise Beaufays. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. 2014. URL http://arxiv.org/abs/1402.1128.

[20] Florian Eyben, Martin Wöllmer, Bjrn Schuller, and Alex Graves. From speech to letters - using a novel neural network architecture for grapheme based ASR. *Proceedings of the 2009 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2009*, pages 376–380, 2009. doi: 10.1109/ASRU.2009.5373257.

[21] Valtteri Ahti. Forecasting commodity prices with nonlinear models. *Helsinki Centre of Economic Research*, 17(268), 2009. ISSN 1795-0562.

[22] R David Mclean. Does Academic Research Destroy Stock Return Predictability? *Journal of Finance*, 2015.

[23] Thomas Fischer and Christopher Krauss. Networks for Financial Market Predictions. *FAU Discussion Papers in Economics, No. 11/2017, Friedrich-Alexander-Universität Erlangen-Nürnberg, Institute for Economics, Erlangen*, pages 1–34, 2017.

[24] Fama Eugene. EFFICIENT CAPITAL MARKETS: A REVIEW OF THEORY AND EMPIRICAL WORK*. *The Journal of Finance*, 25(2):418–420, 1969.

[25] Malkiel George Burton. *A Random Walk down Wall Street : the Time-Tested Strategy for Successful Investing*. W.W. Norton, 2003.

[26] Rob Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. URL https://otexts.com/fpp2/simple-methods.html.

[27] A and Rossen. What are metal prices like? Co-movement, price cycles and long-run trends. 2014.

[28] Kim Han Thung and Chong Yaw Wee. A brief review on multi-task learning. *Multimedia Tools and Applications*, 77(22):29705–29725, 2018. ISSN 15737721. doi: 10.1007/s11042-018-6463-x.

[29] Yu Zhang and Qiang Yang. A Survey on Multi-Task Learning. pages 1–20, 2017. URL http://arxiv.org/abs/1707.08114.

[30] Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. (May), 2017. URL http://arxiv.org/abs/1706.05098.

[31] Ratnadip Adhikari. *An Introductory Study on Time Series Modeling and Forecasting*, volume 1302.6613. LAP Lambert Academic Publishing, 2013. ISBN 9783659335082.

[32] Taylor Smith. Pmdarima documentation, 2018. URL http://www.alkaline-ml.com/pmdarima/1.0.0/modules/generated/pmdarima.arima.auto_arima.html.

[33] Jonathan Taylor Josef Perktold, Skipper Seabold. Statsmodels - ARIMA Documentation, 2009. URL https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARIMA.html.

[34] Christopher Olah. Understanding LSTM Networks, 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[35] R Rojas. Neural Networks - A Systematic Introduction. *Neural Networks*, 7, 1996. URL https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf.

[36] Paul. J. Werbos. Backpropagation through time: What it does and how do we do it, 1990.

[37] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *30th International Conference on Machine Learning, ICML 2013*, (PART 3):2347–2355, 2013.

[38] Keras LSTM Tutorial. URL https://adventuresinmachinelearning.com/keras-lstm-tutorial/.