universität wien

Masterarbeit / Master's Thesis

Titel der Masterarbeit / Title of the Master's Thesis:

Can Machine Learning enhance return predictability of selected metal commodities?

verfasst von / submitted by

Benjamin Reinhard Achim Walter Albrechts BSc

angestrebter Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien 2021 / Vienna 2021

Studienkennzahl lt. Studienblatt / Degree programme code as it appears on the student record sheet:	UA 066 974
Studienrichtung lt. Studienblatt / Degree programme as it appears on the student record sheet:	Masterstudium Banking and Finance
Betreut von / Supervisor:	Univ.-Prof. Mag. Günter Strobl, PhD
Mitbetreut von / Co-Supervisor:	Dipl.-Inform. Dipl.-Volksw. Dr. Christian Westheide

This page is intentionally left blank.

Abstract

This thesis compares predictability of returns of the metal commodities Gold, Silver, Platinum, Palladium and Copper on multiple datasets and across multiple resampling periods. The goal is to determine if deep learning models can confirm and/or enhance the findings of Tharann (2019) on more recent data, who finds strong predictability on the same selection of metal futures out of sample especially after including the *Aruoba-Diebold-Scotti Business Conditions Index* (ADS-Index).
Both the source code as well as a description of the data fields used are provided in order to make all findings reproducible (with minimal installation efforts).
Overall, there are strong indications that metal returns are likely predictable and research should be attractive for many types of investors with different asset re-allocation preferences. Tests are run with daily, weekly, monthly and quarterly data on three different sets of predictive variables. As for the utilized models, both *Long Short Term Memory* (*LSTM*) as well as *Convolutional Neural Networks* (*CNN*) can outperform linear *Ordinary Least Squares* (*OLS*), still one of the most utilized tools in existing research on similar topics, on the underlying datasets.

Kurzdarstellung

Diese Arbeit vergleicht die Vorhersagbarkeit von Renditen der Metallrohstoffe Gold, Silber, Platin, Palladium und Kupfer auf verschiedenen Datensätzen und über mehrere Veranlagungsperiodizitäten. Ziel ist es, festzustellen, ob Deep-Learning-Modelle die Ergebnisse von Tharann (2019) auf aktuelleren Daten bestätigen und/oder verbessern können, wobei jener starke Indikatoren für eine solche Vorhersagbarkeit auf derselben Auswahl von Metall-Futures aufdeckt, insbesondere nach Einbeziehung des *Aruoba-Diebold-Scotti Business Conditions Index* (ADS-Index).
Sowohl der Quellcode als auch eine Beschreibung der verwendeten Datenfelder sind bereitgestellt, um alle Ergebnisse (mit minimalem Installationsaufwand) reproduzierbar machen zu können.
Insgesamt gibt es deutliche Hinweise darauf, dass Metallrenditen vorhersehbar sind und die Forschung für verschiedenste Anleger mit unterschiedlichen Präferenzen bei der Umschichtung von Vermögenswerten attraktiv sein dürfte. Die Tests wurden mit täglichen, wöchentlichen, monatlichen und vierteljährlichen Perioden für drei verschiedene Datensätze durchgeführt. Insgesamt scheinen sowohl *Long Short Term Memory* (*LSTM*) als auch *Convolutional Neural Networks* (*CNN*) lineare *Kleinstquadrateschätzer* (*OLS*), eine häufig verwendete Methode in ähnlichen Veröffentlichungen, auf den verwendeten Daten zu übertreffen.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

This thesis explores the applicability of supervised deep learning to anticipate price movements of metal commodities. Predicting returns is a feat undertaken for many generations of finance researchers now. Evidence on asset price predictability can be traced back to roughly 120 years ago, when Charles Dow experimented with technical analysis and dividend growth to yield information about future price developments on the equity market (Hamilton, 1922). In fact, at times there seemed to be even a consensus among practitioners and, although less so, academics that technical analysis, in essence exploiting past/historic information to infer future return developments, provides informational value to investors (Lettau & Ludvigson, 2001; Brown & Jennings, 1989).

A more recent contribution to this fundamental question but with a slightly less optimistic view for the equity market is Welch & Goyal (2008). They conclude that while their models are indeed able to deliver a positive outperformance, they do not do so systematically. Many are primarily driven by the oil shock in 1975, strongly time-varied, or could even be classified as downright spurious. In a defense of return predictability Cochrane (2008) on the other hand argues that empirical return developments are likely caused by time-variation of risk premia and expected returns, not by changes in dividend growth as Dow suspected. The debate receives further uplift in context of *Factor Timing* or *Smart Beta* strategies, namely attempts to time investments in specific equity factors dynamically according to the current market situation (Durand et al., 2011; Hodges et al., 2017).

However, the question begs why returns would be predictable at all. If they were, any market participant could simply gain an extraordinary profit by trading on predictions early, which would in consequence then price in any potential gains made from such a transaction faster and may ultimately lead to a leveling out. Equity markets are closely monitored by many analysts and institutions with large capital endowments after all. Furthermore, markets provide high liquidity, low transaction cost and broad informational availability. If such an algorithm were possible, why does it not exist already? While surely a puzzle on financial markets in general, commodity markets on the other hand have a tendency to be less liquid than FX or equity. They often require moving physical assets at high costs or even legally exclude financial investors from

participating. This might ultimately open more room for return forecasting. Tharann (2019) utilizes the predictive variables of and the groundwork laid out by Welch & Goyal (2008) to get informed estimations on Gold, Silver, Platinum, Palladium, and Copper. His paper isolates the long-term government bond yield and the *Aruoba-Diebold-Scotti Business Conditions Index* (ADS-Index) developed by Aruoba et al. (2009) as the most influential predictive indicators, showing strong evidence for return predictability at least in metal markets with an out-of-sample R^2 of up to 18.57%. Tharann (2019) finds predictability to be especially strong during the Great Financial Crisis of 2008. This thesis fundamentally builds on this paper and aims to extend the research by investigating if deep learning methods can produce better and more reliable results on a similar feature set.

1.2 Main Goals and Research Questions

This thesis is an investigation into forecasting returns of commodities using deep-learning neural networks. Concisely, it attempts to find answers to or extend the following:

1. Is there any predictive value for the selected metals at all?
2. If so, will applications of machine/deep learning be able to enhance the predictive power?
3. Does predictability vanish or improve using a broader set of features and/or different asset re-allocation periods?

The focus is on the five selected commodity metals shown in Figure 1. Gold, the typically perceived safe haven in times of crisis and held by many financial investors worldwide, Silver, Platinum, Palladium and finally Copper, whereby the latter two are primarily used in industrial contexts. This selection of commodities aims to:

1. Cover a spectrum of metals that feature both longer term value ap- and depreciation.
2. Represent various levels of demand from different types of investors.
3. Limit the set to those with sufficiently many datapoints available for reliable comparisons.

Since trading cost for commodities may be substantial, realizing this strategy is likely only possible when utilizing cash-settled derivatives.

Figure 1: Prices and traded volumes for the included commodities.

2 Dataset and Preprocessing

The key source of all underlying analysis is a database collected from *Bloomberg*. The following section will describe the core data model (also referred to as *Core Model*). It consists of the original predictive variables as outlined by Tharann (2019). Since some of the used data are only available at lower frequencies, corresponding predictive variables are excluded in higher frequency models.

2.1 Commodity (Log-)Returns

Commodity primarily closing prices (`PX_LAST`) of continuously rolled metal futures. Furthermore, information on `PX_OPEN`, `PX_LOW`, and `PX_HIGH` is documented. Returns are calculated by:

$$r_t = log(\frac{\texttt{PX\_LAST}_t}{\texttt{PX\_LAST}_{t-1}})$$

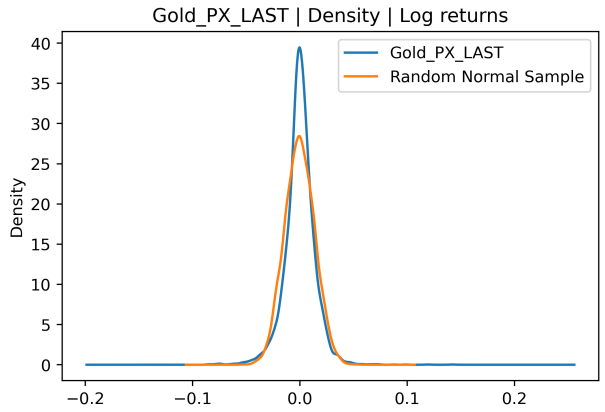
If daily asset reallocation is assumed, the formula incorporates opening prices to allow for an overnight trading delay. Referring to last price quotations would assume instant execution of future return inference and trade. Using the opening prices effectively implies selling the portfolio at the last prices, running all processes over night and then buying at the opening price of the following day:

$$r_t = log(\frac{\texttt{PX\_LAST}_t}{\texttt{PX\_OPEN}_t})$$

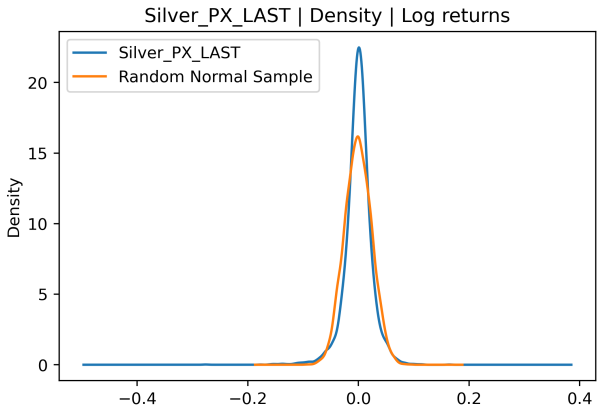
Note that $log(\cdot)$ implies the natural logarithm in this case. Welch & Goyal (2008) represent the commodity premium as:

$$crp_t = r_t - r_{f,t}$$

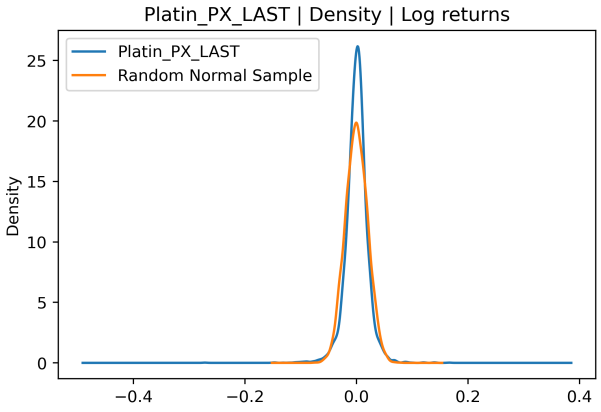
Since a truly risk-free rate $r_{f,t}$ does not exist, many suitable proxies can be discussed. Brooks & Yan (1999) compare two widely applied examples, namely the US-*Treasury-Bill* (T-Bill) rate as well as the British *London Interbank Offered Rate* (LIBOR), whereby they find strong deviations and differing characteristics for both. The yield curve for LIBOR turns out to be steeper (spread between highest and lowest rate of any maturity) and with a smaller curvature



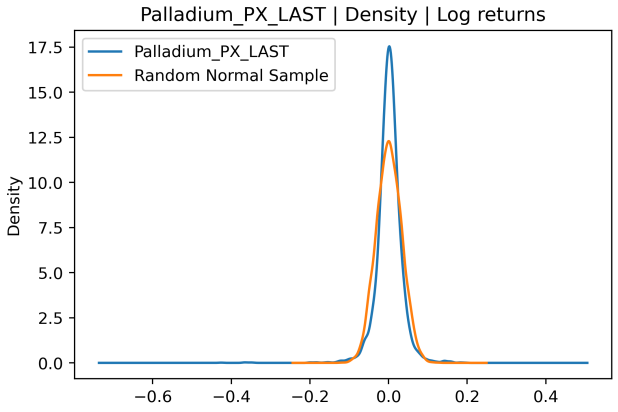
(a) Gold



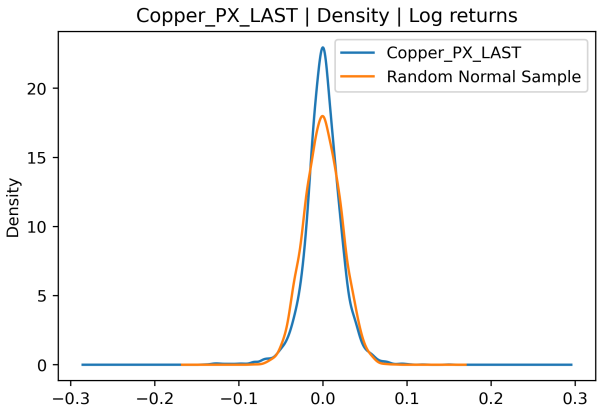
(b) Silver



(c) Platinum



(d) Palladium



(e) Copper

Figure 2: Log returns are not normally distributed but tend to have slim tails.

(of the yield curve itself) than the treasury rate (whereby edge-cases exist at which both rates even move in opposite directions). LIBOR is also generally higher, but the spread between it and the treasury rate is not constant for any given maturity as LIBOR exhibits an overall flatter yield curve. Since there are drastic differences at times, the US-index needs to be considered. This is also appropriate in the context of the *New York Mercantile Exchange* (NYMEX) and therefore US traded commodities. Furthermore, the great LIBOR scandal also highlighted how safe some of these proxies ultimately really are (Hou & Skeie, 2014). However, the 3-months T-Bill is already incorporated in the spread between long term and short term US Bond rates as a predictive variable in the dataset. Deducting it in the dependent variable again would potentially lead to biases during training and inference. It is therefore omitted from the dependent variable.

Tharann (2019) utilizes 12 months excess returns of commodity future contracts rolled over monthly data, defining the excess return as:

$$er_{t+1} = r_{t+1} - f_t$$

whereby

$$r_{t+1} = log(\frac{F_{t+1,T}}{F_{t,T}}) + f_t$$

and f_t being the interest rate on a fully collateralized future contract. A complete collateralization, i.e. buying commodity futures as well as treasury bills, is not the working assumption in this thesis, as the risk-free rate is omitted from the dependent variable. The *Bloomberg* data used in this thesis are continuously rolled contract prices. In general and given spot prices P_t, the price of a future maturing at T can be denoted as:

$$F_{t,T} = E[P_T] \cdot (1 + r_{f,t})^{t-T}$$

The expectation around P_t indicates that future and spot prices need not be the same. Since most assets are likely going up in price, at least in a market with positive inflation, the spot price is typically below the future price. This is a situation referred to as *Contango*. Should spot prices exceed future prices, a commodity is said to be in *Backwardation*.

2.2 Predictive Variables

The *Core Model* consists of the following predictive variables illustrated in Figure 3:

- dividend_payout_ratio: The dividend payout ratio of the *Standard & Poor's 500* (*S&P 500*) (`DVD_PAYOUT_RATIO`).
- default_return_spread: The spread between investment grade *S&P 500* corporate bonds (`SP5IGBIT_PX_LAST`) and the long term US bond rates (`USGG30YR_PX_LAST`).
- default_yield_spread: The log spread between investment grade (`SP5IGBIT_PX_LAST`) and high yield (`SP5HYBIT_PX_LAST`) *S&P 500* corporate bonds.
- dividend_yield: The gross aggregate dividend yield of *S&P 500* stocks (`GROSS_AGGTE_DVD_YLD`).
- inflation: *Organisation for Economic Co-operation and Development* (*OECD*) inflation reports on consumer price indices for the US (`CPIYOY_PX_LAST`).
- lt_us_govbond_rates: The 30 year US bonds rates (`USGG30YR_PX_LAST`).
- stock_var: The variance of *S&P 500* stock prices (`SP5_PX_LAST`).
- term_spread: The spread between the US bond (`USGG30YR_PX_LAST`) with the longest and the bill (`USGG3M_PX_LAST`) with the shortest maturity.
- industry_output_gap: The US industry output gap (`IPYOY_PX_LAST`).
- unemployment: The reported *OECD* unemployment rate for the US.
- ads_index: The ADS-Index as provided by the US *Federal Reserve System* (*FED*).

Core Model is primarily an equivalent dataset to Tharann (2019). However, commodities are not represented by log excess returns on commodity futures but by simple log returns. The primary reason behind this decision was the necessity of practical applicability, since predicting excess returns as $r_e = log(\frac{1+r_A}{1+r_f})$ would still require a model to predict the risk-free rate in order to automate trading decisions. Extensions to this setting are:

- *Extended Model*, which in essence is simply the core_model but expanded by

1. Price to EBITDA ratio on the *S&P 500* as a profit-potential oriented variable (since EBITDA is close to FCF)
2. Return on Assets as a purely profit-oriented variable irrespective of the market capital structure
3. FX-rate of the Euro and US-Dollar, since this rate could influence the decision-making of foreign buyers and sellers

- *Large Model*, which includes the extended_model and the geometric outperformance of MSCI-factors over the MSCI-World as a benchmark. Geometric outperformance is calculated as:

$$r_{op}^{geom} = log(\frac{P_t^{Factor}}{P_{t-1}^{Factor}}) - log(\frac{P_t^{MSCIWorld}}{P_{t-1}^{MSCIWorld}})$$

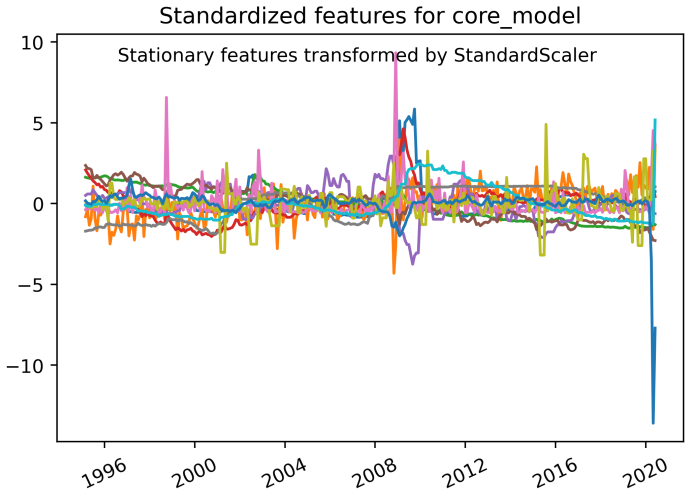
This is indeed geometric and not arithmetic outperformance because of the included log returns. The idea somewhat relates to Hodges et al. (2017) who conclude that factor timing (or smart beta) strategies closely relate to the business cycle, whereby the latter, as argued by Tharann (2019), has some effect on commodities as well.

2.3 Equity Related Components

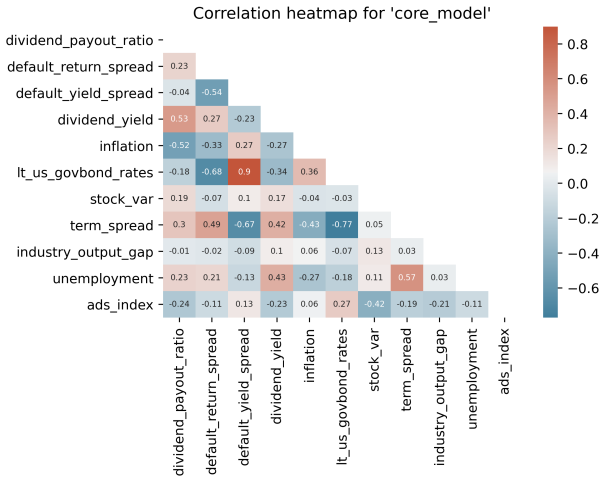
All Equity-related data is sampled from the *S&P 500* as a reference index for the US-market. The *S&P 500* itself is represented as a total return index, corrected for stock splits and dividends (`TOT_RETURN_INDEX_GROSS_DVDS`).

2.3.1 Payout Policy

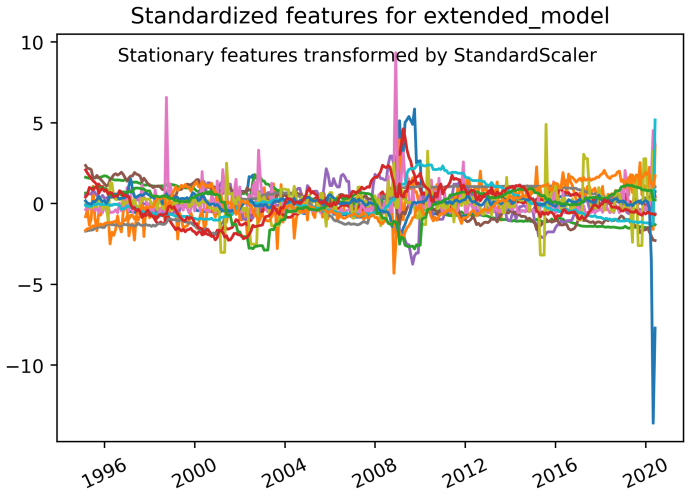
Payout policy is represented in the baseline dataset through the *Dividend Payout Ratio*. Dividends are the result of residual income to a corporation. On the individual level, this implies that when dividends are low or do not increase as anticipated, a companies' income is likely relatively low (et vice versa) compared to previous results. On an aggregate level, this could be used as a higher frequency signal about the overall state of the economy than the *Gross Domestic Product* (GDP) (delayed, quarterly, and corrected ex post) or *Industrial Output*



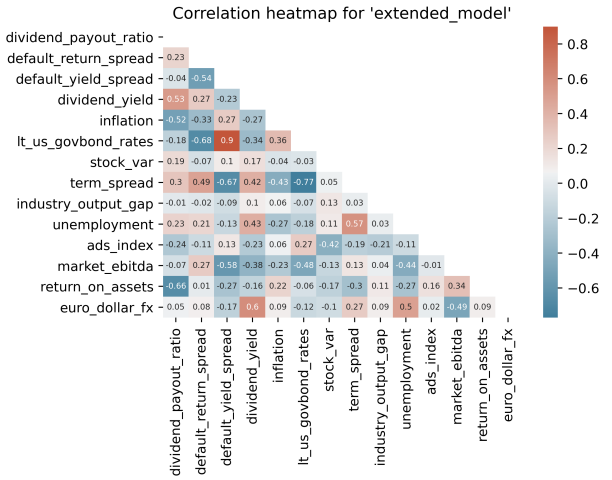
(a) core_model



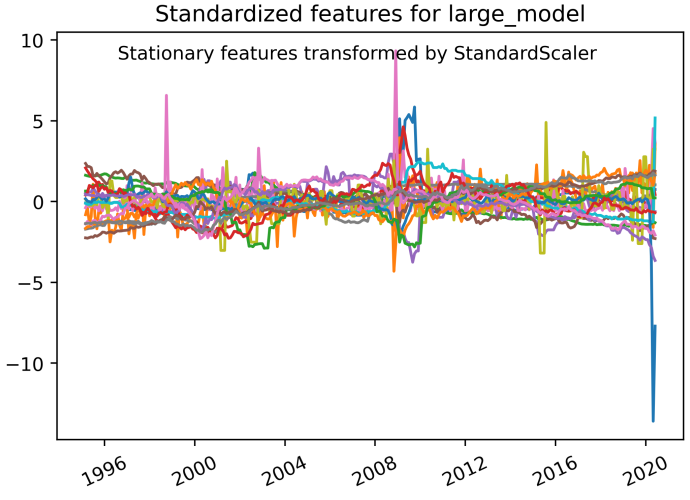
(b) Correlations



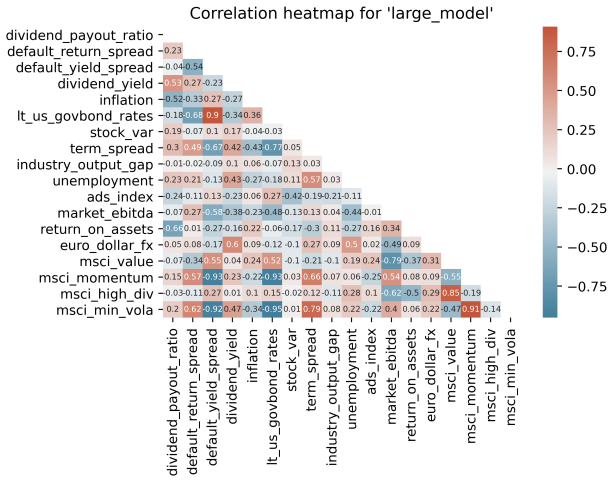
(c) extended_model



(d) Correlations



(e) large_model



(f) Correlations

Figure 3: Stationary and scaled features for the available data models incl. a heatmap of their empirical correlations.

(delayed, monthly). Ultimately, equity holders are residual claimants on the cash flows of their corporation. At least for longer periods in time, not more can be paid out sustainably than has been earned in the previous periods. Going further, reducing or not increasing (total) dividends may also be punished by shareholders. In order to enhance their cash-out they might increase pressure on management by exercising governance by action or exit, impacting the share price of their company in the process. Even just potential stock buyers could change their decisions based on updated information about dividend policy, as they might not be willing to pay a price as high as before the announcement was made public. Empirically, evidence is present when looking at announcement returns around publications of recent financial results. Using event study methodology, MacKinlay (1997) shows that a day before and up to a week later positive/negative abnormal returns are observable for positive/negative earnings announcements, respectively. But on the contrary, in a study assessing both short and long run performance of dividend changing firms, Gunasekarage & Power (2002) show that while dividend cutters are indeed punished in the short run, in the long run they outperform their dividend-increasing counterparts. To correct for any structural changes in the paypout policy such as an increasing preference of share repurchases over dividends, the *Extended Model* includes changes in *EBITDA* of the *S&P 500*. *EBITDA* is the standardized accounting metric closest to free cash flows. It captures a company's income which can either be used for investments or distributed to various stakeholders.

2.3.2 Profitability

In November and December 2020, examples such as *Tesla*, *BYD*, *Nikola* or even *Xiaomi* demonstrated that while in spite of (temporary) negative net income, growth firms can still be expensive to buy. These are exceptions, however, since such management would not be feasible for a mature market proxy such as the *S&P 500*. Furthermore, neither price levels nor their growth rates are constant over time. As such, Shiller (2000, p. 182) finds a clear positive interdependency between the *Dividend Present Value* and *S&P 500* price levels. In economic terms, one would expect that when profitability is relatively high the economy should be in a more healthy state. But looking at empirical results from *shopping goods* pricing, Warner & Barsky (1995) find that markdowns already start to occur when demand is excessively high, or in other words, when profitability is at its peak. This is counter-intuitive, initially. One would expect

prices to go up rather than down once demand outgrows supply or is on a comparatively high level and, hence, prices to mark up (indeed these effects are well explained when looking at the case of Veblen [luxury items] or Giffen goods [interplay of income and substitution effect], but this is a minor puzzle for average convenience or shopping goods). A direct implication could be that declining profitability of firms (on market level) indicates a delayed/future cooldown of the economy with direct effects on commodity markets. Suitable profit indicators to capture these events, even if only with a longer delay, would be both the return on assets (ROA) as well as the return on equity (ROE). However, empirically both ratios exhibit a correlation of over 0.9 over the past 30 years. This may be regarded as soft multicollinearity and may negatively affect interpretability primarily with linear regression models, so a choice needs to be made. Since it allows for abstraction from leverage, ROA is added in the *Extended Model*.

2.3.3 Valuation

As price levels on the stock market are not constant over time, investors might condition their trading behavior on current price levels relative to some measure of profitability. Following the Efficient Market Hypothesis (EMH), markets should be closely following a random walk. The hypothesis is famously linked to Malkiel & Fama (1970), although they were not first to discover and write about it. Traces of this theory can be found almost 80 years earlier in Gibson (1889). The EMH basically describes that markets are efficient at all times and relatively high or low prices are a mere illusion of the spectator, since both (some form of) relevant information as well as rational expectations about the future should be immediately priced in after becoming known. Malkiel & Fama (1970) distinguish among three forms of efficient markets:

1. **Weak Efficiency**: All past trade information is already priced in.
2. **Semi-Strong Efficiency**: Weak Efficiency plus public information is already priced in.
3. **Strong Efficiency**: Semi-Strong Efficiency plus all private (Malkiel & Fama (1970) refer to this as monopolistically accessible) information is already priced in.

A direct consequence of weak form efficiency would be a market that closely follows a random walk. Weak efficiency would furthermore render technical analysis useless. Semi-Strong efficiency would go further and not grant returns to fundamental market analysis as well, while strong efficiency would even make insider trading unfruitful. If only weak efficiency holds, then

predictability as attempted in this thesis should not allow for any dramatic outperformance at all. If that were the case, nobody would try to invest costly resources in developing such a tool to determine future returns and trade on these expectations. But if there is no reward for pricing the information under *Weak Efficiency*, how can they be contained in market prices? Grossman & Stiglitz (1980) investigate this paradox analytically. They find that under perfect information transition there is no incentive for agents to become informed if the acquisition of this information is costly. As a consequence, information could not be priced in at all anymore and information-wise perfect markets could not exist.

Long-term efficiency may be questionable when considering the aforementioned contributions of Shiller (2000). Looking at the *Shiller-CAPE*, defined as the ten year moving average of prices over total earnings, the paper suggests that market prices relative to earnings exhibit temporal ups and downs over time. A CAPE of e.g. 10 to 15 may be considered as cheap while 25+ would be somewhat expensive.

Overall, investors with distant investment horizons may profit from timing their investments relative to market valuation levels. One way of accounting for earnings is dividends (although share repurchases become more common recently). The dividends yield, defined as:

$$dy = \frac{Total\ SPX\ Dividends}{Total\ SPX\ Valuation}$$

is therefore included. This is not to say that high and low yields may not be unambiguous. High dividend yields may signal relative underpricing (consequence: valuation too low) or that a company is in dire straights (consequence: adequate valuation). Similarly, a low yield may reflect an overpriced stock (consequence: valuation too high) or high company growth (consequence: valuation adequate), where future dividends are expected to grow.

2.3.4 Smart Beta

Factor investing has seen uplift in the past decade. What started originally with Fama & French (1992) as an extension to the *Capital Asset Pricing Model* (CAPM), which incorporated risk as the central driver of stock returns, has become its own field of research in the meantime. Today, the *MSCI Inc.* offers indices on various factors, such as *Momentum*, *Value*, *Quality* or *Size* as aggregated portfolios from subsets of a global selection of equities from the *MSCI World* index.

Hodges et al. (2017) analyze if investments into equities could be timed optimally. They subdivide the business cycle in four segments, namely expansion, slowdown, contraction, and recovery. By collecting returns conditional on these subsegments, they find that factor returns are strongly related to the business cycle, with defensive assets like *Quality* working well during slowdowns and *Value* as well as *Size* yielding the highest returns during recoveries.

This is systematic equity prediction on an aggregate level. It is unknown how many investment firms already trade on this phenomenon, but there are likely many who have been actively pricing these findings into factor stocks for a long time now. The idea is that if equity integration with commodity markets really exists, this information could be of relevance to commodity return developments as well.

As this is highly experimental and there is little research done on this topic, geometric smart beta outperformance over the *MSCI World* as defined in section 2.2 is only added to the large data model.

2.4 Aruoba-Diebold-Scotti Index

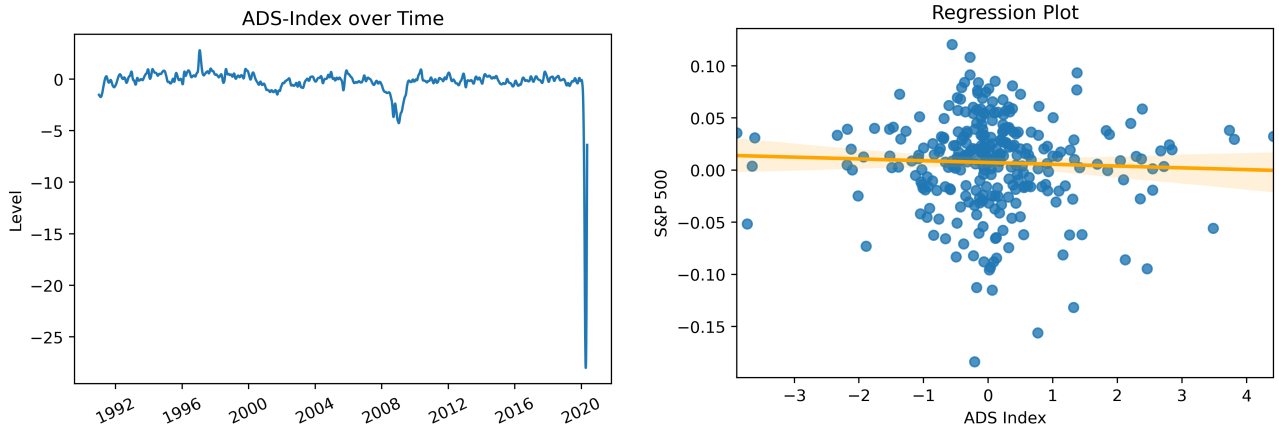


Figure 4: Changes in ADS have an unintuitive relationship with the *S&P 500*. Is it too slow?

The ADS-Index by Aruoba et al. (2009) is an attempt to obtain real business conditions at a higher frequency than GDP would allow for. It contains the following components:

1. Jobless claims (Weekly)
2. Employment (Monthly)
3. Industrial production (Monthly)

4. Real personal income less transfer payments (Monthly)

5. Real manufacturing and trade sales (Monthly)

6. Real GDP (Quarterly)

Since the index is published immediately once new information is available, it can only reflect the information available at that specific point in time, meaning, the most recent but already potentially outdated information is regularly included. Furthermore, it is a mixed sampling frequency indicator ranging from weekly to quarterly data. In any analysis of daily asset reallocations it is excluded throughout this thesis for this reason even though the provided data is updated on a daily basis.
On average, the index is normed at zero. Positive deviations indicate an improvement of overall business conditions while negative deviations signal an overall worsening. The absolute values of the index may be used for comparison of both recession and boom scenarios.
The *FED* regularly provides updated datatables publicly for free. Tharann (2019) finds strong improvements in predictability at least in-sample for the ADS-Index, but less so on new data out of sample. Looking at the US equity market, there is an apparent inverse relation between changes in ADS and returns on the *S&P 500* as illustrated above, although this is probably the opposite of what one would expect.

2.5 Other Predictors

Other predictors include macroeconomic variables such as GDP, industrial output and unemployment rates for the US as indicators of the overall state of the economy. Further higher frequency indicators included in the original dataset by Tharann (2019) are the previously outlined bond indicators such as the term spread on US government bond indices and default spreads between corporate and government bonds.
As US commodities or derivative contracts on them may also be traded by non-US investors, the FX rate on the Euro to Dollar is included in the *Extended Model* as an experimental feature. The idea behind this approach is that when the dollar becomes more expensive, long trades in the US by European investors should become less attractive. Of course, longer term there should be no difference in commodity prices between Europe and the US, but especially in the very short term this might be tradable information.

3 Predictability

The initial setup requires a brief discussion about predictability in general. If commodity prices are a Markov process and follow a random walk, then predictability on past features, meaning already observed variables, should not play a role in determining any future price movement or return. A Markov process in general characterizes a stochastic process in which the future outcome is only determined from the current value, not how one has come there (*Markov chain*, n.d.).

The second question is about the financialization of commodity markets in the early 2000's, which brought more and more financial investors on the table. The hypothesis would be that technical predictive variables gained traction and might have added systematic effects into the price movements due to systematic pricing of information (over other economic principles such as economic order quantities for industrial trade).

3.1 The Random Walk Hypothesis

The first part addresses the Markov Chain hypothesis. A quick test for a random walk, a potential consequence of at least weak market efficiency, can be performed by looking at an arbitrary set of lagged returns. If the process is truly random, then returns should not contain strong explanatory value and, hence, beta factors in a simple *OLS* regression should not be (highly) significant and R^2 should be sufficiently low. The test is conducted as follows:

$$H_0 : \begin{cases} \beta_i & = 0, \ \forall \ i \ \varepsilon \ \mathbb{W}[-n, 0[\\ \alpha & = E[r] \\ R^2 & = 0 \end{cases}$$

$$r_{t=0} = \alpha + \sum_{t=-n}^{t=-1} r_t \beta_t + u$$

Since a statistical test is likely to reject this multi-hypothesis in general as at least one part of it will probably fail, the results, which for daily data are displayed in the table below, should not be interpreted too tightly. On the contrary, each part of the H_0 is discussed separately in order to avoid this rejection bias.

	Gold_PX_LAST	Silver_PX_LAST	Platin_PX_LAST	Palladium_PX_LAST	Copper_PX_LAST
	(1)	(2)	(3)	(4)	(5)
t-1	-0.0002	-0.0103	0.0318	0.0918***	-0.0568***
	(-0.0356 , 0.0353)	(-0.0472 , 0.0266)	(-0.0065 , 0.0700)	(0.0484 , 0.1352)	(-0.0899 , -0.0237)
t-2	-0.0111	0.0114	-0.0319	-0.0271*	-0.0064
	(-0.0408 , 0.0185)	(-0.0198 , 0.0426)	(-0.0926 , 0.0287)	(-0.0581 , 0.0039)	(-0.0390 , 0.0263)
t-3	0.0107	0.0048	-0.0058	-0.0288*	-0.0093
	(-0.0211 , 0.0425)	(-0.0267 , 0.0364)	(-0.0353 , 0.0237)	(-0.0609 , 0.0033)	(-0.0429 , 0.0243)
t-4	0.0118	-0.0162	0.0199	0.0244	0.0116
	(-0.0195 , 0.0432)	(-0.0457 , 0.0134)	(-0.0173 , 0.0572)	(-0.0112 , 0.0600)	(-0.0219 , 0.0451)
t-5	0.0257	0.0068	-0.0024	0.0031	-0.0051
	(-0.0054 , 0.0569)	(-0.0242 , 0.0377)	(-0.0345 , 0.0296)	(-0.0333 , 0.0395)	(-0.0386 , 0.0285)
t-6	-0.0373**	-0.0142	-0.0166	-0.0033	0.0153
	(-0.0672 , -0.0074)	(-0.0477 , 0.0192)	(-0.0514 , 0.0182)	(-0.0330 , 0.0265)	(-0.0158 , 0.0463)
t-7	-0.0178	-0.0033	0.0005	0.0152	-0.0015
	(-0.0482 , 0.0126)	(-0.0319 , 0.0253)	(-0.0287 , 0.0298)	(-0.0179 , 0.0483)	(-0.0344 , 0.0313)
t-8	-0.0071	-0.0318**	-0.0320*	-0.0294	0.0230
	(-0.0374 , 0.0233)	(-0.0609 , -0.0028)	(-0.0643 , 0.0003)	(-0.0700 , 0.0112)	(-0.0081 , 0.0541)
t-9	0.0157	0.0142	-0.0016	0.0007	0.0129
	(-0.0129 , 0.0443)	(-0.0150 , 0.0434)	(-0.0292 , 0.0261)	(-0.0311 , 0.0325)	(-0.0185 , 0.0443)
Observations	7,649	7,649	7,649	7,649	7,649
R^2	0.0032	0.0020	0.0038	0.0113	0.0046
Adjusted R^2	0.0020	0.0009	0.0026	0.0101	0.0034
Residual Std. Error	0.0099(df = 7640)	0.0178(df = 7640)	0.0140(df = 7640)	0.0201(df = 7640)	0.0160(df = 7640)
F Statistic	1.4215 (df = 9.0; 7640.0)	0.8986 (df = 9.0; 7640.0)	0.9751 (df = 9.0; 7640.0)	2.7890*** (df = 9.0; 7640.0)	1.7080* (df = 9.0; 7640.0)
Note:					*p<0.1; **p<0.05; ***p<0.01

Table 1: Random walk? OLS regression of lagged returns.

The explained amount of the variance looks to be relatively high for both Palladium and Copper (compared to the predicted amounts in later analysis further below). We may be looking at momentum effects for Palladium and short term mean-reversal for Copper. For Palladium, the last three days seem to have a reversing impact on a significance level smaller than 10%, while for Copper only the first past observation seems to exhibit some form of relevance (on a significance level of less than 1%). The suspicion is further fueled by high significance of the F-test statistic (all regression coefficients are zero) on the 1% level for Palladium and 10% for Copper. This indicates that the R^2 is probably different from zero, too. We may assume that these metals do not necessarily follow a random walk, but instead offer a minuscule technical advantage, even if the evidence is not so strong for Copper. Although some stars are observable for the other commodities as well, any conclusion here would probably be weak and difficult to justify. We likely cannot rule out a random walk in this case.

3.2 Pre and Post Financialization

This sub-chapter analyzes the previously discussed potential relation of asset prices with increasing participation of technical and more trade-profit oriented traders from the investment sector. Industrial companies likely buy when they observe large demand for their goods and services, but commodity prices might, contrary to speculative market participants, not be their

primary concern. In essence, prices might become more subject to systematic variation due to the influx of speculative and informed traders in the early 2000s and the digitalization of trading platforms via the then rapidly growing internet.

To get a more informed picture of this issue, one may take a look at price movements and the predictive variables identified by Tharann (2019) in the regression tables below. The cut is taken arbitrarily with the *Commodity Futures Modernization Act* in December 2000. Shamsher (2021) argues [in context of the Indian market] that under the existence of financialization, integration with equity stock markets should increase. Tharann (2019) fundamentally builds his analysis and selection of variables on equity integration. Combining these two ideas, if financialization in the US is present, then the predictive variables chosen by Tharann (2019) should contain relatively more explanatory power after December 2000 then what they had before.

The following is a high level investigation if these phenomena exist within the data. By comparing two *OLS*-regressions of real-time (non-shifted) equity data on log commodity returns for all included metals we can see the effects at play for the adjusted R^2. In general, the R^2 can be interpreted as the squared empirical correlation of predicted values vs. actual observations. Typically it can be defined as

$$R^2 := \frac{SSE}{SST} = 1 - \frac{SSR}{SST}$$

with SST as the sum of squared observed, SSE as the sum of squared estimated and, finally, SSR as the sum of squared residual values. The downside to this metric is manifold. For once, the R^2 looses its interpretability in a regression without intercept, it is ambiguous in out-of-sample evaluations (negative actual correlations become positive after applying the square) and it rises with more included explanatory variables, even if those have virtually no explanatory value. The latter point is the reason why one should not take a look at pure R^2, but consider the adjusted version

$$R^2_{adj} := 1 - (1 - R^2)\frac{n-1}{n-k-1}$$

with n datapoints and k explanatory variables without the constant (Wooldridge, 2015).

Looking at the tables below, we can see negative R^2_{adj} in the pre-financialization period up until

December 2000 (with the exception of Palladium). Applying the same regression on the same dataset with later observations, a strong change in the overall picture is observable. For all regressions we can see positive R^2_{adj}. Metrics for Gold and Silver, however, do not seem to be as supportive. Overall it can be concluded that there is at least some indication of financialization in the underlying dataset.

	Gold_PX_LAST	Silver_PX_LAST	Platin_PX_LAST	Palladium_PX_LAST	Copper_PX_LAST
	(1)	(2)	(3)	(4)	(5)
ads_index	-0.5233	2.1425	-0.0924	2.5021	-0.0018
	(-4.1464 , 3.0998)	(-1.7976 , 6.0825)	(-4.8171 , 4.6324)	(-3.5271 , 8.5312)	(-5.1305 , 5.1270)
const	-0.1777	-0.0432	0.2296	-0.7888*	0.2374
	(-0.5484 , 0.1930)	(-0.8736 , 0.7871)	(-0.3644 , 0.8235)	(-1.6428 , 0.0652)	(-0.9153 , 1.3902)
default_return_spread	-0.1207	0.3962	-0.0406	0.6750	0.0580
	(-0.6396 , 0.3982)	(-1.0388 , 1.8311)	(-0.8788 , 0.7977)	(-1.1534 , 2.5033)	(-1.2669 , 1.3828)
default_yield_spread	-0.0084	0.1576	0.7320*	-0.6321	0.2982
	(-0.6177 , 0.6009)	(-1.1634 , 1.4786)	(-0.1096 , 1.5736)	(-2.2525 , 0.9883)	(-1.2047 , 1.8011)
dividend_payout_ratio	0.2082	0.3180	-0.0172	0.2458	0.1457
	(-0.1952 , 0.6115)	(-0.4727 , 1.1088)	(-0.4906 , 0.4562)	(-0.5093 , 1.0010)	(-0.7928 , 1.0842)
dividend_yield	0.3229	5.0872	-7.9048	-11.0840	0.7819
	(-7.7367 , 8.3824)	(-11.5404 , 21.7148)	(-18.4808 , 2.6713)	(-30.1868 , 8.0187)	(-15.5069 , 17.0706)
industry_output_gap	-1.3773	6.1931	-2.7932	-9.2483	1.1647
	(-5.7985 , 3.0438)	(-3.0067 , 15.3930)	(-7.9508 , 2.3643)	(-21.6297 , 3.1331)	(-6.6110 , 8.9404)
inflation	-1.0389	-0.9466	1.0155	-2.9612	2.1883
	(-3.0033 , 0.9255)	(-4.8459 , 2.9528)	(-1.6802 , 3.7112)	(-8.7558 , 2.8333)	(-1.6419 , 6.0186)
lt_us_govbond_rates	0.0227	-2.0238	3.0473**	5.6034*	0.0726
	(-2.2402 , 2.2856)	(-6.1428 , 2.0953)	(0.1544 , 5.9403)	(-0.6149 , 11.8216)	(-4.2434 , 4.3886)
stock_var	2.8585***	4.0782*	0.5605	3.8033	-0.6067
	(0.8923 , 4.8247)	(-0.2333 , 8.3898)	(-2.1550 , 3.2760)	(-1.4866 , 9.0933)	(-5.0091 , 3.7957)
term_spread	-1.7058	-3.8544	-0.6479	-2.4090	1.4102
	(-5.4200 , 2.0084)	(-9.9647 , 2.2560)	(-5.4078 , 4.1120)	(-11.9418 , 7.1238)	(-6.1051 , 8.9256)
unemployment	15.1965	-418.3083	-558.6821	962.5678	-565.2723
	(-474.1214 , 504.5143)	(-1693.8686 , 857.2521)	(-1401.8633 , 284.4991)	(-305.8324 , 2230.9681)	(-2090.7375 , 960.1930)
Observations	70	70	70	70	70
R^2	0.0959	0.1321	0.1460	0.1657	0.0443
Adjusted R^2	-0.0756	-0.0325	-0.0160	0.0074	-0.1369
Residual Std. Error	0.0361(df = 58)	0.0691(df = 58)	0.0447(df = 58)	0.0903(df = 58)	0.0708(df = 58)
F Statistic	1.3861 (df = 11.0; 58.0)	3.8030*** (df = 11.0; 58.0)	2.5194** (df = 11.0; 58.0)	1.2561 (df = 11.0; 58.0)	0.4226 (df = 11.0; 58.0)
Note:					*p<0.1; **p<0.05; ***p<0.01

Table 2: Core_Model Pre-Financialization

	Gold_PX_LAST	Silver_PX_LAST	Platin_PX_LAST	Palladium_PX_LAST	Copper_PX_LAST
	(1)	(2)	(3)	(4)	(5)
ads_index	-0.2693	-0.2289	-0.1142	1.1263***	-0.1371
	(-0.6203 , 0.0817)	(-0.8254 , 0.3676)	(-0.9195 , 0.6911)	(0.4146 , 1.8381)	(-0.8786 , 0.6043)
const	-0.0683	-0.0637	0.0066	-0.0082	-0.0001
	(-0.1755 , 0.0389)	(-0.2734 , 0.1460)	(-0.1337 , 0.1469)	(-0.2066 , 0.1901)	(-0.1496 , 0.1493)
default_return_spread	0.6604*	1.0275**	0.5812	0.3003	0.6957
	(-0.0730 , 1.3938)	(0.1467 , 1.9082)	(-0.1171 , 1.2794)	(-0.6688 , 1.2694)	(-0.8300 , 2.2214)
default_yield_spread	-0.0352	-0.0096	0.0451	-0.0082	0.0632
	(-0.1037 , 0.0334)	(-0.1379 , 0.1187)	(-0.0330 , 0.1232)	(-0.1258 , 0.1094)	(-0.0380 , 0.1643)
dividend_payout_ratio	-0.0403	-0.0746	-0.1020***	-0.0390	-0.0596
	(-0.1013 , 0.0208)	(-0.1738 , 0.0246)	(-0.1731 , -0.0309)	(-0.1317 , 0.0537)	(-0.1653 , 0.0461)
dividend_yield	1.8910	3.0283	4.0310**	3.3492	4.0766
	(-0.7416 , 4.5236)	(-1.1630 , 7.2197)	(0.8280 , 7.2340)	(-2.0225 , 8.7210)	(-1.4822 , 9.6354)
industry_output_gap	-0.5602	-0.2376	-0.2861	0.1515	0.2530
	(-1.5173 , 0.3970)	(-1.6259 , 1.1506)	(-1.7042 , 1.1319)	(-1.8022 , 2.1052)	(-1.4326 , 1.9385)
inflation	-0.5365	-1.5300**	-2.0724***	-1.7075*	-1.7938**
	(-1.3668 , 0.2938)	(-2.9740 , -0.0861)	(-3.6328 , -0.5120)	(-3.5638 , 0.1488)	(-3.2870 , -0.3006)
lt_us_govbond_rates	2.1438*	2.4943	1.4044	-1.2672	0.7597
	(-0.0455 , 4.3332)	(-1.2591 , 6.2477)	(-1.0707 , 3.8795)	(-4.6405 , 2.1061)	(-3.0136 , 4.5330)
stock_var	1.4447	0.7255	0.9207	1.3880	-3.0420**
	(-1.7030 , 4.5924)	(-2.5754 , 4.0264)	(-2.2049 , 4.0462)	(-2.6054 , 5.3814)	(-5.4189 , -0.6652)
term_spread	-0.0531	-0.4585	-0.3883	-1.6030**	-0.8200
	(-0.6934 , 0.5871)	(-1.6655 , 0.7485)	(-1.2565 , 0.4800)	(-2.9139 , -0.2922)	(-2.0509 , 0.4109)
unemployment	-20.9555	3.2985	-23.5951	63.3281	13.3482
	(-73.9169 , 32.0059)	(-85.0246 , 91.6217)	(-91.9813 , 44.7912)	(-31.4432 , 158.0995)	(-76.0591 , 102.7554)
Observations	234	234	234	234	234
R^2	0.0646	0.0637	0.1098	0.1164	0.1134
Adjusted R^2	0.0182	0.0173	0.0657	0.0726	0.0694
Residual Std. Error	0.0476(df = 222)	0.0866(df = 222)	0.0661(df = 222)	0.0966(df = 222)	0.0747(df = 222)
F Statistic	1.1382 (df = 11.0; 222.0)	1.5981 (df = 11.0; 222.0)	3.2452*** (df = 11.0; 222.0)	2.7913*** (df = 11.0; 222.0)	2.7016*** (df = 11.0; 222.0)
Note:					*p<0.1; **p<0.05; ***p<0.01

Table 3: Core_Model Post-Financialization

4 Benchmarking

In order to assess the predictive power, a benchmark on the exact same underlying data is conducted for all available models. These include *OLS* as a baseline and *Long Short Term Memory* (*LSTM*) as well as *Convolutional Neural Networks* (*CNN*) as the challenging Deep Learning architectures. The latter will be discussed in subsequent chapters.

For all three available data models (core, extended and large) as well as for all available resampling periods, models are then trained and hyperparameters tuned automatically by employing grid search (see below in section 6). Resampling in this context refers to changing the data frequency into daily, weekly, monthly and quarterly data. Then, predictions are run out of sample on the most recent 20% of the entire dataset and the overall empirical correlation on this new data (also referred to as *accuracy* and *performance*) is stored.

These results are then visualized in a chart with the *OLS* on the x-axis and the corresponding challenger model on the y-axis for each data model and resampling frequency. All models are trained in parallel to speed the process up.

The resulting charts are then analyzed whereby a 45°-line is drawn on which all points lie where *OLS* performance equals challenger performance. Points which lie above this line can be interpreted as superior models and points below are regarded as inferior.

5 Long Short Term Memory

5.1 Introduction: Neural networks

The baseline for *LSTM* starts with simple *Feedforward Neural Networks* (FFNN) which are likely first proposed by McCulloch & Pitts (1943) in the form of perceptrons. The fundamental idea is simple. A FFNN takes given input tensors in the input layer, wraps a specific activation function around the input and sends it to a set of neurons in the hidden layer with some weights in between to filter or stress a neurons output. The hidden layer takes the output from the input layer as its input and proceeds in the same way directing its output to the output layer (hidden layers are not necessary as we will see later. Multiple hidden layers are of course possible, but the danger of overfitting is increased the deeper the network becomes). This is what is commonly referred to the as the *feed-forward* process of neural networks. In supervised learning, which is what this thesis is about, the so predicted output is then compared to the real values. The discrepancy between these results is then evaluated by a convex cost function. With gradient descent, the backpropagation mechanism is then performed by adjusting the weights of the neural network in the direction of the negative gradient of the specified cost function for the given output error. An illustration is provided in Figure 5.

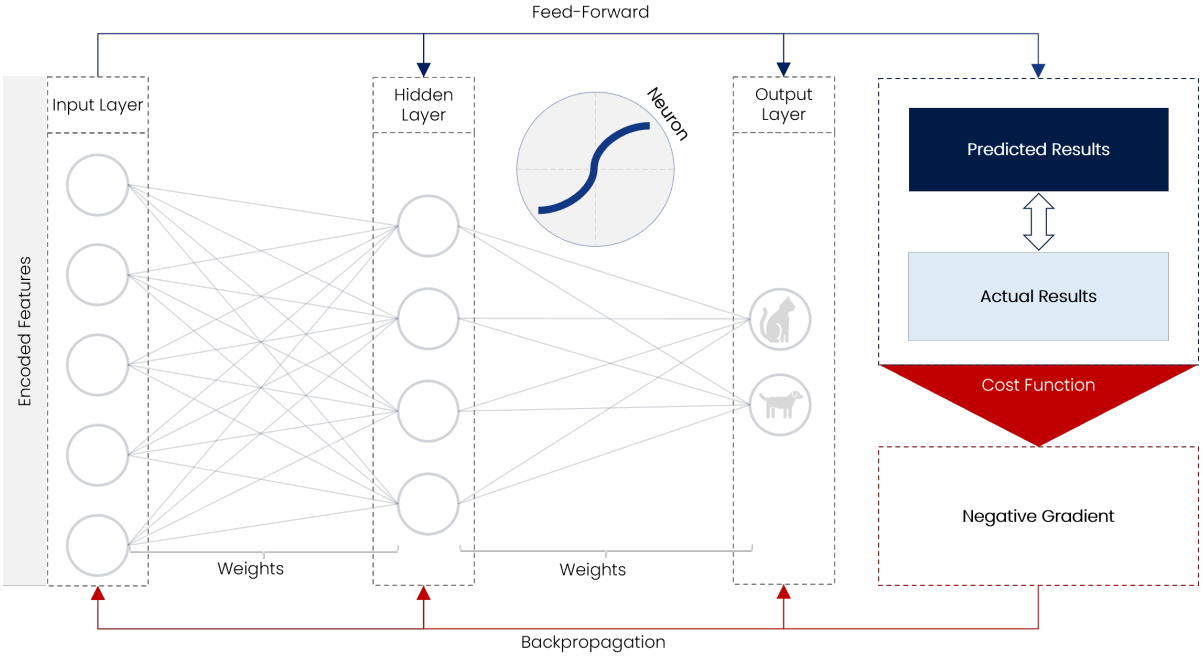


Figure 5: Sketch of a basic FFNN in a supervised learning environment featuring feedforward and backpropagation mechanisms. Biases are abstracted from this illustration.

A compact-form neural network can be denoted as:

$$h = \sigma_h(W_h x + b_h)$$
$$y = \sigma_y(W_y h + b_y)$$

If the cost function equals the euclidian distance (least squares), the activation function $\sigma_y = x$ and there is no hidden layer, the network is equivalent to *OLS* with gradient descent. The weights are then equivalents to the beta coefficients:

$$y = W_y x + b_y$$
$$C = \frac{1}{2}(a - y)^2 \rightarrow Min$$

Long Short-Term Memory on the other hand is a recurrent neural network architecture which dates back to Hochreiter & Schmidhuber (1997). Further refinements were developed in the years after by Gers et al. (2000), who introduce the forget gate and add peephole connections (from the cell to the gates). For a long time, computing limitations held *LSTM*s back. A real breakthrough then happened in 2015, when Google (Danko, 2015) announced they would be using them in their language models from now on. Another famous implementation of *LSTM* was the Starcraft-AI AlphaStart developed by DeepMind who stroke a 100% victory rate over the then worlds leading e-sports team *Team Liquid* in 2018 (Stanford, 2019), challenging the public afterwards and continuing to win almost every match. Today, following market wide adoption, *LSTM*s have become an industry standard for natural language processing (NLP), as the extensive toolkits NVIDIA Nemo (*NVIDIA Nemo: A toolkit for conversational AI*, 2019) and Kaldi ASR (Povey et al., 2011) show. *LSTM*s could be particularly interesting for time series because of their recurrent nature, possibly allowing to identify repeating structural patterns within the financial data we are working with in this thesis.

5.2 Formal Foundations

5.2.1 Compact Recurrent Forms

As discussed, *LSTM* is an extension to simple recurrent neural networks, which are a class of artificial neural networks initially proposed by Elman (1990):

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

whereby

x_t : Input tensor

h_t : Hidden layer

y_t : Output tensor

W : Weight parameter

U : Recurrent weight parameter

b : Bias parameter

σ : Activation functions

An obvious problem with these models is their arbitrarily long tracking of time. While it might only seem to be $t-1$ initially, every iteration contains information of the past iteration. As long as the weights in between the neurons are not zero, everything will be carried forward indefinitely. This may not a good solution for financial time series, especially since the structure of events is unknown. Furthermore, gradients in recurrent neural networks can explode or vanish the farther the included observations reach back in time (discussion in the subsequent section 5.2.2).

For these reasons, *LSTM*s introduce several gates (Gers et al., 2000):

$$
\begin{aligned}
f_t &= \sigma_f(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_i(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_o(W_o x_t + U_o h_{t-1} + b_o) \\
c_t^i &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ c_t^i \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

whereby

f_t : Forget gate

i_t : Input gate activation

o_t : Output gate activation

c_t^i : Cell input activation

c_t : Cell state

5.2.2 Backpropagation and Gradient Descent

Fundamentally, the central problems of recurrent neural nets are exploding or vanishing gradients, which makes them generally unable to handle long time dependencies. Consider the simple recurrent net denoted above. The recurrent component is located in the hidden layer:

$$h(t) = \sigma(W_h x(t) + U_h h(t-1) + b_h)$$

In order to perform backpropagation process, one needs to calculate the loss or cost function C at any step $t = T$ with respect to the hidden state of an earlier step $t = \tau$. To obtain the

gradient of the loss $C(t)$, the derivative needs to be computed:

$$\frac{\partial C(t=T)}{\partial h(t=\tau)} = \frac{\partial C(t=T)}{\partial h(t=T)} \prod_{\tau<t}^{T} \frac{\partial h(t)}{\partial h(t-1)}$$

In general, backpropagation through the network by gradient descent would be performed by updating the weights for a given learning rate λ:

$$W_{new} = W_{old} - \lambda(\frac{\partial C(t=T)}{\partial h(t=\tau)})$$

The Jacobian of the hidden layer at any given time t can be obtained by:

$$\frac{\partial h(t)}{\partial h(t-1)} = diag(\frac{\partial \sigma(U_h h(t-1) + W_h x(t) + b_h)}{\partial U_h})U_h$$

The derivative of the cost function may then be re-written as:

$$\frac{\partial C(t=T)}{\partial h(t=\tau)} = \frac{\partial C(i)}{\partial h(T)} U_h^{T-\tau} \prod_{\tau<t}^{T} diag(\frac{\partial \sigma(U_h h(t-1) + W_h x(t) + b_h)}{\partial U_h})U_h$$

Note that the weight updating process fundamentally relies on this computed gradient. However, the gradient includes the weight matrix $U_h^{T-\tau}$, which is now multiplied by itself for $T-\tau$ times. This multiplier captures the two quintessential problems of recurrent neural nets, namely, the exploding gradients for weights > 1 and the vanishing gradients for weights < 1. A more in-depth description of this problem is provided by Pascanu et al. (2013).

One solution to the vanishing gradient was published by Gers et al. (2000) who introduce forget gates. The idea is to filter out information which is not anymore required since it has become too old. These forget gates correspond to f_t in the compact example above. The necessary equations for implementing backpropagation with a forget gate are given by the following equations (Gers et al., 2000, p. 6ff, notation adapted) with *out* and *in* describing a layers output

and (recurrent) input gate, respectively:

$$\Delta w_{s,d} = \lambda \frac{\partial y^d(t)}{\partial net_d(t)} (\sum_k \frac{\partial y^k(t)}{\partial y^d(t)} e_k(t)) y^s(t-1) \tag{1}$$

$$\frac{\partial \Delta y^k(t)}{\partial \Delta net_k(t)} = \sigma'_k(net_k(t)) \Rightarrow \delta_k(t) = \sigma'_k(net_k(t)) e_k(t) \tag{2}$$

$$\delta_{out_j}(t) = \sigma'_{out_j}(net_{out_j}) (\sum_{v=1}^{S_j} h(\varsigma_{c_j^v}(t)) \sum_k w_{kc_j^v} \delta_k(t)) \tag{3}$$

$$e_{\varsigma_{c_j^v}}(t) = y_{out_j}(t) h'(\varsigma_{c_j^v}(t)) \tag{4}$$

$$\frac{\partial \varsigma_{c_j^v}(t)}{\partial w_{c_j^v s}} = \frac{\partial \varsigma_{c_j^v}(t-1)}{\partial w_{c_j^v s}} y^{f_j}(t) + g'(net_{c_j^v}(t)) y^{in_j}(t) y^s(t-1) \tag{5}$$

$$\frac{\partial \varsigma_{c_j^v}(t)}{\partial w_{w_{in_j} s}} = \frac{\partial \varsigma_{c_j^v}(t-1)}{\partial w_{in_j s}} y^{f_j}(t) + g(net_{c_j^v}(t) y^{in_j}(t)) y_s(t-1) \tag{6}$$

$$\frac{\partial \varsigma_{c_j^v}(t)}{\partial w_{f_j s}} = \frac{\partial \varsigma_{c_j^v}(t-1)}{\partial w_{f_j s}} y^{f_j}(t) + h(\varsigma_{c_j^v}(t)) \sigma'_{f_j}(net_{f_j}(t)) y^s(t-1) \tag{7}$$

$$\frac{\partial \varsigma_{c_j^v}(t=0)}{\partial w_{sd}} = 0 \text{ for d } \epsilon \{f, \text{in}, c_j^v\} \tag{8}$$

$$\Delta w_{c_j^v s}(t) = \lambda e_{\varsigma_{c_j^v}}(t) \frac{\partial \varsigma_{c_j^v}(t)}{\partial w_{c_j^v s}} \tag{9}$$

In order to perform the backpropagation process, weights can then be updated as follows:

$$\Delta w_{sd}(t) = \lambda \sum_{v=1}^{S_j} e_{\varsigma_{c_j^v}}(t) \frac{\partial \varsigma_{c_j^v}(t)}{\partial w_{sd}} \text{ for d } \epsilon \{f, \text{in}, \}$$

whereby

s, d, k : Model output unit (index = k)

j : Memory block index

v : Memory cells in block j

ς : Internal states

$e_{\varsigma_j^v}$: Internal states error at j, v

in : Input gate

out : Output gate

net : The unactivated model output

Only equations (5), (6), and (7) need to be kept in RAM for each iteration during the training process. However, this way of performing gradient descent on all available data is computationally expensive. Therefore, *Stochastic Gradient Descent* (SGD) with momentum is used in the final implementation. SGD works with random subsamples instead of considering the full dataset at each step and can, thus, be calculated faster (note that the Jacobians dimension increases with the amount of included observations).

5.2.3 Activation functions

In the original proposal of neural networks as perceptrons by McCulloch & Pitts (1943), neurons were binary activators ϵ $\{0, 1\}$. Today, more sophisticated functions have replaced these simple structures although their key purpose still remains the same. There are, generally, linear and nonlinear activation functions, with the non-linear ones aiming to enable a network to learn more complex mappings. A multitude of activation functions are available, but one must chose carefully, since they can make a strong difference in reliability and rigidity of any neural model. Changing the output activation function in the final Dense-layer in this thesis' implementation may cause some models to completely lose their ability to learn anything at all (jumping gradients). This chapter discusses available candidates about their suitability to represent financial returns. More specifically, good functions exhibit the following properties:

1. Are differentiable at any given point.

2. Are continuous and monotonous for any given input.

3. Allow negative returns.

4. Are unlimited or at least allow for sufficiently large positive returns.

5. Are lower-bound at -1 (returns cannot exceed -100%).

The original perceptrons binary activation (BIN) can be defined as:

$$\sigma_{bin} = \begin{cases} 1, \text{ if } ax + b > 0 \\ 0, \text{ else} \end{cases}$$

This may be a good activation function if the target were to categorically distinguish between positive and other returns. However, this is not the case and therefore its binary nature makes it unsuitable for specific return prediction. A prominent and often-used continuous example would be the logistic function (LOG) defined as:

$$\sigma_{log} = \frac{1}{1 + e^{-x}}$$

which is unsuitable due to its nature of predicting positive returns only. A function which satisfies properties (1) to (3), (5) would be the *Tangens Hyperbolicus* (TANH):

$$\sigma_{tanh} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

For $x \to \infty$, σ_{tanh} approaches 1 while for $x \to -\infty$, σ_{tanh} is limited to -1. A good application of this function may be the recurrent states within *LSTM*, since extreme shocks might not be carried forward for too long. Limiting these extreme values at the upper bound as well could therefore be a reasonable choice. However, for predicting returns themselves we would like to capture extreme shocks once they may become likely. Thus, σ_{tanh} is not necessarily the best option for the final Dense-layer.

The *Exponential Linear Unit* (ELU) is probably one of the better suited functions and satisfies all properties from (1) to (5) at once as illustrated in Figure 6:

$$\sigma_{elu} = \begin{cases} \gamma(e^x - 1), \ if \ x \leq 0 \\ x, \ else \end{cases}$$

1. The functions individual parts are differentiable: $\frac{\partial \sigma_{elu}}{\partial x} = \begin{cases} \gamma e^x, \ if \ x < 0 \\ 1, \ else \end{cases}$
2. The functions individual parts are continuous and strictly increasing. At $x = 0$ both parts are equal to 0, therefore the function by itself is continuous as well.
3. If $x < 0$, the activated output for input close to zero will be lower. For $x \geq 0$ no change will be applied.
4. Activated output < 0 is allowed if $\gamma > 0$.
5. $\sigma_{elu} = \infty$ if $x \to \infty$, thus, infinite returns are indeed possible.
6. $\sigma_{elu} = -1$ if $\gamma = 1$ and $x \to -\infty$

Since it reflects financial returns most appropriately, ELU with $\gamma = 1$ is the only used activation function of the final Dense Layer which decodes the model predictions back to returns in all neural implementations.

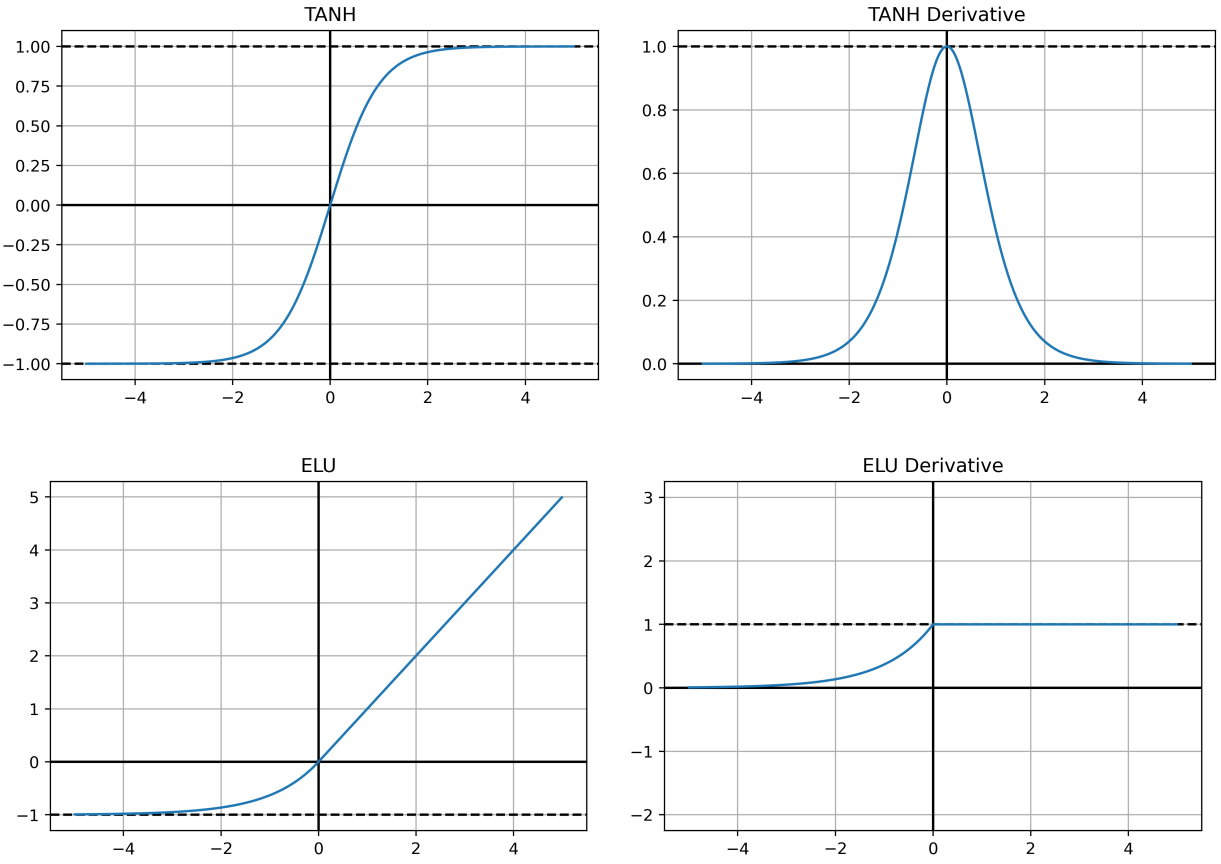


Figure 6: TANH vs. ELU

5.2.4 Data Preparation and Dimensionality

The model follows the standard setup for time-major implementations and therefore expects a three dimensional tensor with the timesteps in the first, the observations in the second and the features in the third dimension. First, the train- (60%), validation- (20%) and testsets (20%) (specific split is configurable in the central config file *config.yml*) are sliced from the original data. Since neural models benefit from feature scaling, two scalers are available. A max-scaler, which limits the maximum amplitude of each data column to 1 as well as a standard scaler, which applies the central limit theorem and converts data to a standard-normal distribution by $\frac{x-\mu}{\sigma} \sim N(0,1)$. The standard scaler is what is used throughout this thesis, although max-scalers are common as well, since they leave the original data structure unchanged. Feature scaling is critical as the activation functions are designed to work with data especially around $\mu = 0$ and $\sigma^2 = 1$.

Using two dimensional data as a *Pandas DataFrame*-object, the method *lag_leads* adds the

required lags to the dataframe containing the input features from the specified data model. Its output is then passed on to the function *to_3d*, which converts the *Pandas DataFrame* to a *Numpy* array. To bring the result into the required three dimensions, the array is then reshaped to have the introduced lags in the first, each observation in the second, and finally the original features in the third dimension.

Due to the recurrent nature of the model, the data is intentionally not reshuffled, since one of the key goals and anticipated benefits over static linear inference is to have *LSTM* identify patterns and structure within the noisy data automatically.

5.2.5 Model Specification

The model itself consists of three layers: The input layer takes the *Numpy* array and computes the input tensor, which is then passed on to a single *LSTM*-kernel, which encodes the returns into a configurably sized, one-dimensional tensor. A single *Dense*-layer then decodes the encoded return vector back into a single return using the ELU activation function. The backward pass is then performed on the validation data to ensure a sanity check in each epoch. The cost function is the euclidean distance between the predicted and observed real returns of the next period.

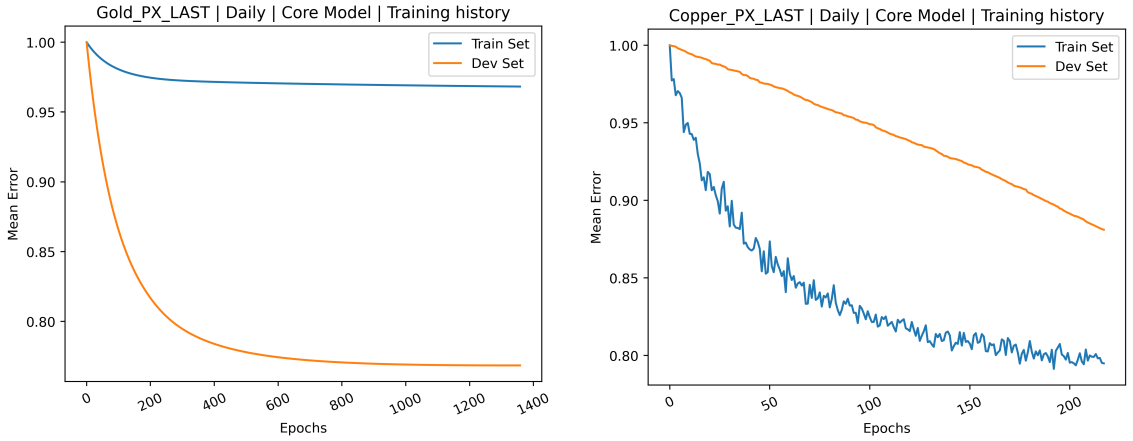
Hyperparameters are available for all three data models as well as for daily, weekly, monthly and quarterly asset reallocation periods. They can be either parsed per model as a *Python* dictionary or, alternatively, they are loaded from *lstm_hyperparams.yml*, where they can be modified in the preset *Yaml* structure.

The class may be used to self-infer alternative hyperparameters which are stored in a dedicated folder (default is "Export/Hyperparameters/LSTM/*") as *Json* or *Yaml* formats and can be sideloaded by adding them as a custom specification when running the training process.

5.2.6 Regularization: Avoiding Overfits

Multiple procedures are in place within the *Lstm* class as well as in its companion functions to avoid over-fits. An over-fit describes a situation in which, given a fixed reference point, prediction in sample improves, but accuracy out of sample deteriorates.

The first measure is the utilization of validation on new data during the training process. This provides a sanity check for the model after each completed epoch. Structural over-fitting, or



(a) Stable learning

(b) Short term instability with overall improvements

Figure 7: Learning curves for two commodities on the Core Model with daily data.

learning by heart, might induce performance deterioration on new data.

Training for too long in general can reduce generalization as well. While each model is preconfigured to be allowed a maximum of 10,000 epochs, a second trigger stops the training process prematurely should the model continue to worsen for more than a predefined number of 25 consecutive iterations on the validation set. Since we are already using the validation data as a sanity check, errors on the training data are not straightforward as well. Thus, early stopping is applied on the training set, too, but with 50 rather than 25 observations of continuous stagnation or worsening to allow for short term volatility.

Dropout layers are utilized on both the recurrent model states as well as the original data. They deactivate a randomly drawn set of neurons and therefore augment the data by inducing noise. This should, ideally, enable the model to filter out noise when loading it for predictions. While a good model should be structurally rigid against dropouts, it must be noted that financial data is noisy already and, therefore, not all models may benefit from this procedure. How many neurons are selected can be configured as well.

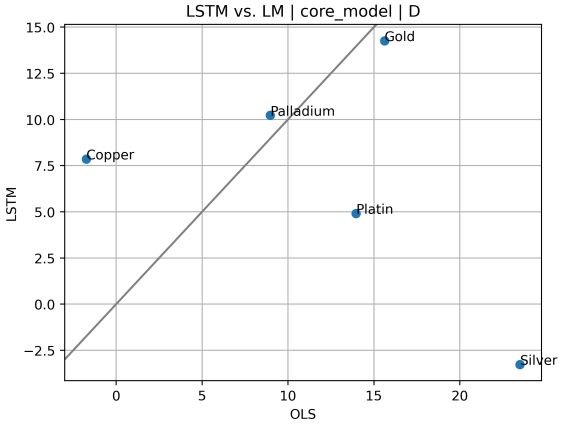
5.3 Results

While *LSTM*'s predicted returns were convincing initially, they only barely stood a robustness test on new data and asked for many additional requirements in the tuning process (such as minimum improvements in learning and a minimum amount of trained epochs). The models validation set consists of entirely new data, and, with the right choice of hyperparameters, learning was somewhat reliable with improving mean absolute percentage errors and euclidean distances on both the train- as well as on the dev- set. However, the test data then showed higher performance erosions than expected ex ante.

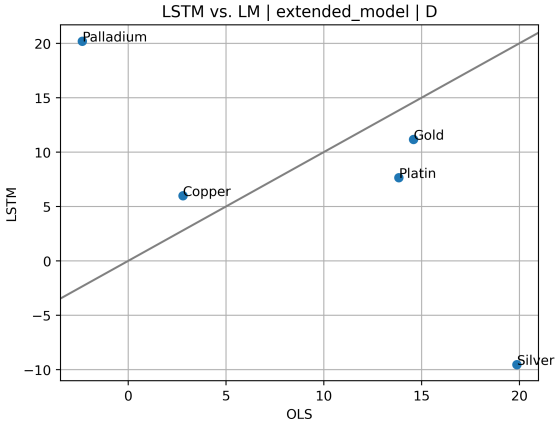
Even though training mostly seemed to improve continuously longer term, some models did not fare so well. Often they improved very little or the early stopping triggers quit the process after a few epochs. Model initialization has the biggest impact on model performance. Merely the starting point defined by the applied seed value often decided upon success or failure as even unreliable models may train for a long time, but optimize a flat local minimum instead of finding a steep local or global optimum. For this reason, training has to be performed multiple times on a range of given seed values in order to find the one that lies closest to a deep minimum (it is not at all clear if the algorithm is really capable of detecting a global minimum). Hence, models need to be executed in the aforementioned hyperparameter training environment which self-monitors the learning behavior and picks the best out of the many runs as the valid model specification.

Although each training process is run in a separate process-container and therefore many seed values can be trained in parallel on a multi-cpu server or in a multi-gpu setup, generally, finding the optimal model still takes a substantial amount of time. Stability issues with consumer grade machines and time required for training per period also drive backtesting out of scope for this thesis. A prototype of a backtest implementation is supplied in the prototyping section of the source code.

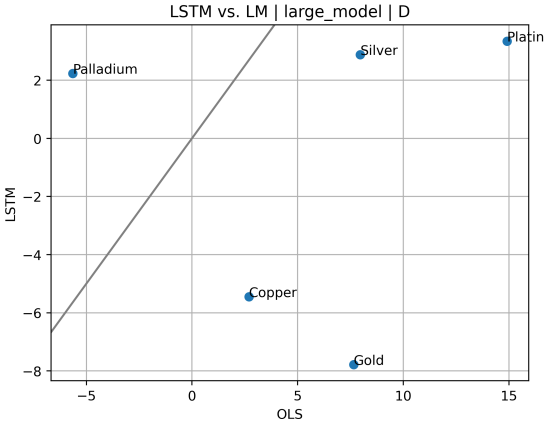
5.3.1 Daily Rebalancing



(a) Core Model



(b) Extended model

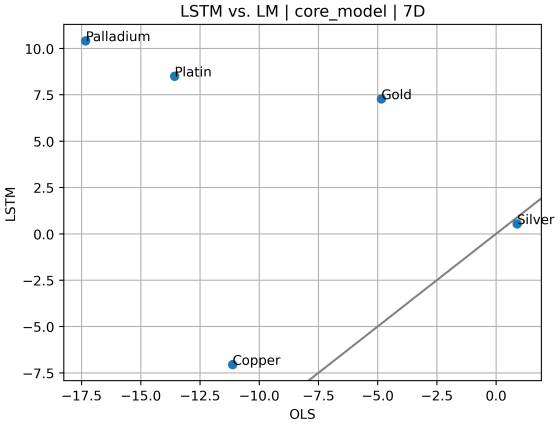


(c) Large model

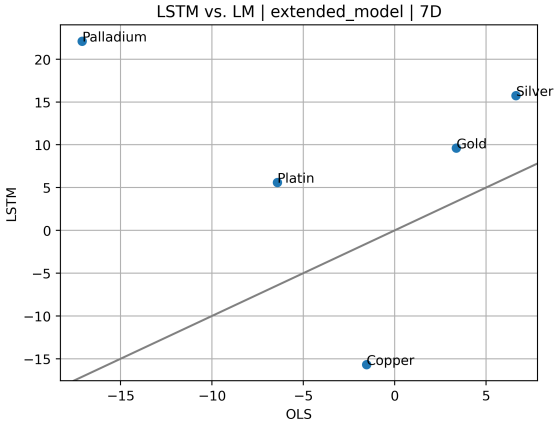
Figure 8: LSTM Benchmarks on daily data

On daily data, all models aside from Silver deliver an overall positive correlation on new data with the restricted dataset. Palladium, to no surprise based on the indications seen in previous chapters, performs well across all three data models. Overall, however, the additional features in the extended data model did not enhance predictability consistently. The *Large Model* with smart beta signals even deteriorated the predictive power of the underlying models quite strongly. Nevertheless, the *LSTM*-models are not better than pure *OLS* with the exception of Silver in the *Extended* and *Large Model*, which performs a lot less than the *OLS*-benchmark. The original dataset by Tharann (2019) (*Core Model* without non-daily features) seems to work best with *LSTM*.

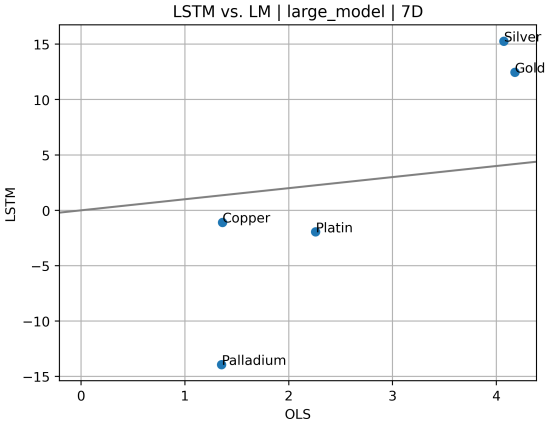
5.3.2 Weekly Rebalancing



(a) Core Model



(b) Extended Model

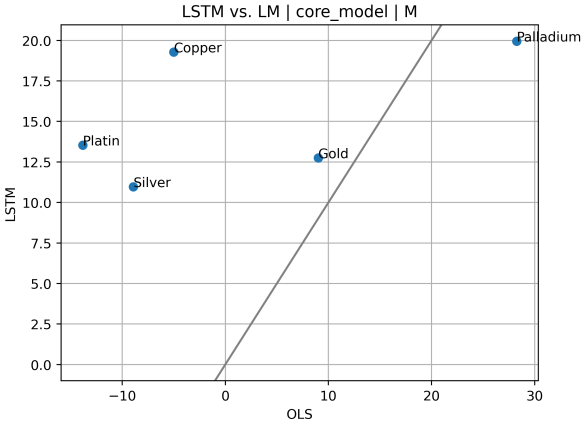


(c) Large Model

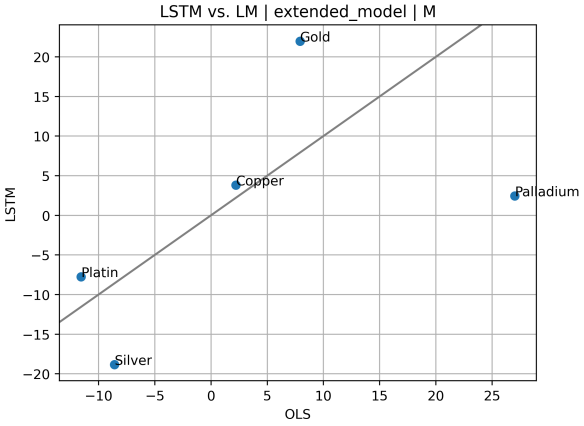
Figure 9: LSTM Benchmarks on weekly data

Similar results as for daily data can be observed on weekly resampling as well. The *Core Model* delivers the most stable performance, with Palladium even reaching an out of sample predictive power of more than 30% with *OLS*. Interestingly, introducing the additional datapoints deteriorated predictive power of *OLS* while simultaneously enhancing *LSTM*, with Palladium showing an increase by a factor of two from the core to the *Large Model*. However, the differences are, with the exception of Palladium, not too large.

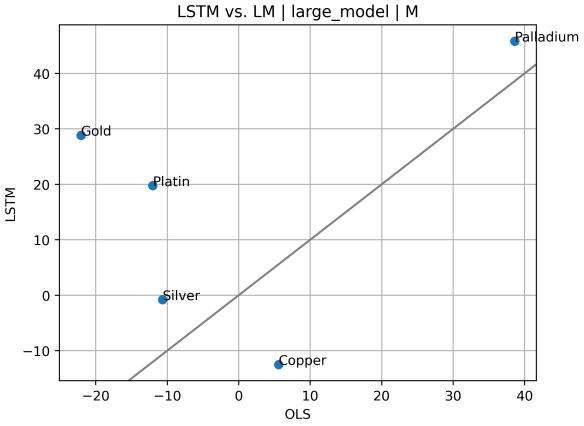
5.3.3 Monthly Rebalancing



(a) Core Model



(b) Extended Model

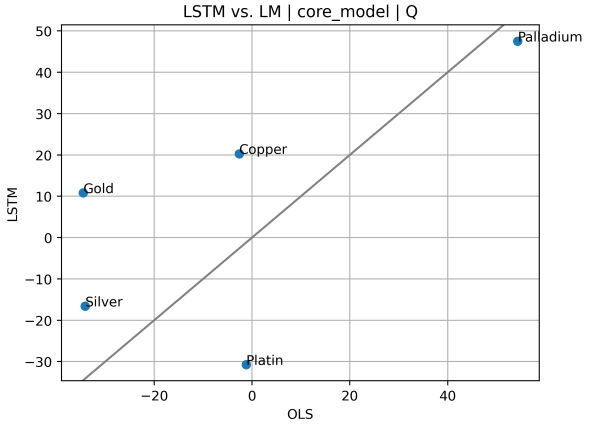


(c) Large Model

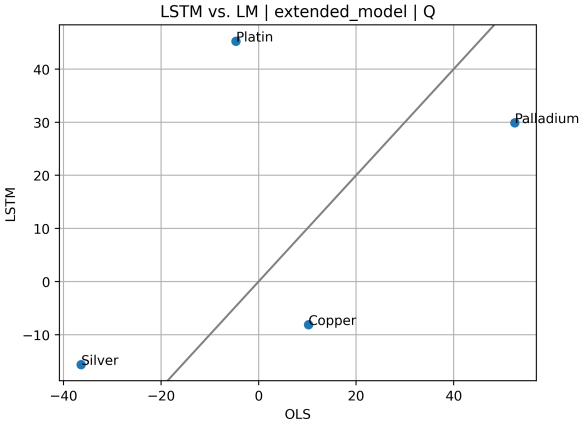
Figure 10: LSTM Benchmarks on monthly data

The story is continued when looking at monthly re-balancing. At this frequency, lagged economic variables such as industrial output as well as the ADS-index are included as well. Predictability of asset returns is relatively solid across all metals for *LSTM* on both the core and the *Large Model*, but not so much for *OLS*. It seems as if *OLS* continues to work well with Palladium. Again, the larger datasets seem to benefit mostly *OLS*, not so much *LSTM*. Smart Beta apparently continues to contribute value to Palladium only, bumping up the *OLS* to almost 0.4 correlation out of sample.

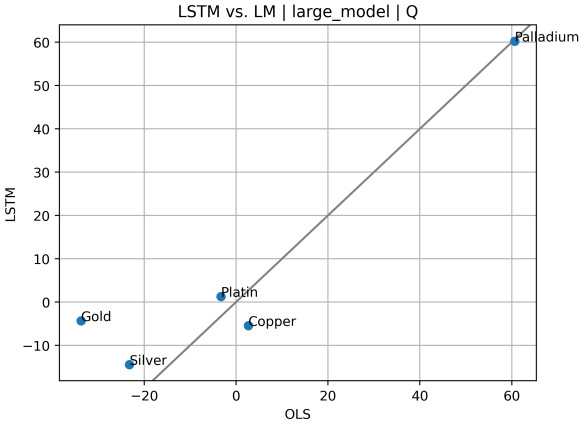
5.3.4 Quarterly Rebalancing



(a) Core Model



(b) Extended Model



(c) Large Model

Figure 11: LSTM Benchmarks on quarterly data

Resampling to quarterly data reduces the amount of available datapoints drastically. One would suspect a neural network's performance to be low in such environments, as they usually benefit from larger amounts of data. Indeed, from all frequencies this is definitely a weaker supporter for *LSTM*, especially on the larger models. The accuracy between predicted and actual returns is close to zero for the *Large Model*, with the notable exception of Palladium.

6 Excursion: A Defense of Grid Search

Grid search is a relatively simple but computationally expensive way of tuning hyperparameters. The underlying principle is easy to code:

1. Define a set of hyperparameters.
2. Specify an initial set of hyperparameters to start from.
3. Define an evaluation set (usually this consists of both train and validation data).
4. Define a set of criteria to be evaluated.
5. For each defined hyperparameter: Train & save the model.
6. For each hyperparameter: Run predictions on the evaluation set.
7. Compare the results and select the best.
8. Repeat (2) to (8) until improvements become small enough.

This is cost-intensive when working with big data (such as audio or video, since a single training cycle without transfer-learning and fine-tuning can take weeks or even months), but for tabular data such as is used throughout this thesis, a modern PC is capable of running a single cycle in a few hours. Furthermore, after a few cycles there are no more improvements observable and since the already optimized parameters are taken in later runs on newer data, good results seem to be achievable within a single cycle.
This is not to say that there are better (less expensive) methods available. Both *LSTM* as well as the *Convolutional Neural Network* (*CNN*) were trained on other algorithms using *ray[tune]* by Liaw et al. (2018). But, preemptively, optimization in context of the data used in this thesis seems to be not as straightforward, maybe due to the low overall explanatory power of the predictive variables. Grid search is the utilized algorithm for model tuning throughout this thesis, therefore.

6.1 Random Search

One alternative is to pick random combinations more systematically using probability distributions. The idea was brought forward in the early 2010s with one notable example being Bergstra & Bengio (2012), who showed that random search can, given the same budget,

1. find models that are as good or better than grid search
2. at a fraction of the computational power.

This result is effectively achieved by searching through a smaller set of hyperparameters. Their argument is that predictive power is mostly impacted by a fraction of the hyperparameters available and, ergo, model optimization should follow primarily these important ones rather than wasting time and energy on parameters which deliver little overall improvements. However, they acknowledge that grid search

1. is simple to implement (with modern libraries this advantage does not hold anymore),
2. can be easily parallelized,
3. is reliable in low dimension spaces.

Especially the latter two arguments are likely the reason why random search turned out to require a lot more time in this thesis' setting. With grid search and GPU- as well as CPU-bound training processes up to 16 parallel cycles could be run on a consumer desktop machine. With more capable hardware and or distributed computing (using tools such as *gRPC* or *Docker Swarm*), a cycle may be completed within the time for the longest individual training and evaluation process (i.e. just a few minutes) using grid search. Since random search takes inference on previous results, parallelization in practice may be limited.
Using *ray* tune with random search did produce inferior results for both *LSTM* as well as *CNN* on given data (with mostly even negative correlation of actual vs. predicted returns).

6.2 Bayesian Optimization

Another alternative to grid search would be Bayesian Optimization. One such Python implementation is provided and maintained by Nogueira (2014). To obtain a deeper understanding, the following description is loosely based on Frazier (2018). *Bayesian Optimization* tries to find a maximum value of an underlying continuous utility function $U(x)$, which is unknown at the time of execution, with a predefined budget of N steps. However, not only is the utility function itself a black box, but also its properties such as the first and second derivatives are unknown. Potentially observable values for $U(x)$ also may or may not be noisy, adding further complication. In order to infer the nature of this utility function, tools such as a *Gaussian Process Regression* (Rasmussen, 2003) may be utilized. The idea is that by varying the hyperparameters, more and more observations are collected to increase the degree of the search algorithm's certainty which of the underlying estimated candidates match the observed results best. But how to determine a function from which so far nothing is known of? First, one has to define a starting point x_0 with an arbitrary set of hyperparameters. The goal will be to utilize an acquisition function $A(x)$ to determine where the next best location x_1 for evaluating the unknown function (closer to its *MAX* or *MIN*) is located. Contrary to U, one knows the first and second order derivatives of A and it is comparatively easy to evaluate. First, one has to choose an appropriate *Gaussian Process* (GP) to determine the probability that any given output matches the actual output of U:

$$P(U) = GP(U; \mu; \sigma^2)$$

Let the underlying distribution of observations be O_{dist}. The conditional probability, hence Bayesian Optimization, of U for any observed distribution of O_{dist} can then be denoted as:

$$P(U|O_{dist}) = GP(U; \mu_{U|O_{dist}}; \sigma^2_{U|O_{dist}})$$

At first sight we have now replaced a complex optimization problem with a simpler one, but what does it actually imply? In general, the goal will be to move where the expected improvement $E[U^* - U(x)]$ is the largest, with U^* denoting a hypothetical optimum of the underlying utility function. This is of course knowledge one does not possess but may try to statistically approximate. Many functions may be used to achieve this such as the *Probability of Improvement*,

in accordance with Brochu et al. (2010) defined as:

$$
\begin{aligned}
PI(x) &= P(U(x) \geq U(x^*)) \\
&= \Psi(\frac{\mu(x) - U(x^*)}{\sigma(x)}) \\
\Psi(x) &= \frac{1}{2}(1 + erf(\frac{x - \mu(x)}{\sigma\sqrt{2}})) \\
erf(x) &= \frac{2}{\sqrt{\pi}} \int_0^x e^{k^2} dk
\end{aligned}
$$

Note that Ψ denotes the cumulative normal distribution function. The acquisition function will be evaluated multiple times on all available datapoints and the lowest (or highest, depending on A) result is chosen as the valid one. It should be noted that while *Probability of Improvement* is one of the earliest and simplest applied functions, it may get stuck in local optima. Therefore, more sophisticated acquisition functions are utilized in modern libraries such as the one by Nogueira (2014).

The next sample will then be drawn from the result of the acquisition function. Note that the purpose of A is not to find the optimum hyperparameter, but merely the next step one should try in order to move closer to the optimum. The target utility function U itself is modeled by the Bayesian statistical model, not the acquisition function. A cooking recipe for this process may look like this (Frazier, 2018):

1. Model U by defining a *Gaussian Process*.
2. Initialize the experiments environment on starting parameters (may be drawn at random).
3. Run training & inference on all available data with the current set of hyperparameters.
4. Update beliefs about the distribution of U based on these results.
5. Obtain the optimum of the acquisition function based on these updated beliefs.
6. Infer the probability via your statistical model for U.
7. Update your hyperparameter. Repeat steps (3) to (6) for a total of $N - 1$ times.
8. Return the hyperparameter that maximizes the statistical model for U.

Implementing *Bayesian Optimization* turned out to be challenging but unfruitful. Ultimately, results were inferior to *Grid Search*.

7 Convolutional Neural Networks

7.1 Introduction

Convolutional neural networks (*CNN*) are a class of deep artificial neural nets initially proposed as a *neocognitron* by Fukushima (1988) and primarily extended and formalized by LeCun et al. (1999). Today, they have become an industry standard on image processing, computer vision and audio encoding for speech recognition. Contrary to the previously described *Long Short Term Memory*, they are not recurrent but advanced feed-forward nets or multilayered perceptrons. Fitting them can be challenging as they are prone to overfitting. Each neuron within a multilayer perceptron is generally, with exceptions such as of dropout layers or deactivated connections, tied to each other neuron in the net as long as its weight $w_{s,d}^{k} \neq 0$.
But since they are *Feedforward Neural Nets*, ultimately, they are less analytically invovled and can therefore be implemented in a more simple and straightforward fashion. They are not inspired by a specific analytic problem, such as the vanishing gradients of *RNN* with *LSTM*, but are a direct analogy to biology. Namely, the detection hierarchy of the visual cortex as shown in Bruce et al. (1981). Neurons seem to, first, extract high level features from simple stimulus patterns, defining the broad locations of interest on the eye-retina. From these positions, the network then gradually moves out and integrates further away and more complex structures into the existing ones. Cells which are farther away from the originally detected pattern also tend to cover a wider area. Both Sato et al. (1980) as well as Bruce et al. (1981) found evidence for this procedure to happen in monkey brains, although further details of this paper are spared from this thesis due to the questionable procedures involving torturing [both], killing and disecting [Sato et al. (1980) only] the animals during the research process.

7.2 The Convolutional Kernel

7.2.1 Causal vs. Non-Causal

The key difference of *Convolutional* from *Feedforward Neural Nets* is the application of the convolutional kernel. On time-series data this can be thought as sliding a window over the input layer and looking at the opening at each point in time. This can be done causally or a non-causally as explained below and interpreted in 7.2.2, with non-causal kernels being the default.

The non-causal approach is a naive cross-correlation detection technique:

$$y(n) = \begin{cases} \sum_{i=0}^{k} x(n+i)h(i), \ if\ n = 0 \\ \sum_{i=0}^{k} x(n+i+(s-1))h(i), \ else \end{cases}$$

whereby:

x : Convolutional layer

n : Length of x

h : Kernel of x

k : Length of h

s : Strides (shift kernel by s)

y : Output

The causal approach can be denoted as:

$$y(n) = \begin{cases} \sum_{i=0}^{k} x(n-i)h(i), \ if\ n = k\text{ - }1 \\ \sum_{i=0}^{k} x(n-i+(s-1))h(i), \ else \end{cases}$$

(Srinivasamurthy, 2018, p. 7ff)

7.2.2 Interpreting Kernels

One observation from the above equations for the non-causal kernel is the not necessarily equal input and output tensor lengths. Keras refers to this as its *valid* mode. Equal tensors can be enforced using the *same* padding parameter in *cnn_hyperparams.yml*. Furthermore, the non-causal kernel relies on future outputs at any given point which induces a look-ahead-bias in context of financial data. Therefore, the causal kernel is used in the implementation. The *padding* hyperparameter to achieve this is set to *causal* in *tensorflow/keras*. The future time-dependency of the non-causal kernel may not be obvious without an example. Consider the case where n = 4, k = 2, and s = 1. For the non-causal kernel, this would result in the following

sequence:

$$y(0) = x(0)h(0) + x(1)h(1)$$
$$y(1) = x(1)h(1) + x(2)h(2)$$
$$y(2) = x(2)h(2) + x(3)h(3)$$

For the causal kernel, this would produce the following sequence:

$$y(1) = x(1)h(0) + x(0)h(1)$$
$$y(2) = x(2)h(0) + x(1)h(1)$$
$$y(3) = x(3)h(0) + x(2)h(1)$$

Clearly, the non-causal approach does not require information about future features already today.

7.2.3 Pooling and Flatten Layers

The output of any *Conv1D*-layer is not in the correct dimensionality to allow convenient interpretation. The pooling layer in Chollet et al. (2015) serves to reduce the dimensionality in order to be able to process it further with a densing layer to a log commodity return. In this thesis' implementation, max pooling is utilized. Here, a window of size ζ slides forward over the ouput of the *Conv1D*-layer with strides s to obtain the max value after each pooling operation. The dimensionality of of the max-pooling's input layer fundamentally depends on the configuration of strides and kernel of the *Conv1D*-layer as outlined in the chapter above. To shape the result into a one-dimensional array, a *Flatten*-layer concatenates the dimensions accordingly.

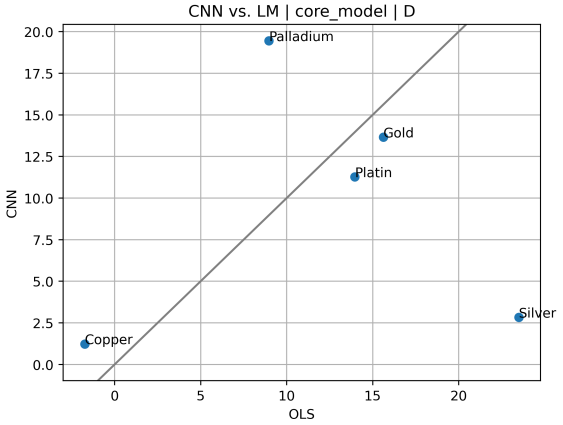
7.3 Activation Functions

For *CNN*s final *Dense* layer the same reasoning applies as for *LSTM*. Financial returns on real assets should not exceed -100% while positive returns are theoretically unlimited. Throughout the implementation, unless otherwise specified, *ELU* is the appropriate activation function. Note that since *CNN*s do not have recurrent layers.

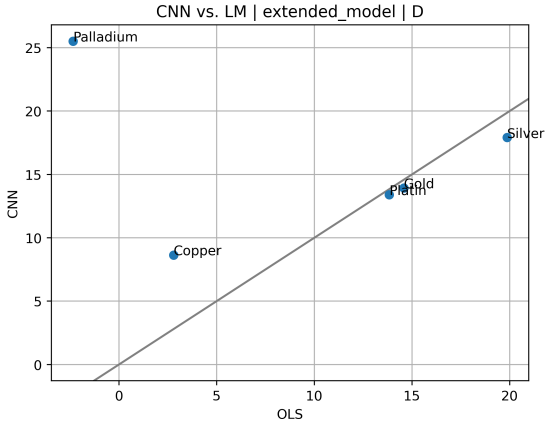
7.4 Results

While *LSTM*s did not perform reliably better than linear models, *CNN*s did so on an at least more consistent (i.e. lower observed variance) level. For the majority of resampling periods and data models, *CNN* outperformed both *OLS* and *LSTM*. With a few exceptions, CNN also produced more reliable results with mostly positive correlations of out of sample predictions with real-world returns.

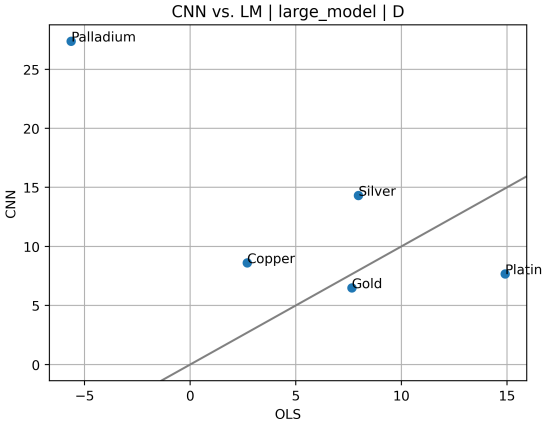
7.4.1 Daily Rebalancing



(a) Core Model



(b) Extended Model

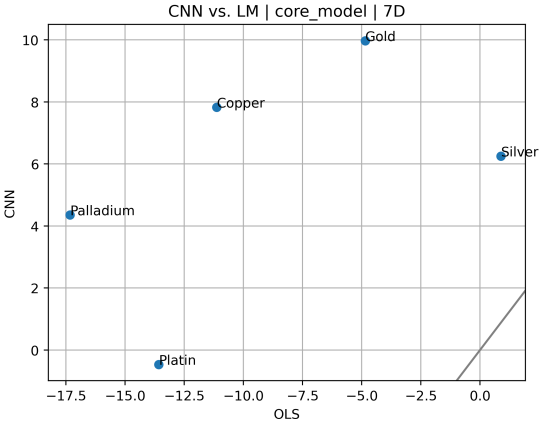


(c) Large Model

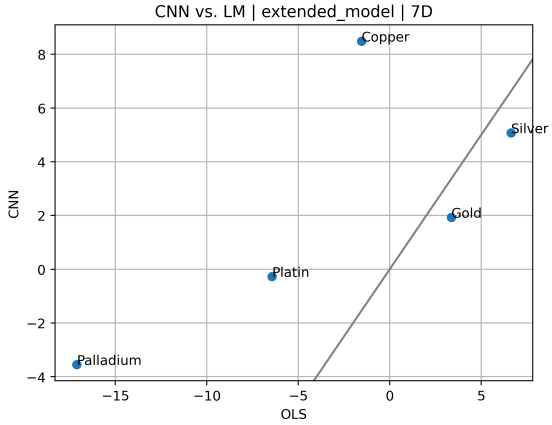
Figure 12: CNN Benchmarks on daily data

Daily asset allocation is clearly better done with *OLS* on this dataset. Adding more data seems to have an ambiguous effect on predictive power for *CNN* but clearly imposes a deterioration of *OLS*' performance.

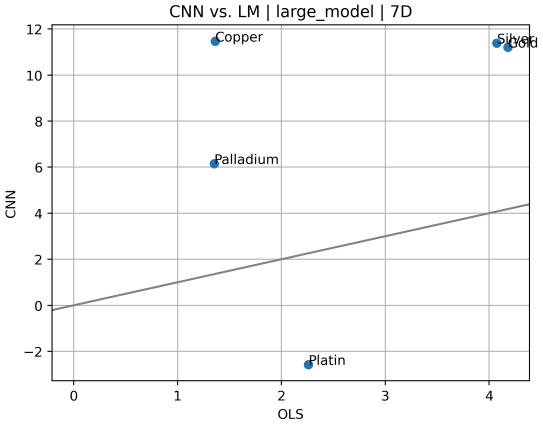
7.4.2 Weekly Rebalancing



(a) Core Model



(b) Extended Model

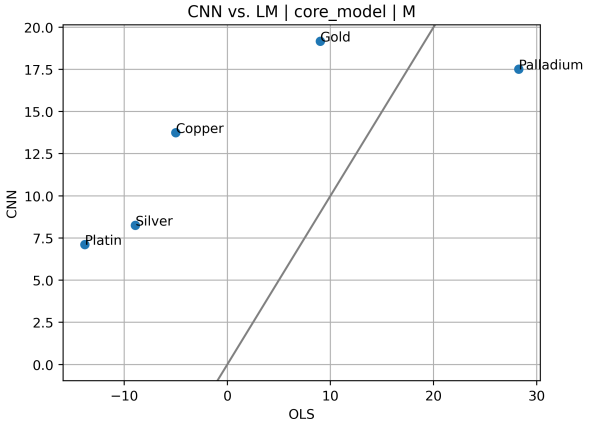


(c) Large Model

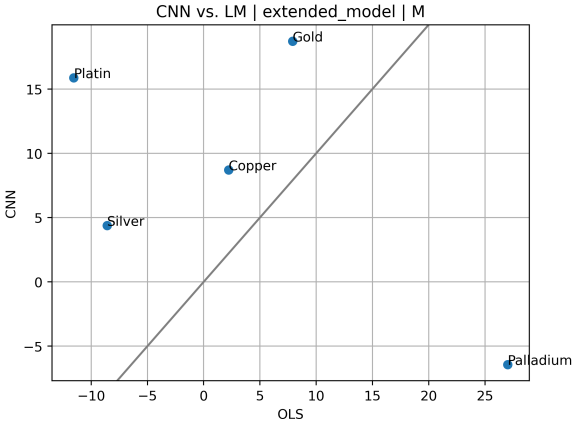
Figure 13: CNN Benchmarks on weekly data

The *Large Model* overall yields the best results, with *CNN* outperforming *OLS* on all commodities but Platinum. Again, the *Extended Model* seems to deteriorate predictive power while the *Large Model* enhances it overall, which is some continuation of what we have seen previously. However, *CNN* is better than *OLS* in this run with all results superseding the ones produced by *OLS* on the *Core Model*. The latter barely even gets correct predictions while *CNN* is relatively correct with the exception of only Platinum, the only commodity which exhibits a slightly negative relation of predicted and observed returns of less than 0.1%, which is unlikely different from zero at all.

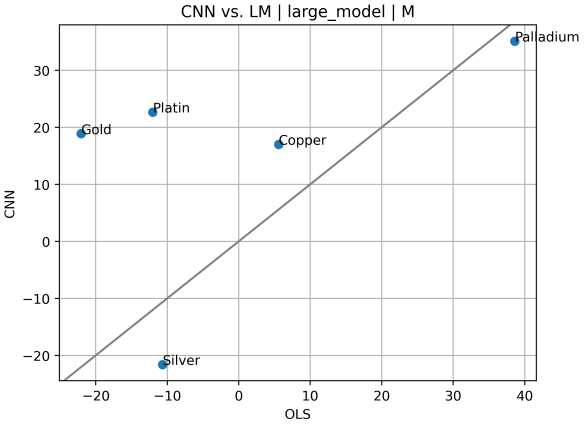
7.4.3 Monthly Rebalancing



(a) Core Model



(b) Extended Model

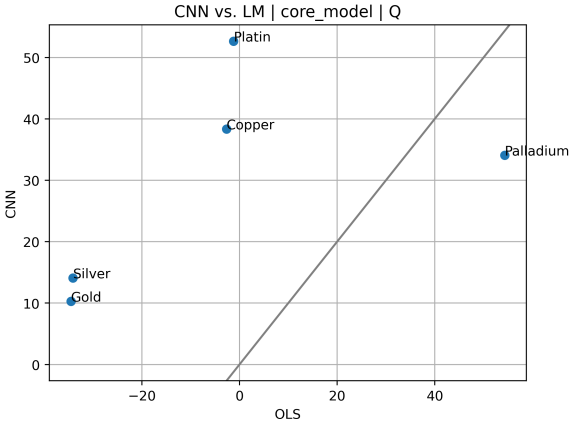


(c) Large Model

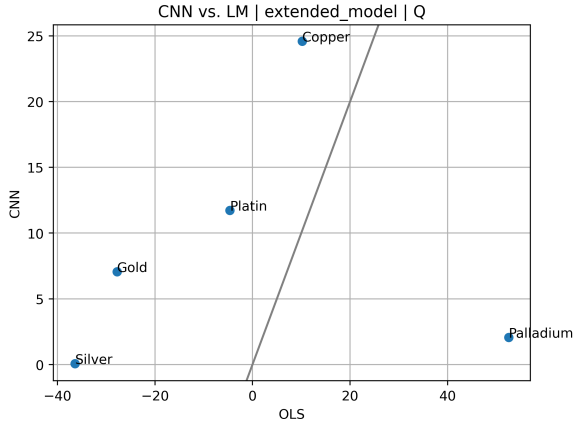
Figure 14: CNN Benchmarks on monthly data

The story is continued on monthly data, with the notable exception of Palladium which, as previously seen, works well with *OLS*. Reliability of *CNN* is drastically reduced with added data with the most consistent results being achieved on *Core Model*.

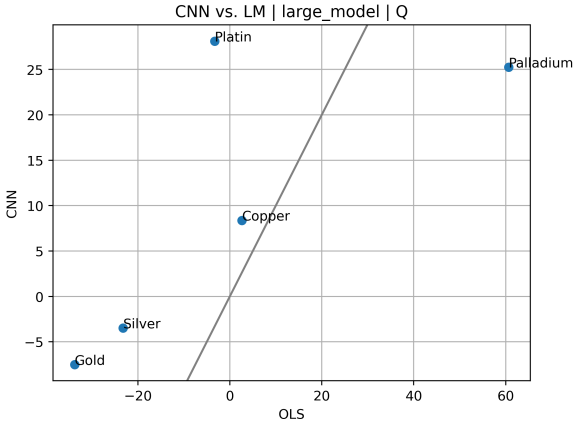
7.4.4 Quarterly Rebalancing



(a) Core Model



(b) Extended Model



(c) Large Model

Figure 15: CNN Benchmarks on quarterly data

Findings on quarterly data also do not differ much from what was already observed previously. The deterioration effects of added data are notable in this example, not only in overall variability of results, but also in a decrease of peak predictive power for *CNN*.

8 Conclusion

Evidence of Tharann (2019) could not only be confirmed on similar data taken from *Bloomberg*, *FED* and the *OECD* instead of *Datastream*, but including daily, weekly and quarterly resampling periods as well. Utilizing this investment strategy may, however, only work with cash-settled derivatives. This thesis does not attempt to backtest the findings in a realistic setting using data and trading delays, taxation issues, platform freezes during low liquidity, or physical storage cost. It also does not provide instructions on how and with what financial contracts one could make use of these findings. What it did find is some support of additional benefits from incorporating the more modern approaches *LSTM* or *CNN* into return-predictive analysis, since there are benefits to be gained from all three methods.

What is also clear after reviewing the results is that machine learning is not drastically superior or will lead to sudden improvements over existing findings over night. Implementation is cumbersome (compared with *OLS*, which is roughly 7 lines of code), training is expensive and results are heavily dependent on well-selected hyperparameters. Furthermore, the benefit of potentially getting slightly more accurate predictions is vastly outweighed by the almost complete lack of standard tests or interpretability of model parameters.

The selected set of features by Tharann (2019) delivered the overall most consistent results. While adding the additional datapoints did enhance predictability on some resampling frequencies, it mostly lead to a clear deterioration out of sample. In a productive setting, *Core Model* and therefore the original dataset is preferable.

While practitioneers and investors may definitely benefit from integrating Deep Learning into their processes, in hindsight, available interpretability is not to be underestimated. The superior feature selection process was mostly done using linear models, after all. Yet another argument on why they are as important for empirical studies as they are as of today.

References

Aruoba, S. B., Diebold, F. X. & Scotti, C. (2009). Real-time measurement of business conditions. *Journal of Business & Economic Statistics*, *27*(4), 417–427.

Bergstra, J. & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, *13*(2).

Brochu, E., Cora, V. M. & De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Brooks, R. & Yan, D. Y. (1999). London inter–bank offer rate (libor) versus treasury rate: Evidence from the parsimonious term structure model. *The Journal of Fixed Income*, *9*(1), 71–83.

Brown, D. P. & Jennings, R. H. (1989). On technical analysis. *The Review of Financial Studies*, *2*(4), 527–551.

Bruce, C., Desimone, R. & Gross, C. G. (1981). Visual properties of neurons in a polysensory area in superior temporal sulcus of the macaque. *Journal of neurophysiology*, *46*(2), 369–384.

Chollet, F. et al. (2015). *Keras.* `https://keras.io`.

Cochrane, J. H. (2008). The dog that did not bark: A defense of return predictability. *The Review of Financial Studies*, *21*(4), 1533–1575.

Danko, Z. (2015). *Neon prescription... or rather, new transcription for google voice.* Retrieved from `https://blog.google/products/google-voice/neon-prescription-or-rather-new/` (Accessed: 2020-06-01)

Durand, R. B., Lim, D. & Zumwalt, J. K. (2011). Fear and the fama-french factors. *Financial Management*, *40*(2), 409–426.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179–211.

Fama, E. F. & French, K. R. (1992). The cross-section of expected stock returns. *Journal of Finance*, *47*(2).

Frazier, P. I. (2018). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, *1*(2), 119–130.

Gers, F. A., Schmidhuber, J. & Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, *12*(10), 2451–2471.

Gibson, G. (1889). The stock markets of london. *Paris and New York, GP Putnam's Sons, New York*.

Grossman, S. J. & Stiglitz, J. E. (1980). On the impossibility of informationally efficient markets. *The American economic review*, *70*(3), 393–408.

Gunasekarage, A. & Power, D. M. (2002). The post–announcement performance of dividend–changing companies: The dividend–signalling hypothesis revisited. *Accounting & Finance*, *42*(2), 131–151.

Hamilton, W. P. (1922). The stock market barometer: a study of its forecast value based on charles h. *Dow's theory of the price movement. Barron's, New York*.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hodges, P., Hogan, K., Peterson, J. R. & Ang, A. (2017). Factor timing with cross-sectional and time-series predictors. *The Journal of Portfolio Management*, *44*(1), 30–43.

Hou, D. & Skeie, D. R. (2014). Libor: Origins, economics, crisis, scandal, and reform. *FRB of New York Staff Report*(667).

LeCun, Y., Haffner, P., Bottou, L. & Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision* (pp. 319–345). Springer.

Lettau, M. & Ludvigson, S. (2001). Consumption, aggregate wealth, and expected stock returns. *the Journal of Finance*, *56*(3), 815–849.

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E. & Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*.

MacKinlay, A. C. (1997). Event studies in economics and finance. *Journal of economic literature*, *35*(1), 13–39.

Malkiel, B. G. & Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, *25*(2), 383–417.

Markov chain. (n.d.). Retrieved from `https://www.lexico.com/en/definition/markov_chain` (Accessed: 2021-07-11)

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

Nogueira, F. (2014). *fmfn/bayesianoptimization.* `https://github.com/fmfn/BayesianOptimization`. GitHub.

Nvidia nemo: A toolkit for conversational ai. (2019). NVIDIA Corp. Retrieved from `https://github.com/NVIDIA/NeMo` (Accessed: 2020-12-07)

Pascanu, R., Mikolov, T. & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310–1318).

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., ... Vesely, K. (2011, December). *The kaldi speech recognition toolkit.* IEEE Signal Processing Society. (IEEE Catalog No.: CFP11SRW-USB)

Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer school on machine learning* (pp. 63–71).

Sato, T., Kawamura, T. & Iwai, E. (1980). Responsiveness of inferotemporal single units to visual pattern stimuli in monkeys performing discrimination. *Experimental Brain Research*, *38*(3), 313–319.

Shamsher, S. (2021). Financialisation of commodities–empirical evidence from the indian financial market. *IIMB Management Review*.

Shiller, R. C. (2000). Irrational exuberance. *Philosophy and Public Policy Quarterly*, *20*(1), 18–23.

Srinivasamurthy, R. S. (2018). *Understanding 1d convolutional neural networks using multiclass time-varying signals.* `https://tigerprints.clemson.edu/cgi/viewcontent.cgi?article=3918&context=all_theses`. Tigerprints.

Stanford, S. (2019). *Deepmind's ai, alphastar showcases significant progress towards agi.* Retrieved from `https://medium.com/@towardai/deepminds-ai-alphastar-showcases-significant-progress-towards-agi-93810c94fbe9` (Accessed: 2020-12-07)

Tharann, B. (2019). Return predictability in metal futures markets: new evidence. *Heliyon*, *5*(6), e01843.

Warner, E. J. & Barsky, R. B. (1995). The timing and magnitude of retail store markdowns: evidence from weekends and holidays. *The Quarterly Journal of Economics*, *110*(2), 321–352.

Welch, I. & Goyal, A. (2008). A comprehensive look at the empirical performance of equity premium prediction. *The Review of Financial Studies*, *21*(4), 1455–1508.

Wooldridge, J. M. (2015). *Introductory econometrics: A modern approach.* Cengage learning.

A Appendix

A.1 Benchmarking: Full Results

	Frequency	Data Model	Commodity	LSTM	CNN	LM
0	Daily	core_model	Copper	7.855765	1.231855	-1.731914
1	Daily	core_model	Gold	14.263361	13.654949	15.622235
2	Daily	core_model	Palladium	10.223678	19.455670	8.968081
3	Daily	core_model	Platin	4.903180	11.281590	13.959542
4	Daily	core_model	Silver	-3.264479	2.837894	23.492696
5	Daily	extended_model	Copper	5.986084	8.642791	2.798055
6	Daily	extended_model	Gold	11.180034	13.931827	14.588291
7	Daily	extended_model	Palladium	20.201646	25.519845	-2.348762
8	Daily	extended_model	Platin	7.661787	13.395938	13.835638
9	Daily	extended_model	Silver	-9.516964	17.909791	19.865214
10	Daily	large_model	Copper	-5.407870	8.603837	2.698088
11	Daily	large_model	Gold	-7.775665	6.496181	7.652523
12	Daily	large_model	Palladium	2.235383	27.380199	-5.642145
13	Daily	large_model	Platin	3.342933	7.671563	14.905031
14	Daily	large_model	Silver	2.879581	14.311927	7.960900
15	Monthly	core_model	Copper	19.283991	13.741453	-4.992170
16	Monthly	core_model	Gold	12.750192	19.170700	9.017649
17	Monthly	core_model	Palladium	19.953426	17.512982	28.238662
18	Monthly	core_model	Platin	13.536538	7.107797	-13.820295
19	Monthly	core_model	Silver	10.970230	8.258135	-8.920266
20	Monthly	extended_model	Copper	3.800193	8.698888	2.219047
21	Monthly	extended_model	Gold	21.979888	18.722205	7.908760
22	Monthly	extended_model	Palladium	2.459666	-6.426083	27.009444
23	Monthly	extended_model	Platin	-7.765537	15.867141	-11.548875
24	Monthly	extended_model	Silver	-18.837286	4.386507	-8.588241
25	Monthly	large_model	Copper	-12.493132	16.977104	5.575502
26	Monthly	large_model	Gold	28.819371	18.873808	-22.045190
27	Monthly	large_model	Palladium	45.808909	35.108402	38.603278
28	Monthly	large_model	Platin	19.747191	22.644017	-12.033417
29	Monthly	large_model	Silver	-0.816430	-21.602102	-10.659283

Table 4: Full benchmarking results (I/II)

	Frequency	Data Model	Commodity	LSTM	CNN	LM
30	Quarterly	core_model	Copper	20.296689	38.328777	-2.636376
31	Quarterly	core_model	Gold	10.828894	10.299016	-34.507260
32	Quarterly	core_model	Palladium	47.531552	34.061132	54.301352
33	Quarterly	core_model	Platin	-30.721093	52.659395	-1.201376
34	Quarterly	core_model	Silver	-16.565804	14.108166	-34.114770
35	Quarterly	extended_model	Copper	-8.126543	24.606863	10.203632
36	Quarterly	extended_model	Gold	0.000000	7.069595	-27.766865
37	Quarterly	extended_model	Palladium	29.884244	2.068325	52.513854
38	Quarterly	extended_model	Platin	45.296327	11.723309	-4.644460
39	Quarterly	extended_model	Silver	-15.636853	0.067011	-36.393122
40	Quarterly	large_model	Copper	-5.441586	8.377852	2.644165
41	Quarterly	large_model	Gold	-4.350958	-7.500744	-33.734450
42	Quarterly	large_model	Palladium	60.262002	25.257954	60.638827
43	Quarterly	large_model	Platin	0.682452	28.122548	-3.316760
44	Quarterly	large_model	Silver	-14.435129	-3.468093	-23.223678
45	Weekly	core_model	Copper	-7.047141	7.827148	-11.127144
46	Weekly	core_model	Gold	7.283705	9.973899	-4.847711
47	Weekly	core_model	Palladium	10.418275	4.358027	-17.331356
48	Weekly	core_model	Platin	8.502982	-0.464822	-13.576997
49	Weekly	core_model	Silver	0.544852	6.248484	0.884102
50	Weekly	extended_model	Copper	-15.675575	8.489647	-1.547097
51	Weekly	extended_model	Gold	9.625003	1.932030	3.370226
52	Weekly	extended_model	Palladium	22.109509	-3.546229	-17.106414
53	Weekly	extended_model	Platin	5.617370	-0.272297	-6.431496
54	Weekly	extended_model	Silver	15.278584	5.077101	6.629064
55	Weekly	large_model	Copper	-1.073381	11.471302	1.361617
56	Weekly	large_model	Gold	0.000000	11.203386	4.179197
57	Weekly	large_model	Palladium	-13.921071	6.148763	1.352354
58	Weekly	large_model	Platin	-2.259976	-2.571379	2.259181
59	Weekly	large_model	Silver	15.274339	11.382509	4.072999

Table 5: Full benchmarking results (II/II)

A.2 Implementation

Python was the language of choice for both linear inference as well as implementing the neural architectures. The goal was to create a maintainable framework for deep commodity predictability, although in practice its application is limited mostly by the insufficient access to a live data API. All neural architecture is build upon the underlying tensorflow-framework v2.5.0 published and maintained by Google, while linear inference is implemented via statsmodels, an open source Python library. The most critical part, however, is the implementation of consistent datasets across all models to ensure their comparability. For this purpose, the classes *Data* as well as *DataModel* were built in a reusable and scalable context to make handling data convenient and reliable across the project.

A.3 The Data Class

Data is the class intended to import the *Bloomberg* datasets into *Pandas Dataframes*. It was developed in order to

1. define a standardized data format,
2. efficiently import the at times large source CSVs (*ADS*-Index),
3. deal with the inconsistent datasets and ensure consistency.

The data class only needs to be imported once and may then be passed on to DataModel in order to obtain standardized predictors and dependent variables. Efficient importing is tackled by using separate processes for each dataset which connect to temporary functions that read and pre-process the source files. These functions are the only ones which need replacing should the models ever be used with a live API. The data class furthermore allows removing of individual assets should they not be needed in the central config file. Excluded commodities were

1. Steel rebar, since it dates back only to 03/2009 and is therefore insufficient to yield a good estimation,
2. Aluminium, since it unsystematically contained longer periods of zero price movements, possibly indicating a temporary trade suspension or de-listing.

Another important issue were the inconsistent date-indices which required alignment for each individual dataset. The prices and total values were forward-filled in order to make the datasets machine-operable. Consistent train-, development-, and test-datasets are created by saving hard limits of different time-periods at the pre-configured levels. The decision for this procedure was mainly driven by the fact that the underlying data exhibited multiple structural breaks for all included commodities. Any model trained on past data must be able to work on new data as well. Using integers with a fixed seed for a shuffled time series is of course possible as well, though it should be noted that both the recurrent as well as the convolutional neural architectures obtain past datapoints as features as well, making bootstrapping and shuffling not really a feasible option on dataset-level. The output of the data class is finally a dictionary of dataframes for each subset within the datapool (such as commodities, equities, government bonds, etc...) as well as class- and therefore project-wide limits for train-, dev-, and test-set.

A.4 The Data Model Class

DataModel is the second most important class throughout the project and has one purpose only: to make pre-processing data as convenient and re-usable as possible. Its primary reason is to separate up to three different data models: *Core Model*, *Extended Model*, and *Large Model*. The final output is a consistent dataset packaged as a dictionary of dataframes with the respective keys x for the predictive variables and y for the dependent variables. DataModel furthermore allows for flexible transformation to analyze different portfolio re-allocation periods. Supported are daily, weekly, monthly and quarterly re-balancing, whereby daily re-balancing uses log returns over last and opening prices to account for a trading delay in between. Finally, a container object collects all data produced by any model in a systematic way throughout this thesis.

A.5 LSTM

A reusable class *Lstm* is available in *lstm.py*, which ensures consistency across all compared data models and resampled periods. It takes the already discussed *Data*-class as its input in the init-function and computes the three available data models from this component, which then become available internally. This is especially important for consistent scalability, as each *Lstm*-instance can then be remotely passed on to its own sub-processes container without needing access to global data models. If this were not the case, interferences with other working processes could be caused in shared memory between individual parallel processes.
Lstm may upon calling it:

1. Prepare the data in the correct format
2. Train the model for each commodity
3. Infer predictions from supplied data after training
4. Return pre-trained models for later use
5. Return a complete history of the training process

A.5.1 Scalable Hyperparameter Design

Neural networks are generally neither unique nor do they produce consistent results for different commodities on the same set of features. As a result, a major part of developing these architectures is to decide upon an optimal set of hyperparameters. This requires intensive computing, a lot of patience and ideally a good algorithm. For *Lstm*, configurable parameters include the following:

1. *lags* [integer]: How many past observations should be included?
2. *epochs* [integer]: Max-cap for epochs to be trained for.
3. *seed* [integer]: Due to the potential pitfall of local optimization, initializing the models weights close to the optimum is one of the most critical aspects to consider. Setting an appropriate seed makes the model not only perfectly reproducible, but it also allows for comparing different predictable powers to identify the optimal initialization.

4. *optimizer* [string]: The optimizer defines the algorithm which performs the gradient descent. For now, this is only Stochastic Gradient Descent (with momentum), primarily for performance reasons. True gradient descent is slow and computationally intensive. To speed it up, SGD performs gradient descent on a random subset only. Momentum helps accelerating the descent further by either looking ahead or applying moving averages to the gradients. The implemented version in this thesis is:

$$velocity_t = momentum * velocity_{t-1} - learning\_rate_t * gradient_t$$
$$weights_{t+1} = weights_t + velocity_t$$

5. *activation* [string]: The activation function for the *LSTM*s output block. Typically, this is *ELU*.

6. *recurrent_activation* [string]: The activation function for the recurrent block. As already indicated above, *TANH* usually delivered empirically superior results to *ELU*. However, this differs from model to model.

7. *use_bias* [boolean]: Indicates if the layer will utilize a bias (a fixed fector) or not. There were only a few models who benefitted from not using a bias.

8. *bias_initialzer* [string]: Determines the initial state of the bias vectors. For most models, setting it to zero turned out to be a good choice.

9. *kernel_initializer* [string]: Determines the initial weights for the *LSTM*s kernel (the non-recurrent components in f_t, i_t, o_t and c_t^i from the compact form). For almost all models, a random normal distribution was selected.

10. *recurrent_initializer* [string]: Determines the initial weights for the *LSTM*s recurrent blocks. (the recurrent components in f_t, i_t, o_t and c_t^i from the compact form). For the majority of models, initializing them orthogonally (eigenvalues from any orthogonal tensors are 1) produced reliable results.

11. *dropout* [float]: The percentage of dropout to be applied to ne non-recurrent neurons. This turned out to be quite different for each model and required a lot of attention.

12. *recurrent_dropout* [float]: The dropout percentage for the recurrent layers. This was an equally mixed situation as for the dropout layers, although models which had already a dropout seemed to benefit from a recurrent dropout as well.

Many more hyperparameters can be edited as well, but they mainly alter the models input/output behavior. A comprehensive overview is available in the configuration file *hyperparams.yml*. A deeper insight to these hyperparameters can be gained by looking at the source code for and the official documentation of *Tensorflow2/Keras* by Chollet et al. (2015).

A.5.2 Training: An Iterative Cycle

Training the model and tuning the hyperparameters proved to be the most challenging and time-consuming task. Each model had to be trained for:

1. Each considered data model.
2. Each considered resampling-/reallocation frequency.
3. Each commodity.
4. Each hyperparameter.
5. Each possible value for any given hyperparameter.

Thus, each tuning loop required at least 5 commodities * 3 data models * 4 resampling frequencies * 12 hyperparameters = 720 cycles. This is a nonviable task without the help of some automation. Therefore the class *ScaleHyperparameters* was created. It is supplied with a predefined selection of hyperparameter candidates which it loops over autonomously. Using either the validation data or random subsets of both train- and validation data, simple performance metrics, namely the empirical correlation, euclidic distance and absolute percentage error were evaluated after each implemented candidate. Ultimately, the best performing hyperparameter was then saved in *Yaml* (or *Json*, alternatively) format as a starting point for manual investigation. This hyperparameter scaling is only tested on *Linux* since the parent thread lasts too long for a *Windows*-based machine and is terminated after a set amount of time has passed or simply gets stuck. Furthermore, ECC-RAM is recommended, since system RAM-errors can break Python's multiprocessing queues (although this problem is a rare occurrence). A complete training cycle on 16 parallelized instances required up to 24 hours. The GPU should not

be used for training since it limits parallel training capabilities more than it enhances individual threat speed (unless a large number of GPUs is utilized)
After each completed cycle, the models were trained and extensively tested by hand using the corresponding two other performance metrics to obtain a better assessment. It is important to mention in this context that the testing set itself remained untouched. Hyperparameters that were deemed worthy were then permanently added into *lstm_hyperparams.yml*.

A.6 CNN

All available hyperparameters for each model can be found in *cnn_hyperparameters.yml* config-file in *yaml* standard. The most important ones shall be discussed in this chapter:

1. *kernel_size* [integer]: Length of the *Conv1D*-window. Also specifies the amount of included lagged observations for all features which is required for the *causal* padding as described above.

2. *epochs* [integer]: Upper-limits the amount of passes over the data during the training process. Per default, this is set to 100,000 passes. Note that early stopping is applied which likely ends the training process prematurely.

3. *seed* [integer]: Random initialization of he weights. Most important for getting a good starting point before the training process is invoked to avoid optimizing local instead of global minima.

4. *optimizer* [string]: Optimizer for the gradient descent. Currently, only *SGD* is supported, but any other option may be added with by replacing the *opt*-variable in *model.compile()*.

5. *filters* [integer]: Dimensionality of the *Conv1D* output space.

6. *strides* [integer]: Strides corresponding to s above, fixing the total stride length. Since this i a *Conv1D*-model on a 2D source, only one-dimensional strides are supported. Furthermore, if $strides \neq 1$ then $dilation\_rate = 1$ et vice versa.

7. *padding* [string]: While it can be changed, note that only *causal* makes sense in the context of this application.

8. *use_bias* [boolean]: Use a fixed vector bias in the *Conv1d* layer. Most models benefitted from it.

9. *kernel_initializer* [string]: Applies weight initialization to all weights within the *Conv1D* layer according to the specified seed if set to a random distribution. Most models empirically benefited from a glorot-normal initialization (truncated normal distribution).

10. *bias_initializer* [string]: Initializer for the bias vector. Applies only if bias is set to *True*. Per default this is configured to be zero.

A.7 Data

	Commodities	Equity	GovBond
0	GC1 Comdty/PX_OPEN	SPX Index/TOT_RETURN_INDEX_GROSS_DVDS	USGG3M Index/PX_OPEN
1	GC1 Comdty/PX_HIGH	SPX Index/DVD_PAYOUT_RATIO	USGG3M Index/PX_HIGH
2	GC1 Comdty/PX_LOW	SPX Index/IDX_EST_DVD_CURR_YR	USGG3M Index/PX_LOW
3	GC1 Comdty/PX_LAST	SPX Index/GROSS_AGGTE_DVD_YLD	USGG3M Index/PX_LAST
4	GC1 Comdty/PX_VOLUME	SPX Index/RETURN_COM_EQY	USGG6M Index/PX_OPEN
5	SI1 Comdty/PX_OPEN	SPX Index/RETURN_ON_ASSET	USGG6M Index/PX_HIGH
6	SI1 Comdty/PX_HIGH	SPX Index/IDX_EST_PRICE_BOOK	USGG6M Index/PX_LOW
7	SI1 Comdty/PX_LOW	SPX Index/PX_TO_EBITDA	USGG6M Index/PX_LAST
8	SI1 Comdty/PX_LAST	SPX Index/PX_VOLUME	USGG12M Index/PX_OPEN
9	SI1 Comdty/PX_VOLUME	SPXT Index/PX_OPEN	USGG12M Index/PX_HIGH
10	PL1 Comdty/PX_OPEN	SPXT Index/PX_HIGH	USGG12M Index/PX_LOW
11	PL1 Comdty/PX_HIGH	SPXT Index/PX_LOW	USGG12M Index/PX_LAST
12	PL1 Comdty/PX_LOW	SPXT Index/PX_LAST	USGG2YR Index/PX_OPEN
13	PL1 Comdty/PX_LAST		USGG2YR Index/PX_HIGH
14	PL1 Comdty/PX_VOLUME		USGG2YR Index/PX_LOW
15	PA1 Comdty/PX_OPEN		USGG2YR Index/PX_LAST
16	PA1 Comdty/PX_HIGH		USGG3YR Index/PX_OPEN
17	PA1 Comdty/PX_LOW		USGG3YR Index/PX_HIGH
18	PA1 Comdty/PX_LAST		USGG3YR Index/PX_LOW
19	PA1 Comdty/PX_VOLUME		USGG3YR Index/PX_LAST
20	HG1 Comdty/PX_OPEN		USGG5YR Index/PX_OPEN
21	HG1 Comdty/PX_HIGH		USGG5YR Index/PX_HIGH
22	HG1 Comdty/PX_LOW		USGG5YR Index/PX_LOW
23	HG1 Comdty/PX_LAST		USGG5YR Index/PX_LAST
24	HG1 Comdty/PX_VOLUME		USGG10YR Index/PX_OPEN
25	RBT1 Comdty/PX_OPEN		USGG10YR Index/PX_HIGH
26	RBT1 Comdty/PX_HIGH		USGG10YR Index/PX_LOW
27	RBT1 Comdty/PX_LOW		USGG10YR Index/PX_LAST
28	RBT1 Comdty/PX_LAST		USGG30YR Index/PX_OPEN
29	RBT1 Comdty/PX_VOLUME		USGG30YR Index/PX_HIGH
30	AL1 Comdty/PX_OPEN		USGG30YR Index/PX_LOW
31	AL1 Comdty/PX_HIGH		USGG30YR Index/PX_LAST
32	AL1 Comdty/PX_LOW		
33	AL1 Comdty/PX_LAST		
34	AL1 Comdty/PX_VOLUME		

	CorpBond	Macro	SmartBeta
0	SP5IGBIT Index/PX_OPEN	GDP CUR$ Index/PX_LAST	MXWO Index/PX_OPEN
1	SP5IGBIT Index/PX_HIGH	CPI YOY Index/PX_LAST	MXWO Index/PX_HIGH
2	SP5IGBIT Index/PX_LOW	EUR Curncy/PX_LAST	MXWO Index/PX_LOW
3	SP5IGBIT Index/PX_LAST	IP YOY Index/PX_LAST	MXWO Index/PX_LAST
4	SP500BDT Index/PX_OPEN	CBOPGAPN Index/PX_LAST	MXWO000V Index/PX_OPEN
5	SP500BDT Index/PX_HIGH		MXWO000V Index/PX_HIGH
6	SP500BDT Index/PX_LOW		MXWO000V Index/PX_LOW
7	SP500BDT Index/PX_LAST		MXWO000V Index/PX_LAST
8	SP5HYBIT Index/PX_OPEN		MXWOQU Index/PX_OPEN
9	SP5HYBIT Index/PX_HIGH		MXWOQU Index/PX_HIGH
10	SP5HYBIT Index/PX_LOW		MXWOQU Index/PX_LOW
11	SP5HYBIT Index/PX_LAST		MXWOQU Index/PX_LAST
12			MXWOMOM Index/PX_OPEN
13			MXWOMOM Index/PX_HIGH
14			MXWOMOM Index/PX_LOW
15			MXWOMOM Index/PX_LAST
16			MXWOSZT Index/PX_OPEN
17			MXWOSZT Index/PX_HIGH
18			MXWOSZT Index/PX_LOW
19			MXWOSZT Index/PX_LAST
20			MXWDHDVD Index/PX_OPEN
21			MXWDHDVD Index/PX_HIGH
22			MXWDHDVD Index/PX_LOW
23			MXWDHDVD Index/PX_LAST
24			M1WOMVOL Index/PX_OPEN
25			M1WOMVOL Index/PX_HIGH
26			M1WOMVOL Index/PX_LOW
27			M1WOMVOL Index/PX_LAST

Table 6: The *Bloomberg* Dataset

The table above contains all data fields which were exported from *Bloomberg L.P.*

Data about unemployment in the US is taken from the following *OECD* website: `https://data.oecd.org/unemp/unemployment-rate.htm`

Data about inflation on consumer price indices is taken from the following *OECD* website: `https://data.oecd.org/price/inflation-cpi.htm`

The ADS-Index is provided by the *FED*: `https://www.philadelphiafed.org/surveys-and-data/real-time-data-research/ads`

All sites were available on the 1st September 2021.

A.8 Source Code

The GitLab repository for this thesis can be found at and cloned from `https://www.gitlab.com/achwalt/MasterThesisCode.git`. Please contact the author to gain access to the source data.